Verifiable Distributed Oblivious Transfer and Mobile Agent Security*

Sheng Zhong
Department of Computer Science
Yale University
New Haven, CT 06520-8285
sheng.zhong@yale.edu

Yang Richard Yang
Department of Computer Science
Yale University
New Haven, CT 06520-8285
yry@cs.yale.edu

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 two novel techniques, *consistency verification of encrypted secret shares* and *consistency verification through re-randomization*. 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

Secure mobile agents and mobile code, Resiliency to corruptions, Oblivious Transfer, Verifiable Secret Sharing

1. INTRODUCTION

The mobile agent is a fundamental building block of the mobile computing paradigm. 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 [15], 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 Tschudin's pioneering work [23], various systems have been designed for this purpose [24, 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 [19] (DOT), 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 (*i.e.*, honest but curious) 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.[1]

The design of VDOT is technically challenging. One might

*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. Because of space limitation, many technical points have been abbreviated or omitted. Details of Verifiable Distributed Oblivious Transfer will be included in a more complete version of this paper.

DIALM-POMC '03 San Diego, California, USA

[1]Besides considering malicious servers, VDOT has a few subtle differences from DOT. Detailed comparisons of VDOT with DOT and other related works [22, 9, 4, 17, 18, 6, 16, 13, 12, 8] can be found in [27].

suggest that the objective of VDOT could be achieved by a secret-sharing scheme with oblivious transfer of each share. However, there are two main technical challenges. First, 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. However, at the same time, in order to protect the sender's privacy, the receiver should be able to reconstruct only one of the two items. In summary, the *first challenge* is to allow the receiver to reconstruct only one item but verify the correctness of both items. Second, when the receiver detects that some shares have been tampered with, the receiver must be able to identify the cheating servers and accuse them with evidence verifiable by the public. In addition, during the identification procedure, the privacy of both the sender and the receiver needs to be protected. In summary, the *second challenge* is to allow the receiver to identify and publicly accuse the cheating servers without compromising the privacy of either the sender or the receiver.

Our VDOT protocol uses novel techniques to address the above two challenges. An overview of the VDOT protocol is as follows. During initialization, Feldman's Verifiable Secret Sharing [10] (VSS) of keys is set up among the servers. 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 her both items (using a variant of the Shamir scheme [26]) 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 encrypted shares of both items are transfered to the receiver during the OT. If cheating is detected, the receiver can perform the cheater-identification procedure, which uses the novel technique *cheater identification through re-randomization* and allows the receiver to identify the cheating servers but still protects the privacy of the sender and the receiver.

We also apply VDOT to mobile agent security and 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. Since we only apply the security mechanism to the security-sensitive portion of an agent, our system is efficient. Since 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.[2] 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, our 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, VDOT identifies malicious servers without compromising the privacy of either the sender or the receiver, using consistency verification through re-randomization. Fourth, we apply VDOT to the problem of mobile agent security and implement the key components of an architecture for mobile agents.

The rest of this paper is organized as follows. In Section 2, we present the definitions; in Section 3, we present how VDOT implements consistency verification of encrypted shares; in Section 4, we present how VDOT implements cheater detection. We prove the security properties of VDOT in Section 5. In Section 6, we present how to apply VDOT to a mobile-agent system. In Section 7, we present implementation issues and report initial performance. We conclude in Section 8.

2. DEFINITIONS

We formulate the problem of VDOT as follows.

Let k be a security parameter. A VDOT protocol involves a sender, a receiver, and a group of servers, $T_1, \ldots, T_t$, where $t = O(\log(k))$. Each of these parties is a probabilistic Turing machine. We assume that the computational power of each party is polynomially bounded in k. In the sequel, unless specified otherwise, when we say polynomial complexity, we mean polynomial in k. We assume an authenticated, untappable channel between the sender (resp., receiver) and each server. Let $s_0, s_1 \in Z_Q$ be the two items held privately by the sender, where Q is a prime of length k. Let $\sigma \in \{0,1\}$ be a private input of the receiver.

A VDOT protocol consists of an initialization stage and a transfer stage. In the initialization stage, the sender sends function $F_j : \{0,1\}^* \rightarrow \{0,1\}^*$ to each server T_j, where F_j depends on (s_0, s_1) and the sender's coin tosses. In the transfer stage, in order to learn s_σ, the receiver first carries out the share-transfer procedure by sending query q_j to server T_j and receiving reply $r_j = F_j(q_j)$ from T_j. Since 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 reconstruct s_σ) or to reject the replies. In the latter case, he may further carry out the cheater-identification procedure and interact with the servers in order to identify the cheater(s). We use μ to denote the receiver's view in the further interaction, if there is any. The receiver finally accuses cheating servers with evidence e.

We summarize the security requirements of a VDOT protocol as follows. Below, when we say *high probability* (resp., *negligible probability*), we mean probability p such that for any polynomial f, there exists k_0 such that, for any $k > k_0$, $p > 1 - \frac{1}{f(k)}$ (resp., $p < \frac{1}{f(k)}$). When we say that two views are *computationally indistinguishable*, we mean that, for any *probabilistic polynomial time (PPT)* distinguisher, if the input is picked at random from the distributions of these two views, respectively, the difference of the probabilities of outputing 1 is negligible.

Definition 1. *(*correctness*) A VDOT protocol is* cor-

[2]The garbled circuit interpreter can also be carried by the agent, in principle. However, requiring each host to provide a garbled circuit interpreter can greatly reduce the communication overhead.

rect, *if there exists a PPT algorithm $\mathcal{R}$ for the receiver that computes s_σ from $(F_1(q_1), \ldots, F_t(q_t))$ when all parties follow the protocol.*

DEFINITION 2. *(*receiver's privacy*) In a VDOT protocol, σ is private against a coalition of the sender and τ_1 servers if, for any colluding group of a dishonest sender and τ_1 dishonest servers, there exists a PPT simulator S that can replace the receiver in the following sense: The view of the above colluding group interacting with S is computationally indistinguishable from the view of this group interacting with the receiver.*

DEFINITION 3. *(*sender's privacy*) One of the two items, s_0 and s_1, is private against a coalition of the receiver and τ_2 servers if, for any colluding group of a dishonest receiver and τ_2 dishonest servers, there exists a PPT simulator S' that takes only one of the two items as input, and can replace the sender in the following sense: The view of the above colluding group interacting with S' and honest servers is computationally indistinguishable from the view of this group interacting with the sender and the honest servers.*

We consider two types of verifiability — *verifiability of reconstruction* and *verifiability of accusation*. For verifiability of reconstruction, we require that cheating be detected if it may lead the receiver to compute a false s_σ. On the other hand, if the cheating behavior of some servers does not affect correct reconstruction of s_σ, it will be unnecessary to detect it.

DEFINITION 4. *(*verifiability of reconstruction*) A VDOT protocol is reconstruction-verifiable if there exists a PPT algorithm for the receiver that accepts the replies $(r_1, \ldots, r_t)$ when $(r_1, \ldots, r_t) = (F_1(q_1), \ldots, F_t(q_t))$ and rejects when*

$$\mathcal{R}(r_1, \ldots, r_t) \neq \mathcal{R}(F_1(q_1), \ldots, F_t(q_t)).$$

Intuitively, when one or more servers try to cheat, an *accusation-verifiable* protocol allows the receiver to identify at least one of the cheating servers and show convincing evidence to the public. At the same time, an accusation-verifiable protocol does not allow the receiver to succeed in falsely accusing any server.

DEFINITION 5. *(*verifiability of accusation*) A VDOT protocol is accusation-verifiable if there exists a PPT cheater-identification algorithm for the receiver and a PPT verification algorithm for the public. If cheating is detected, the cheater-identification algorithm outputs a non-empty set of T_j such that $r_j \neq F_j(q_j)$ and computes e, evidence of cheating, from $(r_1, \ldots, r_t)$ and μ. The public's verification algorithm accepts e if and only if all servers accused by the receiver really cheated in the accused way.*[3]

[3]If for any accused T_j, $r_j = F_j(q_j)$, then obviously the verification algorithm should reject. But even if for each accused T_j, $r_j \neq F_j(q_j)$, the verification algorithm may or may not accept — it accepts only when e is valid evidence of the accused type of cheating. In other words, if a server cheats in one way and the cheater-identification algorithm falsely accuses the server of another type of cheating, the verification algorithm should still reject. The distinct types of cheating are described in Sections 3 and 4.

3. CONSISTENCY VERIFICATION OF ENCRYPTED SHARES AND THE SHARE-TRANSFER PROCEDURE

In this section, we address the first challenge and present the share-transfer procedure of our protocol. This section is organized as follows. 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 review a variant of the Shamir scheme and the Feldman VSS. Next we show how they can be adapted to verify the consistency of secret shares encrypted using ElGamal. Finally we present the share-transfer procedure of our protocol, which is based on the Bellare-Micali OT and the Feldman VSS.

In the sequel, let P, Q be two large primes such that $P = 2Q + 1$. Let G_Q be the quadratic residue subgroup of Z_P^*. Let g be a generator of G_Q. The standard Decisional Diffie-Hellman (DDH) assumption states that, for x, y, z picked uniformly at random from $[0, Q-1]$ $(= Z_Q)$, (g^x, g^y, g^z) is computationally indistinguishable from (g^x, g^y, g^{xy}) [3]. This implies that no PPT algorithm can compute the discrete logarithm of a random element in G_Q.

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. Since 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 $\beta \in Z_Q$ and sets $G_\sigma = g^\beta$, $G_{1-\sigma} = \delta / g^\beta$. 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 $b = 0, 1$, $\hat{s}_b = s_b G_b^\kappa$, where $\kappa \in Z_Q$ is the sender's private key and g^κ her public key.

This OT protocol can be understood as transferring both shares in ElGamal ciphertext. Recall that in the ElGamal encryption scheme, which is semantically secure under the DDH assumption, when plaintext $s \in G_Q$ is encrypted with private key κ using random string $r \in Z_Q$, the ciphertext will be $(sg^{\kappa r}, g^r)$. In the Bellare-Micali OT above, $\hat{s}_b$ can be understood as the first element of the ElGamal ciphertext of s_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 $\hat{s}_b$, a party not knowing κ (*e.g.*, the receiver) must know $\log_g G_b$, the random string used for encryption.

The sender gives both $\hat{s}_0$ and $\hat{s}_1$ to the receiver. Since $G_\sigma^\kappa = (g^\beta)^\kappa = (g^\kappa)^\beta$, the receiver can reconstruct s_σ by computing $s_\sigma = \hat{s}_\sigma / (g^\kappa)^\beta$, where g^κ is public and β is known to the receiver. However, since the receiver does not know $\log_g G_{1-\sigma}$, he cannot compute $s_{1-\sigma}$.

Since a server sends its two shares to the receiver as ElGamal ciphertexts, our first technical challenge can be addressed if we can verify the consistency of secret shares encrypted using ElGamal. Before presenting our scheme, we review a variant of the Shamir scheme and the Feldman VSS.

A Variant of the Shamir Scheme The variant of the Shamir scheme we use is defined on G_Q by applying the homomorphic mapping $\alpha \to g^\alpha$ to the original Shamir scheme on Z_Q. Therefore, with a (t, τ)-secret sharing of $s \in G_Q$,

the jth share of the secret is $s_{(j)} = s\prod_{\theta=1}^{\tau-1} g^{a_\theta j^\theta}$ (for $j = 1, \ldots, t$),[4] where a_θ's are random coefficients.

Now consider the possbility of tampering. We say a share is a *true share* if it has not been tampered with by its holder; otherwise, we say the share is a *false share*. If $\{s_{(j)}|j \in J\}$ are all true shares, and $s_{(\pi)}$ is also a true share, where $\pi \in \{1, 2, \ldots, t\} - J$, we have

$$\prod_{j\in J} s_{(j)}^{\prod_{\theta\in J}^{\theta\neq j} \frac{\pi-\theta}{j-\theta}} = s_{(\pi)}. \quad (1)$$

DEFINITION 6. *In the above variant of the Shamir scheme, let $s_{(1)}, \ldots, s_{(t)}$ be the shares submitted by the servers,* which may or may not be true. *If (1) is satisfied, then we say that $s_{(\pi)}$ is consistent with $\{s_{(j)}|j \in J\}$; otherwise, we say that $s_{(\pi)}$ is inconsistent with $\{s_{(j)}|j \in J\}$.*

LEMMA 7. *If $s_{(\pi)}$ is consistent with $\{s_{(j)}|j \in J\}$, then for any $j_0 \in J$, $s_{(j_0)}$ is consistent with $\{s_{(j)}|j \in J - \{j_0\} \cup \{\pi\}\}$, and the two sets of shares, $\{s_{(j)}|j \in J\}$ and $\{s_{(j)}|j \in J - \{j_0\} \cup \{\pi\}\}$, return the same s if used for reconstructing s.*
Remark Intuitively, this lemma indicates that exchanging $s_{(\pi)}$ with $s_{(j_0)}$ ($j_0 \in J$) does not break the consistency or change the reconstruction, provided that $s_{(\pi)}$ is consistent with $\{s_{(j)}|j \in J\}$. The proof is simple elementary algebra.

LEMMA 8. *One share is true if and only if it is consistent with any set of τ true shares.*

LEMMA 9. *Assume that the number of false shares is less than $\frac{t-\tau}{2}$. For any $J \subseteq \{1, \ldots, t\}$, $|J| = \tau$, $\{s_{(j)}|j \in J\}$ is a set of true shares if and only if more than $\frac{t-\tau}{2}$ shares with indices outside J are consistent with $\{s_{(j)}|j \in J\}$.*

PROOF. If $\{s_{(j)}|j \in J\}$ is a set of true shares, then the left hand side of (1) computes the untampered value of $s_{(\pi)}$. Therefore, each true share $s_{(\pi)}$ is consistent with $\{s_{(j)}|j \in J\}$. Because there are fewer than $\frac{t-\tau}{2}$ false shares, there must be more than $\frac{t-\tau}{2}$ true shares with indices outside J. These true shares are all consistent with $\{s_{(j)}|j \in J\}$.

If more than $\frac{t-\tau}{2}$ shares with indices outside J are consistent with $\{s_{(j)}|j \in J\}$, let $J_0 = J \cup \{\pi | s_{(\pi)}$ is consistent with $\{s_{(j)}|j \in J\}\}$. Because $|J_0| > \tau + \frac{t-\tau}{2}$, and because there are fewer than $\frac{t-\tau}{2}$ false shares, $\{s_{(j)}|j \in J_0\}$ must include more than τ true shares. Take a set of τ true shares, and we know that all other shares with indices in J_0 are consistent with this set of τ true shares, by Lemma 7. Therefore, all shares with indices in J_0 are true, by Lemma 8. Specifically, $\{s_{(j)}|j \in J\}$ is a set of true shares. □

Feldman VSS We can view the Feldman VSS as a combination of the Shamir schemen and its variant described above. Assume $s \in Z_Q$ is a secret shared in the Feldman VSS, and $s_{(j)}$ the jth share of s in the Feldman VSS, held by server T_j. Then $s_{(j)}$ is exactly the jth share of s in the Shamir scheme, while $g^{s_{(j)}}$, T_j's commitment to $s_{(j)}$, can be understood as the jth share of g^s in the variant of the Shamir scheme using the same random coefficients. As we will see shortly, this setup is important to the verification of the consistency of encrypted shares.

[4]We add a pair of parenthesis to the subscript of the jth share, $s_{(j)}$, in order to distinguish it from the two private items, s_0 and s_1.

One way to set up the Feldman VSS among the t servers is to use a single trusted party. Note that if we use a trusted party to set up the Feldman VSS for a protocol, the protocol can be compromised if the trusted party is corrupted. Another possible solution without using a trusted party is to use a general-purpose protocol for secure multi-party computation, which is less efficient. However, we need to set up VSS only once, and then we can reuse the set-up for multiple transfers. Therefore, both of the solutions above will be acceptable. (Although Pedersen's protocol [21] was proposed to efficiently set up VSS without a trusted party, a flaw has been identified [11].)

Consistency Verification on Encrypted Shares With a setup of the Feldman VSS, we can check the consistency of secret shares encrypted using ElGamal. The following lemma articulates this fact.

LEMMA 10. *Consider an ElGamal cryptosystem using generator g. Suppose that t servers use the Feldman VSS with threshold τ to share an implicit global private key κ. Each server T_j regards its share κ_j as its own private key, and uses the commitment to its share, g^{κ_j}, as its public key. Suppose that each server T_j encrypts plaintext $s_{(j)}$ using ElGamal, where $s_{(j)}$ is the (possibly-tampered) jth share of a secret s in the variant of the Shamir scheme using threshold τ. If all servers use the same random string $r \in Z_Q$ for encryption, then there exists a PPT algorithm that takes the ciphertexts of any $\tau+1$ shares of s as input and outputs whether or not these shares are consistent.*

PROOF. The shares are consistent if and only if (1) is satisfied. On the other hand, because κ_j is the untampered jth share of κ in the Feldman VSS, g^{κ_j} is the untampered jth share of g^κ in the variant of the Shamir scheme. Therefore, g^{κ_j}'s are consistent, *i.e.*, they satisfy

$$\prod_{j\in J} (g^{\kappa_j})^{\prod_{\theta\in J}^{\theta\neq j} \frac{\pi-\theta}{j-\theta}} = g^{\kappa_\pi}. \quad (2)$$

Assume that the common random string used for encryption is r. Then (2) implies that (1) $\Leftrightarrow$

$$\prod_{j\in J} (s_{(j)} g^{\kappa_j r})^{\prod_{\theta\in J}^{\theta\neq j} \frac{\pi-\theta}{j-\theta}} = s_{(\pi)} g^{\kappa_\pi r}. \quad (3)$$

Because T_j's ciphertext is $(s_{(j)} g^{\kappa_j r}, g^r)$, (3) can be checked with only ciphertexts. □

In order to simplify formulae, we define

$$CShare(\{c_j|j \in J\}, \pi) = \prod_{j\in J} c_j^{\prod_{\theta\in J}^{\theta\neq j} \frac{\pi-\theta}{j-\theta}}. \quad (4)$$

Therefore, for consistency verification on ciphertexts $\{c_j|j \in J\}$ and c_π, it is sufficient to compare $CShare(\{c_j|j \in J\}, \pi)$ with c_π.

Building the Share-Transfer Procedure of our Protocol Given the above OT and secret sharing schemes, we address the first challenge as follows. The Feldman VSS of keys is established among the servers; the sender shares s_0^2, s_1^2 in the variant of the Shamir scheme[5] and distributes

[5]We use the bijection $\alpha \to \alpha^2$ to map the secret from Z_Q to G_Q. Note that there is an efficient algorithm to compute the modular square root because P is a prime.

the shares among the servers; an instance of Bellare-Micali OT of shares is invoked between each server and the receiver; the receiver can only reconstruct one private item, but he can verify the consistency of the shares of either item, using the ciphertexts transfered.

In order to protect the privacy of the servers, we need to reduce the number of shares the receiver can learn. For this end, although there is no difference in the servers with regard to the shares, we distinguish between two classes of servers: the first τ servers, $T_1, \ldots, T_\tau$, are called *main* servers, while the rest servers are called *verification* servers. Only the main server's shares will be transfered to the receiver using the Bellare-Micali OT. Then the receiver computes the ciphertexts of the verification servers' shares from the ciphertexts of the main servers' shares, using (4), and sends them to the verification servers for comparison. The results of these comparisons reflect whether or not the shares are consistent.

We summarize the initialization stage and the share-transfer procedure of our protocol as follows.

Overall System-Initialization

A Feldman VSS of keys as described in Lemma 10 is set up among $T_1, \ldots, T_t$.

Sender-Initialization

The sender distributes the shares of s_0^2 and $s_1^2 \in G_Q$ (in the variant of Shamir's scheme with threshold τ), respectively, among $T_1, \ldots, T_t$.

The Share-Transfer Procedure

Step 1: The receiver picks $\delta \in G_Q$ uniformly at random according to the public random source. He also picks $\beta \in Z_Q$ 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 $\log_g G_1$. (Note that we assume the random oracle model and all proofs we use are non-interactive.)

Step 2: Each main server T_j first verifies that 1) there is no previous query from the receiver in this session; 2) δ is chosen properly according to the public random source; and 3) $\delta = G_0 G_1$. If all the three conditions are satisfied, T_j computes, for $b = 0, 1$,

$$\hat{s}_{b,j} = s_{b,j} G_b^{\kappa_j}, \tag{5}$$

and sends $(\hat{s}_{0,j}, \hat{s}_{1,j})$ to the receiver, together with a receipt of (G_0, G_1).

Step 3: Using the public key of main server T_j, the receiver computes a share of s_σ^2 by $s_{\sigma,j} = \hat{s}_{\sigma,j} / (g^{\kappa_j})^\beta$. Then the receiver computes

$$s_\sigma^2 = \prod_{j \in J} s_{\sigma,j}^{\prod_{\theta \in J}^{\theta \neq j} \frac{-\theta}{j-\theta}}, \tag{6}$$

where $J = \{1, \ldots, \tau\}$. He further computes s_σ from s_σ^2.

Step 4: The receiver computes, for $b = 0, 1$ and $\pi = \tau + 1, \ldots, t$, $\hat{s'}_{b,\pi} = CShare(\{\hat{s}_{b,j} | j \in J\}, \pi)$. Then he sends $(G_0, G_1, \hat{s'}_{0,\pi}, \hat{s'}_{1,\pi})$ to each verification server T_π.

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

$$\hat{s'}_{b,\pi} = s_{b,\pi} G_b^{\kappa_\pi}, \tag{7}$$

and sends the results of comparisons back to the receiver, together with a receipt of $(G_0, G_1, \hat{s'}_{0,\pi}, \hat{s'}_{1,\pi})$.

Step 6: If for both $b = 0$ and $b = 1$, more than half of the verification servers reply with "yes" (*i.e.*, reply that (7) holds), the receiver outputs s_σ and accuses those verification servers who reply with "no" for either $b = 0$ or $b = 1$. This is called "type-1 accusation" and the evidence consists of the replies from all servers. The protocol finishes successfully. If for either $b = 0$ or $b = 1$, the number of verification servers replying with "yes" is less than half (we require $t \not\equiv \tau \pmod 2$ in order to avoid a tie), a cheater-identification procedure is invoked (see Section 4).

4. CHEATER IDENTIFICATION AND ACCUSATION

In the previous section, we have presented the share-transfer procedure of our protocol and shown how the receiver verifies the consistency of the encrypted shares. In this section, we present a procedure that can further identify and publicly accuse at least one of the cheating servers.

Before the cheater identification, the receiver has already collected τ encrypted shares, $\{\hat{s}_{b,j} | j \in J\}$, for $b = 0, 1$. In addition, the consistency/inconsistency of each $\hat{s}_{b,\pi} = s_{b,\pi} G_b^{\kappa_\pi}$ $(\pi \notin J)$ with $\{\hat{s}_{b,j} | j \in J\}$ has also been revealed by T_π, maybe incorrectly. In order to identify the cheaters, the receiver needs to check the consistency/inconsistency of $\hat{s}_{b,\pi}$ $(\pi \notin J')$ with $\{\hat{s}_{b,j} | j \in J'\}$, where $J' \subseteq \{1, \ldots, t\}$, $|J'| = \tau$ and $J' \neq J$. On the other hand, in order to protect the privacy of the sender, the servers should not reveal more shares to the receiver. That is, the above checking of consistency should not reveal any $\hat{s}_{b,\pi}$ for $\pi \notin J$ to the receiver. Our solution to this problem is called *consistency verification through re-randomization*.

Consistency Verification through Re-Randomization

The main idea of consistency verification through re-randomization is that each server T_j $(j = 1, \ldots, t)$ encrypts $s_{b,j}$ using $r' \in Z_Q$, a common random string *that is unknown to all parties*. Specifically, each server T_j $(j = 1, \ldots, t)$ calculates a new ciphertext of its share, $(s_{b,j} g^{\kappa_j r'}, g^{r'})$, and sends the new ciphertext to the receiver. The receiver can then check for consistency among the new ciphertexts.

In order to prevent servers from further tampering with their shares (or switching back from tampered shares to true shares), for $j \in J$, T_j should prove that its new ciphertext $\tilde{s}_{b,j}$, which is encrypted using random string r', is just a re-randomization of its previous ciphertext $\hat{s}_{b,j}$. For the same reason, for $\pi \notin J$, the receiver checks that the consistency/inconsistency regarding T_π's new ciphertext is the same as that regarding T_π's old ciphertext.

We summarize the cheater-identification procedure of our protocol as follows.

The Cheater-Identification Procedure

Step 1: All the servers pick a common string, $G \in G_Q$, at random, according to the public random source. Each server T_j $(j = 1, \ldots, t)$ computes, for $b = 0, 1$,

$$\tilde{s}_{b,j} = s_{b,j} G^{\kappa_j},$$

and sends $\tilde{s}_{0,j}, \tilde{s}_{1,j}$ to the receiver. Each main server T_j proves to the receiver, in zero knowledge [25],

$$\log_{\frac{G}{G_b}} \frac{\tilde{s}_{b,j}}{\hat{s}_{b,j}} = \log_g g^{\kappa_j}. \tag{8}$$

Step 2: The receiver verifies all main servers' proofs. If a main server provides an invalid proof, the receiver aborts the protocol and accuses the main server. This is called "type-2 accusation" and the evidence is the invalid proof provided by the main server.

Step 3: The receiver verifies that, for $b = 0, 1$, and for each T_π ($\pi \notin J$) that replied with "yes" for this b,

$$\tilde{s}_{b,\pi} = CShare(\{\tilde{s}_{b,j} | j \in J\}, \pi), \tag{9}$$

holds; he also verifies that, for $b = 0, 1$, and for each T_π ($\pi \notin J$) that replied with "no" for this b, (9) does not hold. If the above is not true for any b and any T_π, the receiver aborts the protocol and accuses T_π. This is called "type-3 accusation" and the evidence for a type-3 accusation of T_π is all main servers' replies and proofs, and also T_π's reply together with $\tilde{s}_{b,1}, \ldots, \tilde{s}_{b,\tau}, \tilde{s}_{b,\pi}$.

Step 4: The receiver searches for a set $J' \subseteq \{1, \ldots, t\}$, $|J'| = \tau$ such that for $b = 0, 1$,

$$\tilde{s}_{b,\pi} = CShare(\{\tilde{s}_{b,j} | j \in J'\}, \pi) \tag{10}$$

holds for more than $\frac{t-\tau}{2}$ choices of $\pi \in \{1, \ldots, t\} - J'$. (The complexity of the search is exponential in t. However, since $t = O(\log(k))$, the complexity is still polynomial in k.) The receiver accuses the T_π's for which Equation (10) does not hold for $b = 0$ or $b = 1$. This is called "type-4" accusation and the evidence consists of the replies from all servers, all main servers' proofs, $\tilde{s}_{b,1}, \ldots, \tilde{s}_{b,t}$, and set J'. After a type-4 accusation, the receiver uses

$$s_\sigma^2 = \prod_{j \in J'} s_{\sigma,j}^{\prod_{\theta \in J', \theta \neq j} \frac{-\theta}{j-\theta}}$$

to compute s_σ^2, and then further computes s_σ. The protocol finishes successfully.

5. SECURITY PROPERTIES OF VDOT

Our verifiable distributed OT protocol has security properties as follows.

CLAIM 11. *(correctness) This verifiable distributed OT protocol is correct.*

PROOF. According to the definition of correctness, we need to show that there exists a PPT algorithm for the receiver to compute s_σ from the servers' responses when every party is honest. As described in the protocol, if all parties are honest, the response from each main server T_j is $(\hat{s}_{0,j}, \hat{s}_{1,j})$, which is computed from (5). By (5) we know

$$\begin{aligned} s_{\sigma,j} &= \hat{s}_{\sigma,j}/(G_\sigma)^{\kappa_j} \\ &= \hat{s}_{\sigma,j}/(g^\beta)^{\kappa_j} \\ &= \hat{s}_{\sigma,j}/(g^{\kappa_j})^\beta. \end{aligned}$$

Because g^{κ_j} is public, the receiver can easily compute $s_{\sigma,j}$ ($1 \leq j \leq \tau$) from the (main) servers' responses. Because $s_{\sigma,1}, \ldots, s_{\sigma,\tau}$ are the shares of s_σ^2, the receiver can further reconstruct s_σ^2 from them, using formula (6). Since the modulus P is a prime number, the receiver can compute s_σ from s_σ^2 in polynomial time. □

CLAIM 12. *(receiver's privacy) σ is private against a coalition of the sender and all t servers.*

PROOF. We constructs S as follows.

First, S picks $\delta, G_0 \in G_Q$, and sets $G_1 = \delta/G_0$. Then S sends (G_0, G_1, δ) to each main server, along with a proof that it knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$. Suppose that the response of the main server T_j to S is $(\hat{s}_{0,j}, \hat{s}_{1,j})$, where $(\hat{s}_{0,j}, \hat{s}_{1,j})$ may not have been computed properly because T_j may be dishonest. S computes $(\hat{s'}_{0,\pi}, \hat{s'}_{1,\pi})$, the query to each verification server T_π, by $\hat{s'}_{b,\pi} = CShare(\{\hat{s}_{b,j} | j \in J\}, \pi)$ and sends the query to the verification server.

Finally, if the majority of the verification servers reply with "yes" for both $b = 0$ and $b = 1$, S accuses the verification servers that reply with "no" for either $b = 0$ or $b = 1$. Otherwise, S invokes the cheater-identification procedure and behaves as if it were the receiver. Note that what S does in the cheater-identification procedure is based only on what the servers send to S, and thus is independent of whether S knows $\log_g G_0$ or $\log_g G_1$. □

CLAIM 13. *(sender's privacy) Under the DDH assumption, one of the two items, s_0 and s_1, is private against a coalition of the receiver and any $\lfloor \frac{\tau-1}{2} \rfloor$ servers.*

PROOF. We construct a simulator, S', for any given colluding group of a dishonest receiver and $\lfloor \frac{\tau-1}{2} \rfloor$ servers. Note that it is sufficient to consider a deterministic adversary that controls this group. (See page 22 of [14] for an argument of considering only deterministic adversaries.) Now we consider the possible strategies of this deterministic adversary.

To each honest main server T_j, the dishonest receiver will have to prove that he knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$. Therefore, there are only two possible cases for a given strategy and a given honest main server T_j: (a) the receiver knows $\log_g G_0$ for the G_0 sent to T_j; (b) the receiver does not know $\log_g G_0$, and thus knows $\log_g G_1$ for the G_1 sent to T_j. For a given strategy, if the number of j's that falls into case (a) is greater than or equal to $\frac{\tau}{2}$, we define $\sigma' = 0$; otherwise, we define $\sigma' = 1$. Consequently, S', which takes $s_{\sigma'}$ as input, can be constructed as follows.

When the simulation begins, S' defines $s'_{\sigma'} = s_{\sigma'}$ and picks $s'_{1-\sigma'} \in Z_Q$ uniformly at random. S' computes the shares of ${s'_0}^2$ and ${s'_1}^2$, respectively, in the variant of the Shamir scheme, and distributes the shares among the servers, just as the sender distributes the shares of s_0^2 and s_1^2 in our protocol. Then the honest servers interact with the colluding group just as required in the protocol.

In order to see the computational indistinguishability of the views, we note the following fact. The shares of $s'_{1-\sigma'}$ can be divided into three classes: (1) at most $\frac{\tau}{2}$ shares of $(s'_{1-\sigma'})^2$ are multiplied by a value $G_{1-\sigma'}^{\kappa_j}$ with $\log_g G_{1-\sigma'}$ known to the adversary; (2) only $\lfloor \frac{\tau-1}{2} \rfloor$ shares of $(s'_{1-\sigma'})^2$ are held by the dishonest servers; (3) all other shares are either held by honest verification servers, or held by honest main servers and multiplied by a value $G_{1-\sigma'}^{\kappa_j} = (\frac{\delta}{G_{\sigma'}})^{\kappa_j}$ with $\log_g G_{\sigma'}$ known to the adversary. Because δ is picked according to the public random source, by the DDH assumption, the ciphertext of each share in the third class is indistinguishable from a uniformly random element. On the other hand, the total number of shares in the first two classes is less than τ, and a set of fewer than τ shares in the variant of Shamir scheme is indistinguishable from a set of uniformly random elements. □

CLAIM 14. *(verifiability of reconstruction)The protocol is reconstruction-verifiable if the number of dishonest servers is less than than* $\frac{t-\tau}{2}$.

PROOF. Please see [27]. □

CLAIM 15. *(verifiability of accusation) The protocol is accusation-verifiable if the number of dishonest servers is less than* $\frac{t-\tau}{2}$.

PROOF. Please see [27]. □

6. SYSTEM ARCHITECTURE AND PROTOCOL SPECIFICATION

In this section, we apply VDOT to protect mobile agents. Our system architecture, which is an extension of that in [1], is shown in Figure 1.

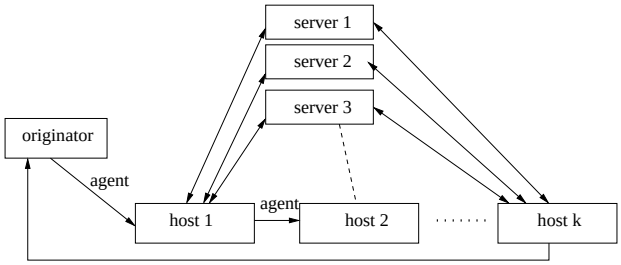

Figure 1: System Architecture

There are three types of entities in our system architecture: the originator, the hosts, and the servers. One motivation for introducing the servers in this mobile computing environment is that the originator may not always be online. Furthermore, since 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. Note that it is straightforward to modify our system if the originator is always online.

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.

6.1 Protocol for Security-Sensitive Computation

The crucial issue of our system is how to evaluate the security-sensitive function of an agent. We present the evaluation protocol in this subsection. For clarity of presentation, we focus on main ideas. Optimizations will be included in a more complete version of this paper.

6.1.1 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. A garbled circuit has 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 clear text so that the host can obtain its local output immediately after the evaluation.

As for table In2, its content is split into secret shares and encrypted using the public keys of the servers. The agent carries the encrypted shares, which will be used to initialize the VDOT protocol.

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 $j+1$. As a result, their garbled versions are also correlated and there is a way to enforce the state transition without revealing these two tables. In particular, a chaining technique 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 [5, 1].

ID	Session identifier
GbCircuit_j	Garbled circuit for host j
GbIn1Host1	Garbled input 1 for host 1
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: Notation

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. 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.

6.1.2 Evaluation of a security-sensitive function

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

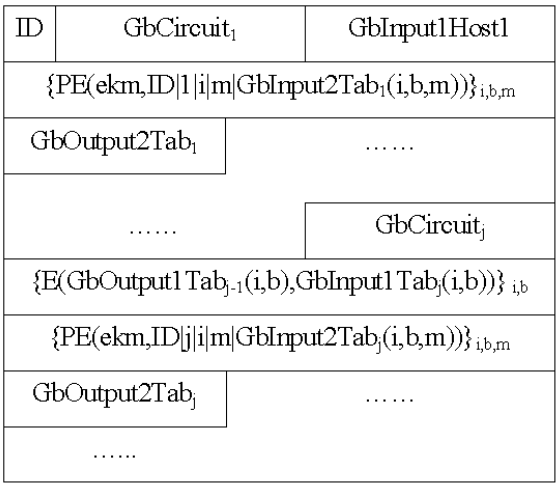

Figure 2: Data Format of a Security-Sensitive Function in an Agent

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.

6.2 Security Analysis

We briefly summarize the security properties of our system.

THEOREM 1. *(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 more than a threshold of servers collude.*

This follows from the security properties of garbled circuits and VDOT. Since 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 local input, no host is able to extract partial private information from the garbled circuit by evaluating it for more than once.

THEOREM 2. *(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 security properties 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.

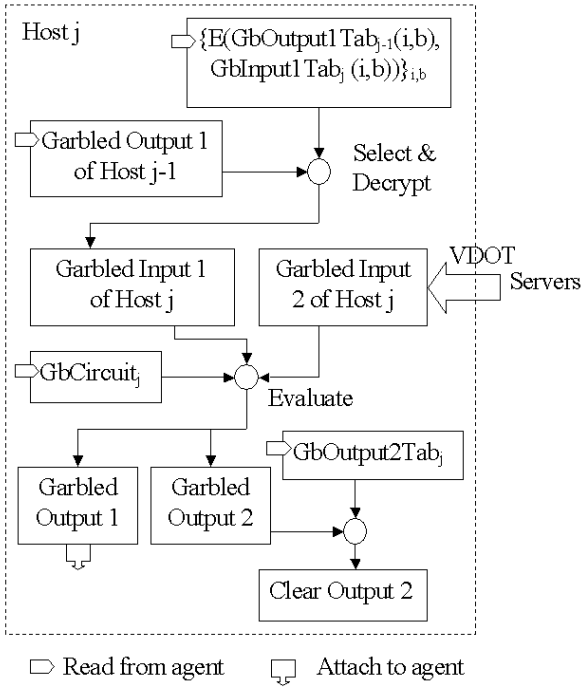

Figure 3: Evaluating a Security-Sensitive Function at Host j

THEOREM 3. *(Cheater Detection) If any server cheats, the host is able to detect the cheater, unless more than a threshold (which is different from the threshold for the originator's privacy) of servers collude.*

This also follows from the security properties of VDOT.

7. IMPLEMENTATION AND PERFORMANCE EVALUATION

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

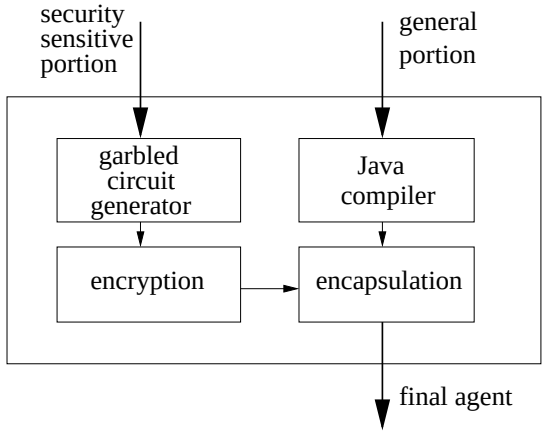

Figure 4: Software Architecture of an Originator

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. In the airline-ticket example, all the user needs to do is to execute the generator and input her desired flight date, source, destination and price threshold. Then the generator immediately outputs a mobile agent on her behalf.

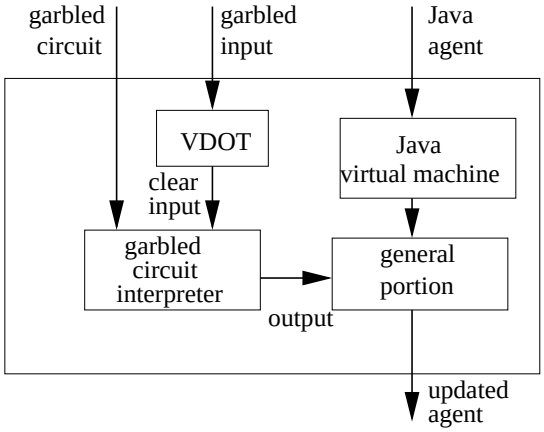

Figure 5: Components of a Host

An agent is sent to hosts for execution. Figure 5 shows the components and the information flow at a host. Since 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.

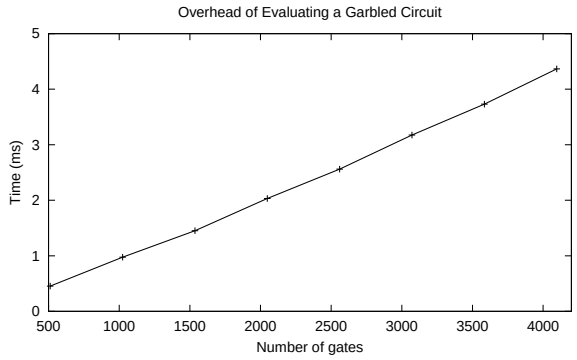

Figure 6: Overhead of Evaluating a Garbled Circuit

In order to interpret garbled circuits, the hosts need to interact with the servers to retrieve translation tables. In particular, if the local inputs of the hosts need to be protected, the hosts and the servers will interact through the VDOT protocol. Since VDOT is a security-intensive process, it is implemented in C++. Since a server may need to serve many hosts, its design should be as simple as possible. In our current design, a server only runs the VDOT protocol.

We next evaluate the overhead of the VDOT protocol, on 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 6 servers and the threshold for the originator's privacy is 3. It is clear that the cost of VDOT is acceptable.

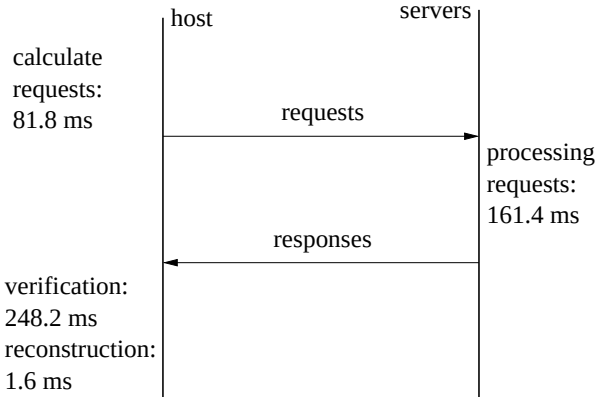

Figure 7: Overhead of VDOT (Number of Servers: 6; Threshold: 3)

8. 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*. It also has *verifiability of reconstruction* and *verifiability of accusation*. 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.

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.

9. REFERENCES

[1] J. Algesheimer, C. Cachin, J. Camenisch, and G. 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] D. Boneh. The decision Diffie-Hellman problem. In *ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63, 1998.

[4] G. Brassard, C. Crepeau, and J.-M. 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] C. Cachin, J. Camenisch, J. Kilian, and J. 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] G. D. Crescenzo, T. Malkin, and R. 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] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology - Proceedings of EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310, 1999.

[12] 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.

[13] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. 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.

[14] O. Goldreich. Secure multi-party computation. Working Draft Version 1.1, 1998.

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

[16] E. Kushilevitz and R. 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.

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

[20] G. C. Necula and P. 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. Pedersen. A threshold cryptosystem without a trusted third party. In *Advances in Cryptology - Proceedings of EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, 1991.

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

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

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

[25] C. P. Schnorr. Efficient indentification and signatures for smart cards. In *Advances in Cryptology - Proceedings of CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, 1990.

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

[27] S. Zhong and Y. R. Yang. Verifiable distributed oblivious transfer. Technical Report Yale/DCS/TR1241, Computer Science Department, Yale University, October 2002.