Universally Composable Secure Mobile Agent Computation[⋆] [⋆⋆]

Ke Xu and Stephen R. Tate

University of North Texas, Denton, TX 76203, USA
{kxu,srt}@cs.unt.edu

Abstract. We study the security challenges faced by the mobile agent paradigm, where code travels and performs computations on remote hosts in an autonomous manner. We define universally composable security for mobile agent computation that is geared toward a complex networking environment where arbitrary protocol instances may be executing concurrently. Our definition provides security for all the participants in the mobile agent system: the originator as well as the hosts. Finally, under the assumption of a universally composable threshold cryptosystem, we present universally composable, multi-agent protocols with provable security against either static, semi-honest or static, malicious adversaries, according to our definition, where in the latter case we need to provide access to a common reference string.

1 Introduction

Mobile agents are goal-directed, autonomous programs capable of migrating from host to host during their execution. From the security point of view, this execution model gives rise to some unique and interesting issues. Mobile agent computation is a distributed computing process, where the participants are the originator who launches the agents and the remote hosts who receive and execute the agents. It is natural to assume no party (i.e., the originator or any of the hosts) trusts anyone else, and mobile agents can be the perfect means for one party to attack another. For example, the agents may try to steal private information from the hosts, or the hosts could manipulate the agents so as to obtain favorable results at the expense of the originator or other hosts' interests. Therefore, the centerpiece of secure mobile agent computation is maintaining the honesty of the agents and the hosts on which they execute. This turns out to be a non-trivial task as the goal may appear to be self-conflicting: The hosts want to monitor the agents' behavior as much as possible, while the originator (and other hosts) wants the agents to have enough privacy and protection to avoid malicious manipulation.

Traditionally, security in mobile agents has been considered along the two separate lines of either protecting the hosts against agent attacks or vice versa, with each having (potentially negative) implications on the other. A recent direction of applying cryptographic protocols from secure multi-party computation to mobile agents has shown the promise of a complete solution that could benefit both sides [16, 5, 2, 17]. Specifically,

⋆ This research is supported in part by NSF award 0208640.

⋆⋆ This work has been published in the *Proceedings of the 7th International Conference on Information Security (ISC '04)*, with slightly different fonts/pagination — when citing page numbers from this paper, please refer to the published version.

the results of [5, 2, 17] are all based on Yao's two-party secure function evaluation protocol [19]. This approach has several advantages over the previous ones. First of all, it looks at the whole picture of mobile agent computation, and aims at providing security for all parties in the system — the originator and the hosts — at the same time. Thus, the assumption is that any party may become malicious and malicious parties can collude with each other, including collusion between the originator and some hosts in the attempt to attack other hosts. Second, it has a solid theoretical basis. In particular, the secure function evaluation problem has been studied extensively in theory and provably sound results exist — from the definition of security [14, 6] to universal protocols for evaluating any polynomial-time function [19, 13, 4, 11]. Harvesting these existing results enables one to study the mobile agent security problem in a well thought-out theoretic context and obtain formally provable solutions, which is the purpose of this paper.

1.1 Universally Composable Security for Mobile Agents

Essentially, mobile agent computation is a distributed computing process, and for many applications it is a process of evaluating a sequence of two-party functions between the originator and each of the hosts. For example, a shopping agent takes the originator's buying criteria and compares it with the offers from different hosts. A search agent matches the information provided by each host against the search phrase from the originator. Often, the result from evaluating one function is used as the input to the next function on the next host. Hence, the functions computed by the agents takes two inputs: one is the current *agent state* which represents the result from the computation on the previous host, and the other is the input provided by the current host. The outcome is a new agent state that will be taken to the next host, and optionally the host may also receive some output from the function, which we refer to as the *local output*. Eventually, the agent goes back to the originator who obtains its desired result from the agent's final state.

Universally composable (UC) security was proposed by Canetti [6] to address the problem of preserving secure protocols in a complex environment, where an unbounded number of protocol instances are executed arbitrarily (i.e., sequentially or concurrently). It is a realistic model for today's networking environment. The contribution of Canetti's work is that it presents a way to study cryptographic protocols which is similar to the previous approach of viewing the protocols as "stand-alone" computation, but the achieved security is geared toward the more complex environment. To this end, in addition to the parties directly involved in the protocol, a new entity called the *environment* is added into the system. Intuitively, the environment represents everything outside of the protocol in question. It provides inputs to the parties and reads their outputs. It also may exchange information freely with the adversary. However, the environment does not see the internals of the protocol execution — messages sent by the parties or the internal states of the parties. Security is defined from the environment's point of view such that no environment can distinguish the execution of the real protocol and a simulation in the ideal model, where an imaginary trusted ideal functionality performs the computation on behalf of all the parties. A *universal composition* theorem asserts such security is preserved when the protocol is executed arbitrarily with other protocols including serving as a subroutine in a larger protocol.

1.2 Our Results

The theme of this work is to treat computing with mobile agents as a secure function evaluation (SFE) problem between the originator and the hosts, with the goal of obtaining universally composable, secure protocols for mobile agent computation. Toward this goal, we first define UC security for mobile agents. Based on the definition, we present two agent protocols. The first one is secure against static, semi-honest adversaries, while the second one adds security against malicious attacks. In both cases we require a universally composable non-interactive threshold cryptosystem, which has not yet been discovered, but several promising advances have been made in this direction [8, 1]. In the case of malicious adversaries, we also augment the model by introducing a common reference string as required by the techniques of [10].

Although there already exist general protocols for two-party and multi-party SFE with UC security [10], one unique restriction in mobile agents prohibits directly applying those results. The autonomous property of mobile agents implies no interaction should happen between the agents and the originator. In other words, the originator should be assumed offline after sending out the agents. On the other hand, all the general results of [10] require continuous communication among the parties[1].

Our technique is based on the two-party SFE protocol due to Yao [19]. In Yao's protocol, one party, Alice, creates an *encrypted circuit* for computing the desired function and sends it to the other party, Bob. Bob evaluates the circuit and obtains the output. However, evaluating the encrypted circuit does not reveal any information about Alice's input other than what the output implies. Cachin et al. [5] first proposed implementing the code of mobile agents as encrypted circuits, a result which was improved by [2, 17]. In particular, Tate and Xu [17] presented a secure multi-agent protocol that does not rely on a trusted third party or trusting any agent or host in the system.

Our results extend this multi-agent protocol with provable UC security, which are lacked by all the prior work. It turned out to be non-trivial to add provable security to the multi-agent protocol. A prerequisite is a formal definition of security in the UC framework, which had not been done before. The cryptographic tools used in the previous protocol have to be reconsidered for UC security. Furthermore, additional techniques are employed in our results so that the resulting security can be proved according to the definition.

In Section 2, we formally define the security for mobile agent computation under the UC framework. Section 3 describes the cryptographic tools used in our protocols. UC-secure mobile agent protocols against static, semi-honest and malicious adversaries are presented in Sections 4 and 5, respectively.

2 Secure Mobile Agent Computation — Definitions

We impose the following requirements on the mobile agent computation.

1. The agent computation can be modeled as the evaluation of a sequence of two-party functions between the originator and each of the hosts, as described in Section 1.1.

[1] In fact, if using the general results, we would not need mobile agents at all, whose sole purpose is to represent the originator and do computation with the hosts.

2. The hosts can be partitioned into *disjoint* subsets, and accordingly the evaluation can be broken down into subproblems, where each problem involves one subset of hosts and can be computed concurrently with the other subproblems. As a result, the originator dispatches one agent to each host subset, and later combines the final states of all the agents into the final result.
3. The participating hosts are predetermined before the protocol starts, as is the partition of the hosts, which determines the total number of agents needed.

Although not every real-world mobile agent application meets these requirements, this is still a fairly general model. (One open problem is to remove the restriction that hosts have to be predetermined, to accommondate free-roaming agents.) We define the notation for host identity and agent functions as follows.

Definition 1. *Assume there are a total of ℓ hosts and n agents in the system, where $n \leq \ell$. The hosts are partitioned into n disjoint subgroups, each group to be visited by exactly one agent. We use the combination of the group id, i, for $i = 1, \ldots, n$, and the host id within that group, j, for $j = 1, \ldots, m_i$ and $\sum_{i=1}^{n} m_i = \ell$, to identify a particular host, noted as $H_{(i,j)}$.*

Definition 2. *Let $f^{(i,j)} : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, for $i = 1, \ldots, n$, $j = 1, \ldots, m_i$, and $\sum_{i=1}^{n} m_i = \ell$, denote the (probabilistic) polynomial-time agent function computed by a mobile agent on host $H_{(i,j)}$. The first parameter is the current agent state, and the second parameter is the input from the host. The function outputs a new agent state as well as a local output to the host. Furthermore, for $i = 1, \ldots, n$, let $x_{(i,0)}$ denote the initial state of mobile agent MA_i and $y_{(i,j)}$ be the input from host $H_{(i,j)}$. Visiting hosts $H_{(i,1)}, \ldots, H_{(i,m_i)}$, agent MA_i computes function $f^{(i,j)}$ at host $H_{(i,j)}$ with the current agent state and the host's input, and obtains the following outputs:*

$$(x_{(i,j)}, z_{(i,j)}) = f^{(i,j)}(x_{(i,j-1)}, y_{(i,j)}) \text{ for } 1 \leq j \leq m_i.$$

$x_{(i,j)}$ is the updated agent state, and $z_{(i,j)}$ is the local output to host $H_{(i,j)}$.

Finally, all n agents return to the originator O with final states $x_{(i,m_i)}$, which are combined to give O's final output, denoted by $\xi = g(x_{(1,m_1)}, \ldots, x_{(n,m_n)})$, where $g(\cdots)$ is the combination function.

2.1 Definition of Security

We define an ideal model for mobile agent computation which captures the desired security properties, as well as a real model in which the secure agent protocol executes. In both models, the parties include the originator, the hosts, an adversary, and the environment. Interactions between the parties in both models follow the general description in [6]. The ideal model uses a trusted ideal functionality defined as follows, to compute the agent functions.

Definition 3. *$\mathcal{F}_{\text{MA}}$ proceeds as follows, interacting with originator O, ℓ hosts whose ids are defined as in Definition 1, and adversary $\mathcal{S}$.*

1. *Upon receiving the initial agent states $x_{(1,0)}, \ldots, x_{(n,0)}$ with session id "sid" from O, store them in a buffer. Send a notification (O_input, sid) to $\mathcal{S}$, where "O_input" is just a text message containing no information about the input value.*

2. *Upon receiving input $y_{(i,j)}$ from host $H_{(i,j)}$, send notification $(H_{(i,j)}\_\mathrm{input}, sid)$ to $\mathcal{S}$. If $x_{(i,j-1)}$ has not been created, save $y_{(i,j)}$. Otherwise, compute $x_{(i,j)}$ and $z_{(i,j)}$, store $x_{(i,j)}$ and return $z_{(i,j)}$ to $H_{(i,j)}$. In addition, if there are saved $y_{(i,k)}$ for $k = j+1, j+2, \ldots$, continue computing the corresponding $x_{(i,k)}$ and $z_{(i,k)}$, returning $z_{(i,k)}$ to host $H_{(i,k)}$, until there are no more buffered inputs.*
3. *Upon receiving a request to reveal intermediate result $x_{(i,j-1)}$ from both O and host $H_{(i,j)}$, send $x_{(i,j-1)}$ to $\mathcal{S}$.*
4. *Once all final agent states $x_{(1,m_1)}, x_{(2,m_2)}, \ldots, x_{(n,m_n)}$ have been computed, combine them using function $g(\cdots)$ as defined in Definition 2, and return the result ξ to O. If a request to reveal the final agent states is received from O, send all the final agent states to $\mathcal{S}$. Halt after this step is done.*

We consider static adversaries who can corrupt the originator and/or the hosts, but only before the protocol starts. In other words, the adversary's choice of which party to corrupt does not depend on the execution of the protocol. Furthermore, we distinguish two types of adversaries: A semi-honest adversary strictly follows the protocol but tries to infer more knowledge from the information it obtained legally, while a malicious adversary may exhibit arbitrary (Byzantine) behavior. As introduced before, the environment reads the outputs of the originator and the hosts. It can also communicate freely with the adversary, and in the end, the environment outputs a binary value.

Definition 4. *Let $\overset{c}{\approx}$ denote computational indistinguishability [12]. Mobile agent protocol π **securely realizes $\mathcal{F}_{\mathrm{MA}}$ against static, semi-honest (or malicious) adversaries under condition** t, if for any such adversary $\mathcal{A}$ interacting with the real-model execution of π and corrupting parties subject to condition t, there exists a corresponding adversary $\mathcal{S}$ corrupting the same set of parties in the ideal model, such that the running time of $\mathcal{S}$ is polynomial in the running time of $\mathcal{A}$ and for all environments $\mathcal{Z}$ we have*

$$\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathrm{IDEAL}_{\mathcal{F}_{\mathrm{MA}},\mathcal{S},\mathcal{Z}}. \tag{1}$$

where $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{MA}},\mathcal{S},\mathcal{Z}}$ denote the ensembles of $\mathcal{Z}$'s binary outputs when it is interacting with the real model execution of π and the ideal model agent computation, respectively.

Due to the Universal Composition Theorem of [6], a complex protocol can be broken into subproblems and each subproblem can be studied separately. Once a secure protocol for a subproblem is obtained, executing the subprotocol in the larger protocol can be treated as running an ideal process with the ideal functionality for the subproblem. This so called *hybrid model* largely reduces the complexity of analyzing the overall protocol, and is used extensively in our work. We begin by introducing the building blocks of our mobile agent protocol and their secure implementations. The first agent protocol is then presented in a hybrid model where the parties have access to the ideal functionalities for the primitives.

3 Cryptographic Tools

The basic idea of the multi-agent protocol of [17] can be summarized as follows. The originator creates n agents each visiting a disjoint subset of hosts. For each host, the originator creates an *encrypted circuit* to compute the agent function for that host. The originator also sets the initial agent states, in the form of *signals* recognizable to the encrypted circuits. Once a host receives an agent, it conducts *oblivious threshold decryption* with enough remote agents to obtain the signals corresponding to its input. Then the host evaluates the encrypted circuit with the current agent state and its own input. The host receives the local output from the circuit, and sends the agent with a new state, which is in the form of signals recognizable by the next encrypted circuit, to the next host in the same subset. In the end, the originator receives back all the agents and combines their final states into the final result. In this section, we describe under the UC framework the building blocks of the multi-agent protocol.

3.1 Encrypted Circuits

Encrypted circuits were introduced by Yao as the centerpiece of his two-party SFE protocol, and are a scrambled form of boolean circuits in which the signals on the wires are random binary strings that represent boolean values. The mappings from the random strings, referred to as the "signals," to their corresponding boolean values, known as the "semantics," are hidden from the evaluator of the circuit. With correctly constructed truth tables for the gates, the encrypted circuit can be evaluated by one who has no knowledge of the semantics of the input signals. Moreover, the output is also in the form of signals, and therefore evaluation does not reveal any information about the input, the output, or any intermediate result within the circuit.

Currently there are several implementations of the encrypted circuit [18, 15, 3]. The result of [18] is particularly appealing due to its full proof of security, which establishes the following.

Theorem 1. *To an evaluator who knows the signals for at most one instance of input and the semantics of the signals for the output it is entitled to learn (if any), real encrypted circuits are computationally indistinguishable from fake encrypted circuits which can be generated in polynomial-time with only the knowledge mentioned above.*

Theorem 2. *Assume an encrypted circuit in which the length of every signal is k. Define the signals obtained on each wire in the evaluation with a particular instance of input as the "on-path" signals for that input, and the rest of the wire signals as the "off-path" signals. For every probabilistic polynomial-time evaluator A with the specific input, every positive polynomial $p(\cdot)$, and sufficiently large k, the probability that A correctly guesses the off-path signal for any given wire in the circuit is strictly less than $\frac{1}{2^k} + \frac{1}{p(k)}$.*

3.2 Oblivious Threshold Decryption

Oblivious threshold decryption (OTD) is a process where a user sends a pair of ciphertexts to enough decryption servers, but receives only one plaintext, while no server learns which plaintext the user receives. It is a combination of oblivious transfer and threshold decryption. In the multi-agent protocol, the agent functions are implemented as encrypted circuits, and each host is an evaluator. The agent functions take inputs from the hosts, and the way for a host to obtain the signals corresponding to its input is essential to the security of the protocol, because both Theorems 1 and 2 rely on the vital assumption that the evaluator holds the signals for only one instance of input, i.e., one signal per input wire. Through OTD, a host receives only the signals for its input and nothing else, and its private input is not revealed to any other party in the system. We define the ideal functionality of OTD to capture both security requirements. Our definition is based on a similar definition for traditional (i.e., non-threshold) public-key encryption from [6].

Definition 5. *Ideal Functionality $\mathcal{F}_{\mathrm{OTD}}$ is defined with a threshold parameter $m \leq n$ and security parameter k for participants $P_1, \ldots, P_n$ and adversary $\mathcal{S}$:*

Key Generation: *On the first activation, expect to receive a message* $(\mathrm{KeyGen}, sid, \mathcal{DS})$ *from some party P_i, where $\mathcal{DS}$ is a set of at least m parties (the "decryption servers"). Then do:*

1. *Hand* (KeyGen, sid) *to $\mathcal{S}$.*
2. *Receive a value e from $\mathcal{S}$, record e, and hand e to P_i.*

Encryption: *Upon receiving a message* $(\mathrm{Encrypt}, sid, e', w)$ *from a party $P_{i'}$:*

1. *Hand* $(\mathrm{Encrypt}, sid, e', |w|)$ *to $\mathcal{S}$.* (*If $e' \neq e$ or e is not yet defined then hand also the entire value w to $\mathcal{S}$.*)
2. *Receive a tag c from $\mathcal{S}$ and hand c to $P_{i'}$. If $e' = e$ then record the pair (c, w).* (*If c appears in a previously recorded pair then halt.*)

Oblivious Decryption Request: *Upon receiving* $(\mathrm{DecryptRequest}, sid, c_0, c_1, b)$ *from a party P_j, where $b \in \{0, 1\}$, record the request as* $(\mathrm{DecryptRequest}, sid, c_0, c_1, b, P_j)$ *and return control to $\mathcal{S}$. We refer to P_j as the "decryption user" for this ciphertext pair.*

Oblivious Decryption Assist: *Upon receiving* $(\mathrm{DecryptAssist}, sid, c_0, c_1)$ *from a party $P_k \in \mathcal{DS}$, where (c_0, c_1) is a ciphertext pair from a previously recorded DecryptRequest and P_k has not yet given a DecryptAssist message for this ciphertext pair:*

1. *Record* $(\mathrm{DecryptAssist}, sid, c_0, c_1, P_k)$ *to track requests for (c_0, c_1).*
2. *If this is the mth recorded DecryptAssist for ciphertext pair (c_0, c_1), then:*
   If an encryption request has recorded a pair (c_b, w_b), then return w_b to P_j (the party that made the corresponding DecryptRequest); otherwise, hand $(\mathrm{DecryptRequest}, sid, c_b)$ *to the adversary, receive a value w from the adversary, and hand w to P_j.*

$\mathcal{F}_{\mathrm{OTD}}$ can be securely realized with non-interactive threshold decryption and oblivious transfer. The ideal functionality $\mathcal{F}_{\mathrm{TD}}$ (for threshold decryption) is very similar to $\mathcal{F}_{\mathrm{OTD}}$ except with only a single ciphertext being provided in the decryption phase, and the ideal functionality $\mathcal{F}_{\mathrm{OT}}$ for oblivious transfer is defined by Canetti et al. in [10] (we are only interested in 1-out-of-2 OT instead of the more general 1-out-of-ℓ OT).

Canetti et al. [10] presented secure realizations of $\mathcal{F}_{\mathrm{OT}}$ in the presence of static and adaptive adversaries. However, to our knowledge, currently there is no published, provably secure implementation of $\mathcal{F}_{\mathrm{TD}}$. The non-interactive threshold cryptosystem of Canetti and Goldwasser [8] is a likely candidate, pending formal proof under the UC model [7]. In light of this, we describe the implementation of $\mathcal{F}_{\mathrm{OTD}}$ in a general sense.

Suppose there exists a non-interactive threshold cryptosystem, NITD, that securely realizes $\mathcal{F}_{\mathrm{TD}}$ (when converted in the obvious way) in the real model through four basic operations.

– Key Generation: This is done by a trusted dealer or through a multi-party protocol by the decryption servers. As a result, the public key is made known to the encryption user, and each server gets its share of the decryption key secretly.
– Encryption: The encryption user runs the encryption algorithm on a plaintext with the encryption key. This is a process involving only the encryption user.
– Decryption: A decryption server applies the decryption algorithm with its share of the decryption key on a ciphertext. The result is a decryption share. Obtaining a decryption share involves only the current server and requires no interaction with the decryption user or any other server.
– Share Combination: After collecting m decryption shares, the decryption user combines them using the combination algorithm and obtains the plaintext. We require no interaction with the servers in this phase.

Then, $\mathcal{F}_{\mathrm{OTD}}$ can be implemented in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model (i.e., assuming access to $\mathcal{F}_{\mathrm{OT}}$ is available) as follows.

Protocol 1. *Oblivious Threshold Decryption in The $\mathcal{F}_{\mathrm{OT}}$-Hybrid Model*

1. *Key Generation: Run the* Key Generation *phase of NITD, giving decryption key shares to all parties in $\mathcal{DS}$.*
2. *Encryption: Run the* Encryption *operation of NITD.*
3. *Oblivious Decryption Request: The decryption user P_j has a bit b and a pair of ciphertexts (c_0, c_1), sends the pair of ciphertexts to m other parties from $\mathcal{DS}$, and then invokes a copy of $\mathcal{F}_{\mathrm{OT}}$ with each such party.*
4. *Oblivious Decryption Assist: Each party invoked in the previous step obtains decryption shares (c_0^i, c_1^i) through the* Decryption *operation of NITD, and completes the $\mathcal{F}_{\mathrm{OT}}$ with P_j. Thus P_j receives c_b^i. Applying* Share Combination *of NITD, P_j combines the m shares it has received to obtain the plaintext corresponding to c_b.*

Lemma 1. *If non-interactive threshold decryption scheme NITD securely realizes ideal functionality $\mathcal{F}_{\mathrm{TD}}$ in the presence of a static, semi-honest adversary that corrupts up to t decryption servers, where $t < m$, then Protocol 1 securely realizes $\mathcal{F}_{\mathrm{OTD}}$ against a static, semi-honest adversary that corrupts up to t servers in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model.*

Proof (Sketch): The goal is to demonstrate for a given real-model adversary $\mathcal{A}$ an adversary $\mathcal{S}_{\mathrm{OTD}}$ in the ideal model, such that no environment can computationally distinguish the ideal process from the execution of Protocol 1. Since NITD securely realizes $\mathcal{F}_{\mathrm{TD}}$, there exists an ideal adversary $\mathcal{S}_{\mathrm{TD}}$ running with $\mathcal{F}_{\mathrm{TD}}$ that simulates protocol NITD. Since the Key Generation and Encryption of Protocol 1 are the same as in NITD, and their counterparts in the corresponding ideal functionalities are also the same, $\mathcal{S}_{\mathrm{TD}}$ can be directly used to simulate these two operations.

For Oblivious Decryption, first notice share collecting is done through ideal oblivious transfer. Therefore, party P_j receives shares for only one ciphertext, and the decryption servers receive only notifications. As a result, if P_j is not corrupted, the view of adversary $\mathcal{A}$ is just the same as what the adversary in NITD sees except there are two ciphertexts. Because of the semantic security of the encryption, there is no distinguishable relation between the two ciphertexts even if their corresponding plaintexts are related. Thus, the simulation is basically running the corresponding part of $\mathcal{S}_{\mathrm{TD}}$ twice to simulate two separate ciphertexts. By corrupting P_j, the adversary would also obtain the decryption shares including those from the honest servers for one ciphertext. This is still simulatable because the ideal adversary $\mathcal{S}_{\mathrm{OTD}}$ gets to generate the public key as well as shares of the decryption key, playing either as the trusted dealer or on behalf of the honest servers, when it simulates the Key Generation phase. Therefore, $\mathcal{S}_{\mathrm{OTD}}$ has the key shares of the honest servers and can obtain their decryption shares for any ciphertext. ■

3.3 Public-Key Encryption

We also employ standard, non-threshold public-key encryption in our multi-agent protocol. A definition of ideal public-key encryption, $\mathcal{F}_{\mathrm{PKE}}$, can be found in [6]. A following paper [9] has proved that for non-adaptive adversaries, this definition is equivalent to CCA security. Hence, a CCA-secure PKE scheme can be trivially turned into a secure realization of the ideal PKE functionality.

4 Mobile Agents with Static, Semi-Honest Adversaries

In this section we present a multi-agent protocol that securely realizes $\mathcal{F}_{\mathrm{MA}}$ against static, semi-honest adversaries, in the hybrid model with access to $\mathcal{F}_{\mathrm{OTD}}$ and $\mathcal{F}_{\mathrm{PKE}}$.

Protocol 2. *Mobile Agent Computation in The $\mathcal{F}_{\mathrm{OTD}}$, $\mathcal{F}_{\mathrm{PKE}}$-Hybrid Model*
The parties are an originator O and ℓ hosts, partitioned into n subsets as described above. The size of each agent's state is n_x bits, and the size of every host input is n_y bits.

1. *Each host $H_{(i,j)}$, upon receiving its input $y_{(i,j)}$ from the environment, sends a notification of this event to the originator O.*
2. *Every host invokes a copy of ideal functionality $\mathcal{F}_{\mathrm{PKE}}$ to generate a public key e, and broadcasts this key to all other hosts and O. In the following, we use the host id (i, j) to identify its corresponding copy of $\mathcal{F}_{\mathrm{PKE}}$.*
3. *Having received its input $(x_{(1,0)}, \ldots, x_{(n,0)})$ from the environment, as well as the input notifications and the public keys from all hosts, O invokes the* Key Generation *operation of ideal functionality $\mathcal{F}_{\mathrm{OTD}}$.*
4. *For each host $H_{(i,j)}$, O creates an encrypted circuit $(\mathcal{C}^{(i,j)}, \mathcal{L}^{(i,j)}, \mathcal{K}^{(i,j)}, \mathcal{U}_x^{(i,j)}, \mathcal{U}_z^{(i,j)})$ to compute function $f^{(i,j)}$, where $\mathcal{C}^{(i,j)}$ is the description of the circuit, $\mathcal{L}^{(i,j)}$ is the list of signals for the input wires corresponding to the agent state, $\mathcal{K}^{(i,j)}$ is the list of signals for the host input, and $\mathcal{U}_x^{(i,j)}$ and $\mathcal{U}_z^{(i,j)}$ are the lists of signals for the two outputs — the new agent state and the local output to the host, respectively. In all these lists, the two signals for each wire are grouped together as a pair, e.g., $(L_{k,0}^{(i,j)}, L_{k,1}^{(i,j)})$, where the signal with semantics 0 is the first element and the signal with semantics 1 is the second element. In addition, O invokes ideal functionality $\mathcal{F}_{\mathrm{OTD}}$ to encrypt each signal in $\mathcal{K}^{(i,j)}$. Denote the encrypted signals as $\overline{\mathcal{K}}^{(i,j)} = ((\overline{K}_{1,0}^{(i,j)}, \overline{K}_{1,1}^{(i,j)}), \ldots, (\overline{K}_{n_y,0}^{(i,j)}, \overline{K}_{n_y,1}^{(i,j)}))$.*

5. *O creates n mobile agents: For agent $\mathcal{MA}_i$, $i = 1, \ldots, n$, its code includes the encrypted circuits $\mathcal{C}^{(i,j)}$, the encrypted signals $\overline{\mathcal{K}}^{(i,j)}$, and $\mathcal{U}_z^{(i,j)}$, for all the hosts in subset i. O also sets, for $k = 1, \ldots, n_x$, $L_k'^{(i,1)} = L^{(i,1)}_{k,x_{(i,0),k}}$, where $x_{(i,0),k}$ is the k-th bit of $x_{(i,0)}$, as the initial state of $\mathcal{MA}_i$. Following our convention, denote this list of signals by $\mathcal{L}'^{(i,1)}$. Then O invokes ideal functionality $\mathcal{F}_{\mathrm{PKE}}^{(i,1)}$ to encrypt the initial agent state. Denote the encrypted agent state by $\overline{\mathcal{L}}'^{(i,1)}$.*
6. *O sends out $\mathcal{MA}_i$ with its encrypted state to $H_{(i,1)}$, for $i = 1, \ldots, n$.*
7. *Every host $H_{(i,j)}$, upon receiving agent $\mathcal{MA}_i$, sends the encrypted agent state $\overline{\mathcal{L}}'^{(i,j)}$ to $\mathcal{F}_{\mathrm{PKE}}^{(i,j)}$ and receives the decryption $\mathcal{L}'^{(i,j)}$.*
8. *Next, $H_{(i,j)}$ sends the encrypted input signal list $\overline{\mathcal{K}}^{(i,j)}$ to m agents including $\mathcal{MA}_i$, the agent on this host. Then $H_{(i,j)}$ and the m agents engage with $\mathcal{F}_{\mathrm{OTD}}$ to obliviously decrypt one signal from each pair in $\overline{\mathcal{K}}^{(i,j)}$. Specifically, $H_{(i,j)}$ makes the DecryptRequest, and the m agents are the decryption servers. For each OTD, the pair of ciphertexts is $(\overline{K}_{k,0}^{(i,j)}, \overline{K}_{k,1}^{(i,j)})$, for $k = 1, \ldots, n_y$, and the selection bit that determines which ciphertext $H_{(i,j)}$ will actually decrypt, is $y_{(i,j),k}$.*
9. *From $\mathcal{F}_{\mathrm{OTD}}$, $H_{(i,j)}$ receives the signals corresponding to its input $y_{(i,j)}$. Denote this list of host input signals as $\mathcal{K}'^{(i,j)}$.*
10. *$H_{(i,j)}$ evaluates the encrypted circuit $\mathcal{C}^{(i,j)}$ using the input signals $\mathcal{L}'^{(i,j)}$ and $\mathcal{K}'^{(i,j)}$, recovers the value of its own local output using the corresponding $\mathcal{U}_z^{(i,j)}$ vector, and sends the agent (consisting of the encrypted circuits and the various signal lists for the following hosts, as well as the updated agent state, which is in the form of signals and encrypted by $\mathcal{F}_{\mathrm{PKE}}^{(i,j+1)}$ using host $H_{(i,j+1)}$'s public key) to the next host $H_{(i,j+1)}$. If this is the last host in the subset, then the agent is returned back to the originator with its state not encrypted. If the transfer was to a different host, that host repeats steps 7 – 10.*
11. *After all n agents return back to O with their final states. O recovers the values they represent and uses function g to combine the results into the final result.*
12. *Every host outputs its local output from the circuit evaluation, and O outputs the result from function g.*

Theorem 3. *Protocol 2 securely realizes ideal functionality $\mathcal{F}_{\mathrm{MA}}$ in the $\mathcal{F}_{\mathrm{OTD}}$, $\mathcal{F}_{\mathrm{PKE}}$-hybrid model against a static, semi-honest adversary who corrupts parties with the following restrictions: if the adversary corrupts hosts only, then it is limited to corrupting hosts from up to $m-1$ out of the n subsets of hosts, where m is the threshold parameter of $\mathcal{F}_{\mathrm{OTD}}$; or the adversary can corrupt the originator and any number of hosts.*

Proof (Sketch): It is easy to verify that Protocol 2 correctly implements the functionality of $\mathcal{F}_{\mathrm{MA}}$ when every party follows the protocol, which is the case with a semi-honest adversary. Therefore, after the protocol execution is over, the originator and the hosts will output the correct results, and so the main task now is constructing an ideal-model adversary $\mathcal{S}$ to simulate the interaction between the environment and the adversary $\mathcal{H}$ in the hybrid model. $\mathcal{S}$ runs a copy of $\mathcal{H}$ internally and forwards the messages from the environment to the internal $\mathcal{H}$. In addition, $\mathcal{S}$ simulates the view $\mathcal{H}$ expects, as explained below. As a result, the internal $\mathcal{H}$ produces the same messages back to the environment as a real $\mathcal{H}$ does, which are forwarded by $\mathcal{S}$ to the environment.

The UC framework assumes open communication channels, and so the view of $\mathcal{H}$ includes the inputs and outputs of the corrupted parties as well as all the messages sent in the system except those between the ideal functionalities and the honest parties. By corrupting the same set of parties in the ideal model, $\mathcal{S}$ can provide the internal $\mathcal{H}$ with the corrupted parties' inputs and outputs. The messages seen by $\mathcal{H}$ include the input notifi cations from the hosts to the originator, the hosts' public keys, all the agents travelling from one host to another, the encrypted input signals $\overline{\mathcal{K}}^{(i,j)}$ sent by a host to remote agents for decryption, and the messages from the ideal PKE and OTD to the corrupted parties. Simulating these messages depends on what parties $\mathcal{H}$ corrupts.

If only the hosts are corrupted and the corrupted hosts are from at most $m-1$ different subsets, adversary $\mathcal{H}$ is not able to control enough agents to invoke $\mathcal{F}_{\mathrm{OTD}}$ to decrypt the encrypted host input signals of its choice. Therefore, for any encrypted circuit contained in an agent, $\mathcal{H}$ knows signals for only one instance of host input to that circuit if the circuit is for a corrupted host, and no input signal at all if the circuit is for an honest host. Furthermore, only the semantics of the signals for local host outputs are known to $\mathcal{H}$, and the agent states, in the form of signals, make no sense to $\mathcal{H}$. Therefore, applying Theorem 1, $\mathcal{S}$ can construct agents containing fake encrypted circuits that are indistinguishable to $\mathcal{H}$, based on the local outputs to the corrupted hosts. To simulate encrypting the lists of host input signals and the agent states, as well as OTD for the corrupted parties at various stages of Protocol 2, $\mathcal{S}$ plays the role of ideal functionality $\mathcal{F}_{\mathrm{OTD}}$ and $\mathcal{F}_{\mathrm{PKE}}$, and interacts with $\mathcal{H}$ accordingly.

Once the originator is corrupted, there is no need to maintain the limit on corrupted hosts, since it is now the adversary $\mathcal{H}$ who creates the agents. Thus, controlling enough agents to decrypt any chosen signals for host input wires becomes irrelevant. In fact, there is no need to simulate the encrypted circuits at all. The new issue, however, is that the agent states, which are represented by signals and previously make no sense to $\mathcal{H}$, now can be easily interpreted, simply because it is $\mathcal{H}$ that creates the circuits and selects the signals and their semantics. Therefore, $\mathcal{S}$ has to obtain from $\mathcal{F}_{\mathrm{MA}}$ the intermediate state of an agent when it comes to a corrupted host. (If the host is not corrupted, the agent state is still unreadable to $\mathcal{H}$ as it is encrypted by the uncorrupted host's public key.) This can be done by sending a request on behalf of O and the corrupted host to $\mathcal{F}_{\mathrm{MA}}$ for revealing the particular intermediate result. The same can be done for revealing the fi nal agent states, which $\mathcal{H}$ sees by controlling O. ■

Substituting ideal functionalities $\mathcal{F}_{\mathrm{OTD}}$ and $\mathcal{F}_{\mathrm{PKE}}$ with their secure realizations in the real model, we can transfer Protocol 2 into a real-model protocol. Security is preserved by the Universal Composition Theorem. However, if the implementation of $\mathcal{F}_{\mathrm{OTD}}$ is secure against t corrupted servers, for $t \leq m-1$, the real-model adversary has to be restricted to corrupting hosts in at most t subsets if the originator is not corrupted.

5 Mobile Agents with Static, Malicious Adversaries

For security against arbitrary malicious attacks, we apply the *universally composable compiler* presented in [10] to Protocol 2. The UC compiler is a general result that transforms a secure protocol with regard to semi-honest adversaries to one secure against malicious adversaries. The basic idea is to force a malicious adversary to behave semi-honestly. For this purpose, the UC compiler begins with having all parties in the protocol jointly generate a uniformly random tape for each party. After this phase, each party obtains a private random tape for use in the protocol, while every other party holds a commitment for the random tape. Next, all parties start execution of the original protocol which is secure against semi-honest adversaries, with the following additional steps. First, when a party receives an input from the environment, it commits the input to everyone else using a functionality known as "commit-and-prove", denoted by $\mathcal{F}_{\mathrm{CP}}$. Second, whenever a party wants to send a message as required by the protocol, it uses the same $\mathcal{F}_{\mathrm{CP}}$ to prove that the message is correctly generated based on the party's committed input, its random tape obtained in the setup phase, and the genuine messages it has received so far. Notice that these three components together uniquely determine the message that should be sent by the party if it is honest. Third, all messages in the system are broadcast through an *authenticated broadcasting channel* to every party, even though it is not intended for everyone. Upon receiving a message, one checks the validity of the message to ensure it was generated based on the three components mentioned before. This is possible because every party holds the commitments of other parties' inputs and random tapes, and all the messages are sent to everyone. As a result, any malicious attacks will be detected by an honest party when a cheating message for the attack is received.

Implied in the description of the compiler is the requirement that every party receives every message sent in the system. This, however, is not possible for the originator, who does not receive any messages after sending out the agents except for receiving the agents back. (The hosts can all stay online and follow the steps required by the compiler.) This restriction can be worked around due to the security of the encrypted circuits (Theorem 2). In brief, the only messages intended for the originator are the agents coming back. As long as the OTD is secure during the agent computation, which is guaranteed by the hosts following the compiler, it is infeasible for a malicious adversary to modify the fi nal agent states, which have to be represented by valid signals.

Protocol 3. *Assuming that the originator and all the hosts share a common reference string, apply the universally composable compiler of [10] to Protocol 2 as follows.*

1. *All participants including the originator O carry out the* random tape generation *phase. Then everyone starts execution of Protocol 2 with additional steps required by the compiler as described above, with every message broadcast to all parties.*
2. *After sending out the agents (i.e., step 6 of Protocol 2), O is excluded from all broadcast messages other than an agent's coming home. All the hosts continue to behave according to Protocol 2 and the compiler.*
3. *When O receives an agent back from the hosts, it checks the validity of the agent's final state. That is, it checks if the strings that represent the agent state are valid signals for the corresponding wires. If not, O ignores this message. Otherwise, it treats the agent according to Protocol 2.*

Theorem 4. *Protocol 3 securely realizes ideal functionality $\mathcal{F}_{\mathrm{MA}}$ against a static, malicious adversary with the same assumption about the adversary's selection of corrupted parties as in Theorem 3.*

References

[1] Abe, M., Fehr, S.: Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. To appear in: CRYPTO 2004.

[2] Algesheimer, J., Cachin, C., Camenisch, J., Karjoth, G.: Cryptographic security for mobile code. In: Proc. of the IEEE Symposium on Security and Privacy. (2001) 2–11

[3] Beaver, D.: Correlated pseudorandomness and the complexity of private computation. In: Proc. of the 28th Annual ACM Symposium on Theory of Computing. (1996) 479–488

[4] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: 20th STOC. (1988) 1–10

[5] Cachin, C., Camenisch, J., Kilian, J., Müller, J.: One-round secure computation and secure autonomous mobile agents. In: Proc. of the 27th ICALP. (2000) 512–523

[6] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proc. of the 42nd FOCS. (2001) 136–145

[7] Canetti, R.: Personal communication (2003)

[8] Canetti, R., Goldwasser, S.: An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In: EuroCrypt'99. (1999) 90–106

[9] Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Advances in Cryptology — Crypto 2003. (2003) 565–582

[10] Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th STOC. (2002) 494–503

[11] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: Proc. of the 20th ACM Symposium on Theory of Computing (STOC). (1988) 11–19

[12] Goldreich, O.: Foundations of Cryptography, Volume 1 Basic Tools. Cambridge University Press (2001)

[13] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: 19th STOC. (1987) 218–229

[14] Micali, S., Rogaway, P.: Secure computation. In: Crypto'91. (1991) 392–404

[15] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: 1st ACM Conference on Electronic Commerce. (1999) 129–139

[16] Sander, T., Tschudin, C.F.: Protecting mobile agents against malicious hosts. In: Mobile Agents and Security. LNCS Vol. 1419. Springer (1998) 379–386

[17] Tate, S.R., Xu, K.: Mobile agent security through multi-agent cryptographic protocols. In: 4th International Conference on Internet Computing. (2003) 462–468

[18] Tate, S.R., Xu, K.: On garbled circuits and constant round secure function evaluation. Technical report, University of North Texas (2003) Available from http://cops.csci.unt.edu/publications/ (Journal version under preparation).

[19] Yao, A.C.: How to generate and exchange secrets. In: Proc. of the 27th IEEE Symposium on Foundations of Computer Science (FOCS). (1986) 162–167