Electronic Fair-Exchange Protocols using Untrusted Servers

Mudhakar Srivatsa, Li Xiong and Ling Liu
College of Computing
Georgia Institute of Technology
{mudhakar, lxiong, lingliu}@cc.gatech.edu

Abstract
Electronic fair-exchage protocols have received significant attention from the research community in the recent past. In loose terms, the fair exchange problem is defined as atomically exchanging electronic items between two parties. All the known fair exchange protocols today utilize a trusted third party server either actively or passively. In this paper, we propose a fair-exchange protocol that guarantees fairness in the exchange of electronic items using untrusted servers. We extend our protocol to operate on large online electronic communities and peer-to-peer systems and demonstrate its security guarantees, scalability and load balancing properties.

1 Introduction

Electronic commerce transactions, especially those that involve the exchange of digital products between transaction parties, have additional requirements as compared to classical barter exchanges. In a typical business environment, a transaction involves fulfillment of some obligation by two parties; a contract describes the penalties if either of the parties fail to meet its obligation. For example, a purchase of products involve the merchant delivering the goods, and *simultaneously*, a customer paying for it. Since, a fraud by either parties in such a transaction is *physically* detectable, and the party responsible for unfair behavior can be penalized. In an electronic transaction a fraud cannot be physically detected. Indeed the faltering party may *vanish* after cheating on a transaction. In such cases, it is next to impossible to enforce the penalties of the contract. Consequently, in an electronic commerce environment two mutually non-trusting parties are reluctant to transact.

In an open electronic commerce environment (non-mutually trusting parties), we need protocols to prevent unfair business dealings by any party involved. However, guaranteeing fairness in an electronic transaction is easier said than done. Say, a customer C contacts an online merchant M for a product P. Now customer C wants to pay for the product only if C receives the right product P. At the same time, the merchant M does not deliver the product P to the customer prior to receiving a proper payment. If merchant were to deliver the product before receiving a proper payment, a fraudulent customer may vanished after receiving the product. On the other hand, if the customer were to make the payment before he/she receives the product, a fraudulent merchant may vanish after receiving the payment.

Fairness in an electronic commerce transaction is a stronger property than security. Security in a transaction can be achieved by ensuring that the parties sign their electronic items (the merchant's product and the customer's payment) before they exchange them. Fairness on the other hand is achieved only if both the parties fulfill their obligation and receive the item it expects, or neither receives any portion of the others item. A fair electronic exchange protocol can be defined as *a protocol that ensures that no player in an electronic commerce transaction can gain an advantage over the other player by misbehaving, misrepresenting or by prematurely aborting the protocol* [12]. In other words, a fair electronic exchange protocol guarantees **exchange atomicity**.

One trivial solution for guaranteeing fairness in electronic transactions is to route all such transactions through an *online trusted third party*. For example, an online trusted third party (ttp) can solve the fair exchange problem in the customer and merchant scenario described above as follows. The customer sends his/her payment to ttp and the merchant sends the product to the same ttp. The ttp verifies that the product sent by the merchant is indeed what the customer wanted and that the payment is indeed what the merchant expects in return. If so, ttp forwards the product to the customer and the payment to the merchant; else ttp aborts the transaction. However, a trusted third party based solution is not scalable, has a single point of failure, and incurs heavy administrative overheads.

In this paper we present electronic fair exchange protocols that do not require a completely trusted server. We achieve fair exchange using a collection of untrusted servers. The untrusted servers could potentially display arbitrary behavior; we use Byzantine failures [6] to model the behavior of untrusted servers. Our protocol is *guaranteed* *to terminate with a successful exchange or with a proof of the malicious behavior of one of the parties provided not more than one-third of the untrusted servers are malicious.*

2 Related Work

Electronic fair exchange protocols have received significant attention from the research community in the recent past. Several interesting questions have already been answered: (i) Is it possible to construct a fair exchange protocol without involving trusted third parties? (ii) Even otherwise, how can we reduce the load in the trusted third parties? (iii) Is it possible to construct exchange protocols for *a weaker version* of fair exchange more efficiently or without involving trusted third parties?

Pagnia *et. al.* [10] show that it is impossible to construct a *strong fair-exchange protocol* in the absence of trusted third parties. Suppose two parties P and Q are interested in exchanging electronic item i_P and i_Q. Assume the description of the item i_Q, denoted as d_Q be known to P and the description of the item i_P (d_P) is known to Q. In other words, the parties P and Q should know precisely what to expect from each other. Let $desc(i)$ denote the description of some item i. The properties of a strong exchange protocol between two parties P and Q are as follows:

- *Effectiveness:* If P and Q behave correctly and do not want to abandon the exchange then when the protocol has completed, P has i_Q such that $desc(i_Q) = d_Q$ and Q has item i_P such that $desc(i_P) = d_P$.

- *Strong Fairness:* When the protocol has completed, either P has i_Q such that $desc(i_Q) = d_Q$, or Q has gained no additional information about i_P. The same conditions similarly count for Q.

- *Timeliness:* P can be sure that the protocol will be completed at a certain point in time. At completion, the state of the exchange as of this point is either final or changes to the state will not degrade the level of fairness reached so far.

[10] shows the impossibility of strong-fair electronic exchange between two parties (in the absence of trusted third parties) by reducing it to a *distributed consensus problem*. It is well known that the asynchronous distributed consensus problem is impossible in the presence of even one faulty process [4, 2].

Given that strong fair-exchange protocols are impossible with involving trusted third parties (*ttp*s), several research efforts have focused on techniques that can significantly reduce the load on the *ttp*s. *Optimistic* fair-exchange protocols guarantee strong fair-exchange while reducing the involvement of *ttp* to only those exchanges that result in a conflict. More concretely, an optimistic fair-exchange protocol is defined as follows:

- An exchange between two non-fraudulent parties does not require a *ttp*.

- The *ttp* is involved only when one of the parties detect a fraud in the transaction. Assuming that most of the parties in an open electronic commerce environment are good, the *ttp* is hopefully involved infrequently.

Micali [8] has proposed a *certified email exchange (CEM)* protocol that satisfies the properties of an optimistic fair-exchange protocol. A certified email exchange is defined as follows: Let a be the message that party P wants to send to party Q and let b be the Q's digitally signed receipt for that message. CEM guarantees that party Q gets the message (a) if and only if party P gets the corresponding receipt (b).

CEM does not require the *ttp* to be *always available*. However, for the duration of time the *ttp* is down, no conflicts can be resolved. The cost of increased conflict resolution time comes with a reward. Note that maintaining a *ttp* online and always available not only increases its maintenance costs and administrative overheads, but also makes it more susceptible to attackers.

Nevertheless, any scheme that uses a trusted third party (or a small collection of them) suffers from several drawbacks. First and most importantly, the *ttp* becomes a single point of failure; if the *ttp* is compromised by an attacker then the attacker may succeed in performing many unfair exchanges. Second, the *ttp* is also susceptible to denial of service attacks wherein a group of malicious nodes may flood fake requests and exhaust all the network bandwidth and processing power available at the *ttp*. Last but not the least, it incurs heavy administrative overheads to maintain the *ttp* servers.

3 Electronic Fair-Exchange using Untrusted Servers

In this section we present a distributed and decentralized fair exchange protocol that does not rely on trusted third parties.

3.1 High-Level Properties

In this paper, we completely avoid the requirement of trusted third party servers by achieving fair-exchange using a collection of untrusted servers. Our protocol tolerates Byzantine failures of up to one-third of the untrusted servers. However, our protocol does not guarantee strong fair-exchange; but provides guarantees that are very close to strong fair-exchange. Our protocol completely satisfies the *effectiveness* and the *timeliness* properties of a strong fair-exchange protocol. However, the strong fairness property is not completely satisfied. When our protocol terminates, the two parties either have completed a successful exchange or with a proof of the malicious behavior of one of the parties. This proof can be used to permanently reprimand the malicious node, there by, making it fatal for the malicious node to attempt such malice. We require that the untrusted servers are always online. Note that this makes the untrusted servers more susceptible to attackers; however, compromising a small fraction of the untrusted servers does not affect the guarantees provided by our protocol.

We extend our protocol to application scenarios including large online electronic commerce communities, peer-to-peer systems etc. Our extended protocol works on large electronic communities and shows the following good properties: completely decentralized, effective load balancing, tolerance to crash and Byzantine failures, and free of administrative costs (since it is devoid of expensive trusted third-party servers).

3.2 Our Protocol

In this section we present a distributed and decentralized fair exchange protocol that does not rely on trusted third parties. In our protocol, two nodes exchange electronic items through a collection of untrusted server(s). The group of untrusted servers may comprise of some malicious nodes, whose actions may be entirely unknown or undefined. Hence, we assume a Byzantine model [6] for the malicious untrusted servers. In the following portions of this section, we present an algorithm for two parties n and m to exchange electronic items P and Q respectively. We also assume that the descriptions of items P and Q (namely, d_P and d_Q) is known to nodes m and n respectively.

In the following sections, we describe the protocol as executed by a non-malicious node n. Our protocol is completely symmetric; hence, if node m were non-malicious then it would execute a symmetric set of steps as that of node n. If node m were malicious then it could execute any arbitrary protocol. The same holds for untrusted servers. We only specify the protocol as executed by a non-malicious servers; the malicious servers may execute any arbitrary protocol.

Scheme I: One Untrusted Server (Trivial Case). Suppose two nodes n and m exchange their electronic items P and Q via one untrusted server s. Nodes n and sends its items P to the untrusted server s along with d_Q, the description of item Q. The untrusted server s on receiving both the items P and Q verifies whether item Q matches the description d_Q sent by node n and if item P matches the description send by node m. If so, the untrusted server s forwards the item Q to node n and item P to node m.

Observe that this electronic exchange protocol is guaranteed to be fair if the server s does not behave maliciously. In the following schemes we add more untrusted servers and guarantee that the electronic exchange is fair as long as only a small fraction (one-third) of them are malicious.

Scheme II: k Untrusted Servers. Now, suppose one relies on k untrusted servers $s_1, s_2, \cdots, s_k$ to exchange electronic items P and Q between nodes n and m. Nodes n and m send their respective items to all the untrusted servers. The non-malicious servers independently execute the same protocol as discussed in Scheme I.

One might *incorrectly* believe that if at least one of the untrusted servers nodes is non-malicious then the items P and Q would be exchanged fairly (since irrespective of the malicious behavior of other untrusted servers, the non-malicious server(s) would anyway exchange the items P and Q fairly). Unfortunately, such a solution is deceptive and it in fact worsens the situation because the fair exchange is not guaranteed as long as at least one of the untrusted servers is malicious. Say the good node n sends its item (P) to all the untrusted servers and the bad node m sends its item to none. If any one of the untrusted servers is malicious, then it may forward node the item P (n's item) to node m.

Scheme III: k Untrusted Servers using Secret Shares It is clear from Scheme II that no untrusted server must ever receive any of the items P and Q *completely*. Hence, node n can divide its item P into k secret shares $\{P_1, P_2, \cdots, P_k\}$ with threshold $h \leq k$. Note that threshold denotes the the minimum number of shares required to reconstruct the secret. Now, node n sends share P_i to untrusted server s_i for $1 \leq i \leq k$. Let us for now overlook how the untrusted server verifies the share P_i and assume that the non-malicious servers execute the same protocol as discussed in Scheme I.

One might *incorrectly* believe that if the number of malicious servers is lesser than the threshold h then the items P and Q would be exchanged fairly. Say node m is malicious and it colludes with a malicious untrusted server s_j. Node m sends $h-1$ shares to some set of $h-1$ servers from $\{s_1, \cdots, s_{j-1}, s_{j+1}, \cdots, s_k\}$. Now, the malicious node m would get $h-1$ shares of item P from these servers and the h^{th} share from the malicious server s_j (which is enough to reconstruct item P); while node n get at most $h-1$ shares of item Q (which is not enough to reconstruct item Q).

Scheme IV: The Final Protocol Let $\langle PK_n, RK_n\rangle$ denote the public and private key pair owned by node n. Also, let us assume that the public-key PK_n is bound to node n through a digital certificate. Node n first sends $I_n = d_P \parallel E_{SK_n}(P) \parallel H(SK_n)$ signed using RK_n directly to node m (so does node m), where E denotes some symmetric key encryption algorithm, H denotes a strong one-way collision free hash function, SK_n is a random key generated by node n for the symmetric key encryption algorithm E. Clearly, node m can extract P from I_n only if it becomes aware of K_n, the key used for encrypting item P.

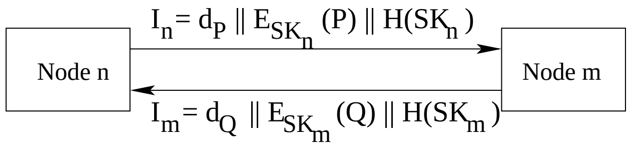
Node n sends key SK_n to node m through the collection of k untrusted servers. The untrusted servers ensure that at the end of the process both node n and node m receive SK_m and SK_n respectively. Let k' denote the maximum number of servers that may be malicious in the set of untrusted servers. Below we present the concrete protocol used for exchanging the secret keys SK_n and SK_m. Note that we specify the actions only for non-malicious nodes and servers; action of malicious nodes may be completely undefined. Also, we specify the protocol as executed by node n; node m executes a symmetric protocol (if it were non-malicious).

1. Node n divides SK_n into k secret shares $\{SK_n^1, SK_n^2, \cdots, SK_n^k\}$ with threshold $thr = k - 2k'$ (the minimum number of shares required to reconstruct the secret). Node n digitally signs and sends $d_P \parallel d_Q \parallel SK_n^j$ to the server s_j.

2. When a non-malicious server s_j receives a share from node n and node m it verifies the signatures and ensures that both of them agree of the description of the items to be exchanged, namely, d_P and d_Q. If so, server s_j sends an OK message to every untrusted server (including itself).

3. When a non-malicious server s_j receives $k - k'$ number of OK messages, it sends $d_P \parallel d_Q \parallel SK_n^j$ to node m and $d_Q \parallel d_P \parallel SK_m^j$ to node n respectively.

4. After node n receives $k - 2k'$ secret shares, it reconstructs the key SK_m. Now, node n attempts to recover the item Q from the message I_m initially sent by node m. Let Q' denote the item obtained on decrypting the item from I_m using symmetric key SK_m. If the description of Q' does not match d_Q, namely $desc(Q') \neq d_Q$, then node n has a proof of node m's malicious behavior. The proof comprises of the initial message I_m and a set of $k - 2k'$ secret shares (both digitally signed by node m) such that the symmetric key SK_m constructed from the secret shares does not match the symmetric key used in I_m.



$I_n = d_P \parallel E_{SK_n}(P) \parallel H(SK_n)$
Node n
Node m
$I_m = d_Q \parallel E_{SK_m}(Q) \parallel H(SK_m)$

Figure 1: Nodes n and m exchange I_n and I_m

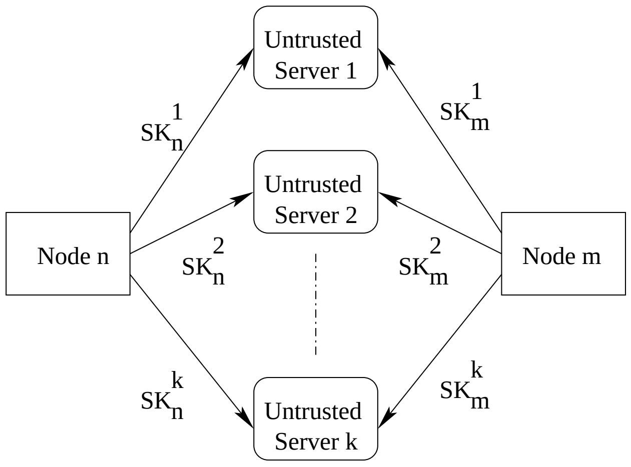


Figure 2: Node n sends $SK_n^1, SK_n^2, \cdots SK_n^k$ to the untrusted servers

Figure 1 shows the initial step of our protocol where the nodes n and m exchange I_n and I_m. Figures 2, 3, 4 and 5 show a graphical illustration of the steps (1), (2), (3) and (4) of the above protocol respectively.

Analysis and Correctness
Why do we set the threshold to be $k - 2k'$? Note that in step (3) we wait for only $k - k'$ messages since we assumed that there could be k' malicious servers and their actions could be undefined (including not sending the OK message). However, of the $k - k'$ OK messages received at least $k - 2k'$ messages are guaranteed to have originated from non-malicious nodes (malicious nodes may report OK without receiving the share OR they may not verify the integrity of the message OR they may simply not forward the share to node n in step (2)). Hence the threshold for the secret share was set to $k - 2k'$ shares. Also, by constraining that $k - 2k' > k'$, the threshold ($k - 2k'$) will be unachievable among the malicious nodes themselves. Hence, this solution can permit at most one-third of the nodes to be malicious ($k' < \frac{k}{3}$).

Why do we need a round of OK messages? Suppose the malicious node m were to *collude* with r malicious

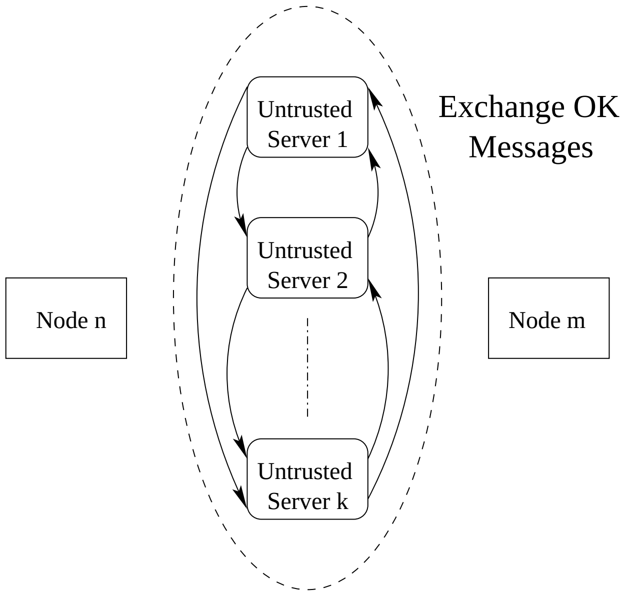


Figure 3: The untrusted servers exchange OK messages

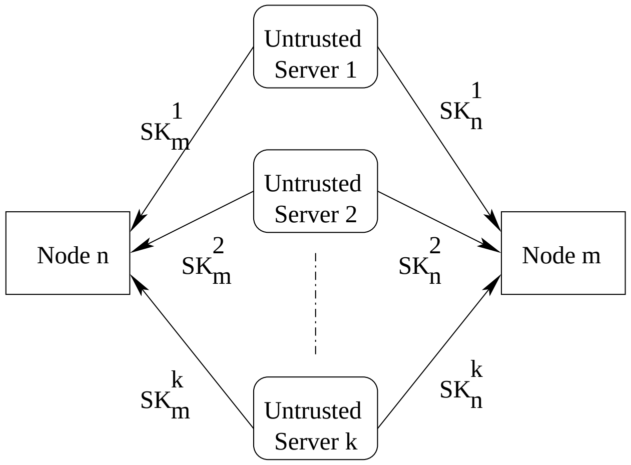


Figure 4: The untrusted servers send $SK_m^1, SK_m^2, \cdots SK_m^k$ to node n

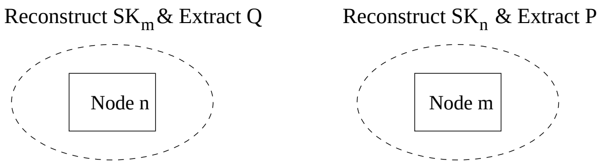


Figure 5: Node n reconstructs SK_m from a $k-2k'$ subset of $SK_m^1, SK_m^2, \cdots SK_m^k$ and extracts item Q

servers (say, $\{s_1, s_2, \cdots, s_r : 1 \leq r \leq k'\}$), then node m may send its secret shares only to $k-2k'-r$ servers in $\{q_{r+1}, \cdots, q_k\}$. Now, node m obtains the item from node n through the servers to whom it sent its shares and its r malicious *friends*, while the non-malicious node n obtains only at most $k-2k'-r$ shares (which is insufficient to reconstruct the item). Even if node m were not aware of any malicious servers in the group, it can send out only $k-2k'+r$ ($r < k'$) shares and *hope* that at least $r+1$ of them reach the malicious servers. Now these malicious servers may choose to forward the share sent by node n to node m and not vice-versa (Note that actions taken by malicious servers are in general undefined). However, if one assumes that node m is not aware of (or does not colludes with) malicious servers then the probability of an unfair exchange is largely reduced. We argue more on the lines of the *probability* of an unfair exchange in our extended protocol presented in Section 4.

Effectiveness. If two non-malicious nodes n and m want to exchange items P and Q then they will always succeed. Since n and m are non-malicious they send out all k correct secret shares to the untrusted servers in step (1). Every non-malicious server that receives it sends out an OK message (since the shares agree on the items d_P and d_Q) in step (2). Hence, every untrusted server would receive at least $k-k'$ number of OK messages and thus send node m's secret share to node n and vice-versa in step (3). Given $k-k'$ correct shares node n can reconstruct the correct symmetric key SK_m and the protocol terminates successfully in step (4). Note that it is not possible for the malicious servers to modify the secret shares since they are digitally signed by the nodes. Hence, even if node n receives a corrupted share from a malicious server, it simply ignores the share if the signature verification fails.

Fairness. Suppose node n were non-malicious and node m were malicious then either the protocol terminates fairly or node n has a proof of node m's malicious behavior. The only way node m succeeds in an unfair exchange is when it has the item P and node n has no proof of node m's malicious behavior. To achieve this node m requires $k-2k'$ secret shares of node n in step (4). Since, $k-2k' > k'$ the node m must receive at least one share from a non-malicious server. However, a non-malicious server would send node n's share to node m only after it receives $k-k'$ number of OK messages in step (3). Hence, node m would have to send out at least $k-2k'$ valid secret shares to non-malicious servers in step (1). Now, these set of $k-2k'$ shares would be available at node n in step (4). If the symmetric key reconstructed from these valid shares (digitally signed by node m) does not match I_m then node n has a proof of malicious behavior of node n.

Now, let us suppose that both node n and node m are malicious. As we have shown above, for node m to successfully obtain $k - 2k'$ secret shares of node n, it has to send out at least $k - 2k'$ shares to non-malicious servers in step (1). Hence, when the two malicious nodes n and m are non-colluding then our protocol either terminates fairly or at least one of the nodes (possibly, both the nodes) has a proof of malicious behavior of the other node. On the other hand, if two nodes are colluding then the fundamental question that arises is: Why go through the fair-exchange protocol at all? If the transacting parties trust each other completely then there is no need for a fair-exchange protocol in the first place.

4 Extended Protocol

In this section we present and discuss our fair-exchange protocol in the context of large online electronic communities and peer-to-peer systems. These are large scale distributed systems comprising of a large number of autonomous nodes. These mutually suspicious nodes may execute transactions that exchange electronic data items between each other. It is very important to ensure fairness in such electronic exchanges. Also, most of the participants in these communities are non-malicious. Nevertheless, the lack of mutual trust makes it necessary for the participants in such communities to cautiously exchange electronic items. Also, the autonomous nature of the participants makes it hard for them to agree on a central trusted third party. Under such scenarios it is highly plausible to extend our protocol that uses untrusted servers to guarantee fair exchange. The fundamental idea is to guarantee fair exchange of electronic data items between two nodes in a community using other nodes in the community (assuming that a significant portion of the community is non-malicious).

Let there be N nodes in a large online community. Let p be the percentage of bad nodes in the system. Let $cf(m)$ denote the collusion-factor for a malicious node m; collusion-factor denotes the fraction of malicious nodes in the system that would collude with node m. Note that since the nodes are largely autonomous, it is quite unlikely that a malicious node from one autonomous organization would collude with other malicious nodes from other organizations.

For simplicity assume that the nodes in the system are identified by identifiers from $\{0, 1, \cdots, N-1\}$. Let the identifier for a node n be denoted by $ID(n)$. We remove this restriction on the domain of identifiers and also permit the number of nodes in the system to vary subsequently.

Let us suppose two nodes n and m are interested in exchanging electronic items P and Q. The first step of the protocol is to uniformly and randomly choose a set of k nodes in the system to play to role of untrusted servers in our fair-exchange protocol. It is very important that the set of k nodes are chosen randomly from the set of N nodes; else a malicious node m could choose a set of his malicious friends as the collection of untrusted servers. To be fair to both nodes n and m, we might want to allow each of them choose $\frac{k}{2}$ nodes to form the group of untrusted servers. However, since our protocol tolerates only $\frac{k}{3}$ malicious nodes amongst the group of untrusted servers, allowing a malicious node to choose $\frac{k}{2}$ untrusted servers is also not feasible.

We overcome this problem by choosing the group of nodes to play the role of untrusted servers as follows: Two nodes n and m choose untrusted server s_j as the server with identifier $H(ID(n) \oplus ID(m) \oplus j) \; mod \; N$ (for $1 \leq j \leq k$), where H denotes a publicly known strong one-way collision free hash function and $\oplus$ denotes bitwise exclusive-or operation. Hence, neither node n nor node m has a say in determining the collection of untrusted servers assuming the uniform and random properties of the hash function H and that nodes cannot spoof their identifiers. Also, an implicit assumption in our discussion is that it is indeed possible to correctly locate a node given its identifier. Putting aside these considerations for now, our algorithm for selecting the group of untrusted servers has several interesting properties:

- The group of untrusted servers can be determined independently by node n and m provided they know $ID(m)$ and $ID(n)$ respectively.
- It is possible that the selection procedure does not yield us k different nodes; hence the actual size of the group of untrusted servers might be lesser than k (although with high probability the selection process results in k distinct nodes provided $k \ll N$).
- Although the fraction of malicious nodes in the entire community $p \ll \frac{1}{3}$, there is a non-zero probability that a randomly selected group of untrusted servers consists of more than one-third malicious nodes.

Before we proceed with the security analysis of our extended protocol, we discuss existing solutions that provides the infrastructural facilities required by our protocol. Today, most online electronic communities and peer-to-peer systems employ an overlay network to carry out their operations. The currently active (online) members of the electronic community connect with each other to form the overlay network. Interestingly enough, all the infrastructural facilities required for our protocol can be derived using a special class of overlay networks called distributed hash table (DHT) based overlay networks. Before we proceed with the discussion on our extended protocol, we take a short detour to the DHT-based overlay networks.

4.1 DHT-based Overlay Networks

In the recent past, most of the research on overlay networks was targeted at improving the performance of search. This led to the emergence of a class of overlay networks that include Chord [14], CAN [11], Pastry [13] and Tapestry [1]. These techniques are fundamentally based on consistent hashing [5], but slightly differ in algorithmic and implementation details. All of them store the mapping between a particular *key* and node responsible for that key in a distributed manner across the network, rather than storing them at a single location like a conventional hash table. This is achieved by maintaining a small routing table at each node. Further more, given a *key*, these techniques guarantee the location of the node that is responsible for the key in a bounded number of hops within the network. To achieve this, each node is given an identifier and is made responsible for a certain set of keys. This assignment is typically done by normalizing the key and the node identifier to a common space (like hashing them using the same hash function) and having policies like numerical closeness or contiguous regions between two node identifiers, to identify the regions which each peer will be responsible for.

For example, in our context the keys $\{key_j = ID(n) \oplus ID(m) \oplus j : 1 \leq j \leq k\}$) denote the collection of identifiers that are a part of the untrusted servers group for an electronic exchange between node n and node m. All the available nodes' IP addresses are hashed using a hash function H and each of them store a small routing table (for example, for Chord the routing table has only b entries for an b bit hash function) to locate other nodes. Now, to locate the untrusted server s_j, the key key_j is hashed using the same hash function H and depending upon the policy, the node responsible for that file is obtained. This operation of locating the appropriate node is called a *lookup*.

A typical DHT-based P2P system will provide the following guarantees:

- A lookup operation for any key is guaranteed to succeed is guaranteed to succeed within a small and bounded number of hops.
- The key identifier space is uniformly (statistically) divided among all currently active peers.
- The system is capable of handling dynamic peer joins and leaves.

4.2 Extended Protocol: Security Analysis

In this section, we present an in-depth security analysis of our extended protocol. For the sake of simplicity we assume that there are a static set of N nodes in the system and the node identifiers are chosen from $\{0, 1, \cdots, N-1\}$. The key concerns include the following: (i) How to select k nodes randomly? (ii) What if there are more than one-third malicious nodes in the collection of untrusted servers? (iii) What is the effect of collusion amongst the bad nodes on the extended protocol?

Before we proceed with the discussion on these issues we emphasize the importance of disallowing the malicious nodes from spoofing fake identities. It has been shown by Douceur in the Sybil attack paper [3] that the bad nodes may potentially amplify their strength by a factor that is proportional to the number of identities they can spoof simultaneously. One could tie down an identity to a node through digital certification based mechanisms or enforce a secure login procedure for nodes wanting to join the overlay network (comprising of the online electronic community).

4.2.1 Untrusted Servers: Selection Procedure

We address our first concern on the size of the selected group of untrusted servers. We answer this question in three steps: (i) What is the probability that the group size could be lesser than k? (ii) Why don't we permit the nodes n and/or m to use some random values $rand_j$ for generating key_j, i.e., why not use $key_j = ID(n) \oplus ID(m) \oplus rand_j$ instead of $key_j = ID(m) \oplus ID(n) \oplus j$? (iii) How can one change the group size dynamically?

We first address the first issue. The probability that the selection procedure yields less than k untrusted servers is $1 - e^{-\frac{k(k-1)}{2N}}$ by Birthday paradox [15]. Note that while N, the number of nodes in the online community could be of the order of a few thousands, k, the number of untrusted servers required for fair-exchange is very small (about 4 to 10). Birthday paradox shows that unless k is of the order of $\sqrt{N}$ the probability of a collision is extremely small. Hence, the probability that a selection yields lesser than k untrusted servers is negligibly small.

Now, we address the second issue. We show that it is very important that the j^{th} key is not chosen as $key_j = ID(n) \oplus ID(m) \oplus rand_j$, where $rand_j$ denotes some random number. One might suppose that allowing the nodes n and m to choose these random numbers may not give them any advantage since the untrusted servers are chosen as a hash of the keys (recall, $s_j = H(key_j) \ mod \ N$) thereby, making the process of choosing a *favorable* $rand_j$ as hard as attempting to invert the hash function H. A $rand_j$ is favorable for a malicious node m if using $rand_j$ results in the selection of a malicious server node s_j. Assuming that the size of the hash space is 2^{128} one might incorrectly conclude that about half of the hash space has to be search before the malicious node m can identify a favorable $rand_j$.

Unfortunately, this is not true, since we choose server s_j from a domain of size N (recall $s_j = H(key_j)\ mod\ N$). Hence, with a reasonably high probability, in about $O(N)$ attempts the malicious node m would be able to choose a favorable $rand_j$. Hence, a malicious node may obtain favorable $rand_j$'s with extremely small effort; the XOR operations are computationally very cheap and one can compute about 1 million hashes (using MD5 [7] from OpenSSL library [9]) in just one second. Hence, it is very important that the keys $\{key_j\}$ are not chosen using some random integers generated by the transacting parties. We propose to generate key_j as follows: $key_j = ID(m) \oplus ID(n) \oplus f(j)$, where $f(j)$ is some deterministic injective function on the domain of integers. One simple example of such an injective function $f(j)$ is $f(j) = j$.

Now, let us briefly explore the third issue. Suppose the group size of untrusted servers turned out to be less than k. One option would be to simply run the fair-exchange protocol using a smaller number of untrusted servers. This would impact the probability of a fair-exchange as discussed in the next section. One can also dynamically increase the group size provided both the nodes n and m agree on doing so. This can be achieved by systematically searching for other untrusted servers by constructing keys using $f(j)$ for $j = k+1, k+2, \cdots$. Nevertheless, we emphasize that with very high probability this additional search would not be required.

This server selection scheme has several advantages. First, by randomly choosing untrusted servers the selection scheme guarantees good load balancing properties. Second, there is no small set of malicious nodes that can disrupt the transactions of node n with *all* the other nodes in the system. However, if there are more than one-third malicious nodes in the set of untrusted servers chosen for an exchange between node n and m then all exchanges between the nodes n and m may be disrupted. However, as we have pointed out, randomizing $f(j)$ breaks the system's security guarantees. One can work around this problem by using an *externally observable and verifiable event* as follows. For example, one could define a time varying $f(j)$ at time t as $f(t, j) = j + hour\_of\_day(t)$. The nodes n and m may exchange their *current hour of the day* as a part of the initial message I_n and I_m (recall Section 3.2). Only if the two nodes agree on this value the exchange protocol is initiated. Note that two nodes would most probably agree on the *current hour of the day* for most time instants t assuming that the maximum clock skew between two nodes is much smaller than one hour. Also observe that a malicious node does not have a choice in choosing the *current hour of the day*; and the set of untrusted servers for an exchange between two given nodes n and m change every one hour.

4.2.2 Untrusted Servers: Large Fraction of Malicious Servers

Now, we have discussed techniques to choose untrusted servers *randomly* from the collection of all nodes in the system. Nevertheless there is non-zero probability that a randomly chosen collection of untrusted servers consists of more than one-third malicious nodes. We address this issue in multiple steps: (i) What is the probability that more than one-third nodes in a randomly chosen set of untrusted servers are malicious? (ii) What are the chances that a exchange terminates unfairly when more than one-third of the servers are malicious?

We address the first issue now. Let p denote the fraction of malicious nodes in the entire system. A random set of k nodes has more than one-third of them malicious with a probability given by $\sum_{v \geq k/3}^{k} binom(p; v, k)$, where $binom(p; v, k)$ denotes the probability in a binomial distribution for v successes from k trials where the probability of success in any trial is p. For small values of p the probability that more than one-third nodes turn out to be malicious is extremely small. Also, for small values of p, the probability that more than one-third untrusted servers are malicious decreases with the number of untrusted servers k. Hence, in theory, one could *always* increase the probability of fair-exchange by increasing k. However, this higher security guarantee comes at the cost of increased number of messages exchanged by our protocol.

Now we address the second issue. What if more than one-third the servers are malicious? Suppose two nodes n and m want to exchange an electronic item. If the malicious node m colludes with the malicious nodes in the collection of untrusted servers then the resulting exchange would be unfair. To illustrate this, say $k' > \frac{k}{3}$ nodes are malicious. We have shown that when there are k' malicious servers, the threshold for secret sharing should be set to $thr = k - 2k'$. However, when $k' > \frac{k}{3}$, the set of malicious servers can re-compute the secret among themselves, since $thr < k'$. Hence node m can reconstruct the symmetric key used by node n for encrypting the electronic item P (and consequently the item P) even without sending out one of its secret shares.

However, this assume complete collusion ($cf = 1$). In a realistic scenario involving a large collection of autonomous nodes, it is very unlikely that all the malicious nodes would collude with each other. This leads us into the next section that analyzes the extended protocol under different collusion models.

4.2.3 Untrusted Servers: Incomplete Collusion

So far we have developed techniques to randomly choose a quorum of untrusted servers. Also, we have shown that the probability that more than one-third of such randomly

Threshold	Outcome of Fair Exchange
$thr \leq r$	unfair exchange
$r < thr \leq k - k' - r$	fair exchange or a proof of malicious behavior
$thr > k - k' - r \bigwedge$ $thr > r$	Pr(fair exchange or proof of malicious behavior) = $\sum_{nms=thr}^{q} binom(1-p; nms, q)$ Pr(unfair exchange) = $\sum_{nms=k-k'-r}^{thr} binom(1-p; nms, q)$

Figure 6: Probability of Fair-Exchange Under Partial Collusive Settings

chosen nodes turn out malicious is very small (for small p). Nonetheless, when more than one-third servers are malicious and all malicious nodes collude with each other the exchange would be unfair. Interestingly, under the assumption that the collusion factor $cf < 1$, we have two interesting outcomes: (i) The probability of a successful fair-exchange increases, (ii) If the malicious node m is not careful the good node n might get a proof of the malicious behavior of node m. We analyze these issues from the perspective of the probability of a fair-exchange between a non-malicious node n and a malicious node m; transaction between two bad nodes is not of interest to us; transaction between two good nodes is always fair (if one of the nodes (say, node n) is successful in extracting item Q from node m, it may implicitly send item P in plain-text (or send K_n, the symmetric key used for encrypting item P) directly to node m).

Suppose two nodes n and m are interested in exchanging electronic items P and Q. Let $cf(m)$ denote the fraction of malicious nodes that collude with node m. If all untrusted servers that do not collude with node m execute the exchange protocol correctly, then the effective fraction of malicious nodes with respect to node m decreases to $p * cf(m)$. This makes it even more unlikely that one-third of the untrusted servers are malicious nodes that collude with node m (see Section 4.2.2). An interesting question that arises is that can a malicious node m succeed in an unfair exchange when the number of untrusted servers that collude with node m is fewer than one-third by utilizing other malicious untrusted servers that do not collude with node m?

Before we proceed, we formalize the nature of collusions we explore in this paper. It is important to remember that there could be other means through which malicious nodes could collude that we have not explored in this paper. Say two nodes n and m are interested in performing an electronic exchange. If the nodes n and m were non-malicious the set of untrusted servers may collude to perform a Denial-of-Service attack by preventing the fair-exchange from succeeding. If node m were malicious, the set of untrusted servers that collude with node m will try to extract the item P from node n but not give away a proof of malicious behavior of node m. However, the malicious nodes that do not collude with node m would attempt an DoS attack on the electronic exchange (instead of helping node m in achieving an unfair exchange).

Suppose a malicious node m colludes with r malicious servers, say $\{s_1, s_2, \cdots, s_r : 1 \leq r \leq k'\}$ such that $r < \frac{k}{3}$. Recall that k denotes the total number of untrusted servers and k' denotes the number of malicious servers. If $r \geq \frac{k}{3}$ then it is very easy to perform an unfair exchange. We claim that if $r < \frac{k}{3}$ there is no strategy that would guarantee an unfair exchange. The difficulty for the malicious node m arises because it cannot distinguish the malicious from the non-malicious servers in the set $\{s_{r+1}, s_{r+2}, \cdots s_k\}$. The malicious servers $\{s_{r+1}, \cdots, s_{k'}\}$ perform a denial of service attack by not sending the OK message required for the protocol to continue. This serves them two purposes. If the two transacting nodes n and m are non-malicious then by not sending the OK message the malicious nodes may prevent an exchange between the two nodes. On the other hand, if node m were malicious then sending an OK message would only result in either the non-malicious node n getting a proof of malicious behavior of node m or the malicious node m performing an unfair exchange; both of which are of no interest to the non-colluding malicious servers.

The key problem for a malicious node m is that it is not aware of these $k' - r$ non-colluding malicious servers do not send the OK message. We now quantitatively analyze the probability that a malicious node m would succeed in an unfair exchange. Note that in this discussion k' could be larger than $\frac{k}{3}$. For the malicious node m to succeed in an unfair exchange at least $k - k'$ servers must send the OK message. The r malicious friends of node m would anyway send the OK message; however, in order obtain the remaining $k-k'-r$ number of OK messages the malicious node would have send $k - k' - r$ shares to non-malicious servers. If the threshold thr used for generating secrets is lesser than or equal to $k - k' - r$ then node n receives the correct item Q from node m or a proof of malicious behavior by node m. Even otherwise, if threshold thr were greater than $k - k' - r$, the fact that node m cannot distinguish between non-malicious servers and non-colluding malicious servers makes this task difficult. For instance, node m would have to send out $q \geq k - k' - r$ secret shares and *hope* that the number of malicious servers nms it reaches is constrained by $k - k' - r \leq nms < thr$. On the other hand, if more that thr shares out of the q shares reaches non-malicious servers then node n gets the item Q or a proof of malicious behavior of node m. The fact that the node n gets a cryptographically secure and proovable piece of data from node m makes this a tough bet for the malicious node m; since one mistake by node m may result in it being reprimanded from the online community. The only way node m can play it safe is to send the correct item Q and the correct key SK_m; so that when node m fails in making an unfair exchange, node n gets the correct item Q and not a proof of its malicious behavior. In the event that node m succeeds in making an unfair exchange then node n gets neither the item Q nor a proof of node m's malicious behavior. From an adverserial point of view, the malicious node m would choose a q such that probability of unfair exchange is maximized; while the probability of fair exchange (or the proof of malicious behavior) to be minimized. Table 6 summarizes the results of the above discussion.

The epitome of this discussion is that one could exploit the partial collusive settings that is commonly observed in large scale systems comprising of autonomous nodes to achieve the following: (i) Tolerate larger fractions of malicious servers in the group of untrusted servers ($k' > \frac{k}{3}$), (ii) Largely reduce the probability of an unfair exchange, and (iii) Heavily constrain a malicious node m to always use the correct item Q and send the correct key K_m lest it gives away a proof of its malicious behavior to the node n.

5 Conclusion

In this paper, we have proposed an electronic fair-exchange protocol that functions without involving a trusted third party server. Eliminating the requirement of trusted servers reduces administrative costs and avoids a single point of failure. We present an extended verion of our protocol that operates on a large scale online electronic communities and peer-to-peer systems. We have quantitatively analyzed the security guarantees provided by our protocol. Our extended protocol is completely distributed, scalable, highly fault-tolerant and shows very good load balancing properties.

References

[1] J. K. B. Zhao and A. Joseph. Tapestry: An infrastructure for fault-tolerance wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, 2001.

[2] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. In *Journal of the ACM 34*, 1987.

[3] J. Douceur. The sybil attack. In *2nd Annual IPTPS Workshop*, 2002.

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. In *Journal of the ACM 32*, 1985.

[5] D. Karger, E. Lehman, F. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees. In *Proceedings of ACM Symposium on theory of Computing (STOC)*, November 2000.

[6] L. Lamport, R. Shostak, and M. Pease. The byzantine general problem. In *ACM Transactions on Programming Languages and Systems*, 1982.

[7] MD5. The md5 message-digest algorithm. http://www.ietf.org/rfc/rfc1321.txt, 1992.

[8] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of the 22nd Annual Symposium on Principle of Distributed Computing*, 1999.

[9] OpenSSL. Openssl. http://www.openssl.org/.

[10] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Darmstadt, Germany, 1999.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM Annual Conference on Data Communication*, August 2001.

[12] I. Ray and I. Ray. Fair exchange in e-commerce. In *ACM SIGEcomm Exchange*, 2001.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, November 2001.

[14] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM Annual Conference on Data Communication*, August 2001.

[15] Wikipedia. Birthday paradox. http://www.wikipedia.org/wiki/Birthday_paradox.