One-Round Secure Computation and Secure Autonomous Mobile Agents

(Extended Abstract)

Christian Cachin[1], Jan Camenisch[1], Joe Kilian[2], and Joy Müller[1]

Abstract. This paper investigates one-round secure computation between two distrusting parties: Alice and Bob each have private inputs to a common function, but only Alice, acting as the receiver, is to learn the output; the protocol is limited to one message from Alice to Bob followed by one message from Bob to Alice. A model in which Bob may be computationally unbounded is investigated, which corresponds to information-theoretic security for Alice. It is shown that

1. for honest-but-curious behavior and unbounded Bob, any function computable by a polynomial-size circuit can be computed securely assuming the hardness of the decisional Diffie-Hellman problem;
2. for malicious behavior by both (bounded) parties, any function computable by a polynomial-size circuit can be computed securely, in a public-key framework, assuming the hardness of the decisional Diffie-Hellman problem.

The results are applied to secure autonomous mobile agents, which migrate between several distrusting hosts before returning to their originator. A scheme is presented for protecting the agent's secrets such that only the originator learns the output of the computation.

1 Introduction

Suppose Alice has a secret input x, Bob has a secret input y, and they wish to compute $g(x, y)$ securely using *one round* of interaction: Alice should learn $g(x, y)$ but nothing else about y and Bob should learn nothing at all. Communication is restricted to one message from Alice to Bob followed by one message from Bob to Alice. Without the restriction on the number of rounds, this is the problem of secure function evaluation introduced by Yao [26] and Goldreich et al. [17]. It is known that under cryptographic assumptions, every function can be computed securely and using a (small) constant number of rounds.

The problem is closely related to the question of "computing with encrypted data" [22]: Alice holds some input x, Bob holds a function f, and Alice should learn $f(x)$ in a one-round protocol, where Alice sends to Bob an "encryption" of x, Bob computes f on the "encrypted" data x and sends the result to Alice, who "decrypts" this to $f(x)$.

The dual of this is "computing with encrypted functions," where Alice holds a function f, Bob holds an input y, and Alice should get $f(y)$ in a one-round

[1] IBM Zurich Research Laboratory, CH-8803 Rüschlikon, Switzerland, {cca,jca,jmu}@zurich.ibm.com.

[2] NEC Research Institute, Princeton, NJ 08540, USA, joe@research.nj.nec.com.

protocol. This scenario has received considerable attention recently because it corresponds to protecting mobile code that is running on a potentially malicious host, which might be spying on the secrets of the code [24, 25].

In the next paragraphs, honest-but-curious behavior is assumed before we turn to arbitrary malicious behavior. Honest-but-curious behavior models a passively cheating party who follows the protocol, but might try to infer illegitimate information later on.

Homomorphic Encryption and Computing with Encrypted Data. One popular approach to "computing with encrypted data" is to search for a public-key encryption scheme (E, D) with the following homomorphic property: given $E(x)$ and $E(y)$ one can efficiently compute $E(x + y)$ and $E(xy)$. Now, if Alice knows the private key D and sends Bob the public key E together with the encrypted data $E(x)$, then Bob can without interaction compute $E(f(x))$ and send it back to Alice. Although this has been a prominent open problem for years [15], it is still unknown whether such homomorphic encryption schemes exist. On the one hand, Boneh and Lipton [7] have shown that all such *deterministic* encryption schemes are insecure; on the other hand, Sander, Young, and Yung [25] propose a scheme that allows the necessary operations on encrypted data, but comes at the cost of a multiplicative blowup per gate, which limits the possible computations to functions with log-depth circuits.

Computational Assumptions. Note that the above approach to "computing with encrypted data" assumes a computationally bounded Bob, who cannot learn anything about the encrypted values. Alice, however, knows all secrets involved and seems not restricted in her computational power. Thus, the distinguishing feature of "computing with encrypted data" seems to be that it remains *secure against an unbounded Alice*. (In fact, the protocol of Sander et al. [25] is information-theoretically secure for Bob.)

Assume instead that Alice, the receiver of the output, is bounded and Bob is unbounded and consider the same question: is there a one-round secure computation protocol for all efficiently computable functions? We give a positive answer in Section 4.1: *any function computable by a polynomial-sized circuit has a one-round secure computation scheme in this model*. The result is obtained by combining Yao's "encrypted circuit" method [26] for secure computation with a one-round oblivious transfer protocol [4]. To our knowledge, this is the first one-round secure computation protocol for arbitrary polynomial-time computations and gives a partial answer to the long-standing open question of computing with encrypted data mentioned above.

If both parties are bounded, the above solution applies as well (we can even obtain stronger results, see below). Conversely, it is well known that secure computation between two unbounded parties with "full information" is impossible for arbitrary functions and limited to trivial functions g where $g(x, y)$ gives full information about y. The following table summarizes the current state of one-round secure computation (both supply input, only Alice receives output):

Alice	Bob	securely computable functions	reference
unbounded	unbounded	only trivial ones	BGW [5]
unbounded	bounded	log-depth circuits	Sander et al. [25]
bounded	unbounded	polynomial-size circuits	this paper
bounded	bounded	polynomial-size circuits	this paper

Malicious Parties. We also investigate the *malicious model*, where both parties might be actively cheating. One cannot demand that Bob ever sends a second message, but if he does, and Alice accepts, the model ensures that Alice obtains $g(x, y)$ for her input x and some y. We show that if Alice and Bob are both computationally bounded, then a one-round protocol exists also in the malicious model, provided they share a random string and that Alice has a public key for which she is guaranteed to know the private key. This is a realistic model, which is also used elsewhere (e.g., [10]).

These results seem essentially optimal because one round of communication is needed to implement oblivious transfer [20].

Securing Autonomous Mobile Agents. One-round secure computation has been recognized as the solution for keeping the privacy of mobile code intact [24]. Here, a code originator O sends one message containing a protected description of the mobile code to host H, which "runs" the program and sends some output back to O, who decodes the output. (This is an instance of "computing with encrypted functions.") The results in this paper on one-round secure computation directly yield mobile code privacy for *all polynomial-time mobile computations*. This is a vast improvement over both the solutions of Sander and Tschudin [24] (which works for functions representable as polynomials) and the one of Sander et al. [25] (which works for functions computable by log-depth circuits).

In our solution the relative complexities of the computations by O and H are similar; for example, if H runs a long, complex computation with a short output, then O's decoding work is proportional to the complex computation, despite the output being short. We do not know if there are general schemes with "small" decoding complexity for O.

The above models are limited to mobile code that visits only one host, however. In Section 5, a protocol is presented that allows an autonomous mobile agent to visit *several distrusting hosts*, which need not be fixed ahead of time. This flexibility is one of the main benefits of the mobile code paradigm. As with an unencrypted autonomous agent, the communication flow must correspond to a closed path starting and ending at O. The secure computation protocol involves constructing a cascade of Yao-style circuits by the hosts and its evaluation by O. No host learns anything about the agent's or the other hosts' secrets.

Related Work. Protocols for two-party secure function evaluation between a bounded and an unbounded party have previously been proposed by Chaum, Damgård, and van de Graaf [11] and by Abadi and Feigenbaum [1]. The former hides the inputs of one party information-theoretically and the latter hides the circuit information-theoretically from the other party (regardless of who receives

the output). Both protocols have round complexity proportional to the depth of the circuit, however.

The work of Abadi, Feigenbaum, and Kilian [2] on information hiding from an oracle assumes an all-powerful oracle that helps a user with insufficient resources in computing $f(x)$ for his input x; the approach is to transform x into an encrypted instance y and have the oracle compute $f(y)$ such that it learns nothing about x but the user can infer $f(x)$ from $f(y)$. The two main differences to our model are (1) that Bob may also provide an input and (2) that the oracle is limited to computing $f(\cdot)$.

Feige, Kilian, and Naor [13] consider a related model in which two parties perform secure computation by sending a single message each to a third party.

2 Definitions

Recall the three scenarios of one-round secure computation introduced above: *computing with encrypted functions*, *computing with encrypted data*, and *secure function evaluation*. Using a universal circuit for g in secure function evaluation, it is straightforward to realize the first two scenarios from the third one by supplying f as input (at the cost of a polynomial expansion). An equivalence in the other directions is possible by letting f be g with one party's inputs fixed.

The remainder of this section presents definitions for one-round secure computation using secure function evaluation. Formal definitions may be constructed using the methods in [3, 9, 18] and are provided in the full version of the paper.

The security parameter is denoted by k and a quantity ϵ_k is called *negligible* (as a function of k) if for all $c > 0$ there exists a constant k_0 such that $\epsilon_k < \frac{1}{k^c}$ for all $k > k_0$. Throughout we assume that the security parameter k, as well as other system parameters, are always part of the input to all algorithms and protocols.

Honest-but-Curious Model. This definition captures one-round secure computation if both parties follow the protocol. A scheme has to ensure correctness, privacy for Alice, and privacy for Bob.

More precisely, a one-round secure computation scheme in the honest-but-curious model consists of three probabilistic polynomial-time algorithms $A_1(\cdot)$, $A_2(\cdot,\cdot)$, and $B(\cdot,\cdot)$ such that (1) $\forall x \in \mathcal{X}$, $\forall y \in \mathcal{Y}$, if $A_1(x)$ outputs (s, m_1) and $B(y, m_1)$ outputs m_2, then $A_2(s, m_2)$ outputs $g(x, y)$ with all but negligible probability; (2) there exists a simulator $\mathsf{sim}_{\mathsf{Bob}}$ that outputs (s, m_1) such that $\forall x \in \mathcal{X}$, no efficient algorithm can distinguish between the distributions output by $\mathsf{sim}_{\mathsf{Bob}}$ and the output of $A_1(x)$; (3) there exists a simulator $\mathsf{sim}_{\mathsf{Alice}}$ that outputs (s, m_1) such that $\forall x \in \mathcal{X}$ and $\forall y \in \mathcal{Y}$, if m_1 is computed from $A_1(x)$ and m_2 from $B(y, m_1)$, then no efficient algorithm can distinguish between the distributions on (x, m_1, m_2) induced by the real protocol and by $\mathsf{sim}_{\mathsf{Alice}}$.

We say that the scheme is secure for bounded Alice and *unbounded Bob* if the distinguisher in (2) is an arbitrary algorithm and the one in (3) is polynomial-time; similarly, we say it is secure for *bounded Alice and bounded Bob* if both distinguishers are polynomial-time algorithms.

In the model above, A_1 is Alice's query generator that outputs a message m_1 sent to Bob and a secret s, B is Bob's algorithm that outputs message m_2 that is sent to Alice, and A_2 is Alice's decoding algorithm that interprets Bob's answer using s. (All algorithms are for a fixed function g.)

Malicious Model. The malicious model allows arbitrary behavior for (bounded) Alice and Bob. We must ensure that for every strategy of Alice, Bob's reply does not reveal more to her about y than what follows from the function output $g(x, y)$ on a particular x. Bob, on the other hand, must be bound to compute m_2 such that Alice can recover $g(x, y)$ for her x and on *some* legal y, chosen independently from x, or have Alice reject. Intuitively, this can be solved by having both parties supply a zero-knowledge proof with their message that it is well-formed. However, a formal proof of security requires that these proofs are proofs of knowledge. To this end, we use a public-key model [21], where each party has registered a public key and a public source of randomness is available (see Section 4.3).

3 Tools

3.1 Oblivious Transfer

A ubiquitous tool in secure computation is *oblivious transfer*. We use a one-out-of-two oblivious transfer also known as ANDOS (all-or-nothing-disclosure-of-secrets [8]): a sender S has two input values a_0 and a_1, which are strings of arbitrary known length, and a receiver R has a bit c; R obtains a_c, but should not learn anything about $a_{c\oplus 1}$ and S should not learn c.

Let G be a group of large prime order q (of length polynomial in k) such that $p = 2q + 1$ is prime and $G \subset \mathbb{Z}_p$ and let $g \in G$ be a generator. (Note that this allows efficient sampling from G with uniform distribution.) Consider two distributions D_0 and D_1 over G^4, where $D_0 = (g, g^\alpha, g^\beta, g^\gamma)$ with $g \stackrel{R}{\leftarrow} G$ and $\alpha, \beta, \gamma \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and $D_1 = (g, g^\alpha, g^\beta, g^{\alpha\beta})$ with $g \stackrel{R}{\leftarrow} G$ and $\alpha, \beta \stackrel{R}{\leftarrow} \mathbb{Z}_q$. The *Decisional Diffie-Hellman (DDH) assumption* is that there exists no probabilistic polynomial-time algorithm that distinguishes with non-negligible probability between D_0 and D_1.

The following is a sketch of the ANDOS protocol between a sender Bob and a receiver Alice [4], denoted $\mathsf{OT}(c)(a_0, a_1)$. Alice's private input is a bit c and Bob's private inputs are $a_0, a_1 \in G$. Common inputs are p and g.

1. Bob chooses $\delta \stackrel{R}{\leftarrow} G$ and sends δ to Alice.
2. Alice chooses $\alpha \stackrel{R}{\leftarrow} \mathbb{Z}_q$, computes $\beta_c = g^\alpha$, $\beta_{c\oplus 1} = \delta/\beta_c$, and sends β_0, β_1 to Bob.
3. Bob verifies that $\beta_0\beta_1 = \delta$ and aborts if not. Otherwise, he chooses $r_0, r_1 \stackrel{R}{\leftarrow} \mathbb{Z}_q$, computes $(e_0, f_0) = (g^{r_0}, a_0{\beta_0}^{r_0})$ and $(e_1, f_1) = (g^{r_1}, a_1{\beta_1}^{r_1})$, and sends (e_0, f_0, e_1, f_1) to Alice.
4. Alice obtains a_c by computing $f_c/{e_c}^\alpha$.

It is easy to see that if both parties follow the protocol, Alice obtains a_c. Consider security for Alice: c is perfectly hidden from Bob, i.e., in an information-theoretic sense, because β_0 and β_1 are uniformly random among all group elements with product δ. Thus the protocol is secure for Alice against an arbitrarily behaving unbounded Bob.

Consider security for Bob. In the *honest-but-curious model*, Alice chooses β_0 and β_1 honest, i.e., such that $\beta_{c\oplus 1}$ is a random public key and, under the DDH assumption, $(e_{c\oplus 1}, f_{c\oplus 1})$ is a semantically secure encryption of $a_{c\oplus 1}$. Hence the protocol is secure for Bob against a bounded Alice. Furthermore, Step 1 of the protocol is not even needed and Alice may compute $\delta \stackrel{R}{\leftarrow} G$ herself in Step 2. The resulting protocol, denoted by $\mathsf{OT\text{-}1}(c)(a_0, a_1)$, has only one round of interaction.

Assuming *malicious behavior*, a one-round version is also possible in the public-key model using shared random information σ; this version is denoted by $\mathsf{OT\text{-}2}(c)(a_0, a_1)$. Here, Step 1 can again be omitted and Alice chooses δ herself, using the sampling algorithm in G with σ as random source. Intuitively, she then sends δ along with β_0, β_1 to Bob, who verifies that the choice of δ is correct according to σ. However, Alice must also supply a "non-interactive proof of knowledge" of α, the discrete logarithm of either β_0 or β_1 (we refer to Section 4.3 for how this can be done). With these changes, the protocol can be proved secure for Bob against an arbitrarily behaving bounded Alice.

3.2 Encrypted Circuit Construction

Yao's encrypted circuit construction implements secure function evaluation between Alice and Bob such that Alice receives the output $z = g(x, y)$ [26].

We give an abstract version of Yao's construction describing only those properties essential to our analysis. A more detailed treatment of Yao's protocol is found in the literature (e.g., [23]). Let $(x_1, \ldots, x_{n_x})$, $(y_1, \ldots, y_{n_y})$, and $(z_1, \ldots, z_{n_z})$ denote the binary representation of x, y, and z, respectively, and let C denote a polynomial-sized circuit computing $g(\cdot, \cdot)$. Yao's construction consists of three procedures: (1) an algorithm $\mathsf{construct}$ that Bob uses to construct an encrypted circuit, (2) an interactive protocol $\mathsf{transfer}$ between Alice and Bob, and (3) an algorithm $\mathsf{evaluate}$ allowing Alice to retrieve $g(x, y)$. Additionally, the proof of security requires a simulation result.

More precisely, the probabilistic algorithm $\mathsf{construct}(C, y)$ outputs the values $\mathcal{C}, (K_{1,0}, K_{1,1}), \ldots, (K_{n_x,0}, K_{n_x,1}), (U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1})$. The first part $\mathcal{C}$ is a representation for C, with input y hardwired in. It may be viewed as an encrypted version of the n_x-input circuit $C(\cdot, y)$. In order to compute $C(x, y)$, one needs a k-bit key for each input bit x_i; the key $K_{i,b}$ corresponds to the key used for the input $x_i = b$. The pairs $(U_{i,0}, U_{i,1})$ represent the output bits, i.e., if decryption of the circuit produces $U_{i,b}$, then the output bit z_i is set to b.

The $\mathsf{transfer}$ protocol consists of n_x parallel executions of ANDOS. In the i-th execution, Bob has input $(K_{i,0}, K_{i,1})$ and Alice has input x_i. That is, Alice learns $K_{1,x_1}, \ldots, K_{n_x,x_{n_x}}$, but nothing more, whereas Bob learns nothing about $x_1, \ldots, x_{n_x}$. Bob also sends $\mathcal{C}$ and $(U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1})$ to Alice.

The algorithm $\mathsf{evaluate}(\mathcal{C}, K_{1,x_1}, \ldots, K_{n_x,x_{n_x}})$ outputs either a special symbol `reject` or $U_{1,z_1}, \ldots, U_{n_z,z_{n_z}}$. From the latter Alice can recover z, and if Alice and Bob obey the protocol, then $z = g(x, y)$.

A key element of the security analysis is the existence of a polynomial-time simulator $\mathsf{sim}_{\mathsf{Yao}}(C, x, g(x, y))$ that outputs a tuple $\mathcal{C}, K_{1,x_1}, \ldots, K_{n_x,x_{n_x}}$, $(U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1})$; the distribution of the simulator's output is computationally indistinguishable from that induced on these same variables by $\mathsf{construct}(C, y)$ and x. Intuitively, given x and $g(x, y)$, the simulator can simulate Alice's view obtained by running Yao's protocol with an ideal (information-theoretically secure) ANDOS.

The existence of $\mathsf{construct}$, $\mathsf{evaluate}$, and $\mathsf{sim}_{\mathsf{Yao}}$ may be based on the existence of pseudo-random functions [16]; efficient implementations of pseudo-random functions can be based on the DDH assumption [19].

4 One-round Secure Computation for Polynomial-Size Circuits

The basic idea of our one-round secure computation protocols is to combine the one-round oblivious transfer protocols with the encrypted circuit construction.

4.1 Honest Behavior

In the honest case, we use the one-round oblivious transfer protocol $\mathsf{OT\text{-}1}$ and send Bob's reply in $\mathsf{OT\text{-}1}$ along with the encrypted circuit computing g. The resulting scheme consists of the three following algorithms A_1, A_2, and B (using the notation above).

$A_1(x)$: Compute the first messages of Alice for n_x parallel oblivious transfer protocols: Let $(\delta^{(i)}, \beta_0^{(i)}, \beta_1^{(i)})$ be computed as in Step 2 of protocol $\mathsf{OT\text{-}1}$ with input x_i of Alice for $i = 1, \ldots, n_x$. Output $s = (\alpha^{(1)}, \ldots, \alpha^{(n_x)})$ and $m_1 = ((\delta^{(1)}, \beta_0^{(1)}, \beta_1^{(1)}), \ldots, (\delta^{(n_x)}, \beta_0^{(n_x)}, \beta_1^{(n_x)}))$.

$B(y, m_1)$: Invoke $\mathsf{construct}(C, y)$ to obtain $(\mathcal{C}, (K_{1,0}, K_{1,1}), \ldots, (K_{n_x,0}, K_{n_x,1}), (U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1}))$. Next, for each $i = 1, \ldots, n_x$, execute Step 3 of protocol $\mathsf{OT\text{-}1}$ using $(\delta^{(i)}, \beta_0^{(i)}, \beta_1^{(i)})$ (taken from m_1) and with Bob's inputs (a_0, a_1) set to $(K_{i,0}, K_{i,1})$. (Provided $|G|$ is sufficiently large, such an encoding of binary strings in G is possible.) Denote the output of this step by $m_{2,i} = (e_0^{(i)}, f_0^{(i)}, e_1^{(i)}, f_1^{(i)})$. Output $m_2 = (\mathcal{C}, m_{2,1}, \ldots, m_{2,n_x}, (U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1}))$.

$A_2(s, m_2)$: For $i = 1, \ldots, n_x$ execute Step 4 of protocol $\mathsf{OT\text{-}1}$ using x_i as c, $(e_{x_i}^{(i)}, f_{x_i}^{(i)})$ (taken from m_2) as (e_c, f_c), and $\alpha^{(i)}$ (taken from s) as α, hence recovering $K_{1,x_1}, \ldots, K_{n_x,x_{n_x}}$. Finally, invoke $\mathsf{evaluate}(\mathcal{C}, K_{1,x_1}, \ldots, K_{n_x,x_{n_x}})$ to obtain $U_{1,z_1}, \ldots, U_{n_z,z_{n_z}}$ and output z.

4.2 Analysis of the Honest-party Case

Our description of Yao's protocol assumed an ideal implementation of ANDOS. We now analyze the above protocol using the oblivious transfer protocol $\mathsf{OT\text{-}1}$. When both parties are honest, the combined protocol's correctness follows easily from the correctness of Yao's protocol and the oblivious transfer protocol. To show privacy, we construct simulators for Alice's and Bob's views. Note that this separation of privacy and correctness is not valid for parties with arbitrary behavior.

Let $\mathit{View}_{\mathit{Alice}}(x, y)$ and $\mathit{View}_{\mathit{Bob}}(x, y)$ denote Alice's and Bob's view of the protocol (C is always a fixed common input, and is dropped for notational convenience). We must simulate each player's view given only the information they are supposed to learn. That is, Bob is allowed to learn y, and Alice is allowed to learn x and $g(x, y)$.

To simulate $\mathit{View}_{\mathit{Bob}}(x, y)$, we define simulator $\mathsf{sim}_{\mathsf{Bob}}(y)$ as follows:

1. Choose Alice's input $x = 0^{n_x}$.
2. Engage in the secure function evaluation protocol with Bob where the simulated Alice plays as she would be given x. Return the view obtained by Bob during the execution of this protocol.

Lemma 1. *For all values of (C, y), $\mathsf{sim}_{\mathsf{Bob}}(y)$ and $\mathit{View}_{\mathit{Bob}}(y)$ are identically distributed.*

The proof follows from the fact that in every execution of the $\mathsf{OT\text{-}1}$ sub-protocol, Alice's message is independent of her input.

Next, we simulate $\mathit{View}_{\mathit{Alice}}(x, y)$. Define $\mathsf{sim}_{\mathsf{Alice}}(x, g(x, y))$ as follows:

1. The simulator invokes the simulator $\mathsf{sim}_{\mathsf{Yao}}(C, x, g(x, y))$ so as to obtain $\mathcal{C}, K_{1,x_1}, \ldots, K_{n_x,x_{n_x}}, (U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1})$.
2. For $i = 1, \ldots, n_x$, the simulator chooses $K_{i,x_i \oplus 1} = 0^k$.
3. The simulator engages in the protocol $\mathsf{transfer}$ with Alice exactly as would Bob, given input pairs $(K_{1,0}, K_{1,1}), \ldots, (K_{n_x,0}, K_{n_x,1})$ and encrypted circuit $\mathcal{C}$. The simulator returns Alice's view of this protocol.

Lemma 2. *For all values of x and y, $\mathit{View}_{\mathit{Alice}}(x, y)$ and $\mathsf{sim}_{\mathsf{Alice}}(x, g(x, y))$ are computationally indistinguishable.*

The proof works by a hybrid argument (omitted). Our first result follows.

Theorem 1. *Under the DDH assumption, (A_1, A_2, B) are a one-round secure computation scheme in the honest-but-curious model, with perfect security against unbounded Bob.*

4.3 Allowing Malicious Behavior

For polynomially bounded, arbitrarily malicious parties, we obtain secure one-round computation in a model with certified public-keys and public randomness.

First, because we can no longer trust Alice to choose δ at random, we replace protocol OT-1 by protocol OT-2 (using public randomness) in the above construction. Then Bob must prove that his messages in OT-2 are consistent with a correct construction of $\mathcal{C}, \{(K_{i,0}, K_{i,1})\}$ and Alice must prove that she knows the discrete logarithm for one element of each pair $(\beta_0^{(i)}, \beta_1^{(i)})$ (e.g., using a result by Cramer et al. [12]). In the security proof for protocol OT-2 one extracts the discrete logarithms from Alice and thereby obtains her input x (x_i corresponds to the element of $(\beta_0^{(i)}, \beta_1^{(i)})$ of which Alice knows the discrete logarithm).

A *fallacious* step would be to use a public random string to implement non-interactive zero-knowledge proofs (NIZKP) [6, 14] that each player's message is well formed. The formal complication to this method is that "standard" NIZKP are *not* proofs of knowledge. Instead, we use the "public-key" scenario for non-interactive proofs of knowledge, put forth by Simon and Rackoff [21], as follows. Each player has a public-key P that is certified by some trusted center once and forever. The player convinces the center, via a standard zero-knowledge proof of knowledge, that he knows the corresponding secret key S for P. Henceforth, the secret key is assumed available to the simulator/extractor. To make a non-interactive proof of knowledge of the solution to some problem in *NP*, the player simply encrypts, using P, whatever it is he wishes to show knowledge of, and then non-interactively prove (using standard NIZKP) that the encryption, if decrypted, would yield a solution to the problem. The extractor, who knows S, can then recover the solution as well. Details are omitted from this extended abstract.

5 Securing Autonomous Mobile Agents

The mobile agent paradigm has several attractive features. One of them is the flexibility of delegating a task to an autonomous agent, who roams the net, visits different sites, collects information, computes intermediate results, and returns to the originator only when the computation is finished. No interaction with the originator is needed in-between.

Sander and Tschudin [24] recognized that mobile code can be protected against a curious host using the approach of "computing with encrypted functions." However, their solution addresses only the case of agents who return home after visiting one host. We consider *autonomous agents* here that leave the originator without a fixed list of hosts to visit in mind and consider the question: How does the agent migrate securely from one host to another?

5.1 Model

More formally, there is the agent's originator O and ℓ hosts $H_1, \ldots, H_\ell$ that run the agent. The state of the agent is represented by some $x \in \mathcal{X}$. The initial state is chosen by O. All that is known about the computation is represented by $g_j : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}$ associated with H_j, which updates the agent's state according

to H_j's secret input y_j. This models arbitrary polynomial-time computations provided the functions g_j are representable by polynomial-size circuits.

A novel feature of our protocol is that neither the hosts nor the path taken by the agent need to be fixed or known beforehand. Only for the simplicity of description do we assume that the agent travels from O to host H_1, then from host H_j to host H_{j+1} for $j = 1, \ldots, \ell - 1$ and then from H_ℓ back to O; the generalization can be derived easily. The agent is autonomous because either a host may decide where to send the agent next or because the agent, besides the encrypted part, consists also of conventional code that computes where to go next based on non-private information. (Note that migration decisions cannot depend on the private state of the computation, as they would be observable by the hosts and thereby leak information about the internal state!)

A scheme for *secure computation by autonomous mobile agents* consists of efficient algorithms $A_1(\cdot)$, $A_2(\cdot,\cdot)$, $B_1(\cdot,\cdot)$, $\ldots$, $B_\ell(\cdot,\cdot)$. The agent's computation proceeds as follows: first, O runs $A_1(x)$ on input x and thereby obtains a secret s and a message m_0, which O sends to H_1; likewise, for $j = 1, \ldots, \ell$, H_j runs $B_j(y_j, m_{j-1})$ on input y_j, message m_{j-1} and obtains m_j, which it sends to H_{j+1} (with the exception that H_ℓ sends m_ℓ to O). Finally, upon receiving m_ℓ, O obtains the desired result by invoking $A_2(s, m_\ell)$. We require:

Correctness: $\forall x \in \mathcal{X}$ and $\forall y_j \in \mathcal{Y}$, the decoding algorithm $A_2(s, m_\ell)$ outputs $z = g_\ell(\cdots g_2(g_1(x, y_1), y_2) \cdots , y_\ell)$;

Privacy: (1) the inputs and computations of the visited hosts remain hidden from the other hosts: for all j, message m_j does not give information about x and $y_{j'}$ for $j' \leq j$; (2) the originator should learn only the output of the computation, but nothing else about the inputs of the hosts: $\forall x \in \mathcal{X}$ and $\forall y_j \in \mathcal{Y}$, $(j = 1, \ldots, \ell)$, given only x, s, and z (as above), m_ℓ can be simulated efficiently.

Honest-but-curious behavior is assumed on behalf of all parties throughout this section (dishonest behavior can be prevented analogously to the two-party case).

5.2 Protocol

Our protocol for secure computation by autonomous mobile agents is an extension of the one-round secure computation protocol in Section 4.1 to multiple hosts, which take over the part of Bob. O proceeds as Alice, sending the first message and receiving the encrypted circuit computing $g_\ell(\cdots(g_1(x, y_1), y_2) \cdots , y_\ell)$. Each host H_j contributes the part of encrypted circuit representing its function g_j; thus the resulting encrypted circuit is a cascade of sub-circuits. H_1 generates the key pairs representing O's input and computes the answers for the oblivious transfer protocol; these are attached to the computation and reach O with the message from H_ℓ. To extend the cascade of sub-circuits, H_j encrypts each input key of its sub-circuit with the corresponding output key from the preceding sub-circuit. This is done using a symmetric encryption algorithm $\mathsf{enc}_K(\cdot)$, realized in the same way as the encryptions for single gates in Yao's construction; in

particular, this scheme has the property that given a key K, one can efficiently check if a ciphertext represents an encryption under key K.

We describe algorithms A_1, A_2, and $B_1, \ldots, B_\ell$ using notation from above.

$A_1(x)$: Compute the first message (of Alice) for n_x parallel oblivious transfer protocols. This results in $s = (\alpha^{(1)}, \ldots, \alpha^{(n_x)})$ and $\widetilde{m}_0 = \big((\delta^{(1)}, \beta_0^{(1)}, \beta_1^{(1)}), \ldots, (\delta^{(n_x)}, \beta_0^{(n_x)}, \beta_1^{(n_x)})\big)$ computed as in $\mathsf{OT\text{-}1}$. Output s and $m_0 = (\widetilde{m}_0, \emptyset)$.

$B_j(y_j, m_{j-1})$: Invoke $\mathsf{construct}(C_j, y_j)$ to obtain

$$\mathcal{C}_j, (K_{1,0}^{(j)}, K_{1,1}^{(j)}), \ldots, (K_{n_x,0}^{(j)}, K_{n_x,1}^{(j)}), (U_{1,0}^{(j)}, U_{1,1}^{(j)}), \ldots, (U_{n_x,0}^{(j)}, U_{n_x,1}^{(j)}).$$

If $j = 1$, then execute Step 3 of protocol $\mathsf{OT\text{-}1}$ using $(\delta^{(i)}, \beta_0^{(i)}, \beta_1^{(i)})$ (taken from $\widetilde{m}_0$) and with Bob's input set to $(K_{i,0}^{(1)}, K_{i,1}^{(1)})$. Denote the output of the $\mathsf{OT\text{-}1}$ step by $\widetilde{m}^{(i)} = (e_0^{(i)}, f_0^{(i)}, e_1^{(i)}, f_1^{(i)})$. Set $\widetilde{m}_1 = (\widetilde{m}^{(1)}, \ldots, \widetilde{m}^{(n_x)}, \mathcal{C}_1)$ and output $m_1 = (\widetilde{m}_1, (U_{1,0}^{(1)}, U_{1,1}^{(1)}), \ldots, (U_{n_x,0}^{(1)}, U_{n_x,1}^{(1)}))$.
If $1 < j \leq \ell$, then the outputs of $\mathcal{C}_{j-1}$ are recoded as inputs to $\mathcal{C}_j$. To this end, for $i = 1, \ldots, n_x$ do the following: choose a random bit c_i and, for $b \in \{0, 1\}$, encrypt key $K_{i,b}^{(j)}$ under $U_{i,b}^{(j-1)}$ (taken from m_{j-1}) as $V_{i,b\oplus c_i}^{(j)} = \mathsf{enc}_{U_{i,b}^{(j-1)}}(K_{i,b}^{(j)})$ Next, set $\widetilde{m}_j = (\widetilde{m}_{j-1}, \mathcal{C}_j, (V_{1,0}^{(j)}, V_{1,1}^{(j)}), \ldots, (V_{n_x,0}^{(j)}, V_{n_x,1}^{(j)}))$ and then output $m_j = (\widetilde{m}_j, (U_{1,0}^{(j)}, U_{1,1}^{(j)}), \ldots, (U_{n_x,0}^{(j)}, U_{n_x,1}^{(j)}))$.

$A_2(s, m_\ell)$: Run Step 4 of protocol $\mathsf{OT\text{-}1}$ and obtain input keys $K_{1,x_1}^{(1)}, \ldots, K_{n_x,x_{n_x}}^{(1)}$ of $\mathcal{C}_1$. Now, run algorithm $\mathsf{evaluate}(\mathcal{C}_1, K_{1,x_1}^{(1)}, \ldots, K_{n_x,x_{n_x}}^{(1)})$ to obtain the output keys of $\mathcal{C}_1$. Each one of these decrypts one ciphertext $V_{i,b}^{(2)}$ to an input key of $\mathcal{C}_2$, which can then be evaluated and then will allow to decrypt the input keys of $\mathcal{C}_3$. Proceeding similarly for all circuits $\mathcal{C}_3, \ldots, \mathcal{C}_\ell$ will eventually reveal $U_{1,z_1}^{(\ell)}, \ldots, U_{n_x,z_{n_x}}^{(\ell)}$ from which the result z can be retrieved.

As for the security of the protocol, note that each host sees an encrypted circuit representing the computation so far, like Alice in the original protocol but lacking the secrets to decrypt the oblivious transfers. A simulator for each host's view is straightforward. When the encrypted circuit reaches O, it consists only of information that has been constructed using the same method as in the original protocol; thus, the security follows from the original argument.

References

1. M. Abadi and J. Feigenbaum, "Secure circuit evaluation: A protocol based on hiding information from an oracle," *Journal of Cryptology*, vol. 2, pp. 1–12, 1990.
2. M. Abadi, J. Feigenbaum, and J. Kilian, "On hiding information from an oracle," *Journal of Computer and System Sciences*, vol. 39, pp. 21–50, 1989.
3. D. Beaver, "Foundations of secure interactive computing," in *Proc. CRYPTO '91* (J. Feigenbaum, ed.), LNCS 576, 1992.
4. M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications," in *Proc. CRYPTO '89* (G. Brassard, ed.), LNCS 435, pp. 547–557, 1990.

5. M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proc. 20th STOC*, pp. 1–10, 1988.
6. M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge proof systems and its applications," in *Proc. 20th STOC*, pp. 103–112, 1988.
7. D. Boneh and R. J. Lipton, "Searching for elements in black box fields and applications," in *Proc. CRYPTO '96*, LNCS 1109, 1996.
8. G. Brassard, C. Crépeau, and J.-M. Robert, "Information theoretic reductions among disclosure problems," in *Proc. 27th FOCS*, 1986.
9. R. Canetti, "Security and composition of multi-party cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.
10. R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali, "Resettable zero-knowledge," in *Proc. 32nd STOC*, 2000.
11. D. Chaum, I. Damgård, and J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result," in *Proc. CRYPTO '87* (C. Pomerance, ed.), LNCS 293, 1988.
12. R. Cramer, I. Damgård, and B. Schoemakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Proc. CRYPTO '94* (Y. G. Desmedt, ed.), LNCS 839, 1994.
13. U. Feige, J. Kilian, and M. Naor, "A minimal model for secure computation (extended abstract)," in *Proc. 26th STOC*, pp. 554–563, 1994.
14. U. Feige, D. Lapidot, and A. Shamir, "Multiple noninteractive zero knowledge proofs under general assumptions," *SIAM Journal on Computing*, vol. 29, no. 1, pp. 1–28, 1999.
15. J. Feigenbaum and M. Merritt, "Open questions, talk abstracts, and summary of discussions," in *Distributed Computing and Cryptography*, AMS, 1991.
16. O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, pp. 792–807, Oct. 1986.
17. O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *Proc. 19th STOC*, pp. 218–229, 1987.
18. S. Micali and P. Rogaway, "Secure computation," in *Proc. CRYPTO '91* (J. Feigenbaum, ed.), LNCS 576, pp. 392–404, 1992.
19. M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," in *Proc. 38th FOCS*, 1997.
20. R. Ostrovsky, R. Venkatesan, and M. Yung, "Fair games against an all-powerful adversary," in *Advances in Computational Complexity Theory*, AMS, 1993.
21. C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Proc. CRYPTO '91* (J. Feigenbaum, ed.), LNCS 576, pp. 433–444, 1992.
22. R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of Secure Computation* (R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, eds.), pp. 169–177, Academic Press, 1978.
23. P. Rogaway, *The Round Complexity of Secure Protocols*. PhD thesis, MIT, 1991.
24. T. Sander and C. F. Tschudin, "Protecting mobile agents against malicious hosts," in *Mobile Agents and Security* (G. Vigna, ed.), LNCS 1419, 1998.
25. T. Sander, A. Young, and M. Yung, "Non-interactive CryptoComputing for NC^1," in *Proc. 40th FOCS*, 1999.
26. A. C. Yao, "How to generate and exchange secrets," in *Proc. 27th FOCS*, pp. 162–167, 1986.