The following paper was originally published in the
Proceedings of the Second USENIX Workshop on Electronic Commerce
Oakland, California, November 1996

Agora: A Minimal Distributed Protocol for Electronic Commerce

Eran Gabber and Abraham Silberschatz
Bell Labs

Agora: A Minimal Distributed Protocol for Electronic Commerce

Eran Gabber and Abraham Silberschatz
Bell Laboratories
700 Mountain Ave.
Murray Hill, NJ 07974

{*eran, avi*}*@research.bell-labs.com*

Abstract

Agora[1] is a Web protocol for electronic commerce, which is intended to support high-volume of transactions each with low incurred cost. Agora has the following novel properties:

- *Minimal.* The incurred cost of Agora transactions is close to free Web browsing, where cost is determined by the number of messages.
- *Distributed.* Agora is fully distributed. Merchants can authenticate customers without access to a central authority. Customers with valid accounts can purchase from any merchant without any preparations (such as prior registration at the merchant or at a broker).
- *On-line arbitration.* An on-line arbiter can settle certain customer/merchant disputes.
- *Fraud control.* Agora can limit the degree of fraud to a pre-determined (low) level.

Agora is authenticated, secure and can not be repudiated. It can use regular (insecure) communication channels.

1 Introduction

The wide spread use of electronic commerce on the World Wide Web will require mechanisms for dealing with high volume of low-priced transactions. By low-priced transactions we mean transactions of low monetary value, which is close to or lower than the cost incurred in communicating with a bank. Thus, low-priced transactions implies that merchants can not afford to communicate with the bank for every transaction. High transactions volume implies distributed protocols, since any centralized resource will become a bottleneck.

In our terminology, *banks* are financial institutes that manage *customer* and *merchant* accounts. Banks provide transaction processing capabilities similar to current credit card companies. Merchants are vendors that would like to sell goods electronically. Customers are users (people, robots, etc.) that would like to purchase goods electronically. In our scheme, we assume that banks are trusted entities, while customers and merchants are assumed to be non-trusted entities. In case of a dispute, a fourth type of entity, *arbiters*, are used to settle the dispute between customers and merchants. Arbiters are assumed to be trusted.

The World Wide Web requires a new class of protocols (Web commerce protocols), that are optimized for purchase of page like merchandise (e.g., Web pages and information that can be represented as Web pages). A Web commerce protocol should be comparable to free Web browsing (in terms of message overhead) in order to achieve low incurred transaction cost.

We assume that the number of messages, rather than message length, determines the overhead (and cost) of a protocol. In other words, the optimization criteria should be number of messages and not message length, provided that message length remains within some reasonable bounds. For example, in TCP or UDP protocols, the overhead of a short message is similar to a longer message that fits into the same number of IP packets.

Agora is a fully distributed commerce protocol that requires a total of four messages to complete a transaction (including transfer of goods) without any access to a central authority. When Agora is

[1] Agora is an ancient Greek marketplace. It is also the name of the Israeli coin of the lowest denomination.

employed as a Web commerce protocol, its messages can be piggybacked on existing HTTP messages without any additional messages. Hence, Agora is *minimal*, since it generates the same number of message as free Web browsing.

Agora is a credit based protocol; that is, a transaction involves transfer of an account identifier and not the actual funds. Merchants use account identifiers in order to get the funds from the bank.

Fully distributed credit based protocols are susceptible to certain types of fraud, since the bank is not contacted during the execution of every transaction. For example, customers may exceed their credit limit, and since the bank is not involved in every transaction, the transaction will be approved. As another example, if a thief gets hold of the identification string and the private key of a legitimate customer, the thief can go on an unrestricted electronic shopping spree. To overcome this inherent flaw, we present in Section 5 the enhanced Agora protocol, that uses a probabilistic algorithm to limit the amount of fraud to a pre-determined (low) level. The overhead of the enhanced Agora protocol is similar to the basic Agora protocol.

We implemented a prototype of Agora using standard Web tools: a Netscape browser and an NCSA server. The customer side was implemented by a Java applet, and the merchant side was implemented by a set of CGI scripts and a small database.

The reminder of the paper is organized as follows. Section 2 describes work on related electronic commerce protocols. Section 3 discusses the basic Agora protocol while Section 4 discusses its properties. Section 5 describes the enhanced Agora protocol, which can limit fraud to a known (low) level. Section 6 discusses the on-line arbitration protocol, which can settle certain customer/merchant disputes. In Section 7 we discuss several limitations of the Agora protocol. Section 8 discusses scalability issues. The paper concludes with a description of the implementation. The Appendix describes the embedding of the Agora protocol in HTTP Messages.

2 Related Work

Millicent [4] is a distributed, low-overhead, digital cash protocol. Its properties are similar to Agora. Millicent introduces a *scrip*, which is a digital money that is honored by a single vendor. Millicent is designed to support high volume of low-priced transactions. Millicent's overhead is similar to Agora when used for Web commerce. However, Millicent requires preparation before a purchase from a new vendor, namely, purchasing a scrip from a broker. These preparations can involve as many as 6 messages. Another complication is handling of change. Normally, the value of the scrip is higher than the price of the goods. Thus the merchant returns the change to the user in the form of another scrip. Since this scrip is honored only by issuing merchant, the customer is forced to redeem it by the broker.

NetBill [2] is a robust electronic commerce protocol that provides atomicity (customer pays for delivered merchandise only) and anonymity via pseudonyms. However, NetBill requires 8 messages to complete a transaction, of which two are to/from the bank. Thus, NetBill is not applicable for low-priced transactions that are less expensive then the cost of contacting the bank.

The Secure Electronic Transaction (SET) protocol of Visa and MasterCard [8] is a credit based protocol, like Agora. However, SET requires communication with the bank for every transaction. That is, SET is not distributed, and its incurred cost prohibits low-priced transactions.

Digital cash protocols, such as DigiCash [1], provide total anonymity. However, these protocols were not designed for Web commerce, and lack proper arbitration (e.g., customer sent digital cash and didn't receive the merchandise).

The chrg-http protocol [7] is a recent Web commerce protocol for low-priced transactions. It was implemented with standard Web tools (Mosaic browser and CERN server). However, it ignores the complexity of real world digital cash and credit transactions. Instead, chrg-http assumes that merchants will bill customers at regular intervals. Customers must register with merchants before any purchase.

Since Agora uses digital signatures to authenticate messages, its does not require a secure communication channel, such as Netscape's Secure Sockets Layer (SSL) [3].

Agora implements an application layer security on top of HTTP. In this aspect it is similar to Secure HTTP (S-HTTP)[6].

3 Basic Protocol

The Agora protocol, as described in the introduction, involves four entities: banks, merchants, customers, and arbiters. Agora supports multiple banks. Each bank has a unique identifier, *Bid*, a private (secret) key K_b^s, and a public key K_b^p.

Customers and *merchants* must have an account with some bank before any sale or purchase can be

name	description	format
Cid	Customer ID	$Bid \parallel expiration\_date \parallel Cacct \parallel K_c^p$
$SCid$	Signed customer ID	$Cid \parallel S_b(H(Cid))$
Mid	Merchant ID	$Bid \parallel expiration\_date \parallel Mname \parallel Macct \parallel K_m^p$
$SMid$	Signed merchant ID	$Mid \parallel S_b(H(Mid))$

Table 1: Customer and Merchant Identification

initiated. Customer account name is denoted by $Cacct$, and merchant account name is denoted by $Macct$. Merchants are also identified by a unique public name, $Mname$, which can be their IP address or their domain name. Customers and merchants have private keys, K_c^s and K_m^s, respectively, and public keys K_c^p and K_m^p, respectively. Customers and merchants identify themselves by presenting signed IDs, which were obtained from the bank. $SCid$ is a customer's signed ID, and $SMid$ is a merchant's signed ID.

Arbiters are trusted entities (not necessarily the bank), that are responsible for settling certain disputes between customers and merchants. Arbiters have the authority to repudiate committed transactions. Each arbiter has a unique identifier, Aid, a private key K_a^s, and a public key K_a^p.

The Agora protocol uses digital signatures to sign messages. In the following discussion we will assume that we use a public key cryptography algorithm, such as RSA, for computing digital signatures. However, Agora can use any public key digital signature scheme with the appropriate changes of signature and verification operations.

The signature of message msg by entity x is denoted by $S_x(msg)$. The signature is computed by $S_x(msg) = E_k(H(msg))$, where k is the signer's private key K_x^s, E is an encryption function, and H is a secure one-way hash function, such as MD5 or SHA.

Signature $S_x(msg)$ is verified when $H(msg)$ is equal to $D_k(S_x(msg))$, where D is a decryption function, and k is the signer's public key K_x^p.

We will use the following notation for message signing:

$S_a(msg)$	Signing a message with arbiter's private key K_a^s
$S_b(msg)$	Signing a message with bank's private key K_b^s
$S_c(msg)$	Signing a message with customer's private key K_c^s
$S_m(msg)$	Signing a message with merchant's private key K_m^s

3.1 Accounts

All customers and merchants must have an account with a bank. The bank uses a *billing period*, during which customer and merchant IDs are valid. The bank generates new IDs for customers and merchants every billing period. The IDs expire by the end of the current billing period or by the end of the next period (a grace period). In this way, the bank can force customers to pay their bills on time.

The protocol assumes that all participants change their private and public keys every billing period in order to prevent brute force attacks. The bank delivers its new public key to customers and merchants together with their new IDs.

Table 1 depicts the generation of customer and merchant IDs. The notation $x \parallel y \parallel z$ will be used to denote the concatenation of x, y, and z. For example, a Cid might be: `visa:0796:0123456789:abcdef9876+deadbeef`, where ":" is a field separator, bank ID is `visa`, expiration date is `0796`, account name is `0123456789`, and customer's public key is `abcdef9876+deadbeef`.

3.2 Transactions

An Agora transaction consists of four messages with the control flow as depicted in Figure 1.

- **M0.** Customer asks for a price quotation from the merchant.
- **M1.** Merchant replies with:

$$\begin{aligned} M1 &= SMid \parallel seq \parallel price \parallel \\ &\quad S_m(H(Mid \parallel seq \parallel price)) \end{aligned}$$

  Where seq is a unique transaction ID and $price$ is the requested price of the goods. Seq must be unique for the merchant only (not among merchants).

- **M2.** Customer verifies $SMid$, checks Mid expiration date, compares $Mname$ with the expected merchant name, extracts K_m^p from Mid, and verifies the seq and $price$. Customer replies

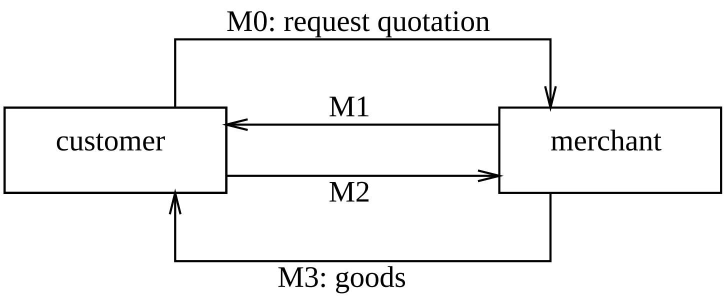

Figure 1: An Agora Transaction

with:

$$M2 = SCid \,||\, seq \,||\, price \,||\, S_c(H(Cid \,||\, Mid \,||\, seq \,||\, price)$$

- **M3.** Merchant verifies $SCid$, checks Cid expiration date, extracts K_c^p from Cid, verifies signature of seq and $price$, and compares seq and $price$ with $M1$. If $M2$ passes all tests, then merchant commits the transaction and supplies the goods to the customer. Merchant should not charge the customer for a reload of goods that were already paid for.

Merchants submit transactions to the bank for billing by the end of the billing period. The pair $M1$ and $M2$ constitute a proof of the transaction.

4 Properties

The basic Agora protocol outlined in Section 3 is minimal, distributed, authenticated, and secure. We elaborate on each of these properties below.

4.1 Minimal

A minimal Web commerce protocol should generate the same number of messages as free Web browsing. Figure 2 depicts the HTTP messages needed to access a page in the free browsing case. We will denote pages that require payment for viewing by *pay-per-view* pages. Menu pages are pages that refer to pay-per-view pages, when payment is enforced by the protocol.

Agora protocol can be piggybacked on the regular HTTP messages generated during free browsing, as depicted in Figure 3. The embedding of Agora messages in HTTP is explained below. A more detailed explanation of the embedding can be found in the Appendix.

- Message *M0* is a normal `GET` request for a menu page.
- The merchant generates *M1* messages for all pay-pay-view pages referenced by the menu. $M1$ messages are embedded in the menu in such a way that they are not displayed by the browser. See an example in the Appendix.
- When the customer decides to purchase a page, he generates *M2* from the appropriate *M1* message. The subsequent `GET` request for the pay-per-view page includes *M2* as a parameter.
- The server responds with the contents of the pay-per-view page.

Note that we start a transaction and generate a *M1* message for each pay-per-view page mentioned in the menu. If the customer doesn't select a page, the corresponding *M2* messages will never be generated. A periodic garbage collection should remove old outstanding transactions.

Agora is indeed a *minimal* protocol, since it can be completely embedded in HTTP messages. Agora does not generate any additional message compared with free Web browsing. However, some messages are longer, especially menu pages that contain multiple *M1* messages.

4.2 Distribution

Agora is fully distributed, since both customers and merchants can verify the identity of others locally using K_b^p, the bank's public key. There is no need for access to the bank, and no preparation is needed before the transaction. Merchants need to contact the bank once in the billing period to transfer accumulated transactions.

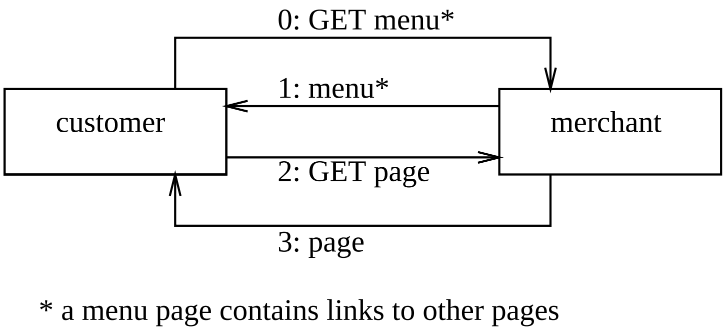

Figure 2: HTTP messages for free browsing

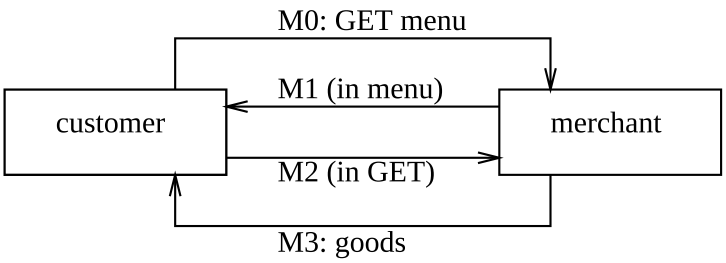

Figure 3: Piggybacking on HTTP messages

4.3 Authentication

Both $M1$ and $M2$ are signed by merchant and customer, respectively. Customers and merchants should verify signatures using K_m^p and K_c^p, which are a part of of Mid and Cid, respectively. Once $M2$ has been verified, it can not be repudiated by customer.

Customers and merchants can not create bogus accounts, since all IDs are signed by the bank's private key.

4.4 Security

Agora is secure against an adversary who can intercept, destroy, modify and replay messages. We now briefly outline various attacks by adversary and how to counter them.

- **Replay**
  An old $M2$ message can not be used for any other transaction by any merchant, since $M2$ contains a unique transaction ID and the merchant's ID. Replaying $M0$ generates a new $M1$. However, old $M2$ messages never match new $M1$s.

- **Double-Charging**
  Merchants can not double-charge customers, since each transaction has a unique sequence number seq. Double charging will be detected immediately by the bank.

- **Alteration**
  Merchants can not alter the amount charged for the goods, since the customer signed $M2$, which contains the original price.

- **Man-in-the-Middle**
  The man-in-the-middle $\mathcal{M}$ can intercept and modify all messages between customer and merchant. $\mathcal{M}$ replaces $M1$ with a new $M1$ that contains a higher price quotation, his $SMid$ and signs with his own key. Once the customer agrees to purchase, $\mathcal{M}$ submits $M2$ to the bank, pays the merchant for the original price and forwards the goods to the customer. $\mathcal{M}$ pockets the difference between the inflated price and the original price.

  Man-in-the-middle attack is impossible, since the customer can verify that $M1$ contains the expected $Mname$ in the $SMid$ part.

4.5 Anonymity

Agora can be made semi-anonymous by using account aliases and not real account names in Cid. The bank should allow customers to generate new aliases for their account in every billing period. Customer should be allowed to use several aliases. The

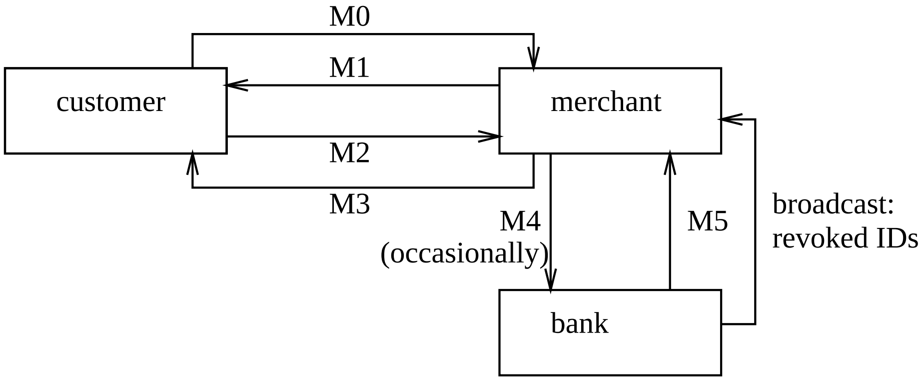

Figure 4: E-Agora

aliases should be unique, in the sense that no two customers will be allowed to use the same alias in the same billing period. Each alias should be associated with a distinct set of public and private keys. The bank will keep a translation table from aliases to real account names.

Since each customer may have several aliases, there is no single unique key that can be used for combining information about him.

Aliases provide some level of anonymity. However, full anonymity can not be achieved with current HTTP protocol, since it reveals much information to the server.

4.6 Untrusted Communication Channels

The protocol can use untrusted and unencrypted communication channels, since eavesdropping is not a security threat (see above), and any alteration of messages will be detected by verification of digital signatures.

5 Enhanced Protocol for Limiting Fraud

There are two major types of fraud that the basic Agora protocol is susceptible to: fraud by customers and fraudulent use of stolen IDs. More specific:

- Customers may exceed their credit limit by mistake or on purpose.
- Thieves may go on an electronic shopping spree with stolen $SCid$s and K_c^ss. Since the rate of electronic purchases is very high, fraudulent purchases may create a considerable loss.

Both types of fraud are due to the distributed verification of customer IDs without access to the bank.

Credit-card companies currently allow a low level of fraud. Customer liability is limited by amount L if the customer has notified the bank within time T of the theft.

The enhanced Agora protocol denoted by E-Agora, also allows a low level of fraud. The protocol uses the following parameters: L denotes the customer liability limit, T denotes the notification period, R denotes the maximal purchase rate per period T, p is a probability $0 \leq p \leq 1$, and M denotes a price threshold. Section 5.1 explains the use of p and M.

We do not consider a low rate of fraudulent purchases as a major problem, since the customer have ample time to detect it and notify the bank. Also the extent of damage is limited. A low rate is defined by purchases not exceeding threshold R per period T.

5.1 New Messages

The E-Agora protocol addresses the fraud problems by the following additions to the basic Agora. Merchants occasionally communicate with the bank, and the bank may broadcast a list of revoked Cid's to merchants. The messages and control flow of E-Agora protocol are depicted in Figure 4.

Merchants are required to maintain a current list of revoked Cids and refuse to accept any transaction belonging to a revoked Cid. A Cid is removed from the list after its normal expiration date.

A merchant contacts the bank after verification of $M2$ if either one of the following two conditions is true:

- The sum of current and previous purchases by the same customer exceeds a threshold M.

- A uniformly distributed random value r in the range $[0,1]$ is $\leq p$.

In all other cases, a merchant does *not* contact the bank during the transaction (as is done in basic Agora).

If any of the above two conditions is true, the merchant sends message $M4$ to the bank:

$$M4 = Mid \,||\, Cid \,||\, \overline{price} \,||\, S_m(Mid \,||\, Cid \,||\, \overline{price})$$

where $\overline{price}$ is the sum of current and previous purchases by the same customer, which were not already sent to the bank.

After receiving message $M4$, the bank updates the customer balance. If the customer's purchase rate exceeds rate R per period T or the customer exceeded his credit limit, the bank does not approve the transaction and also revokes Cid. Otherwise, the bank accepts the transaction.

When processing of message $M4$ is completed, the bank replies to the merchant with $M5$:

$$M5 = Cid \,||\, code \,||\, S_b(Cid \,||\, code)$$

where *code* denotes whether the bank accepted or rejected the transaction.

5.2 Batch Processing

Merchants should communicate with the bank once in every period T and transfer batches of transactions. Usually this communication will be performed at off-peak hours. In this way, customer balance in the bank is never out of date by more than T. To reduce overhead, the bank may piggyback a list of revoked Cids on any message to the merchant.

5.3 Selection of Parameters

The parameters M, R and p are derived from L and T as follows.

- $M = p \times L$
- $R = \frac{M}{T}$

The value of p is chosen so that it reduce the overhead of sending $M4$ and $M5$. p should not be too small, in order to keep M reasonable large. M should represent a transaction value that is cost effective to communicate to the bank. Also, R should be an acceptable purchase rate.

As an example of a reasonable choice of parameters, L is \$50.00, T is 24 hours, p is 0.1, M is \$5.00, and R is \$5.00 per 24 hours. Since the protocol is intended for low-priced transactions, such as 0.1 cent, these limits are acceptable.

5.4 Discussion

A thief is a person who manages to illegally get hold of some other customer's ID and private key. A thief who go on a shopping spree will be stopped as soon as he spends L or less. A shopping spree is defined as frequent purchases in a short period of time. If the thief is purchasing expensive items (value more than M), each item will be immediately deducted from the customer's account. The thief will be stopped after spending R during period T. If the thief is purchasing from a single merchant, even cheap items, he will also be detected as soon as he spends more than R. If the thief is buying less than M from multiple merchants, his purchases will be reported to the bank with probability p. He will be stopped as soon as his reported purchase rate exceeds R, which means that M purchases were reported to the bank. This is equivalent to total purchases of L.

A customer who exceeds his credit limit will be handled in a similar way. Since account balance is updated every period T by batch processing, the bank can immediately detect over the limit purchases of expensive items (more than M). If the customer purchase cheap items from many merchants, he will be detected with probability p by a merchant, which means that he will make $\frac{1}{p}$ purchases on the average before being detected. The maximal value of over the limit purchases is $\frac{M}{p} = L$.

6 On-Line Arbitration

Certain disputes may occur in the scheme we have outlined above. For example, a customer may send message $M2$, but receive partial goods or different goods than ordered. The customer may even not receive the goods at all. In this case the customer would like to get the goods or get a refund, since he already agreed to pay. A trusted arbiter can settle the above disputes. The arbiter can not settle more complicated disputes, such as false advertising by merchants. These disputes should be settled by humans. Figure 5 depicts the arbitration protocol and its control flow.

The arbitration protocol makes the following modification to E-Agora protocol:

- The merchant includes a secure hash value of the goods in $M1$.

- The customer sends $M2$ to the merchant and expects the goods. If the customer doesn't receive the goods at all, or if the checksum of the received goods is different than what appeared

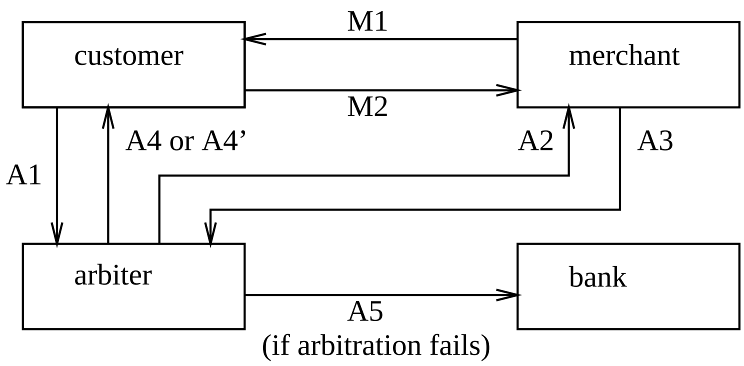

Figure 5: Arbitration Protocol

in $M1$, then the customer contacts the arbiter by sending message $A1$, where $A1 = M1 \parallel M2$.

- The arbiter asks the merchant for the goods by sending message $A2$. The arbiter signs the request, so the merchant can validate it. This prevents eavesdroppers from asking for goods that somebody else paid for.

- If the merchant replies with the correct goods in message $A3$, then the arbiter forwards it to the customer in message $A4$.

- In all other cases (merchant doesn't reply or replies with incorrect goods) the the arbiter repudiates the transaction. The arbiter sends a repudiation message $A5$ to the bank with a copy to the customer ($A4'$).

7 Caveats

The Agora arbitration protocol assumes that the goods are static, that is, their hash value do not change from the time $M1$ is sent to the customer until the goods are actually delivered. Another problem is that the arbitration protocol can not detect delivery delays of time-sensitive information, such as stock quotes. This means that the arbitration can not deal with dynamic or time-sensitive information.

Agora security is based on the security of the bank's signature. A security breach at the bank will allow an intruder to sign fraudulent IDs and confirm the resulting transactions. Since all other protocols assume that the bank can be trusted, we will also do so.

Agora generates new $SCid$ for all customers every billing period. We have to use some highly secure protocol to transfer $SCid$ and K_b^p.

Another problem is stealing $SCid$ and K_c^s from the customer. This problem exists in all other protocols too.

8 Scalability

Any practical electronic commerce protocol must handle a large number of merchants, customers and transactions. We assume that the total number of active accounts is 10^9, and 1% of all accounts are revoked every billing period (10^7 accounts). Most revocations are due to lost or misplaced account identifiers.

The main implementation complexity of the E-Agora protocol is that all merchants must maintain a current list of revoked Cids, and refuse to accept any transaction belonging to a revoked Cid. The bank can efficiently broadcast a list of revoked Cids using some Internet broadcast protocol. Merchants can easily store a list of 10^7 revoked Cids in their local disk, since this list should not consume more than 100MB per billing period. The list is cleaned by the end of the billing period.

Checking a Cid against the list of revoked Cids can be performed by an application of a Bloom filter[5]. A Bloom filter consists of m bits of memory initialized to zero and k hash transformations on Cid. All hash transformation produce an integer in the range $[0, m-1]$. Each revoked Cid is stored in the local disk, and the k hash transformations are applied on it. The corresponding bits of the filter are set to 1.

To check a Cid against the revoked list, the k hash transformations are computed. If any of the corresponding bits are *not* set, this Cid has not been revoked. If all corresponding bits in the filter are set, we have to check the local revoked list to ensure that this Cid has been indeed revoked, since there is a possibility of a *filter error*. Filter errors occur when the k bits have been set by a collision with other Cids.

The following table depicts the probability of filter errors for representative m and k. In all cases, we assume that the filter contains 10^7 revoked Cids. For computation of filter error probability, see [5].

m	k	prob.(filter error)
2^{26} bits (8MB)	4	0.040648
2^{27} bits (16MB)	8	0.001652
2^{28} bits (32MB)	8	0.000019

Since the entire Bloom filter can be stored in main memory, we can check a given Cid against the local revoked list in a constant time. I/O operations are required only for revoked IDs and infrequent filter errors.

9 Implementation

We implemented the Agora protocol using standard Web tools: a Netscape Navigator 2.0 browser and a NCSA 1.4.1 server. The customer side is implemented by a Java applet, and the merchant side is implemented by a set of CGI scripts and a small data base. There was no need to modify either the browser or server software. We also implemented a bank of limited capabilities, that can only generate signed IDs. The current version of the bank does not respond to $M4$ messages, neither it can revoke IDs. We did not implement an arbiter yet.

The implementation is straight-forward, except for the Java applet, which had a security problem and a performance problem.

A secure Java environment prevents applets from accessing local files if the applet is loaded from the network. That is, an applet can not keep local state on the customer machine. However, for our scheme, we need to keep some local state, such as balance and a transaction log in the local machine. We evaded this problem by writing a stateless applet, that contained hard coded $SCid$ and K_c^s.

Other known security problems with Netscape Navigator may allow an intruder to eavesdrop and then switch pages of the browser code or any program that the browser executes. The solution of this problem is outside the scope of the Agora protocol.

We implemented public key encryption entirely in Java (without native code), which is about 200 times slower than native C implementation. The performance will improve with JIT (Just In Time) compiler technology.

10 Conclusions

Agora is a simple, credit-based, low-overhead protocol for electronic commerce. The protocol is ideally suited for high-volume low-priced transactions, such as pay-per-view access to Web pages. The protocol can be piggybacked on current HTTP messages.

References

[1] David Chaum. Achieving Electronic Privacy. *Scientific American*, pages 96–101, August 1992.

[2] Benjamin Cox, J. D. Tygar, and Marvin Sirbu. NetBill Security and Transaction Protocol. In *First USENIX Workshop on Electronic Commerce*, New York, New York, July 11–12 1995.

[3] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol, Version 3.0. Internet draft `<draft-freier-ssl-version3-01.txt>`, March 1996. Found at `http://home.netscape.com/eng/ssl3/`.

[4] Mark S. Manasse. The Millicent Protocols for Electronic Commerce. In *First USENIX Workshop on Electronic Commerce*, New York, New York, July 11–12 1995.

[5] M. V. Ramakrishna. Practical Performance of Bloom Filters and Parallel Free-Text Searching. *Communications of the ACM*, 32(10):1237–1239, October 1989.

[6] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. Internet draft `<draft-ietf-wts-shttp-00.txt>`, July 1995. ⇒ `http://www.eit.com/creations/s-http/`.

[7] Lei Tang and Steven Low. Chrg-http: A Tool for Micropayments on the World Wide Web. In *Proceedings of the 6th USENIX Security Symposium*, San Jose, California, July 22–25 1996.

[8] Visa and MasterCard. Secure Electronic Transaction (SET) Specification, Book 2: Technical Spec-
ifications. Found at `http://www.visa.com/cgi-bin/vee/sf/set/settech.html?2+0`, February 1996.

Appendix

Embedding Agora Protocol in HTTP Messages

- Message *M0* is a normal `GET` request for a menu page.
- The merchant generates an applet activation for each reference to a pay-pay-view page in the menu. Instead of generating one $M1$ message for each reference in the page, the merchant generates a single $M1'$ message, which contains a concatenation of all transaction IDs and prices of the corresponding $M1$ messages. $M1'$ is constructed as follows (compare with Section 3.2):

  $$\begin{aligned} M1' &= SMid \,\|\, seq_1 \,\|\, price_1 \,\|\, \ldots \,\|\, seq_n \,\|\, price_n \,\| \\ &\quad S_m(H(Mid \,\|\, seq_1 \,\|\, price_1 \,\|\, \ldots \,\|\, seq_n \,\|\, price_n)) \end{aligned}$$

  Replacing multiple $M1$s with a single $M1'$ reduces the number of digital signatures without compromising the security of Agora.

  For example, a menu page contains the following references:

  ```
  A picture of
  <a href="protected/Everest.mpg" price="0.001">Mount Everest</a>.
  A picture of
  <a href="protected/ufo.mpg" price="0.001">
  a UFO over the White House</a>.
  ```

  The resulting page contains the following applet activations:

  ```
  A picture of
  <applet code="buyer.class" width=xxx height=xxx name="1">
  <param name="m1" value=M1'>
  <param name="nr_items" value="2">
  <param text="Mount Everest">
  .
  </applet>
  A picture of
  <applet code="buyer.class" width=xxx height=xxx name="2">
  <param text="a UFO over the White House">
  </applet>
  .
  ```

  The applet name is its sequence number in the page. The lengthy $M1'$ message is given only to the first applet, which sends it to all other applets. Each applet extracts its own transaction ID and price from $M1'$.

  The resulting menu page is longer than the original page, since it contains several applet activations and one $M1'$ message instead of direct links to pages. However, the resulting page is still contained in a single message.

- When the customer decides to purchase a page, he generates $M2$ from the appropriate parts of $M1'$ message. The subsequent `GET` request for the pay-per-view page includes $M2$ as a parameter. For example:

  ```
  GET http://server.address/cgi-bin/gatekeeper?M2
  ```

- The server responds with the contents of the pay-per-view page.