Verifiable Distributed Oblivious Transfer and Mobile Agent Security*

Sheng Zhong Yang Richard Yang
Computer Science Department, Yale University
New Haven, CT 06520, U. S. A.

Abstract

The mobile agent is a fundamental building block of the mobile computing paradigm. In mobile agent security, oblivious transfer (OT) from a trusted party can be used to protect the agent's privacy and the hosts' privacy. In this paper, we introduce a new cryptographic primitive called *Verifiable Distributed Oblivious Transfer (VDOT)*, which allows us to replace a single trusted party with a group of threshold trusted servers. The design of VDOT uses a novel technique called *consistency verification of encrypted secret shares*. VDOT protects the privacy of both the sender and the receiver against malicious attacks of the servers. We also show the design of a system to apply VDOT to protect the privacy of mobile agents. Our design partitions an agent into the general portion and the security-sensitive portion. We also implement the key components of our system. As far as we know, this is the first effort to implement a system that protects the privacy of mobile agents. Our preliminary evaluation shows that protecting mobile agents not only is possible, but also can be implemented efficiently.

Keywords: Mobile Agent, Privacy, Oblivious Transfer, Verifiable Secret Sharing.

1 Introduction

As an important paradigm of computation, the mobile agent has a lot of potential applications in electronic commerce. However, the success of the mobile agents depends on security. In the past, the focus of mobile-agent security has been on protecting the safety and the integrity of visited hosts. To achieve this objective, researchers have proposed novel techniques such as the Sandbox architecture [14], which restricts the access of a visiting mobile agent, and proof-carrying code [20], which allows a host to efficiently verify that the visiting mobile agent will not do harm to the host.

However, in mobile agent computing, it is as important to protect the privacy of the agent from the hosts as to protect the privacy of the hosts from the agent. Since Sander and

*This work was supported in part by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795. Sheng Zhong was supported by ONR grant N00014-01-1-0795 and NSF grants ANI-0207399 and CCR-TC-0208972. Yang Richard Yang was supported in part by NSF grant ANI-0207399. A preliminary version of this paper was presented at the DialM-POMC Joint Workshop on Foundations of Mobile Computing in 2003.

Tschudin's pioneering work [25], various systems have been designed for this purpose [26, 5, 1]. In particular, Algesheimer, Cachin, Camenisch, and Karjoth [1] present a nice and general solution that has provable security. However, the security of this system relies on a single trusted party which carries out oblivious transfer (OT). If the trusted party is compromised, the privacy of both the agent and the hosts can be violated.

The security of [1] can be significantly strengthened if the single trusted party is replaced by a group of threshold trusted servers. For this end, a "threshold extension" of OT is needed. One possible solution is to use Naor and Pinkas's distributed OT (DOT) [19], which involves a sender, a receiver, and a group of servers. In DOT, the sender has two items and the receiver chooses to receive one of them. First, the sender distributes to each server some data derived from her items, in such a secure way that no single server can figure out any information about her items. Then the receiver queries the servers. From the servers' responses, the receiver is able to reconstruct one and only one of the two items. Furthermore, the receiver has no information about the other item and the sender has no information about which of the items the receiver has chosen.

However, DOT assumes semi-honest servers. If some servers are malicious, they can mislead the receiver to reconstruct a false item. To deal with such malicious servers, we propose a new cryptographic primitive called "Verifiable Distributed Oblivious Transfer," or VDOT for short.

Challenges and Contributions The design of VDOT is technically challenging. One might suggest that the objective of VDOT could be achieved by a secret-sharing scheme with oblivious transfer of each share. However, there are two somewhat conflicting goals that need to be achieved. On the one hand, the receiver must be able to verify the correctness of *both* items; otherwise, a malicious server could violate the receiver's privacy by tampering with its share of one item and observing whether or not this attack is detected by the receiver. On the other hand, in order to protect the sender's privacy, the receiver should be able to reconstruct *only one* of the two items. In summary, the major technical challenge is to allow the receiver to reconstruct only one item but verify the correctness of both items.

Our VDOT protocol uses a novel technique to address the above challenge. An overview of the VDOT protocol is as follows. During initialization, a global private key is shared in the Feldman VSS. An advantage of this setup is that the consistency of secret shares encrypted using ElGamal can be verified. Before each transfer, the sender distributes the shares of both items among the servers. During the transfer procedure, the receiver invokes the one-round OT protocol by Bellare and Micali [2, 5], with each server in a quorum (called *main servers*) in order to get the share of the item he chooses. Although the receiver can reconstruct only one item, he can verify the consistency of both items through the help of the remaining servers (called *verification servers*), because the *encryptions of* the shares of both items are transfered to the receiver during the OT.

We then apply VDOT to mobile agent security to implement the key components of a mobile agent architecture. As far as we know, this is the first effort to implement a system that protects the privacy of mobile agents. To write an agent in our system, the designer extracts the security-sensitive portion of the agent into a function. Then the function is encoded as a garbled circuit, which is carried by the agent. Because we only apply the security mechanism

to the security-sensitive portion of an agent, our system is efficient. Because the result of the security-sensitive portion is interpreted by the normal portion of the agent, all that a host needs to provide is an interpreter of garbled circuits. As a result, our system provides a general-purpose solution. We measure the overhead of our system and show that the overhead is acceptable. In other words, our preliminary evaluation shows that protecting mobile agents not only is possible, but also can be implemented efficiently.

In summary, the contributions of this paper are as follows. First, we introduce a new cryptographic primitive, VDOT, which can be used in situations where proxies of OT are needed but no single proxy can be trusted. In particular, VDOT can be used to strengthen the security of the mobile agent system designed in [1]. Second, the design of VDOT uses a novel technique to achieve consistency verification of encrypted secret shares. Third, we apply VDOT to the problem of mobile agent security to implement the key components of an architecture for mobile agents.

Organization The rest of this paper is organized as follows. In Subsection 1.1, we discuss related work. In Section 2, we define the security requirements for VDOT. (In principle, we can use the general definitions of secure multi-party computation with respect to malicious adversaries in [13]; however, these general definitions are much more complicated than the definitions we give specifically for VDOT.) In Section 3, we explain the technique of *consistency verification on encrypted shares* and present the VDOT protocol. We prove the security properties of VDOT in Section 4. In Section 5, we show how to apply VDOT to a mobile-agent system. In Section 6, we present implementation issues and report initial performance evaluation. We conclude this paper in Section 7.

1.1 Related Work

OT Protocols Oblivious Transfer was first introduced by Rabin [23]. Later, several variations were proposed, *e.g.*, 1-out-of-2 OT [9], 1-out-of-N OT [4], k-out-of-N OT [17], and adaptive k-out-of-n OT [18]. Our VDOT can be viewed as an extension of DOT [19], which introduces a group of servers to the 1-out-of-2 OT scenario. The major difference is that, as we have explained, our VDOT protocol considers potentially malicious servers, while the DOT protocol considers semi-honest servers. Another difference is that our model allows the receiver to communicate with all servers.

PIR/SPIR Protocols A problem similar to OT is private information retrieval (PIR) [6], in which a user (analogous to the receiver in OT) privately retrieves a bit from a database (analogous to the sender in OT). However, in PIR, only the user's privacy is protected, and the amount of communication is required to be small. In order to get nontrivial solutions with information-theoretic privacy, it is often assumed that there are two or more copies of the database, held by database servers that do not communicate with each other. With computational assumptions, a PIR protocol with a single copy of the database can be constructed [15]. Gertner, Ishai, Kushilevitz, and Malkin added the privacy of the database to the PIR model [12]. The result is called symmetric PIR (SPIR). The difference between SPIR and OT is that the former further requires small-communication overhead.

Interestingly, Gertner, Goldwasser, and Malkin introduced auxiliary servers to PIR [11], just as Naor and Pinkas introduced a group of servers to OT. However, in the Gertner-Goldwasser-Malkin model, the database itself is still involved in the protocol after the initialization stage, and the auxiliary servers may contain no information about the data at all (in the case of "total independence"). Therefore, it is significantly different from the models of distributed OT and verifiable distributed OT. The relationship between OT and PIR/SPIR is further studied in [8].

Cheating Prevention in Secret Sharing Another topic related with this paper is Verifiable Secret Sharing (VSS), which can be used as a tool for cheating detection. Although several VSS schemes were proposed in the literature, *e.g.*, [21] and [24], our protocol is based on the scheme in [10]. Besides VSS, coding-theory-based techniques [16, 3] and cheating-immune secret sharing [22, 30] can also be used for cheating detection; however, the contexts of these schemes are different from this paper.

Threshold-based Mobile-Agent Computation An extension of [1] based on threshold cryptography was proposed by Tate and Xu [28]. The problem they study is analogous to ours but they base their solution on multiple agents rather than multiple proxy servers. Consequently, their protocol is different from ours.

2 Definitions

We formulate the problem of VDOT as follows. A VDOT protocol involves a sender, a receiver, and a group of servers, $T_1, \ldots, T_n$. Each of the honest parties is a probabilistic Turing machine who is restricted to run in time polynomial in a security parameter s, while all the dishonest parties are controlled by an adversary who is also a probabilistic Turing machine running in time polynomial in s. We assume an authenticated, untappable channel between the sender (resp., receiver) and each server. Let x_0, x_1 be the two items held privately by the sender. Let $\sigma \in \{0,1\}$ be a private input of the receiver.

A VDOT protocol consists of an initialization stage and a transfer stage.[1] In the initialization stage, the sender sends a function $F_j : \{0,1\}^* \to \{0,1\}^*$ to each server T_j, where F_j depends on (x_0, x_1) and the sender's coin tosses. In the transfer stage, in order to learn x_σ, the receiver sends query q_j to server T_j and receiving reply $r_j = F_j(q_j)$ from T_j. Because the receiver may not send his queries all at once, q_j may depend on the replies to previous queries. After receiving replies from the servers, the receiver decides either to accept the replies (and gives an output $O(r_1, \ldots, r_n)$ which is supposed to be x_σ) or to reject the replies (and output $\perp$ which means cheating is detected).

Now we summarize the security requirements of a VDOT protocol. In the follows, by saying ϵ is *negligible* in s we mean, for all positive polynomial $p(\cdot)$, there exists $s_0 \in \mathcal{N}^+$ such that,

[1] We assume that all participants of the protocol, including the malicious ones, will proceed to the end of the protocol. Therefore, there is no *fairness* problem. This is a reasonable assumption because we *detect* cheating when any participant aborts the protocol, and in many realistic scenarios it is good enough to detect cheating.

for any $s > s_0$,

$$\epsilon(s) < \frac{1}{p(s)}.$$

We say that an event happens with *high probability* if the probability that it does not happen is negligible in s.

Definition 1 *(*correctness*) A VDOT protocol is correct if the receiver's outputs x_σ when all parties follow the protocol.*

Definition 2 *(*receiver's privacy*) A VDOT protocol protects the receiver's privacy against a coalition of the sender and t_1 servers if, for σ chosen uniformly at random from $\{0,1\}$, for any probabilistic polynomial-time adversary that controls a colluding group of the sender and t_1 servers, when all parties out of the colluding group are honest, the probability that the adversary outputs σ is at most $\frac{1}{2} + \epsilon$, where ϵ is negligible in s.*

Definition 3 *(*sender's privacy*) A VDOT protocol protects the sender's privacy against a coalition of the receiver and t_2 servers if, for (x_0, x_1) chosen uniformly at random, for any probabilistic polynomial-time adversary that controls a colluding group of the receiver and t_2 servers, for any random tape the adversary uses, when all parties out of the colluding group are honest, there exists $\sigma' \in \{0,1\}$ such that the probability that the adversary outputs $x_{1-\sigma'}$ is at most $\frac{1}{2} + \epsilon$, where ϵ is negligible in s.*

For verifiability, we require that cheating be detected if it may lead the receiver to compute a false x_σ. On the other hand, if the cheating behavior of some servers does not affect correct reconstruction of x_σ, it will be unnecessary to detect it.

Definition 4 *(*verifiability of reconstruction*) A VDOT protocol is verifiable if, when the sender and the receiver are honest, there exists a probabilistic polynomial-time algorithm V such that*

- $V(r_1, \ldots, r_n) =$ *"accept" if no server cheats;*
- $V(r_1, \ldots, r_n) =$ *"reject" with high probability if* $O(r_1, \ldots, r_n) \neq x_\sigma$.

Remark In the above definition, we do not have any requirement of V's output if some server is cheating but $O(r_1, \ldots, r_n) = x_\sigma$. In this case, both acceptance and rejection will be fine, because there is cheating but it does not affect the reconstruction.

3 VDOT Protocol

In this section, we address the technical challenge mentioned in Section 1 and present our protocol. Before describing our protocol in details, we first review an adapted version of the Bellare-Micali OT protocol, which can be understood as transferring both items encrypted using ElGamal. Then we show how to verify the consistency of secret shares encrypted using ElGamal, which is the key technical contribution of our protocol.

Let p, q be large primes such that $p = 2q + 1$. Denote by G_q the quadratic residue subgroup of Z_p^*. Suppose that g is a generator of G_q. We assume that q has s bits.

3.1 Bellare-Micali OT

Assume that there exists a public random source. In this adapted version of Bellare-Micali OT, the receiver first picks $\delta \in G_q$ using the public random source. Because the receiver has no control over the public random source, he does not know $\log_g \delta$, the discrete logarithm of δ. The receiver then picks $\beta \in [0, q-1]$ and sets

$$G_\sigma = g^\beta, \qquad G_{1-\sigma} = \delta / g^\beta.$$

Note that the receiver knows $\log_g G_\sigma$ but not $\log_g G_{1-\sigma}$. The receiver sends G_0, G_1, δ to the sender, along with a proof that he knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$, using a result by Cramer *et al.* [7]. The sender first verifies that δ has been chosen properly according to the public random source, and $\delta = G_0 G_1$. Then the sender computes, for $b = 0, 1$,

$$\hat{x}_b = x_b G_b^k,$$

where $k \in [0, q-1]$ is the sender's private key and $K = g^k$ her public key.

This OT protocol can be understood as transferring both items in ElGamal ciphertexts. Recall that in the ElGamal encryption scheme, which is semantically secure under the DDH assumption, when cleartext $m \in G_q$ is encrypted with public key K using random string $r \in [0, q-1]$, the ciphertext will be $(mK^r, g^r) = (m(g^r)^k, g^r)$. In the Bellare-Micali OT above, $\hat{x}_b$ can be understood as the first element of the ElGamal ciphertext of x_b, encrypted using random string $\log_g G_b$. For convenience, hereafter we often refer to the first element of an ElGamal ciphertext as *the ciphertext*. In order to decrypt $\hat{x}_b$, a party not knowing k (*e.g.*, the receiver) must know $\log_g G_b$, the random string used for encryption.

The sender gives both $\hat{x}_0$ and $\hat{x}_1$ to the receiver. Because $G_\sigma^k = (g^\beta)^k = K^\beta$, the receiver can reconstruct x_σ by computing

$$x_\sigma = \hat{x}_\sigma / K^\beta,$$

where K is public and β is known to the receiver. However, because the receiver does not know $\log_g G_{1-\sigma}$, he cannot compute $x_{1-\sigma}$.

3.2 Consistency Verification

The basis of our VDOT protocol is actually a distributed version of the above Bellare-Micali OT. The sender distributes shares of the two items, x_0 and x_1 respectively, among the servers; then each server runs the above Bellare-Micali OT with the receiver, to transfer the shares of the two items, such that the shares of x_σ, but not $x_{1-\sigma}$, can be received by the receiver. The privacy properties of our protocol are based on the privacy properties of Bellare-Micali OT. Therefore, our remaining question is how the receiver detects cheating if any server does not transfer the correct share.

To detect cheating, the receiver can verify the *consistency* of shares. More precisely, suppose that $(s_1, \ldots, s_n)$ are the shares of a secret using (n, t)-Shamir secret sharing [27]. Then, for any quorum J ($|J|$=t), any $i \notin J$, it must hold that

$$\sum_{j \in J} s_j \cdot \prod_{l \in J}^{l \neq j} \frac{i-l}{j-l} = s_i.$$

Now suppose that we consider a variant of Shamir scheme by applying a homomorphic mapping $\alpha \to g^\alpha$ to the Shamir scheme. Then, for secret shares $(s_1, \ldots, s_n)$, it must hold that

$$\prod_{j \in J} s_j^{\prod_{l \in J}^{l \neq j} \frac{i-l}{j-l}} = s_i.$$

We say that s_i is consistent with $\{s_j | j \in J\}$ whenever the above equation holds. Therefore, if no share is corrupted, any share should be consistent with any disjoint quorum. But if some shares are corrupted while others are not, with high probability there is inconsistency that can be detected.

However, note that the receiver needs to detect inconsistency of shares of *either* item. For $x_{1-\sigma}$, the receiver only sees the encryptions of its shares, but not its shares in cleartext. To allow the receiver to detect inconsistency on encrypted shares, we need to use a property of Feldman VSS [10].

Specifically, suppose that the servers share k using (n, t)-Feldman VSS. Denote by k_j the share of k held by T_j, and $K_j = g^{k_j}$ the corresponding committment. Then clearly,

$$\prod_{j \in J} K_j^{\prod_{l \in J}^{l \neq j} \frac{i-l}{j-l}} = K_i.$$

Therefore, for the shares $(s_1, \ldots, s_n)$ in the above variant of Shamir scheme,

$$\begin{aligned} \prod_{j \in J} s_j^{\prod_{l \in J}^{l \neq j} \frac{i-l}{j-l}} = s_i \qquad & \Leftrightarrow \qquad \prod_{j \in J} (s_j K_j^\beta)^{\prod_{l \in J}^{l \neq j} \frac{i-l}{j-l}} = s_i K_i^\beta \\ & \Leftrightarrow \qquad \prod_{j \in J} (s_j K_j^{1-\beta})^{\prod_{l \in J}^{l \neq j} \frac{i-l}{j-l}} = s_i K_i^{1-\beta}. \end{aligned}$$

The above means that, to verify consistency among shares (the equation on the left side), the receiver only needs to check an identity on the right side, which only involves encrypted shares (which the receiver is able to see).

3.3 VDOT Protocol Specification

In this subsection, we give the full details of our protocol. A server T_j is called "main server" if $1 \leq j \leq t$; it is called "verification server" otherwise.

System Initialization An (n, t)-Feldman VSS is set up among $T_1, \ldots, T_n$ to share k, the sender's private key.

Step 0: The sender distributes the shares of x_0 and x_1 (in the variant of Shamir's scheme with threshold t), respectively, among $T_1, \ldots, T_n$.

Step 1: The receiver picks $\delta \in G_q$ uniformly at random according to the public random source. He also picks $\beta \in [0, q-1]$ uniformly at random, and computes $G_\sigma = g^\beta$ and $G_{1-\sigma} = \delta / g^\beta$. He sends query (G_0, G_1, δ) to each main server, along with a proof that he knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$.

Step 2: Each main server T_j first verifies that 1) the receiver's proof is valid; 2) δ is chosen properly according to the public random source; and 3) $\delta = G_0 G_1$ ($G_0, G_1, \delta \in G_q$). If all the three conditions are satisfied, T_j computes, for $b = 0, 1$,

$$\hat{s}_{b,j} = s_{b,j} G_b^{\kappa_j}, \tag{1}$$

where $s_{b,j}$ is T_j's share of x_b in the variant of Shamir scheme. T_j sends $(\hat{s}_{0,j}, \hat{s}_{1,j})$ to the receiver.

Step 3: The receiver checks, for each j, that $\hat{s}_{0,j}, \hat{s}_{1,j} \in G_q$. Using the public key of main server T_j, the receiver computes each share of x_σ by

$$s_{\sigma,j} = \hat{s}_{\sigma,j} / K_j^{\beta}.$$

Then the receiver computes

$$x_\sigma = \prod_{j \in \{1,\ldots,t\}} s_{\sigma,j}^{\prod_{l \in \{1,\ldots,t\}}^{l \neq j} \frac{-l}{j-l}}. \tag{2}$$

Step 4: The receiver computes, for $b = 0, 1$ and $i = t+1, \ldots, n$,

$$\hat{s'}_{b,i} = \prod_{j \in \{1,\ldots,t\}} \hat{s}_{b,j}^{\prod_{l \in \{1,\ldots,t\}}^{l \neq j} \frac{i-l}{j-l}}.$$

Then he sends $(G_0, G_1, \hat{s'}_{0,i}, \hat{s'}_{1,i})$ to each verification server T_i.

Step 5: Each verification server T_i tests, for $b = 0, 1$,

$$\hat{s'}_{b,i} \quad = \quad s_{b,i} G_b^{k_i}, \tag{3}$$

and sends the results of comparisons back to the receiver.

Step 6: If for both $b = 0$ and $b = 1$, more than half of the verification servers reply with "yes" (*i.e.*, reply that (3) holds), the receiver accepts the servers' replies and outputs x_σ. Otherwise, the receiver rejects.

4 Security Properties of VDOT

Our VDOT protocol has security properties as follows.

Claim 5 *This VDOT protocol is correct.*

Claim 6 *The VDOT protocol protects the receiver's privacy against a coalition of the sender and all the servers.*

Proof: Consider an adversary that controls the sender and all the servers. It is clear that, no matter how adversary cheats, the message sequnce M sent by the receiver follow a distribution that is *symmetric* in σ. That is, $\forall m \in \{0,1\}^*$,

$$Prob[M = m|\sigma = 0] = Prob[M = m|\sigma = 1] = Pr[M = m] = p_m.$$

Suppose that the adversary's output $O_A = O_A(M, \kappa)$, where κ represents the adversary's knowledge. Then,

$$\begin{aligned}
Prob[O_A = \sigma] &= Prob[O_A = 0|\sigma = 0]Prob[\sigma = 0] + Prob[O_A = 1|\sigma = 1]Prob[\sigma = 1] \\
&= \sum_m Prob[O_A = 0|\sigma = 0 \wedge M = m]Prob[M = m|\sigma = 0]/2 \\
&\quad + \sum_m Prob[O_A = 1|\sigma = 1 \wedge M = m]Prob[M = m|\sigma = 1]/2 \\
&= \sum_m \frac{p_m}{2} Prob[O_A(m, \kappa) = 0|\sigma = 0 \wedge M = m] \\
&\quad + \sum_m \frac{p_m}{2} Prob[O_A(m, \kappa) = 1|\sigma = 1 \wedge M = m].
\end{aligned}$$

Because each m is a constant string, and κ is independent of σ,

$$\begin{aligned}
Prob[O_A = \sigma] &= \sum_m \frac{p_m}{2} Prob[O_A(m, \kappa) = 0|M = m] + \sum_m \frac{p_m}{2} Prob[O_A(m, \kappa) = 1|M = m] \\
&= \sum_m \frac{p_m}{2} \\
&= \frac{1}{2}.
\end{aligned}$$

□

Claim 7 *Under the DDH assumption, the VDOT protocol protects the sender's privacy against a coalition of the receiver and $t-1$ servers.*

Proof: Note that there is at least one honest main server. Because the receiver's proof has been checked by the honest main server(s), it must be the case that the receiver knows either $\log_g G_0$ or $\log_g G_1$. Suppose that the receiver knows $\log_g G_{\sigma'}$; we will show that the adversary outputs $x_{1-\sigma'}$ with probability less than $\frac{1}{2} + \epsilon$.

First, it is clear that, whenever the adversary computes $x_{1-\sigma'}$, it can derive each share of $x_{1-\sigma'}$. For example, it can derive $s_{1-\sigma',j}$, where T_j is an honest main server. Therefore, it will be sufficient if we can show that the adversary computes $s_{1-\sigma',j}$ with probability less than $\frac{1}{2} + \epsilon$.

To determine the probability that the adversary computes $s_{1-\sigma',j}$, let's look at the adversary's interaction with the honest parties. All the adversary learns from the honest parties is the $\hat{s}_{1-\sigma',j}$s from the honest main servers and the replies indicating whether $\hat{s'}_{1-\sigma',i} = s_{1-\sigma',i} G_{1-\sigma'}^{k_i}$ from the honest verification servers. The latter can be ignored because the adversary can compute such replies by checking the following identity itself:

$$\hat{s'}_{1-\sigma',i} = \hat{s}_{1-\sigma',j}^{\prod_{l \in \{1,\ldots,t\}}^{l \neq j} \frac{i-l}{j-l}}.$$

However, the former ($\hat{s}_{1-\sigma',j}$s) are ElGamal encryptions of $s_{1-\sigma',j}$s. Because the ElGamal encryption scheme is semantically secure under the DDH assumption, the probability that the adversary computes $s_{1-\sigma',j}$ must be less than $\frac{1}{2}+\epsilon$. □

Claim 8 *The VDOT protocol is verifiable if the number of dishonest servers is not more than than* $\frac{n-t}{2}$.

Proof: We construct a verification algorithm V as follows. If the majority of the verification servers reply with "yes," then V outputs "accept;" otherwise, V outputs "reject." It is clear that, if no server cheats, all verification server will reply with "yes" and V will output "accept." In the remainder of this proof, we still need to show that, if the receiver's output is not equal to x_σ, then the majority of the verification servers reply with "no." Equivalently, we show that, when the majority of the verification servers reply with "yes," the receiver's output must equal x_σ.

Because the number of dishonest servers is not more than $\frac{n-t}{2}$, among the main servers and the verification servers that reply with "yes," there are at least t that are honest. Suppose that there are h honest main servers and $t-h$ dishonest main servers. Then there are at least $t-h$ honest verification servers that reply with "yes." For each such honest verification server T_i, it must hold that,

$$\hat{s'}_{\sigma,i} = s_{\sigma,i} G_\sigma^{k_i},$$

which implies

$$s_{\sigma,i} G_\sigma^{k_i} = \prod_{j\in\{1,\ldots,t\}} \hat{s}_{\sigma,j}^{\prod_{l\in\{1,\ldots,t\}}^{l\neq j} \frac{i-l}{j-l}}.$$

For $j \in \{1,\ldots,t\}$, we define $\overline{s}_{\sigma,j}$ using $\hat{s}_{\sigma,j} = \overline{s}_{\sigma,j} G_\sigma^{k_j}$. Note that, if T_j is honest, then it must be the case that $\overline{s}_{\sigma,j} = s_{\sigma,j}$. Therefore, we can simplify the above equation as

$$s_{\sigma,i} = \prod_{j\in\{1,\ldots,t\}} \overline{s}_{\sigma,j}^{\prod_{l\in\{1,\ldots,t\}}^{l\neq j} \frac{i-l}{j-l}}.$$

We now have at least $t-h$ $s_{\sigma,i}$s and t $\overline{s}_{\sigma,j}$s that are consistent. Exclude the $t-h$ $\overline{s}_{\sigma,j}$s from dishonest main servers. Then we have at least $t-h$ $s_{\sigma,i}$s and h $\overline{s}_{\sigma,j}$s that are consistent, where each $\overline{s}_{\sigma,j} = s_{\sigma,j}$. These t items uniquely define a degree-$(t-1)$ polynomial, which must be the original polynomial used for secret sharing. The output of the receiver equals this polynomial evaluated at 0, which is exactly x_σ. □

5 Protocol for Mobile-Agent Computation

5.1 A Global Picture of Mobile-Agent Computation

In this section, we apply VDOT to design a secure protocol for mobile agents. Our system architecture, which is a threshold extension to that in [1], is shown in Figure 1.

There are three types of entities in our system architecture: the originator, the hosts, and the servers. The reason for introducing the servers in this mobile computing environment is

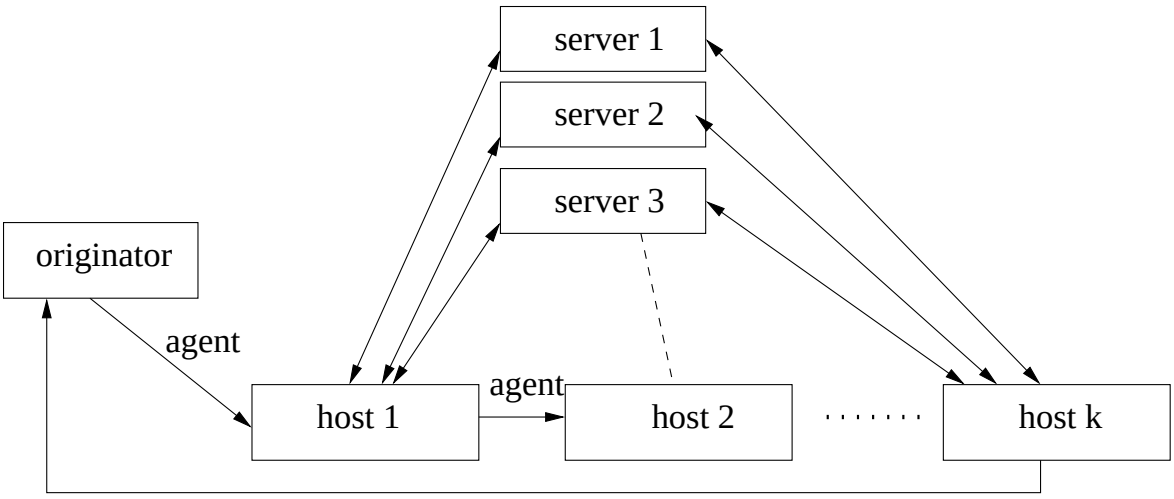

Figure 1: System Architecture for Mobile Agent Computation

that the originator may not always be online. Furthermore, because the majority of Internet users are still using dial-up service, they do not have persistent connections. In such scenarios, the servers serve as a proxy to the originator.

Next we briefly discuss each of the entities.

- Originators The responsibility of an originator is to create an agent and send the agent to the hosts. To improve efficiency, we partition an agent into the security-sensitive portion and the general portion.
- Hosts The responsibility of a host is to run the general portion of an agent and interpret the garbled-circuit portion of the agent. In order to interpret a garbled circuit, the host needs to run the VDOT protocol with the servers to get the appropriate entries from the translation table.
- Servers The responsibility of the servers is to serve as a proxy for an originator and provide translation tables to the hosts through the VDOT protocol.

5.2 Protocol Design of Mobile-Agent Computation

The crucial part of our protocol is how to evaluate the security-sensitive function of an agent. Therefore, we start our presentation of the protocol by describing the encoding of the security-sensitive function.

5.2.1 Encoding of a security-sensitive function

For each host, the originator of an agent encodes the security-sensitive function by a garbled circuit and attaches the circuit to the agent. It is proven in [29] that a garbled circuit never reveals any information (to any polynomially bounded adversary) when it is evaluated. However, to evaluate a garbled circuit, a host needs four translation tables:

- (table In1) A table that translates clear input 1 (the previous state) to garbled input 1.
- (table In2) A table that translates clear input 2 (the local input) to garbled input 2.
- (table Out1) A table that translates garbled output 1 to clear output 1 (the new state);
- (table Out2) A table that translates garbled output 2 to clear output 2 (the local output).

Among the four tables, table Out2 is attached to the agent in cleartext so that the host can obtain its local output immediately after the evaluation.

Tables In1 and Out1 encode the state of the agent. Note that the clear output 1 at host j should be the same as the clear input 1 at host $j+1$. A chaining technique [5, 1] is used to combine the entries of table Out1 at host j with the corresponding entries of table In1 at host $j+1$, the next host. Therefore, as long as host j attaches its garbled output 1 to the agent, host $j+1$ is able to obtain its garbled input 1 which corresponds to the agent's state after visiting host j.

Now the only remaining table is In2. For each bit of input 2, the agent originator holds two items —- the garbled inputs for 0 and 1. Note that we must guarantee that the host receives the item corresponding to its real input bit, but not the other item, because otherwise the host would be able to test the agent with all possible inputs to violate the originator's privacy. So, a VDOT is invoked, with the originator as the sender, the host as the receiver, and the servers as the servers in VDOT. Through this VDOT, the host obtains the garbled input corresponding to its real input bit.

ID	Session identifier
GbCircuit_j	Garbled circuit for host j
GbIn1Host1	Garbled input 1 for host 1
ek_m	The encryption (public) key of the mth server
GbIn1Tab_j(i,b)	The entry of table In1 for host j when the i-th bit of input 1 is b
GbIn2Tab_j(i,b,m)	The m-th share of the entry of table In2 for host j when the i-th bit of input 2 is b
GbOut1Tab_j(i,b)	The entry of table Out1 for host j when the i-th bit of output 1 is b
GbOut2Tab_j	The translation table Out2 for host j

Table 1: Notations in Figure 2

Figure 2 summarizes the data format carried by an agent for a security-sensitive function (the notation is explained in Table 1). In our protocol, we use both asymmetric encryption and symmetric encryption. Here, we denote by $PE(ek, m)$ the asymmetric encryption of cleartext m with encryption key ek; we denote by $E(k, m)$ the symmetric encryption of cleartext m with key k. In our protocol, we require that it be easy to verify whether or not a ciphertext is encrypted with a key in the symmetric encryption scheme. Note that this property can be implemented by adding redundancy to the cleartext before encryption.

Figure 2: Data Format of a Security-Sensitive Function in an Agent

5.2.2 Protocol Summary for Mobile-Agent Computation

When an agent arrives at a host, since In1 is chained to Out1 of the previous host, the host uses the garbled output 1 of the previous host to retrieve its garbled input 1. The host then executes VDOT to obtain the value of garbled input 2 corresponding to its local input.

With both garbled input 1 and garbled input 2, the host evaluates the garbled circuit. After the evaluation, the host uses the attached table Out2 to get its local output. Then it attaches its garbled output 1 to the agent so that the next host can retrieve its garbled input 1 from the agent.

Figure 3 shows the information flow of our protocol at a host.

The last host sends the agent back to the originator. The originator then translates the garbled output 1 to determine the final state of the computation.

5.3 Security Analysis of the Mobile-Agent Protocol

We briefly analyze the security properties of our protocol for mobile-agent computation.

- Originator's Privacy The originator's private information in the security-sensitive portion of the agent is private against any hosts and any servers, unless t or more servers collude.

  This follows from the property of sender's privacy of VDOT. Because only the originator knows the translation tables of input 1 and output 1, the state information (in which the originator's private information is hidden) is private against other parties. Because VDOT ensures that each host can only evaluate the garbled circuit with one value of its

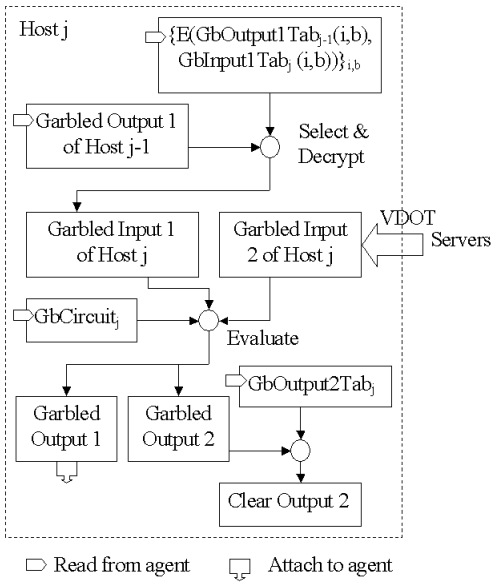

Figure 3: Evaluating a Security-Sensitive Function at Host j

local input, no host is able to extract partial private information from the garbled circuit by evaluating it for multiple times.

- Host's Privacy A host's local input to the agent and local output from the agent are private against the originator, any other hosts and any servers, no matter how many parties involved collude.

  The privacy of local input follows from the property of receiver's privacy of VDOT. Because the local input is not revealed in VDOT, there is no way for other parties to learn about it. The privacy of the local output is obvious.

- Cheating Detection If the servers cheat to change the garbled input 2, the host is able to detect cheating, unless $\frac{n-t}{2}$ or more servers collude.

  This follows from the verifiability of VDOT.

6 Implementation and Performance Evaluation

We have implemented VDOT and garbled circuits, the key components of system. We have also measured preliminary performance.

In our software design, the general portion of an agent will be implemented in Java, while the security-sensitive portion will be encoded as a garbled circuit. Figure 4 shows the components and the information flow at an originator. In our current implementation, a user needs to manually generate a garbled circuit, which should be very small for many applications. In the future, we expect that an automatic circuit generator will be built. Using the automatic circuit generator, a user can generate a circuit for her own use by specifying her own parameters. For example, to build an agent that searches for airline tickets, all a user needs to do is to execute

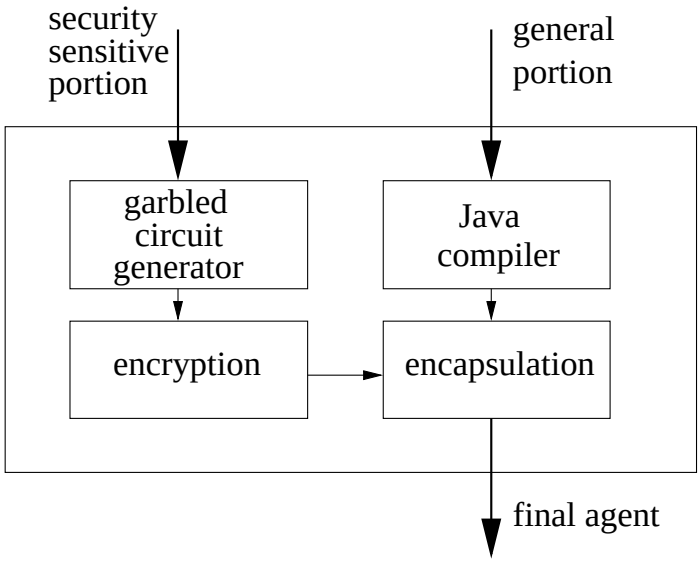

Figure 4: Software Architecture of an Originator

the generator and input her desired flight date, source, destination and price threshold. Then the generator immediately outputs a mobile agent on her behalf.

An agent is sent to hosts for execution. Figure 5 shows the components and the information flow at a host. Because garbled circuits are general purpose and are represented in a platform-independent format, for the purpose of efficiency, our current interpreter is implemented in C.

Obviously, one potential major overhead will be the evaluation of garbled circuits. However, measurement of our prototype interpreter shows that the overhead is very small. Figure 6 shows the overhead of evaluating random garbled circuits of different sizes. The result shows that the overhead of evaluating a garbled circuit of several hundred gates is pretty small.

In order to interpret garbled circuits, the hosts need to interact with the servers through the VDOT protocol to retrieve garbled input 2. Our prototype of VDOT is implemented in C++. We evaluate the overhead of the VDOT protocol on machines with Intel 1.0GHz CPU running Linux. Figure 7 shows the steps of the VDOT protocol and labels the cost of each computational step. The setting of the evaluation is $n = 6$ and $t = 3$. It is clear that the cost of VDOT is acceptable.

7 Conclusions

In this paper, we presented our VDOT scheme. We showed that VDOT is *correct* and it protects both the *receiver's privacy* and the *sender's privacy*. In addition, the correct reconstruction of the transfered item is *verifiable*. Note that VDOT can be further extended to consider a malicious sender, who may intentionally mislead the servers so that they can be accused of cheating. The protocol we have shown can be easily adapted to solve this problem, if revised slightly. We ignore this extension for simplicity.

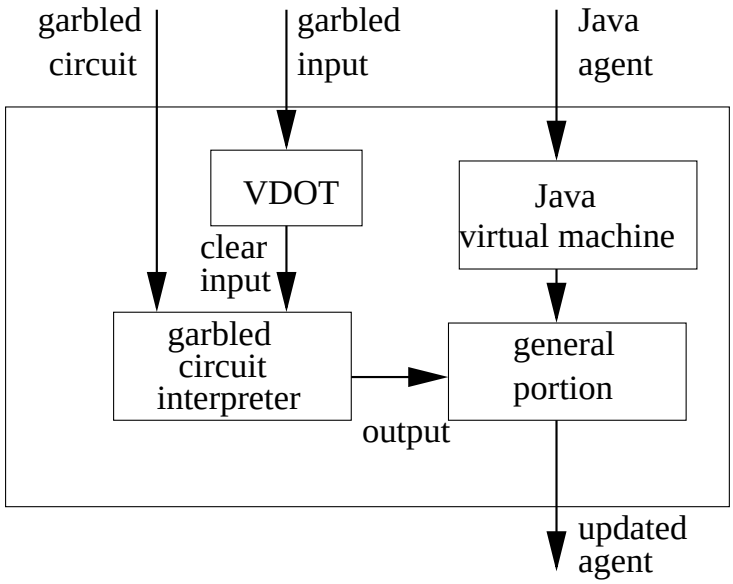

Figure 5: Components of a Host

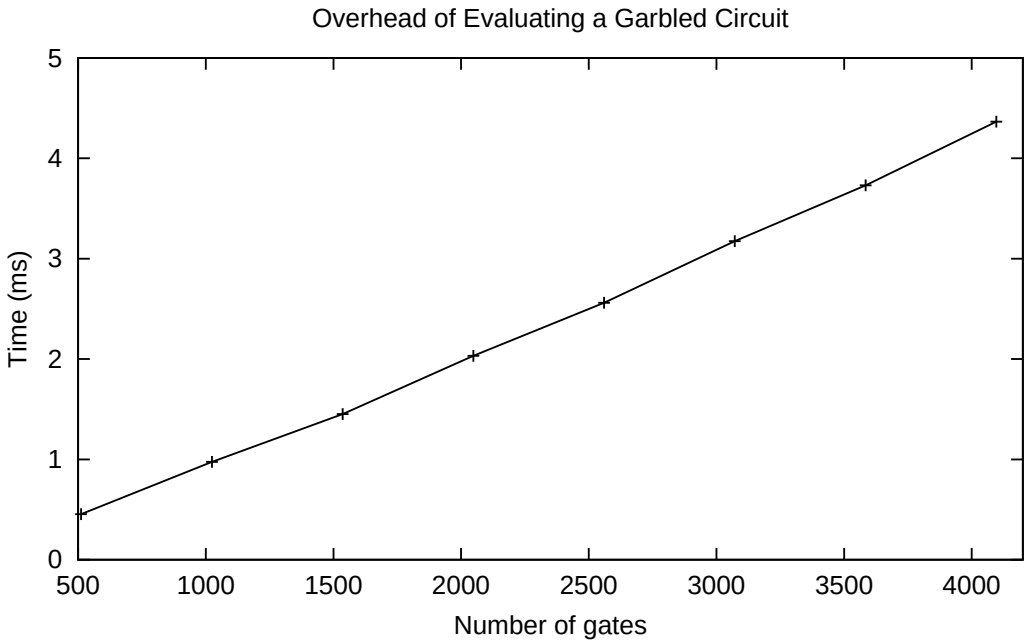

Figure 6: Overhead of Evaluating a Garbled Circuit

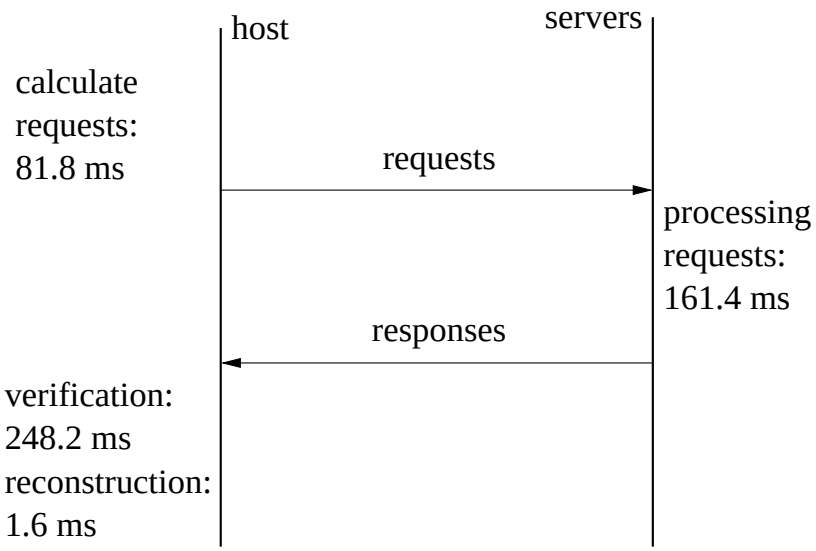

Figure 7: Overhead of VDOT ($(n, t) = (6, 3)$)

We also presented a system for secure mobile-agent computation. Our system partitions an agent into the general portion and the security-sensitive portion. Our system protects the privacy of both the originator and the hosts, without using any single trusted party. We also designed and implemented the key components of our system. As far as we know, this is the first effort to implement a system that protects the privacy of mobile agents. Our preliminary evaluation shows that protecting mobile agents is not only possible, but also can be implemented efficiently.

Our current major implementation effort is seamless integration between the general portion and the security-sensitive portion of an agent. A strong programming support environment is desired.

References

[1] Joy Algesheimer, Christian Cachin, Jan Camenisch, and Gunter Karjoth. Cryptographic security for mobile code. In *IEEE Symposium on Security and Privacy*, pages 2–11. IEEE, 2001.

[2] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology - Proceedings of CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 547–557, 1990.

[3] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, 1968.

[4] Gilles Brassard, Claude Crepeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *Advances in Cryptology - Proceedings of CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pages 234–238, 1986.

[5] Christian Cachin, Jan Camenisch, Joe Kilian, and Joy Muller. One-round secure computation and secure autonomous mobile agents. In *Automata, Languages and Programming, 27th International Colloquium,*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523, 2000.

[6] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–982, November 1998.

[7] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - Proceedings of CRYPTO 94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.

[8] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *Advances in Cryptology - Proceedings of EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138, 2000.

[9] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28:637–647, 1985.

[10] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 427–437, 1987.

[11] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In *RANDOM'98*, volume 1518 of *Lecture Notes in Computer Science*, pages 200–217, 1998.

[12] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 151–160, 1998.

[13] Oded Goldreich. Secure multi-party computation. Working Draft Version 1.1, 1998.

[14] L. Gong. Java security architecture (JDK1.2). Technical report, Sun Microsystems, 1998.

[15] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[16] R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[17] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 245–254, 1999.

[18] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *Advances in Cryptology - Proceedings of CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 573–590, 1999.

[19] Moni Naor and Benny Pinkas. Distributed oblivious transfer. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 205–219, 2000.

[20] George C. Necula and Peter Lee. Safe, untrusted agents using proof-carrying code. In *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 61–91, 1998.

[21] T. P. Pedersen. Non-interactive and information-theoretical secure verifiable secret sharing. In *Advances in Cryptology - Proceedings of CRYPTO 91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, 1992.

[22] Josef Pieprzyk and Xian-Mo Zhang. Cheating prevention in secret sharing over $GF(p^t)$. In *INDOCRYPT 2001*, volume 2247 of *Lecture Notes in Computer Science*, pages 79–90, 2001.

[23] M. Rabin. How to exchange secrets by oblivious transfer. *Tech. memo TR-81, Aiken Computation Laboratory, Havard U.*, 1981.

[24] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the 21th Annual ACM Symposium on the Theory of Computing*, pages 73–85, 1989.

[25] Tomas Sander and Christian Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 44–60, 1998.

[26] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 554–567. ACM, 1998.

[27] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[28] S. R. Tate and K. Xu. Mobile agent security through multi-agent cryptographic protocols. In *Proceedings of the 4th International Conference on Internet Computing (IC 2003)*, pages 462–468, 2003.

[29] A. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[30] Xian-Mo Zhang and Josef Pieprzyk. Cheating immune secret sharing. In *ICICS 2001*, volume 2229 of *Lecture Notes in Computer Science*, pages 144–149, 2001.

ACKNOWLEDGEMENT

The authors are indebted to Joan Feigenbaum and the anonymous reviewers for their insightful comments.