ExchangeGuard: A Distributed Protocol for Electronic Fair-Exchange

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

Abstract[1]

Electronic fair-exchange 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 centralized trusted third party server either actively or passively. In this paper, we propose a distributed protocol for exchange of electronic items using untrusted servers. We perform detailed security analysis and show that the protocol guarantees effectiveness and fairness with Byzantine failures of up to one third of the untrusted servers. We also give the probability of a fair exchange otherwise. Finally we discuss how to deploy the protocol to 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 transacting 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, 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. More concretely, 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

[1]This research is partially supported by NSF CNS CCR, NSF ITR, DoE SciDAC, DARPA, CERCS Research Grant, IBM Faculty Award, IBM SUR grant, HP Equipment Grant, and LLNL LDRD.

Any opinions, findings, and conclusions or recommendations expressed in the project material are those of the authors and do not necessarily reflect the views of the sponsors.

is not scalable, has a single point of failure, incurs heavy administrative overheads, and is susceptible to Denial-of-Service (DoS) and host compromise attacks.

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 [5] 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.*

The rest of this paper is organized as follows. Section 2 summarizes the work done in the field of electronic fair-exchange protocols and positions our work appropriately. Section 3 presents our protocol followed by a detailed analysis and proof of correctness in Section 4. Section 5 describes our extended protocol and sketches techniques to practically deploy the protocol on a large scale. Finally, the paper concludes in Section 6.

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?

Pagnia *et. al.* [11] show that it is impossible to construct a *strong fair-exchange protocol* in the absence of 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.

Pagnia *et. al.* [11] shows the impossibility of strong-fair electronic exchange between two parties (in the absence of trusted third parties) by reducing it to an *asynchronous distributed consensus problem*. It is well known that the asynchronous distributed consensus problem is impossible in the presence of even one faulty process [4, 1].

Given that strong fair-exchange protocols are impossible with involving third parties (tp), several research efforts have focused on techniques that can significantly reduce the load on the tps. *Optimistic* fair-exchange protocols guarantee strong fair-exchange by utilizing trusted third parties, while reducing the involvement of a tp 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 trusted tp.
- The trusted tp is involved only when one of the parties detect a fraud in the electronic exchange. Assuming that most of the parties in an open electronic commerce environment are good, the trusted tp is hopefully involved infrequently.

Micali [7] 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 trusted tp to be *always available*. However, for the duration of time the trusted tp is down, no conflicts can be resolved. The cost of increased conflict resolution time comes with a reward. Note that maintaining a trusted tp 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 trusted tp becomes a single point of failure; if the trusted tp is compromised by an attacker then the attacker may succeed in performing many unfair exchanges. Second, the trusted tp 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 trusted tp. Last but not the least, it incurs heavy administrative overheads to maintain the trusted tp servers.

This paper presents an electronic fair-exchange protocol without relying on completely trusted servers. To the best of our knowledge this is first attempt to develop fair-exchange

protocols using untrusted servers. The guarantees provided by our protocol is *very close* (but not equivalent to) that of a strong fair-exchange 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 The Protocol

In this section we present **XChange**, 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 fair exchange or an honest party has a proof of malicious behavior of the fraudlent party. In the event that the latter happens, the malicious party might have knowledge of the other party's electronic item. Nonetheless, the proof of misbehavior 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.

3.2 Electronic Exchange Using Untrusted Servers

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 [5] 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 section, 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.

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 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 no more than one-third of them are malicious.

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 the previous scheme.

Supposing a malicious node m sends its item Q to at least one non-malicious server 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). Also, if node m were to send its item Q to all the untrusted server then it is quite likely that at least one of the untrusted servers is non-malicious and hence, the exchange terminates 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.

k **Untrusted Servers using Secret Shares.** It is clear from previous scheme 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 $thr \leq k$. A (thr, k) threshold secret sharing

scheme proposed by Shamir [14] presents a technique to divide a data item D into k pieces in such a way that D is easily reconstructable from any thr pieces but even complete knowledge of $thr-1$ pieces reveals absolutely no information about D. 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 the first scheme. Note that a major improvement in Scheme III over schemes I and II is that the untrusted servers do not directly learn anything about the items P and Q as long as the number of malicious untrusted servers is lesser than the threshold thr.

One might be tempted to believe that if the number of malicious servers is lesser than the threshold thr 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 $thr-1$ shares to some set of $thr-1$ servers from $\{s_1, \cdots, s_{j-1}, s_{j+1}, \cdots, s_k\}$. Now, the malicious node m would get $thr-1$ shares of item P from these servers and the thr^{th} share from the malicious server s_j (which is enough to reconstruct item P); while node n get at most $thr-1$ shares of item Q (which is not enough to reconstruct item Q). In general, if node m were to collude with r malicious servers then it could send out $thr-r$ shares to the non-malicious servers and receive $thr-r$ shares of item P from non-malicious servers and the remaining r shares from the set of malicious servers. Node n would not be able to reconstruct item Q (or a proof of node m's misbehavior) since it has only $thr-r$ shares of item Q.

3.3 XChange Protocol

We now present our XChange protocol. 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). We use k to denote the total number of untrusted servers, k' to denote the number of malicious servers ($k' \leq k$) and r denote the number of untrusted servers that colludes with a malicious party m.

Phase 1: Initial Exchange. As the first phase, we simplify the problem of exchanging (possibly large) electronic items P and Q to exchanging two relatively small keys. The main motivation in doing so is the huge performance gains one would obtain in terms of the cost of computing the secret shares and the messaging cost. 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 PKI based digital certificate [9]. Node n first sends $I_n = d_P \parallel E_{SK_n}(P)$ and $sig_n(I_n)$ directly to node m where, $sig_n(X)$ denotes the digital signature of X as signed by node n, E denotes some symmetric key encryption algorithm (like DES [3] or AES [8]) and SK_n is a random key generated by node n for the symmetric key encryption algorithm E. Clearly, node m can efficiently extract the item P from I_n only if it becomes aware of SK_n, the key used for encrypting item P. When node n receives I_m and $sig_m(I_m)$, it checks the validity of this signature before proceeding to phase 2. In the remaining phases of this protocol, node n and m exchange the secret keys SK_n and SK_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.

Phase 2: Exchanging Parties Submitting Secret Shares to Untrusted Servers. Node n divides SK_n into k secret shares $\{SK_n^1, SK_n^2, \cdots, SK_n^k\}$ with threshold $thr = k - 2k'$. We defer the discussion of how the threshold was selected to the end of this section. Node n then sends $U_j = d_P \parallel d_Q \parallel SK_n^j$ and $sig_n(U_j)$ to the server s_j for $j = 1, 2, \cdots, k$.

Phase 3: Untrusted Servers Verifying Secret Shares and Exchanging OK Messages. 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 all untrusted servers (including itself).

Phase 4: Untrusted Servers Forwarding Secret Shares to Nodes. When a non-malicious server s_j receives $k-k'$ number of OK messages, it sends U_j and $sig_n(U_j)$ to node m and vice-versa. The untrusted servers 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).

Why do we need a round of OK messages? Suppose the malicious node m were to *collude* with r malicious 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). Essentially, a round of OK messages forces the malicious node m to send out at least $k-2k'$ $(= thr)$ shares to the non-malicious servers. 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 ma-

licious 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 5.

Phase 5: Secret Share Reconstruction and Item Construction. After node n receives $k - 2k'$ secret shares, it reconstructs the key SK_m. Let the reconstructed key be 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 either the malicious node m must have either sent an incorrect electronic items ($Q' \neq Q$) or an incorrect key ($SK'_m \neq SK_m$). Nonetheless, 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 (all of which are digitally signed by node m) such that the electronic item extracting using the symmetric key SK'_m constructed from these secret shares does not match the description of the electronic item used in I_m.

Now we revisit the threshold thr used for dividing secret shares in Phase 2 and discuss how to determine it. Note that of the $k - k'$ OK messages received in Phase 3, only $k - 2k'$ 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 Phase 4). Hence, in order for node n to reconstruct the secret shares, the threshold should not exceed $k - 2k'$ ($thr <= k - 2k'$). On the other hand, the threshold should be unachievable between a malicious node and his r malicious friends ($thr > r$). So, if $r < k - 2k'$, we can always pick a threshold thr ($r < thr <= k-2k'$) such that the fairness is guaranteed. Assuming worse case collusion, $r = k'$, the fairness is guaranteed as long as $k' < thr <= k - 2k'$). Hence, this solution can guarantee complete fairness with up to one-third of the nodes to be malicious ($k' < \frac{k}{3}$). We will give a more formal and detailed analysis of the effectives and fairness of the protocol in next section.

4 Analysis

4.1 Threat Model

Recall that we have k untrusted servers among which k' are malicious. A malicious node m may collude with r servers out of the k' malicious ones. The malicious node m is *not aware* of the $k' - r$ malicious but non-colluding servers. We first formalize the malicious node behaviors and 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. The collusion model presented in this paper is a highly pragmatic model for large scale online electronic-communities discussed in Section 5.

For a malicious node n, the goal is to have an unfair exchange, i.e. to get the item from n but not giving away his. 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 a DoS attack on the electronic exchange (instead of helping node m in achieving an unfair exchange). 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. Table 1 summarizes the behavior of various untrusted servers. Additionally, one could view the failure model for colluding servers as *Byzantine failures* and that of non-colluding malicious servers as *crash failures*. Our analysis analyzes the XChange protocol under a combination of these two failure models; we motivate the need of such a model from a pragmatic stand point in Section 5.

Note that we do not consider collusion between node m and node n. If they were to collude, then: 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.

4.2 Effectiveness

Theorem 4.1 *The XChange protocol guarantees effectiveness for two non-malicious nodes n and m.*

Since nodes n and m are non-malicious they send out all k correct secret shares to the untrusted servers in phase (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 phase (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 phase (3). Given $k - k'$ correct shares node n can reconstruct the correct symmetric key SK_m and the protocol terminates successfully in phase (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.

Untrusted Server	Goal	Action
non-malicious servers	fair-exchange	complete protocol
malicious non-colluding servers	denial-of-service	not send OK messages
malicious colluding servers	unfair-exchange	send OK messages; deliver node n's secret shares to node m; NOT deliver node m's secret shares to node n

Table 1: Partial Collusion of Untrusted Servers: Goals and Actions

Threshold	Outcome of Fair Exchange
$thr \leq r$	guaranteed unfair exchange
$r < thr \leq k - k' - r$	guaranteed fair exchange or a proof of malicious behavior
$thr > k - k' - r \bigwedge$ $thr > r$	outcome of the fair exchange protocol is uncertain Pr(fair exchange or proof of malicious behavior) = 1− Pr(unfair exchange) Pr(unfair exchange) = $\sum_{nms=max(k-k'-r,thr-r)}^{min(thr-1,q)} \frac{\binom{k-k'}{nms} * \binom{k'}{q-nms}}{\binom{k}{q}}$

Table 2: Probability of Fair-Exchange under Partial Collusive Settings

4.3 Fairness

Now we analyze the fairness of the XChange protocol. Suppose node n were non-malicious and node m were malicious. 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.

As we have discussed earlier in the threat model, a malicious node m will always receive r shares from the untrusted servers it is colluding with. If the threshold thr used for constructing secret shares were to be less than or equal to r, then it is straightforward for node m to perform an unfair exchange since it can construct SK_n from the r shares it receives from its colluding servers.

Corollary 4.2 *Guaranteed Unfair Exchange. If $thr \leq r$, then a malicious node can perform a guaranteed unfair exchange.*

Now we consider the case when $r < thr$. Since the threshold is not achievable between node m and its colluding servers, it needs additional shares to construct SK_n. So it has to send out some shares to other servers in order to obtain additional shares from node n.

Lemma 4.3 *Assume a malicious node m sends out q shares ($0 < q \leq k$), among which nms shares reach non-malicious untrusted servers. The following conditions have to be satisfied in order for node m to get an unfair exchange.*
C1: $nms \geq k - k' - r$
C2: $nms + r \geq thr$
C3: $nms < thr$

Node m needs additional shares to construct SK_n. However, a non-malicious server would send node n's share to node m only after it receives $k - k'$ number of OK messages according to Phase 3 in our protocol. The r malicious friends of node m would anyway send the OK message. Since the malicious non-colluding servers do not send OK message, in order to obtain the remaining $k - k' - r$ number of OK messages, m should send at least $k - k' - r$ shares reaching non-malicious servers. Hence C1: $nms \geq k - k' - r$. Once the nms non-malicious servers have received shares from both n and m and received the required number of OK messages, they will forward the shares to both n and m. So node m receives nms shares from these non-malicious servers and r shares from his malicious friends. The total number of secret shares must exceeds the threshold ($\geq thr$) in order to reconstruct SK_n and successfully extract item P. Hence C2, $nms + r \geq thr$. At the same time, node n also receives nms shares from non-malicious servers. For node m to get an unfair exchange, node n must not have the required number of secret shares ($< thr$) so that it can neither extract the item Q nor can it obtain a proof of malicious behavior of node m. Hence C3, $nms < thr$.

Theorem 4.4 *Guaranteed Fair Exchange. If $r < thr \leq k - k' - r$, then there is a guaranteed fair exchange or the non-malicious nodes gets a proof of malicious behavior of the malicious node.*

If the threshold thr used for generating secrets is less 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. Note that the only way to satisfy condition C1 is to send at least $k - k' - r$ shares to non-malicious servers; however this would falsify condition C3. Note that $r < thr \leq k - k' - r$ puts a bound on the number of colluding malicious servers r, namely, $r < \frac{k-k'}{2}$.

Theorem 4.5 *Uncertain Outcome. If $thr > k - k' - r \bigwedge thr > r$, the XChange protocol may have an uncertain outcome. They are given by the following probabilities:*
Pr(fair exchange or proof of malicious behavior) = 1− Pr(unfair exchange)
Pr(unfair exchange)

$= \sum_{nms=max(k-k'-r,thr-r)}^{min(thr-1,q)} \frac{\binom{k-k'}{nms} * \binom{k'-r}{q-nms}}{\binom{k-r}{q}}$

Uncertainty in Fair Exchange. If the 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 its task difficult. To satisfy condition C1, node m would have to send out $q \geq k - k' - r$ secret shares and *hope* that it reaches at least $k - k' - r$ non-malicious servers (nms). In order to satisfy constraint C2, the node m would have to *hope* that the number of non-malicious servers nms it reaches is constrained by $nms + r \geq thr$. To satisfy condition C3, node m would have to *hope* that $nms < thr$. 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; or if $nms < k - k' - r$ we do not get the required number of OK messages and hence the protocol terminates without the exchange of the electronic items (still the exchange is fair); or if $nms < thr - r$ then the maximum number of shares received by the malicious node m equals $nms + r$, which is lesser than the threshold thr and hence the protocol terminates without the exchange of the electronic items (still the exchange is fair).

Nonetheless, the fact that the node n may get a cryptographically secure and provable piece of data from node m with a *non-zero probability* makes this a tough bet for the malicious node m; since one mistake by node m may result in it being *permanently 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.

Summary. Table 2 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 SK_m lest it gives away a proof of its malicious behavior to the node n.

5 Extended Protocol for Large Online E-Communities

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. The fundamental goal the extended XChange protocol is to achieve fair exchange of electronic data items between two nodes in an e-community using other nodes in the same e-community.

5.1 Extended XChange Protocol

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 labeled with 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 [15].

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 augment phase 0 to our XChange protocol for selecting untrusted servers.

Phase 0: Two exchanging parties n and m choose a group of nodes to play the role of untrusted servers as follows. Two nodes n and m choose untrusted server s_j as that server whose identifier is $H(ID(n) \oplus ID(m) \oplus j) \ mod \ N$ (for $1 \leq j \leq k$), where H denotes a strong one-way collision free hash function (like MD5 [6] or SHA1 [13]) and $\oplus$ denotes bitwise exclusive-or operation. Hence, neither node

N	Mean time (ms)	E[Att]	Pr($Att \leq 4N$)	Pr($Att \leq 8N$)
1024	2.32	2112	0.80	0.96
2048	4.22	4301	0.76	0.95
4096	8.47	8157	0.76	0.97
8192	16.08	16421	0.84	0.99

Table 3: Effort required to choose a favorable $rand_j$

n nor node m independently 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. Interested readers may refer to the long version of this paper for further details [15].

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 $key_j = ID(m) \oplus ID(n) \oplus j$? 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 $s_j = H(key_j)\ mod\ N$). In fact, with a reasonably high probability, in about $O(N)$ attempts the malicious node m would be able to choose a favorable $rand_j$. Moreover the computational effort required to identify a favorable $rand_j$ is very small; the XOR operations are computationally very cheap and one can compute about 1 million hashes (using MD5 [6] from OpenSSL library [10]) in just one second [2]. Table 3 shows the mean time and the expected number of attempts (Att) to find a favorable $rand_j$; and the probability that a favorable $rand_j$ is found in $O(N)$ attempts. Therefore, 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 publicly known deterministic injective function on the domain of integers. One simple example of such an injective function $f(j)$ is $f(j) = j$.

[2]As measured on a 900MHz Intel Pentium III processor running RedHat Linux 9.0

5.2 Analysis

In this section, we present an in-depth security analysis of our extended XChange protocol. The key concerns include the following: (i) Does the extended XChange protocol indeed select k untrusted servers uniformly and 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 [2] 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).

5.2.1 Number of Untrusted Servers

We study our selection procedure for selecting a group of untrusted servers. We explore this issue in three steps: (i) What is the probability that the group size could be lesser than k? (ii) How can one change the group size dynamically?

Now we address the first issue using theorem 5.1.

Theorem 5.1 *The untrusted selection procedure (phase 0) yields k untrusted servers with probability $e^{-\frac{k(k-1)}{2N}}$*

The proof of this theorem directly follows from the Birthday paradox [16]. Birthday paradox estimates the probability of collision in a hash function; in our scenario, a collision would imply that two of the k nodes that are randomly chosen from N nodes in the system happen to be identical. In other words, a collision in the hash function would imply that our server selection procedure actually yieled less than k servers.

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, see Section 5.2.2). Birthday paradox shows that unless k is of the order of $\sqrt{N}$ the probability of a collision is extremely small. Figure 1 shows the probability of a collision with $N = 1024$ nodes for different values of the group size k. From Figure 1 it is apparent that the probability that a selection yields lesser than k untrusted servers is very small for small group sizes.

Now we address the second 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

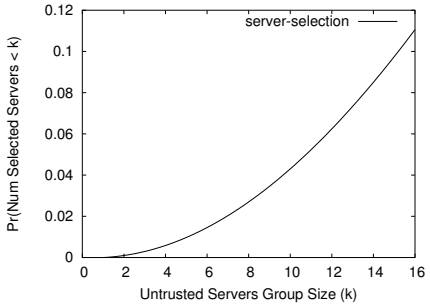

Figure 1: Probability of Collision for Varying Group Size (k) and $N = 1024$ nodes

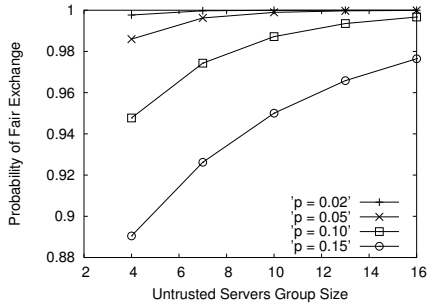

Figure 2: Probability of Fair-Exchange for Varying Group Size (k) and Different Fraction of Malicious Nodes (p)

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.

Discussion. Our 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. 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 function $f(t, j) = j + hour\_of\_day(t)$. 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 of randomly choosing the *current hour of the day*; and that the set of untrusted servers for an exchange between two given nodes n and m change every one hour.

5.2.2 Fairness and Effectiveness

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 using theorem 5.2. Let p denote the fraction of malicious nodes in the entire system. Let $binom(p; v, k)$ denotes the probability in a binomial distribution for v successes from k trials where the probability of success in any trial is p.

Theorem 5.2 *The probability that our server selection scheme chooses smaller than one-third malicious servers in a collection of k servers is* $\sum_{v=0}^{\lfloor k/3 \rfloor} binom(p; v, k)$.

The proof of this theorem follows from the fact that our server selection scheme chooses servers uniformly and randomly from a large pool of nodes wherein the probability that a randomly chosen node is malicious is p.

Figure 2 shows the probability of fair-exchange (the probability that the number of malicious servers is lesser by $\frac{k}{3}$). 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 (provided $p < \frac{1}{3}$). 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? Note that presence of more than one-third malicious servers does not guarantee an unfair exchange because only colluding servers would aim at achieving an unfair exchange, while the non-colluding malicious servers would attempt a denial-of-service attack. We have studied the extended XChange protocol under the

threat model discussed in Section 4. Interested readers may refer to the long version of this paper for detailed experimental results [15].

6 Conclusion

In this paper, we have proposed XChange, an electronic fair-exchange protocol that functions without requiring a trusted third party server(s). Eliminating the requirement of trusted servers reduces administrative costs, avoids a single point of failure and shields the system from denial-of-service and host compromise attacks. We have shown the correctness and quantitatively analyzed the security guarantees provided by the XChange protocol. We presented an extended XChange protocol that operates on a large scale online electronic communities and peer-to-peer systems. The extended XChange protocol is completely distributed, scalable, highly fault-tolerant and shows very good load balancing properties. We also studied the security guarantees offered by the extended XChange protocol and outlined techniques to construct a pragmatic implementation of the extended XChange protocol using DHT-based overlay networks.

References

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

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

[3] FIPS. Data encryption standard (des). http://www.itl.nist.gov/fipspubs/fip46-2.htm.

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

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

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

[8] NIST. Aes: Advanced encryption standard. http://csrc.nist.gov/CryptoToolkit/aes/.

[9] NIST. Digital signature standard (dss). http://csrc.nist.gov/cryptval/dss.htm.

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

[11] 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.

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

[13] SHA1. Us secure hash algorithm i. http://www.ietf.org/rfc/rfc3174.txt, 2001.

[14] A. Shamir. How to share a secret. In *Communications of ACM*, 1979.

[15] M. Srivatsa, L. Xiong, and L. Liu. Exchangeguard: A distributed protocol for electronic fair-exchange. http://www.cc.gatech.edu/ mudhakar/sguard/exchange.pdf.

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