Cryptographic Security for Mobile Code

Joy Algesheimer Christian Cachin Jan Camenisch Günter Karjoth

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

February 22, 2001

Abstract

This paper addresses the protection of mobile code against cheating and potentially malicious hosts. We point out that the recent approach based on computing with "encrypted functions" is limited to the case where only the code originator learns the result of the computation and the host running the code must not notice anything at all. We argue that if the host is to receive some output of the computation, then securing mobile code requires minimal trust in a third party. Tamper-proof hardware installed on each host has been proposed for this purpose. In this paper we introduce a new approach for securely executing (fragments of) mobile code that relies on a minimally trusted third party. This party is a *generic* independent entity, called the *secure computation service*, which performs some operations on behalf of the mobile application, but does not learn anything about the encrypted computation. Because it is universal, the secure computation service needs to be only minimally trusted and can serve many different applications. We present a protocol based on tools from theoretical cryptography that is quite practical for computing small functions.

1 Introduction

Mobile code is an important programming paradigm for our increasingly networked world. It provides a flexible way to structure cooperative computation in distributed systems. Already today, the Internet is full of mobile code fragments, such as Java applets, which represent only the simplest form of mobile code.

Mobile agents are mobile code that acts autonomously on behalf of a user for continuous collecting, filtering, and processing of information. They combine the benefits of the agent paradigm, such as reacting to a changing environment and autonomous operation, with the features of remote code execution; they operate in computer networks and are capable of moving from server to server as necessary to fulfill their goals. Important applications include mobile computing, where bandwidth is limited or users are disconnected, data retrieval from large repositories, and configuration management of software and networks. Today's vision of mobile agents roaming the Internet may soon become reality as the paradigm is incorporated in large-scale applications.

Although sound definitions of mobile computations are still under debate (e.g., [FPV98]), we assume here that *mobile code* is a program that is produced by one entity, called the *originator*, and is subsequently *transferred* to a second entity, the *host*, immediately before it is executed

by the host. In other words, no manual intervention (such as performing an installation or running a setup routine) is required on behalf of the host; mobile code comes ready to run. Moreover, *mobile agents* are capable of continued, autonomous operation disconnected from the originator and migrate freely to other hosts during their lifetime. Such agents have also been called *itinerant agents*.

Mobile Code Security. Two security problems arise in the area of mobile code: (1) protecting the host from malicious code and (2) protecting the code from malicious hosts. The first problem has received considerable attention because of the imminent threat of computer viruses and Trojan horses—nothing but prominent members of the mobile agent family. Current solutions are to run mobile code in a sandbox with fine-grained access control and to apply code signing for exploiting a trust relation with the code producer. We address the second problem in this paper: protecting the mobile application. Solutions for this are far less developed, but this problem needs to be solved for making the mobile agent metaphor useful in many contexts.

Mobile code is exposed to various security threats: a malicious host may examine the code, try to learn the secrets carried by an agent, and exploit this knowledge in its interaction with the agent to gain an unfair advantage. A host might also try to manipulate the result of a computation. We do not address denial-of-service attacks here, such as killing the agent. Our goal is to achieve secrecy for mobile applications and integrity for their outputs in the traditional sense of information security.

Protecting mobile code was deemed impossible by some mobile code researchers until Sander and Tschudin [ST98] realized that protocols from theoretical cryptography could be useful to execute mobile code in an *encrypted form* on an untrusted host. However, most such protocols for so-called secure computation [GMW87, AF90] require several rounds of interaction and are therefore not applicable in our context. Sander and Tschudin concluded that only functions representable as polynomials can be computed securely in this manner. Subsequent work of Sander et al. extends this to all functions computable by circuits of logarithmic depth [SYY99].

Recently some of us together with Kilian have found a protocol for computing all polynomial-time functions efficiently [CCKM00], which solves the mobile code privacy problem in this form. In particular, this protocol allows any polynomial-size circuit to be evaluated securely in polynomial time using only one round of interaction.

However, this approach has a serious drawback: no information about the encrypted computation must leak to the host and only the originator may receive any output. This rules out any *active* mobile code that performs some immediate action on the host (like a mobile agent in a shopping scenario that accepts or rejects an offer of its host based on a secret strategy [Yee99]). The impossibility of protecting active mobile code is demonstrated in Section 2 below; the basic problem is that a malicious host can observe the output of the computation and simply run the code again with a different input.

The only existing defense for active mobile code against a malicious host uses trusted hardware. This has been proposed by Yee [Yee99] and by Wilhelm et al. [WSB99] and entails running mobile code exclusively inside tamper-proof hardware, encrypting it as soon as it leaves the trusted environment. The implicit assumption one must make here is that all users trust the manufacturer of the hardware. Such an assumption seems very strong and it is unclear whether the benefits of the mobile code *software* paradigm justify the deployment of an expensive *hardware* infrastructure (unlike the example of a DVD player using tamper-proof hardware, which primarily provides the functionality of playing video).

Our Contribution. In this paper, we introduce an architecture for secure execution of active mobile code fragments that needs no additional client hardware. Instead, we propose a *generic secure computation service* that performs some cryptographic operations on behalf of the mobile code; it guarantees privacy as well as integrity of the computation to the code originator and its host. Moreover, the computation service itself does not learn anything about the computation; it must only be trusted not to collude with the originator or the host.

Our architecture builds on tools for secure computation from cryptology and applies them in new ways. In particular, we employ Yao's "encrypted circuit construction" for scrambling a circuit that computes the desired function [Yao86]. Such methods had been thought of theoretical interest only, but current technology makes them appear practical for small tasks where maintaining privacy justifies this overhead.

The generic nature of the proposed computation service has several benefits:

- Its cost can be shared across many applications because it is generic; nothing about its usage must be known before deploying it.
- The trust placed in its integrity is universal and not bound to a particular service or to an application context; secure computation servers may be set up and operated by independent entities.
- It is based on software and commodity hardware only and therefore much cheaper to build and operate than any solution involving specialized hardware.

In many respects, the secure computation service resembles other generic security services like a public-key infrastructure (PKI) or an anonymous re-mailer. These services also enhance security and privacy where needed.

Organization of the Paper. Section 2 introduces a formal model for mobile computations, formulates the desired security properties, and reviews prior work for protecting mobile code. It is shown why the approach based on "one-round secure computation" is not suitable for securing active mobile code. Our architecture is introduced in Section 3, and Section 4 illustrates two applications: a comparison shopping agent and a generalized auction scheme. Conclusions are drawn in Section 5.

2 Protecting Mobile Agents

This section formalizes mobile agent computations and states our desired security conditions. The formal model is then used to argue why protecting active mobile agents purely by software is impossible without further assumptions.

2.1 Model

The defining element of a mobile code computation is that it proceeds autonomously and independently of the originator. We model mobile agent computation as follows.

Participants: There are an originator O and ℓ hosts $H_1, \ldots, H_\ell$, on which the mobile agent runs.

Non-interactive communication: Each participant sends and receives only a single message. We denote by m_0 the message that O sends to H_1 and by m_j the message that H_j sends to H_{j+1} for $j = 1, \ldots, l-1$, and by m_ℓ the message that the last host H_ℓ returns to O.

Computation: Let the state of the mobile agent be an element of a set $\mathcal{X}$. Its initial state x_0 is determined by O. Let the input by H_j be an element of a set $\mathcal{Y}$ and the output to H_j an element of $\mathcal{Z}$ (input and output domains are the same at all hosts for simplicity). The agent computation on host H_j is represented by two functions

$$g_j : \mathcal{X} \times \mathcal{Y} \to \mathcal{X} \quad \text{and} \quad h_j : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$$

that determine the new state $x_j = g_j(x_{j-1}, y_j)$ of the agent and the output $z_j = h_j(x_{j-1}, y_j)$. O obtains the final state $\xi = x_\ell \in \mathcal{X}$ of the agent. The functions g_i and h_i are known to all parties.

A (non-interactive) *secure mobile computing scheme* consists of $2\ell + 2$ algorithms $\mathcal{A}_0$, $\mathcal{A}_1, \ldots, \mathcal{A}_\ell$, $\mathcal{B}_1, \ldots, \mathcal{B}_\ell$, and $\mathcal{D}$ such that for all $j = 1, \ldots, \ell$ and $x_0 \in \mathcal{X}$, $y_j \in \mathcal{Y}$, and with

$$\begin{aligned} m_0 &= \mathcal{A}_0(x_0) \\ m_j &= \mathcal{A}_j(m_{j-1}, y_j) \text{ for } j = 1, \ldots, \ell \\ z_j &= \mathcal{B}_j(m_{j-1}, y_j) \text{ for } j = 1, \ldots, \ell \\ \xi &= \mathcal{D}(m_\ell) \end{aligned}$$

the following two conditions hold.

Correctness: $\xi = g_\ell(x_{\ell-1}, y_\ell)$ and $z_j = h_j(x_{j-1}, y_j)$ for $j = 1, \ldots, \ell$, using

$$x_{j'} = g_{j'}(\cdots(g_2(g_1(x_0, y_1), y_2)\cdots), y_{j'})$$

for $j' = 1, \ldots, \ell - 1$.

Privacy: The inputs, outputs, and the computations of all hosts remain hidden from the originator and from all other hosts, except for what follows from their outputs: O learns only ξ but nothing else about any y_j than what follows from x_0 and ξ, and similarly, H_j learns only z_j but nothing about x_0 and $y_{j'}$ for $j' < j$ than what follows from z_j and y_j.

These requirements can be defined formally using the simulation approach from cryptography [Bea91, MR92, Gol98, Can00]. Note that in the definition of privacy, the information that a host H_j learns about the input x_{j-1} to its part of the computation depends on the combination of the agent output z_j and the host's private input y_j.

For simplicity, the model assumes that the order in which the agent visits all hosts is fixed. It can be extended to allow for the sequence to depend on z_j by introducing a function $\pi : \mathcal{Z} \to \{1, \ldots, \ell\}$ and sending the mobile agent to $H_{\pi(z_j)}$ from H_j.

In the special case of mobile code applications with a single host H, the function g yields O's output ξ and h gives H's output z.

2.2 Software-only Solutions

Sander and Tschudin [ST98] were the first to realize that a software-only solution to protecting mobile code from a malicious host is indeed feasible for small programs using cryptographic

techniques. They proposed to use so-called *homomorphic public-key encryption schemes* that allow for non-interactive addition or multiplication of two encrypted messages by manipulating ciphertext only. In this way, the host can compute any function $g(\cdot, y)$ on a hidden input x that is representable by a polynomial (in the single-host scenario).

This approach was later improved by Sander et al. [SYY99] to non-interactive evaluation of all functions $g(\cdot, y)$ on a hidden input x that can be represented by circuits of logarithmic depth [SYY99]. Cachin et al. [CCKM00] further generalized this to arbitrary functions, provided they can be represented by a polynomial-size circuit; they also described how to realize secure mobile agent applications with multiple hosts in this way.

However, all those solutions address only the secure evaluation of g_j for updating the agent's state and producing the final result, but ignore how to realize h_j for producing output at H_j. More precisely, they are restricted to functions $h_j : \mathcal{Y} \to \mathcal{Z}$ such that the output from the agent to the host must not depend on anything else than its own input.

In fact, it is not hard to see that this is the best one can achieve under the given circumstances. Towards a contradiction suppose there exists an active agent that also outputs some value to its host, for example, in a shopping agent application indicating whether or not to accept an offer. Assume for simplicity that the agent's decision is solely based on the price y_j offered by H_j and that it will buy the cheapest offer; the state of the agent is $x_{j-1} = c$ indicating a secret threshold c chosen by the originator, below which it will accept the offer. Because of the communication constraints in our model it must be that running algorithm $\mathcal{B}_j$ on m_{j-1} and y_j immediately yields z_j. Then H_j can determine whether the agent is willing to accept y_j or not, i.e., whether $y_j < c$. But nothing prevents a malicious host from running $\mathcal{B}_j$ again with some other y'_j and continuing in this way until the agent has leaked c completely, applying simple binary search.

This shows that software-only protection for the privacy of a mobile shopping agent application is not possible. In fact, we can conclude the following.

Proposition 1. *(Non-interactive) secure mobile computing schemes do not exist. In particular, any scheme in which some host is to learn information that depends on the agent's current state cannot be secure.*

As a consequence of this, we must extend our model above in order to obtain privacy and integrity for active mobile agents. Allowing for communication between each host and the originator would solve the problem as mentioned earlier; but it would destroy the benefits of the mobile agent paradigm where the originator may be poorly connected or temporarily off-line. The only alternative seems to extend the model by at least one trusted element.

One such extension, proposed by Yee [Yee99] and by Wilhelm et al. [WSB99], uses trusted and tamper-proof hardware modules at every host, such as smart cards or cryptographic co-processors. Each one of these hardware modules possesses a public key and mobile code can be executed securely using this infrastructure in the following way: After generating the mobile agent code, the originator encrypts it under the public key of H_1's module. Upon receiving some encrypted mobile agent, a host H_j passes it along to its hardware module, together with H_j's input y_j. The module decrypts the code, executes it on the inputs provided and encrypts the output again under the public key of the module in H_{j+1}. Then it returns this encryption to the host, together with z_j, the output intended for H_j. The host sends the encrypted code and the encrypted data to the next host in the sequence.

To guarantee privacy in the formal model discussed above, each hardware module must be trusted to execute the code properly and only once. Furthermore, all trusted modules must be produced and initialized by a trusted, external entity.

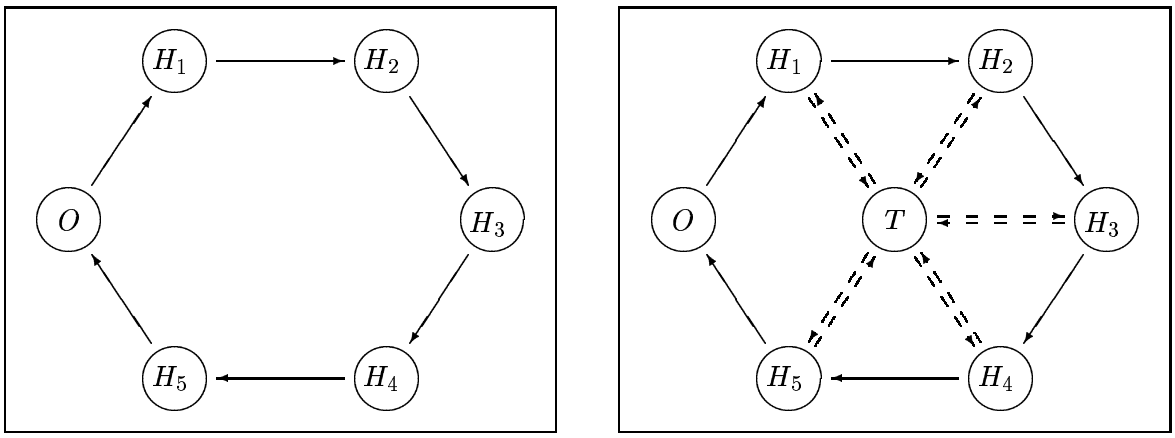

Figure 1: The communication flow of a traditional mobile agent (left) and using the generic secure computation service (right).

In the next section we introduce an alternative extension that is based on a minimally trusted party, the secure computation service.

3 Generic Secure Computation Service

Suppose there exists a third party T that is on-line and connected to all hosts running agent applications and is at their disposal for securing agent computations. Is it possible to realize such a secure mobile computing scheme in which T itself does not gain any information about the computation, no matter how it behaves? All computations should proceed with minimal or no interaction. We give a positive answer below and describe a scheme with these properties under assumptions that (1) T does not collude with the originator against any host, and (2) T does not collude with any host against the originator or against any other host.

Our scheme is generic and not bound to any particular application. Hence the service of T may be offered as a public service for "secure mobile agent computation" on the Internet. The two trust assumptions seem reasonable for such a generic, independent entity. Clients who use this service in the role of O or H (e.g., for comparison shopping) do not have to fear that T has "second thoughts" trying to violate their privacy (e.g., of customer profiling and collecting marketing data). Moreover, T itself has an interest to maintain its reputation as a security provider.

The scheme described below extends the communication pattern of mobile agent computations by two messages from each host to T and back. Figure 1 shows the communication in traditional mobile agent computation and in our scheme.

Our technique is based on encrypting a binary digital circuit that realizes the part of the agent computation in which privacy must be maintained. Although, in principle, such circuits may model arbitrary computations, the associated costs are prohibitive for larger applications. But for small parts of an agent application, like the comparison function of the shopping agent, the overhead seems reasonable.

We proceed by reviewing the encrypted circuit construction for interactive secure protocols.

3.1 Encrypted Circuit Construction

The encrypted circuit construction of Yao [Yao86] is an interactive protocol for secure function evaluation between two parties. We describe it for a binary function $g(\cdot,\cdot)$ and parties Alice (with input x) and Bob (with input y). Bob receives the output $z = g(x,y)$ but learns nothing else and Alice learns nothing at all. We give an abstract version of Yao's construction describing only the properties necessary here (more details can be found in the literature [Fra93, Rog91]).

Let $(x_1,\ldots,x_{n_x})$, $(y_1,\ldots,y_{n_y})$, and $(z_1,\ldots,z_{n_z})$ denote the binary representations of x, y, and z, respectively, and let C denote a polynomial-sized binary circuit computing g. The essential components of Yao's construction are (1) an algorithm construct that Alice uses to construct an encrypted circuit, (2) a transfer protocol between Alice and Bob, and (3) an algorithm evaluate allowing Bob to retrieve $g(x,y)$. More precisely, these procedures are as follows.

(1) The probabilistic algorithm construct(C) takes the circuit as input and outputs the tuple

$$(\mathcal{C},\mathcal{L},\mathcal{K},\mathcal{U}),$$

where $\mathcal{C}$ may be viewed as an encrypted version of the n_x+n_y-input circuit $C(\cdot,\cdot)$ and where $\mathcal{L},\mathcal{K}$, and $\mathcal{U}$ denote lists of "key pairs"

$$\begin{aligned}\mathcal{L} &= (L_{1,0},L_{1,1}),\ldots,(L_{n_x,0},L_{n_x,1})\\ \mathcal{K} &= (K_{1,0},K_{1,1}),\ldots,(K_{n_y,0},K_{n_y,1})\\ \mathcal{U} &= (U_{1,0},U_{1,1}),\ldots,(U_{n_z,0},U_{n_z,1}),\end{aligned}$$

corresponding to x, y, and z, respectively.

In order to compute $C(x,y)$ from the encryption $\mathcal{C}$, Bob needs one "key" for each input bit: $L_{i,b}$ corresponds to input bit $x_i = b$ and $K_{i,b}$ corresponds to input bit $y_i = b$. The keys $U_{i,0}$ and $U_{i,1}$ represent the output bits of the encrypted circuit, i.e., if evaluation of the encrypted circuit produces $U_{i,b}$, then the output bit z_i is set to b.

The particular method in which $\mathcal{C}$ is encrypted ensures that for every gate in the circuit, given two keys representing its input bits, the key representing the resulting output bit can be readily computed, but no information is revealed about which cleartext bit it represents.

(2) Alice and Bob engage in a protocol for *oblivious transfer* [EGL85] or "all-or-nothing-disclosure-of-secrets" [BCR86]. This is an interactive two-party protocol for a sender with input two messages m_0 and m_1 and a chooser with input a bit σ. At the end, the chooser receives m_σ but does not learn anything about $m_{\sigma\oplus 1}$, and the sender has no information about σ.

More precisely, Alice acts as the sender of $K_{i,0}$ and $K_{i,1}$, and Bob obtains for every bit y_i of his input the value $K'_i = K_{i,y_i}$ but learns nothing about $K_{i,y_i\oplus 1}$. At the same time, Alice learns nothing about y_i.

In addition, Alice computes the keys representing x as $L'_i = L_{i,x_i}$ for $i = 1,\ldots,n_x$ and sends

$$\mathcal{C}, L'_1,\ldots,L'_{n_x},\mathcal{U}$$

to Bob.

(3) The algorithm $\mathsf{evaluate}(\mathcal{C}, L'_1, \ldots, L'_{n_x}, K'_1, \ldots, K'_{n_y})$ takes as inputs the encrypted circuit, a representation of x, and a representation of y by the respective keys. It outputs the keys $U'_1, \ldots, U'_{n_z}$ from which Bob can recover z, and if Alice and Bob obey the protocol, then $z = g(x, y)$.

The security of this construction can be proved in the appropriate formal models. Implementing the construct and evaluate algorithms requires pseudo-random functions [GGM86], which are realized in practice by block ciphers. Block ciphers are very fast cryptographic primitives, even if implemented in software.

3.2 Basic Scheme

We first show how to use the encrypted circuit construction for realizing secure mobile code computation with a single host. The extension to multiple hosts is considered in Section 3.3.

Assume T has published the public key of an encryption scheme. We denote the corresponding encryption and decryption operations by $E_T(\cdot)$ and $D_T(\cdot)$, respectively. Assume further that all parties can communicate over secure authenticated links, which could be realized by using standard public-key encryption and digital signatures.

The basic idea is that O constructs an encrypted circuit $\mathcal{C}$ computing the two values ξ and z. It sends $\mathcal{C}$ to H, but encrypts all keys in $\mathcal{K}$ for T and does not include the key pairs in $\mathcal{U}$ which correspond to ξ (denoted by $\mathcal{U}_x$) so that H will not learn anything about ξ. Next H selects from $\mathcal{K}$ the encrypted keys representing y and invokes T to decrypt them in a single round of interaction. Then H evaluates the circuit and obtains z; it also returns the keys in the circuit output representing ξ to O, who can determine ξ from this.

We now give the details. Let C be the binary circuit computing $(\xi, z) = (g(x, y), h(x, y))$ from the same inputs with $n_x + n_y$ input bits $x_1, \ldots, x_{n_x}, y_1, \ldots, y_{n_y}$ and $n_x + n_z$ output bits $\xi_1, \ldots, \xi_{n_x}, z_1, \ldots, z_{n_z}$, slightly modifying the notation from the previous section. The scheme proceeds in five steps.

1. O chooses a string id that uniquely identifies the computation, e.g., containing the name of O, a description of g and h, and a sequence counter. O invokes $\mathsf{construct}(C)$ and obtains $(\mathcal{C}, \mathcal{L}, \mathcal{K}, \mathcal{U})$ as above with $\mathcal{U}$ consisting of $n_x + n_z$ key pairs in total. We let $\mathcal{U}_x$ denote the pairs in $\mathcal{U}$ with indices $1, \ldots, n_x$ and $\mathcal{U}_z$ denote those with indices $n_x + 1, \ldots, n_x + n_z$.

   For $i = 1, \ldots, n_y$ and $b \in \{0, 1\}$, it computes

   $$\bar{K}_{i,b} \;=\; E_T(id \| i \| K_{i,b}).$$

   Let $\bar{\mathcal{K}}$ denote the list of pairs of all such $\bar{K}$. Then O lets $L'_i = L_{i,x_i}$ as above for $i = 1, \ldots, n_x$ and sends

   $$id, \mathcal{C}, L'_1, \ldots, L'_{n_x}, \bar{\mathcal{K}}, \mathcal{U}_z$$

   to H.

2. H sets $\bar{K}'_i = \bar{K}_{i,y_i}$ for $i = 1, \ldots, n_y$ to be the encryptions representing its input y and sends them to T along with id.

3. T decrypts $\bar{K}'_i$ for $i = 1, \ldots, n_y$ and verifies that the ith decrypted string contains the identifier id and index i. If all checks are successful, T returns the decrypted keys $K'_1, \ldots, K'_{n_y}$ to H.

4. H invokes $\mathsf{evaluate}(\mathcal{C}, L'_1, \ldots, L'_{n_x}, K'_1, \ldots, K'_{n_y})$ and obtains $U'_1, \ldots, U'_{n_x+n_z}$. Then H determines $z = (z_1, \ldots, z_{n_z})$ such that $U_{n_x+i,z_i} = U'_{n_x+i}$ for $i = 1, \ldots, n_z$ and forwards the remaining values $U'_1, \ldots, U'_{n_x}$ to O.

5. O determines its output $\xi = (\xi_1, \ldots, \xi_{n_x})$ such that $U_{i,\xi_i} = U'_i$ for $i = 1, \ldots, n_x$.

This is the basic form of the scheme, and it works under the assumption that all parties follow the protocol but try to infer more information later (this is also called honest-but-curious behavior).

The scheme is as secure as the original encrypted circuit construction, assuming that the public-key encryption scheme used by T is semantically secure and that T does not collude with either H or O. More precisely, for the originator O and for H, it corresponds directly to Yao's method with oblivious transfer realized by encryption and a third party T. By itself, T does not obtain any useful information about the circuit and sees only random keys. But if T would collude with O, they could learn the input of H together, whereas if T and H colluded, they could evaluate the circuit for several different input values of H and thus gain information about O's input.

In order to prevent deviations from the protocol and make the scheme robust, one must take additional steps and require that every party proves, using zero-knowledge proofs, that it correctly carried out all operations according to the protocol. T must use a public-key cryptosystem that is secure against adaptive chosen-ciphertext attacks (which is equivalent to it being non-malleable [DDN00]). O and H must also commit to their inputs; moreover, every party should verify the correctness proofs and abort if this fails. In a practical system, all of these can be realized in the so-called "random oracle model" [FS87, BR93] using a secure hash function. In this case, the public-key encryption scheme and the pseudo-random functions for circuit encryption should be implemented with discrete logarithms based on the hardness of the Diffie-Hellman problem [NR97]. Details are omitted here.

A variation of the protocol that is easier to make robust is presented in Section 3.4.

3.3 Extension for Mobile Agents

We can extend the method above to a general mobile computing scheme with hosts $H_1, \ldots, H_\ell$ in the model of Section 2.1. The generalization is the natural one in which each host executes steps 2–4 of the basic scheme above and sends the agent to the next host afterwards.

The originator must prepare one encrypted circuit for each host and there must be a way for incorporating the encrypted state x_{j-1} from $\mathcal{C}^{(j-1)}$ into $\mathcal{C}^{(j)}$ for $j > 1$. This can be done by using the output keys $U'^{(j-1)}_1, \ldots, U'^{(j-1)}_{n_x}$ from $\mathcal{C}^{(j-1)}$ for decrypting a hidden representation of the inputs to $\mathcal{C}^{(j)}$.

Suppose there is a symmetric cryptosystem with encryption and decryption operations under key κ denoted by $E_\kappa(\cdot)$ and $D_\kappa(\cdot)$, respectively. The cryptosystem must include sufficient redundancy such that given a potential key U and a ciphertext c, it can be determined with high probability whether c results from an encryption under U.

The modifications to the scheme are now as follows.

1. The originator obtains $\mathcal{C}^{(j)}$, $\mathcal{L}^{(j)}$, $\mathcal{K}^{(j)}$, $\mathcal{U}^{(j)}$, and $\bar{\mathcal{K}}^{(j)}$ for $j = 1, \ldots, \ell$ in the same way as for C above. However, it selects the values $L'_i = L'^{(1)}_{i,x_i}$ only for $\mathcal{C}^{(1)}$. The identifier in the jth stage is set to $id \| j$. The originator also prepares two encryptions

$$E_{U^{(j-1)}_{i,0}}(L^{(j)}_{i,0}) \quad \text{and} \quad E_{U^{(j-1)}_{i,1}}(L^{(j)}_{i,1})$$

for each $j > 1$ and $i = 1, \ldots, n_x$, and randomly permutes them before assigning them to $V_{i,0}^{(j)}$ and $V_{i,1}^{(j)}$; call the list of such pairs $\mathcal{V}^{(j)}$.

Then O sends

$$id, L'_1, \ldots, L'_{n_x}, \mathcal{C}^{(1)}, \bar{\mathcal{K}}^{(1)}, \mathcal{U}_z^{(1)} \quad \text{and} \quad \mathcal{C}^{(j)}, \bar{\mathcal{K}}^{(j)}, \mathcal{U}_z^{(j)}, \mathcal{V}^{(j)} \quad \text{for } j = 2, \ldots, \ell$$

to H_1 in a single message. Note that nothing in the data for stage j is linked to the identity of H_j and so the sequence in which the hosts are visited can be determined dynamically.

2. For $j > 1$, when H_j runs step 2 of the basic scheme, it has received $\mathcal{V}^{(j)}$ and $U_1'^{(j-1)}, \ldots, U_{n_x}'^{(j-1)}$ from H_{j-1}, who has before evaluated $\mathcal{C}^{(j-1)}$.

   The host interprets each $U_i'^{(j-1)}$ as a symmetric key to E, determines which one of the ciphertexts $V_{i,0}^{(j)}$ and $V_{i,1}^{(j)}$ it decrypts, and then decrypts the one that matches. This yields $L_i^{(j)}$, an oblivious representation of the ith bit in the current state x_j of the mobile agent. Those keys are then needed to evaluate $\mathcal{C}^{(j)}$.

3. When H_j has obtained its output from evaluating $\mathcal{C}^{(j)}$, it forwards all data that it has received from H_{j-1}, together with $U_1'^{(j)}, \ldots, U_{n_x}'^{(j)}$, to H_{j+1}. At the end of the circle, H_ℓ returns only the $U_i'^{(\ell)}$ to O.

In order to make this scheme robust, the same measures as described above should be taken. In particular, T must ensure that it decrypts ciphertexts containing a particular identifier $id\|j$ in at most one execution of step 3.

3.4 Variation

In this section we present a different scheme in the same model for which robustness can be added at much lower cost.

The main difference is that the trusted party generates the encrypted circuit. Because it is trusted to follow the protocol one does not have to add a costly zero-knowledge proof for correctness of the whole circuit. Therefore, the operations of the other parties and the corresponding proofs ensuring robustness become much simpler. T has to know g and h for constructing the circuit, but it may obtain a description of C from O in the first protocol message.

We use a three-party oblivious transfer protocol introduced by Naor et al. [NPS99] in which the role of the chooser is separated among the chooser and a third party, called the receiver. Compared to the standard notion of oblivious transfer (see Section 3.1), the receiver gets the output message m_σ specified by the chooser, who itself learns nothing. This so-called "proxy" oblivious transfer can be realized using three message flows: from chooser to receiver and from receiver to sender and back.

The protocol needs also a one-round implementation of standard oblivious transfer between two parties, which can be realized using the methods of Cachin et al. [CCKM00] or Sander et al. [SYY99].

Note that the resulting overall structure of this protocol is similar to the auction scheme of Naor et al. [NPS99].

Protocol. As in the basic scheme the essential component here is the encrypted circuit construction. The protocol is described for the basic case of mobile code with a single host H.

Suppose O employs a public-key encryption scheme with encryption and decryption operations denoted by $E_O(\cdot)$ and $D_O(\cdot)$, respectively. O starts the computation as the chooser in n_x parallel three-party oblivious transfers, one for each bit of x. It sends these hidden choices to H, who acts as the receiver in the three-party oblivious transfers, together with C and $E_O(\cdot)$. H forwards the appropriate data to T, who acts as the sender; it will send the key pairs $\mathcal{L}$ in the three-party oblivious transfer. Furthermore, H also prepares its input to n_y parallel one-round oblivious transfers (playing the role of the chooser), one for each bit of y. It sends these to T, together with the descriptions of C and $E_O(\cdot)$; T will send the key pairs $\mathcal{K}$ in the one-round oblivious transfers.

T invokes $\mathsf{construct}(C)$ to obtain $\mathcal{C}$ and the key pairs $\mathcal{L}$, $\mathcal{K}$, and $\mathcal{U}$. It replies to H with $E_O(\mathcal{U}_x)$, $\mathcal{C}$, $\mathcal{U}_z$, and the final flows in all oblivious transfer protocols.

From this H can determine the keys $L'_1, \ldots, L'_{n_x}$ representing x and the keys $K'_1, \ldots, K'_{n_y}$ representing y. It runs $\mathsf{evaluate}(\mathcal{C}, L'_1, \ldots, L'_{n_x}, K'_1, \ldots, K'_{n_y})$ to obtain $U'_1, \ldots, U'_{n_x+n_z}$ as above. Then it determines its output z from $U'_{n_x+1}, \ldots, U'_{n_x+n_z}$ and from $\mathcal{U}_z$, and it forwards $U'_1, \ldots, U'_{n_x}$ together with $E_O(\mathcal{U}_x)$ to O. This enables O to obtain its output ξ.

Extension for Mobile Agents. We show how to extend the protocol from a single host to ℓ hosts $H_1, \ldots, H_\ell$. The protocol starts as before for the first host. However, the steps for $H_2, \ldots, H_{\ell-1}$ are slightly different: three-party oblivious transfer and encryption under E_O are not needed. Instead, T encrypts the keys $\mathcal{L}^{(j)}$ in the input of $C^{(j)}$ and representing the state x_{j-1} of the mobile agent under the output keys in $\mathcal{U}^{(j-1)}$ from $\mathcal{C}^{(j-1)}$ as before in $\mathcal{V}^{(j)}$. The keys $\mathcal{U}^{(j-1)}$ can be stored by T between step $j-1$ and step j or they can be sent along with the protocol flow and are transmitted to T via H_{j-1} and H_j (in this case, they must be encrypted using $E_T(\cdot)$). In addition, the last host obtains $\mathcal{U}_x$ encrypted with $E_O(\cdot)$ from T and forwards this to O as before.

Discussion. The communication pattern is the same as in the basic scheme: there is one message from O to H_1, one from each H_{j-1} to H_j and one from H_ℓ to O, plus one communication flow between each host and the trusted party. Robustness can be added as before by using non-malleable public-key encryption schemes and non-interactive zero-knowledge proofs. However, the result will be much more practical because zero-knowledge proofs are not needed for the potentially large encrypted circuit in our trust model—only for the relatively few steps pertaining to the oblivious transfers. Moreover, the encrypted circuit construction can be implemented by a block cipher instead of public-key operations.

4 Applications

We discuss two applications of mobile agents that greatly benefit from privacy support for in mobile code: comparison shopping and a complex auction scheme.

4.1 Comparison Shopping

A mobile agent visits several vendor sites and compares offers—not just based on price, but also on other attributes. The originator wants to maintain the privacy of his preferences, but a vendor has an interest to learn the buyer's strategy as well as information about other vendor's

offers. For complex offers where the price is determined individually for each customer based on its needs, such as in the insurance market, the vendor wants to keep its method of calculating the price secret. All these requirements can be fulfilled by the secure mobile computing scheme.

An electronic negotiation between a buyer and a single vendor can take place using the scheme for secure mobile code that visits a single host. Typically, the vendor acts as the originator and downloads an applet to the buyer's browser (as is already quite common on the Internet). The applet is executed with the help of the trusted computation service by the buyer and the offer is displayed to the buyer. The vendor may obtain some information as well, but it would have to spell out clearly in a "privacy statement" accompanying the applet which information it obtains, such that it can be verified by an independent entity.

A shopping agents that goes out and collects offers from several vendors can be realized as well, but this requires prior agreement on the data format of the offers. It seems therefore restricted to highly structured areas where privacy is important.

4.2 Generalized Auctions

Auctions with generalized bidding strategies present an interesting application area for secure mobile agents. Bidding agents can implement a complex strategy being a function of time and other participants' behavior, which gives the bidder more flexibility compared to traditional single-parameter auctions based purely on price. Sandholm and Huai [SH00] present a mobile agent system to conduct such auctions.

Recently the German UMTS licenses were sold employing a sealed-bid, multiple-round, multiple-lot auction. It provides an interesting example of a real-world generalized auction. Telecom operators could buy either two or three packets of frequencies out of twelve available frequency packets. In each round the bidders had to submit their bids, which had to be increased by a minimum amount over the previous bids. The winners for each frequency were announced at the end of the round. The bidding stopped after a round with no more new bids. During each round the bidders were isolated and under close supervision by the authorities to prevent coalitions. They had to enter the bids into a computer system, which played the role of the auctioneer and computed the winner. The German UMTS auction in August 2000 lasted for 173 rounds during almost three weeks and raised about 99 billion DEM.

As the value of the lots is interrelated, a bidder is interested to define his bidding behavior as dynamically as possible, for example making the valuation of a lot depend on other winning bids that he observed in the previous rounds. If the bidders can express their strategies as a computable function, then one may construct a circuit to compute the auction function, i.e., the outcome of the auction, with the strategies as the private inputs of all participants. This would require an auction agent that visits each bidder only once.

However, in the likely case that the bidders are unable to express their strategies mathematically, each round of the auction could also be performed securely by an auction applet that visits each bidder once and returns to the auctioneer. There it outputs the winning bids or the end of the auction if the bids did not exceed the minimum increment. If the scheme for secure mobile computing is used, then there is *no* single entity that sees all bids (like the auctioneer, its computer system, or its operators).

Generalized auctions are common in electricity markets, equities trading, bandwidth auctions, and transportation exchanges, and bidders often have preferences over combination of items.

4.3 Implementation Note

Although encrypted circuits can be constructed for an arbitrary function and any mobile code application in theory, a practical implementation will only represent privacy-critical parts in this way and execute the remaining parts in the form of conventional programs. Thus, the circuits are rather small and processing them is realistic with current technology. It seems feasible to include them as an add-on to an existing mobile code platform, such as Aglets (`http://aglets.org`).

The comparison shopping agent, for example, could compute most bookkeeping functions in unencrypted form and hide only its preferences and the best offer so far. The same holds for the auction applications. If the encrypted circuit construction is realized using AES with 128-bit keys, an encryption of a binary gate may be stored in 96 bytes, including 64 bits for redundancy. An encrypted circuit that outputs the maximum of two n-bit numbers using specialized comparison gates can be represented by far less than n kilobytes and requires about $100n$ block cipher operations.

5 Discussion

Our scheme for secure mobile agent computing provides an attractive alternative to using trusted hardware. Let us compare the trust assumptions in these two approaches. The proposed scheme relies on the assumption that T does not actively collude with any of the participants against another one. When using trusted hardware, which is also generic, all parties have to trust the hardware manufacturer in the same way.

There is however a small difference because the trusted hardware can observe all computations and may possibly leak some information about this (covert channels could easily be realized and might go to a large government organization). The on-line secure computation service in our approach does not learn anything about the computation, except that it takes place and has a certain size; there is no information that can be leaked.

Otherwise, the differences between the server-based approach and trusted hardware are clearly the speed advantage of secure hardware compared to the encrypted circuit construction on the one hand, and the high cost and low flexibility of trusted hardware on the other hand. Secure hardware is also more flexible for local agent computations involving user interaction.

Note that server-aided computations are quite common for other cryptographic applications and have been studied extensively [AFK87, BQ95, LL95]; these are protocols in which a powerful server performs some computation on behalf of a client device with limited processing capabilities such as a smart card. The server provides computing power to the client, but should not learn anything about the secrets of the client.

References

[AF90] M. Abadi and J. Feigenbaum, *Secure circuit evaluation: A protocol based on hiding information from an oracle*, Journal of Cryptology **2** (1990), 1–12.

[AFK87] M. Abadi, J. Feigenbaum, and J. Kilian, *On hiding information from an oracle*, Proc. 19th Annual ACM Symposium on Theory of Computing (STOC), 1987, pp. 195–203.

[BCR86] G. Brassard, C. Crépeau, and J.-M. Robert, *Information-theoretic reductions among disclosure problems*, Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS), 1986.

[Bea91] D. Beaver, *Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority*, Journal of Cryptology **4** (1991), no. 2, 75–122.

[BQ95] P. Béguin and J.-J. Quisquater, *Fast server-aided RSA signatures secure against active attacks*, Advances in Cryptology: CRYPTO '95 (D. Coppersmith, ed.), Lecture Notes in Computer Science, vol. 963, Springer, 1995.

[BR93] M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, Proc. 1st ACM Conference on Computer and Communications Security, 1993.

[Can00] R. Canetti, *Security and composition of multi-party cryptographic protocols*, Journal of Cryptology **13** (2000), no. 1, 143–202.

[CCKM00] C. Cachin, J. Camenisch, J. Kilian, and J. Müller, *One-round secure computation and secure autonomous mobile agents*, Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP) (U. Montanari, J. P. Rolim, and E. Welzl, eds.), Lecture Notes in Computer Science, vol. 1853, Springer, July 2000, pp. 512–523.

[DDN00] D. Dolev, C. Dwork, and M. Naor, *Non-malleable cryptography*, SIAM Journal on Computing **30** (2000), no. 2, 391–437.

[EGL85] S. Even, O. Goldreich, and A. Lempel, *A randomized protocol for signing contracts*, Communications of the ACM **28** (1985), 637–647.

[FPV98] A. Fuggetta, G. P. Picco, and G. Vigna, *Understanding code mobility*, IEEE Transactions on Software Engineering **24** (1998), no. 5, 342–361.

[Fra93] M. K. Franklin, *Complexity and security of distributed protocols*, Ph.D. thesis, Columbia University, 1993.

[FS87] A. Fiat and A. Shamir, *How to prove yourself: Practical solutions to identification and signature problems*, Advances in Cryptology: CRYPTO '86 (A. M. Odlyzko, ed.), Lecture Notes in Computer Science, vol. 263, Springer, 1987, pp. 186–194.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, Journal of the ACM **33** (1986), no. 4, 792–807.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game or a completeness theorem for protocols with honest majority*, Proc. 19th Annual ACM Symposium on Theory of Computing (STOC), 1987, pp. 218–229.

[Gol98] O. Goldreich, *Secure multi-party computation*, Manuscript, 1998, (Version 1.1).

[LL95] C. H. Lim and P. J. Lee, *Security and performance of server-aided RSA computation protocols*, Advances in Cryptology: CRYPTO '95 (D. Coppersmith, ed.), Lecture Notes in Computer Science, vol. 963, Springer, 1995.

[MR92] S. Micali and P. Rogaway, *Secure computation*, Advances in Cryptology: CRYPTO '91 (J. Feigenbaum, ed.), Lecture Notes in Computer Science, vol. 576, Springer, 1992, pp. 392–404.

[NPS99] M. Naor, B. Pinkas, and R. Sumner, *Privacy preserving auctions and mechanism design*, Proc. 1st ACM Conference on Electronic Commerce, 1999.

[NR97] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS), 1997.

[Rog91] P. Rogaway, *The round complexity of secure protocols*, Ph.D. thesis, Laboratory for Computer Science, MIT, April 1991.

[SH00] T. Sandholm and Q. Huai, *Nomad: Mobile agent system for an internet-based auction house*, IEEE Internet Computing **4** (2000), no. 2, 80–86.

[ST98] T. Sander and C. F. Tschudin, *Protecting mobile agents against malicious hosts*, Mobile Agents and Security (G. Vigna, ed.), Lecture Notes in Computer Science, vol. 1419, 1998.

[SYY99] T. Sander, A. Young, and M. Yung, *Non-interactive CryptoComputing for NC^1*, Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS), 1999.

[WSB99] U. G. Wilhelm, S. Staamann, and L. Buttyán, *Introducing trusted third parties to the mobile agent paradigm*, Secure Internet Programming (J. Vitek and C. D. Jensen, eds.), Lecture Notes in Computer Science, vol. 1603, Springer, 1999, pp. 469–489.

[Yao86] A. C. Yao, *How to generate and exchange secrets*, Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS), 1986, pp. 162–167.

[Yee99] B. Yee, *A sanctuary for mobile agents*, Secure Internet Programming (J. Vitek and C. D. Jensen, eds.), Lecture Notes in Computer Science, vol. 1603, Springer, 1999, pp. 261–273.