Submitted to the forthcoming LNCS on "Mobile Agent Security", November 11, 1997

Protecting Mobile Agents Against Malicious Hosts

Tomas Sander and Christian F. Tschudin

International Computer Science Institute
1947 Center Street, Berkeley, CA 94704, USA
{sander|tschudin}@icsi.berkeley.edu

Abstract. A key element of any mobile code based distributed system are the security mechanisms available to protect (a) the host against potentially hostile actions of a code fragment under execution and (b) the mobile code against tampering attempts by the executing host. Many techniques for the first problem (a) have been developed. The second problem (b) seems to be much harder: It is the general belief that computation privacy for mobile code cannot be provided without tamper resistant hardware. Furthermore it is doubted that an agent can keep a secret (e.g., a secret key to generate digital signatures). There is an error in reasoning in the arguments supporting these beliefs which we are going to point out.

In this paper we describe software-only approaches for providing computation privacy for mobile code in the important case that the mobile code fragment computes an algebraic circuit (a polynomial). We further describe an approach how a mobile agent can digitally sign his output securely.

1 Introduction

Can a program actively protect itself against its execution environment that tries to divert the intended execution towards a malicious goal? After a little thought this seems to be a problem impossible to solve because it leads to an infinite recourse. The assessment routine that would detect wrong execution of code or tampering of data and that would try to counter them would also be subject to diversion. For mobile code applications, more specifically for mobile software agents which are designed to run on potentially arbitrary computers, this problem is of primordial importance. Without strong guarantees on computation integrity and privacy, mobile programs would always remain vulnerable to "hijacking" and "brainwashing".

Consider for example the following scenario. A customized mobile air-fare agent is sent out with the order to visit in turn the servers of several airlines, to query their databases for finding a suitable flight, and, once the best offer has been determined, to book the flight. A simple and profitable attack would be to tamper the agent's state and code such that it forgets the already visited servers and erroneously selects an offering of the malicious server's airline. Another attack would be to rise prices up to some threshold used internally by the agent or, if applicable, to steal the agent's electronic money without further processing. Even more problematic would be a scenario where the mobile agent wants to book the flight and therefor wants to digitally sign an order: here the problem

is that the agent must carry the user's private key and do some computation using the key.
This simple example exposes three fundamental problems of executing mobile code in an untrusted environment:

(i) Can a mobile agent *protect itself against tampering* by a malicious host? (code and execution integrity)
(ii) Can a mobile agent *conceal the program* it wants to have executed? (code privacy)
(iii) Can a mobile agent remotely *sign a document* without disclosing the user's private key? (computing with secrets in public)

Some of the problems mentioned in the shopping agent scenario are alleviated by using state–of–the–art cryptographic algorithms and specially managed trusted servers, others disappear when tamper-resistant hardware can be relied on. But true "self–defense" of mobile agents against malicious hosts is only possible if there exists a nucleus of pure software operations for which computation integrity and privacy can be mathematically proven. The question and challenge thus is whether mobile code can *carry out cryptographic primitives* even though the code a) is executed inside untrusted computing environments and b) should run autonomously i.e., without interactions with its originating site.
We firmly believe that in many cases fully software based *cryptographic* solutions exist for protecting mobile agents against malicious hosts. This belief contradicts the folklore saying that because a host must execute and thus is in possession of an agent's code, the agent is to the host's full mercy. In section 2 we will first discuss the problems of cleartext programs where we also sketch an approach to effectively hide computations from a malicious host. Section 3 describes in more depth the approach of computing with encrypted functions for which we give in section 4 first possible solutions that are based on homomorphic encryption schemes and composition techniques. A solution for the signature problem that relies on another technique is proposed in section 5. Section 6 concludes this paper.

2 Protection of Mobile Agents from Malicious Hosts

An appealing feature of mobile agent technology is that a user can delegate a task into the net: the agent autonomously roams around, locates information, computes intermediate results and triggers remote actions all its way long without interaction with the originator. Cryptographic solutions for securing the execution of mobile agents thus should conform to the requirement that they do not introduce interactive protocols (involving the originator of the agent). The other goal we want to reach is that the agent should be able to execute security sensitive computations even in an untrusted execution environment. Before discussing the feasibility of such solutions we briefly review other techniques that were proposed in order to alleviate the threats to mobile code.

2.1 Detection of Tampering vs. Prevention

An effective approach of securing mobile agents consists in letting them circulate in trusted execution environments only. Thus, by setting up a trusted network of nodes,

by encrypting agents as they are sent from node to node and by authenticating the host before an agents transports itself to it (as well as authenticating the agents before it enters a host), we can make it highly unlikely that an agent encounters a malicious host. However, this severely hurts the concept of an open agent system where new servers can join the system as new needs show up. It also relies on the trust model to be effective which means that this approach can not exclude the possibility that although a host is trusted it behaves maliciously.

Detection of tampering concentrates more on the longlived interests a host may have when it joins the mobile agent network: by threatening to bar a host from further business, we restrict the security measures to detecting tampering a posteriori. Should tampering be observed one could forward this information to rating agencies that maintain records on a server's trustworthiness (social control). Alternatively one could attempt legal steps to recover possible losses and to make punitive claims. Several (cryptographic and non-cryptographic) techniques have been proposed in this direction which require the host to do some extra work in order to proof later on that it duly executed a specific mobile agent [18, 15]. It would also be possible to add "dummy data items" to an agents which are offered as potential objects to tamper with: a returning agent may then be checked to see whether these items were modified or not [9]: Clearly the last approach lacks the necessary cryptographic strength that would be required to serve as a proof in a court room. In general, detection approaches are ineffective for attacks where the culprit may not be identified or does not existing anymore once a fraud is detected.

A further step towards protecting a mobile agent against malicious hosts is to make tampering difficult or expensive. Code obfuscation, for example, proposes to make the agent's program illegible and thus difficult to manipulate [7]. One major problem is that unless provably effective techniques can be applied here, this remains an arms race where each new masquerade technique is immediately paralleled by countermeasures (see for example the case of Java byte code obfuscators [14, 4]).

Still another approach would be to protect an application as a whole instead of protecting the individual agents it is composed of (see e.g., [18] for a discussion of such an approach and further references). A specific task may be split into several mobile agents that collaborate with each other from different computation platforms, using secret sharing schemes. One of these platforms may be a Trusted Mobile Agent Computing Base that the originator trusts. Also if the execution platform for the complimentary agent is chosen at random the threat of a collusion attack may be unlikely.

2.2 Where Tamperproof Solutions are Required

Despite all possible attacks against mobile agents that may be countered with some of the techniques described above there still remain cases where we need *provably* secure prevention. For example, detection of tampering is inappropriate for E-money if the amount of money involved is too low to justify legal actions or law enforcement is difficult. Also remote digital signing of contracts would become virtually impossible because divulging the user's private key would deprive the user from the possibility to proof that she did not order some good. So unless one wants to exclude certain types of actions a user wants to delegate to its mobile agent, there clearly are cases where prevention of tampering and privacy guarantees are mandatory. Therefore, in this

paper we concentrate on positive solutions for the problem of providing provably strong protection to individual mobile agents against tampering and spying attacks.

2.3 A General Belief on the Vulnerability of Mobile Agents

There is a widespread belief in the mobile agent community that an entity which executes a given program has fully control over its execution, that the entity may potentially fully understand the program and therefore eventually can change it in any way it wants. Without linking the execution of a program to some trusted "safe haven" there would therefore be no way to let a mobile agent do security sensitive operations. This view has been expressed at several places in the literature. Chess, for example, writes in [2]:

> "It is impossible to **prevent** agent tampering unless trusted (and tamper-resistant) hardware is available [. . .]. Without such hardware, a malicious [host] can always modify/manipulate the agent. [. . .] As alluded to above, it is impossible to keep an agent private unless its itinerary is known in advance."

While this belief is often restated, there are no rigorous arguments that could be used to verify this statement. However, there are some intuitive arguments that seem to strongly support this view. The following points certainly apply for a mobile agent's program:

- Cleartext data can be read and changed.
- Cleartext programs can be manipulated.
- Cleartext messages, e.g., to the originator, can be faked.

But the important point is that there is no intrinsic reason why programs have to be executed in cleartext form: In the same sense that you can communicate some ciphermessage to another party without understanding it, we would like a computer to execute a cipherprogram without understanding it. So our claim is that the folklore about the mobile agent's vulnerability is wrong because it tacitly assumes that a mobile agent consists of cleartext data and cleartext programs.

2.4 The Possibility of Executing Encrypted Programs

While later on in this paper we will give examples for how one can encrypt functions such that the executing entity will not learn anything substantial about them, we will briefly review the consequences this has.
A first and nice observation is that if we can execute encrypted programs without decrypting them, we automatically have a) code privacy and b) code integrity in the sense that specific tampering is not possible. Attacks by a malicious host would then be reduced to actions "at the surface of the mobile agent": denial of service, random modifications of the program or of its output as well as replay attacks.
However, a solution for the protection of mobile code is susceptible to new attacks that were not possible in a non-mobile program environment. Because an attacker can run the program in a fully controllable setting and arbitrarily often, he could collect sufficient data to recover essential information about the program, even if it is encrypted.

2.5 Realizing Cryptographic Primitives in Unsecured Hosts

The biggest challenge is indeed the question whether useful cryptographic primitives can be realized with mobile code. If in fact sufficiently many and powerful cryptographic services can be run even at distrusted places, we can envisage fully mobile agent based software environments that can be trusted. They can keep secrets, take care of their replication for increased redundancy, perform cryptographic protocols with other mobile agents and disclose their content if told so (after they verified themselves authorization, of course). In this paper we present in section 5 a first cryptographic primitive that a mobile agent could implement in a secure way, namely digital signing.

3 Computing with Encrypted Functions

Instead of using the more general term 'program' as we did in the previous sections we will from now on differentiate between a *function* and the *program* that implements it. Thus, our goal is to encrypt functions such that their transformation can again be implemented as programs. The resulting program will consist of cleartext instructions that a processor or interpreter understands. What the processor will not be able to understand is the "program's function".
Closely related to our goal of computing with encrypted function (CEF) is the problem of computing with encrypted data (CED). Starting from this point we will discuss in this section the limits of the solutions proposed so far and explore some structural constraints on encrypting functions that are imposed by the desired non-interactiveness for mobile agents. We then identify an important class of such encryptable functions, namely polynomials and rational functions.

3.1 Computing with Encrypted Data

The problem of computing with encrypted data (CED) has been described by Abadi and Feigenbaum [1] in the following way:

> Bob has an algorithm to compute a function f and is willing to compute $f(x)$ for Alice. Alice wants to compute f on her private input x but does not want to reveal x to Bob. Furthermore Alice should not learn anything substantial about the algorithm of Bob for computing f.

Their proposed solution yields a highly *interactive* protocol to this problem in the model of "Boolean circuits": it allows Alice to encrypt the input data x in such a way that Bob can compute $f(x)$ for her without getting to know the cleartext x. Abadi and Feigenbaum also point to the following relation between CED and computing with encrypted functions:

> *"It is also clear from this description that the distinction between 'data' and 'circuits' is unnecessary. If C has the ability to hide a circuit, then he can also hide some private data, simply by hardwiring it into the circuit. Conversely, in protocols in which C has the ability to hide data, he can also hide a circuit through a detour: C can run the protocol, take the circuit for f to be universal circuit, and use an encoding of the circuit he wants to hide as an input."*

Assume that Alice wants to have her function g executed by Bob. Thus, by letting Bob's algorithm f be a universal circuit, we can encrypt the Boolean circuit computing g as input data for Bob's universal circuit. While it is therefore principally possible to compute with encrypted functions, Abadi and Feigenbaum's solution is interactive and requires several rounds of message exchanges between Alice and Bob. The number of communication rounds is related to the depth of a Boolean circuit which represents the universal circuit and may be quite large. Hence the needed amount of interactiveness contradicts the spirit of mobile agents which should be able to perform their computation in a most autonomous way.
Another also important drawback is that the reduction of CEF to CED is infeasible. It is true that a Boolean function (circuit) $f : \mathcal{B}^n \rightarrow \mathcal{B}$ can be realized by substituting in a universal Boolean function $U : \mathcal{B}^m \rightarrow \mathcal{B}$ certain variables specifying the function f. However, for a function $U : \mathcal{B}^m \rightarrow \mathcal{B}$ that is universal for the class of boolean functions $\{f : \mathcal{B}^n \rightarrow \mathcal{B}\}$ we necessarily have that $m \geq \frac{2^n}{\log(2n+2)}$ [11]. Thus we get an exponential blowup by using universal Boolean functions which means that this principally correct approach is computationally infeasible.
So seemingly there is a difference between hiding "data" and hiding "circuits". The infeasibility of the reduction suggests that CEF may me much harder then CED. We will show for an important instance of algebraic circuits that actually the opposite might be case because we can significantly weaken the conditions on the encryption function.

3.2 Non-interactive Computing with Encrypted Functions

With the requirements of mobile agents in mind we can now state the problem that we want to solve:

> Alice has an algorithm to compute a function f. Bob has an input x and is willing to compute $f(x)$ for her, but Alice wants Bob to learn nothing substantial about f. Moreover, Bob should not need to interact with Alice during the computation of $f(x)$.

A Protocol for "Non-Interactive Computing with Encrypted Functions". For letting Alice and Bob work together in the way described before, we assume that a function f can be transformed (encrypted) to some other function $E(f)$. The encryption hides the function f and may or may not also contain the encryption of the output data. We let the notation $P(f)$ stand for the program that implements the function f. Figure 1 depicts a protocol in which Alice does not send to Bob the program $P(f)$ for the plain function f but the program $P(E(f))$ for the encrypted function $E(f)$. Bob only learns about the program $P(E(f))$ that he has to apply to his input x and the result of this computation that he has to return to Alice.
The simple protocol of figure 1 for non-interactive computing with encrypted functions looks like this:

(1) Alice encrypts f.
(2) Alice creates a program $P(E(f))$ which implements $E(f)$.
(3) Alice sends $P(E(f))$ to Bob.

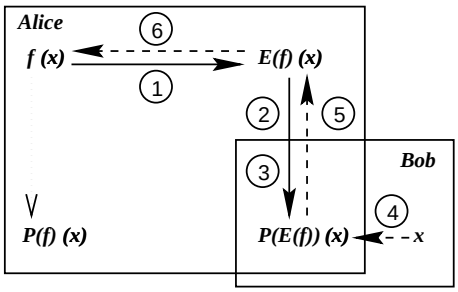

Fig. 1. Computing with encrypted functions.

(4) Bob executes $P(E(f))$ at x.
(5) Bob sends $P(E(f))(x)$ to Alice.
(6) Alice decrypts $P(E(f))(x)$ and obtains $f(x)$.

3.3 Computing with Encrypted Functions using Composition

A simple way of achieving code privacy consists in Alice diverting the function herself. Thus, Bob is asked to compute some systematically modified function $E(f)$ whose results can be ungarbled exclusively by Alice.

Let us describe an easy example. Assume Alice wants to evaluate a linear map A at Bob's input x on Bob's computer. She does not want to reveal A to Bob, so she picks at random an invertible matrix S and computes $B := SA$. She sends B to Bob, Bob computes $y := Bx$ and sends y back to Alice. Alice computes $S^{-1}y$ and obtains the result Ax without having disclosed A to Bob.

This is an example for CEF where the encryption of a function can be realized efficiently. The encryption of matrix A also garbles the output of the computation, so Alice has to decrypt the result sent back by Bob.

The matrix example can be generalized: we encrypt a polynomial f by composing it with another function g. Assume f is a rational function (the quotient of two polynomials) and s is a rational function too such that Alice is able to invert s efficiently, then let $E(f) := s \circ f$. The security of this method is based on the difficulty of decomposing the resulting $E(f)$:

Decomposition Problem: Given a multivariate rational function h that is known to be decomposable, find s and f such that $h = s \circ f$.

Interestingly, there are already results on the hardness of decomposing rational functions. No polynomial time algorithm for decomposing multivariate rational functions is known [19]! Furthermore, ways to construct rational functions s that we use to garble f and that are easy to invert have been proposed by Shamir [13] in a different context. An in-depth analysis of such a composition approach remains to be carried out. But it shows that interesting candidates for doing CEF indeed exist.

3.4 Encrypting Polynomials and Rational Functions

The challenge is to find encryption schemes for arbitrary functions. In a first approach we identify specific function classes for which we can find encrypting transformations. An interesting class are polynomials and rational functions. At the current stage we have to leave it open whether the CEF approach is applicable to arbitrary functions – we even can not claim to have achieved a complete solution for the case of all polynomials. However, within the restricted setting of polynomials and rational functions we can prove first positive results that falsify the "general belief on mobile code vulnerability" for non trivial cases. Rational functions represent a very rich and important class of functions. By studying them we hope to find principles that apply to a much broader class of functions.

On the Relation Between General Programs, Boolean Circuits and Algebraic Circuits. What are the fundamental limits for the set of functions that can be computed in encrypted form? Our goal is to map a cleartext program P to an encrypted program P_E such that Alice can recover $P(x)$ from $P_E(x)$ for some unknown input x of Bob. The encrypted program can not just be any data stream but has to be an executable program. Therefore most of the ordinary *data* encryption techniques can not be applied. Furthermore we want that the cleartext program and its encrypted form are "compatible" with each other. The mathematical analogue is the one of an algebraic structure i.e., a domain set M and a number of operations and relations on M. The mathematical analogue for "compatible transformations" are homomorphisms that are compatible with the operations and relations. Assume for example that $(G,+)$ is a group. A homomorphism $\varphi : (G,+) \rightarrow (G,+)$ is a map such that $\varphi(x+y) = \varphi(x) + \varphi(y)$. Programs are usually build out of a small set of building blocks – the instructions the programming language provides. What we need is the computational analogue to homomorphisms which respects the building blocks. Processing *data* via "homomorphic functions" has been studied e.g., in [12] and [6].
In the following section we focus on algebraic homomorphic encryption schemes i.e., maps between algebraic structures (we study rings) that are compliant with the compatibility constraint. Programs thus would be represented by algebraic circuits or polynomials. From a cryptographic viewpoint the mathematical "compatibility" requirement by using homomorphisms may be too strong. Recall that the map we use to transform a cleartext program into an encrypted program should be hard to invert for an adversary. There might not be enough (mathematical) homomorphisms available as encryption functions or they might be too easy to invert and therefore not be suited for one-way encryption functions. Consider for example the ring $\mathbb{Z}/n\mathbb{Z}$ where the only functions that are additively homomorphic are linear functions $x \rightarrow cx$ which turn out to be totally insecure. However, the required "compatibility" is considerably weaker in a computational framework than in the mathematical framework: instead of requiring for a map between groups that $\varphi(x+y) = \varphi(x)+\varphi(y)$, it is for computational purposes sufficient that $\varphi(x+y)$ can be efficiently computed from $\varphi(x)$ and $\varphi(y)$. Exponentiation in $\mathbb{Z}/p\mathbb{Z}$ is an example for such a map.
How do algebraic circuits relate to Boolean circuits? It is known that (under reasonable conditions) every algebraic circuit on finite fields can be simulated efficiently by Boolean

circuits. The converse of this is wide open (cf. [16] for a discussion of these issues). Shifting from a Turing machine model to Boolean circuits is not a restriction: Every language in $\mathcal{P}$ i.e., which can be recognized by a deterministic Turing machine in polynomial time, can be recognized by uniform Boolean circuits of polynomial size [17]. So there may be programs that can not be efficiently simulated using algebraic circuits (but which are feasible for Boolean circuits). It would be very interesting to derive methods for evaluating encrypted Boolean circuits non-interactively by conceding that some information about the original circuit may be revealed. The requirement to hide *all* information about a Boolean circuit is too strong because it leads to the use of universal Boolean circuits which we already had identified as computationally infeasible before. But for many applications it might be enough to hide only partial information about the Boolean circuit which allows to circumvent the universal construction.

4 Homomorphic Encryption Schemes

What are the structural requirements of encryption functions that can map one function to another one? We start this investigation by looking at systems which enable to do computations on encrypted data. In the beginning of the 90s Feigenbaum and Merritt asked the following question [5]:

> Is there an encryption function E such that both $E(x + y)$ and $E(xy)$ are easy to compute from $E(x)$ and $E(y)$?

Encryption functions $E : R \to S$ for rings R and S having the property stated above are called *algebraic homomorphic encryption schemes*. Their ability to serve as encryption schemes for non-interactive computing with encrypted data and encrypted functions depends on the homomorphic properties a specific homomorphic encryption scheme (HES) has.

Definition 1. Let R and S be rings. We call an (encryption) function $E : R \to S$

- *additively homomorphic* if there is an efficient algorithm PLUS to compute $E(x+y)$ from $E(x)$ and $E(y)$ that does not reveal x and y,
- *multiplicatively homomorphic* if there is an efficient algorithm MULT to compute $E(xy)$ from $E(x)$ and $E(y)$ that does not reveal x and y,
- *mixed multiplicatively homomorphic* if there is an efficient algorithm MIXED-MULT to compute $E(xy)$ from $E(x)$ and y that does not reveal x,
- *algebraically homomorphic* if it is additively and multiplicatively homomorphic.

4.1 Homomorphic Schemes for Computing with Encrypted Data

In [5] Feigenbaum and Merritt wonder whether there exist algebraic homomorphic encryption schemes. The reason for this is the following

Proposition 2. *Algebraic homomorphic one-way trapdoor functions $E : R \to S$ allow non-interactive CED for the evaluation of polynomials (resp. algebraic circuits) $p \in R[X_1, \ldots, X_s]$.*

Proof. Assume E is algebraic (i.e., additively and multiplicatively) homomorphic and let $p = \sum a_{i_1 \ldots i_s} X_1^{i_1} X_s^{i_s}$ be the polynomial to be evaluated at some encrypted value $E(x)$. Using the following protocol, Bob computes $E(p(x))$ without knowing x.

(1) Alice computes $E(x)$ on her private input x.
(2) Alice sends the function E, the algorithms PLUS and MULT and the value $E(x)$ to Bob.
(3) Bob writes a program P that implements p in the following way:
- each coefficient $a_{i_1 \ldots i_s}$ of p is replaced by $E(a_{i_1 \ldots i_s})$,
- each multiplication in the evaluation of p becomes a call to MULT,
- each addition in the evaluation of p becomes a call to PLUS.

Using this program P, Bob computes $P(E(x))$.
(4) Bob sends $P(E(x)) = E(p(x))$ back to Alice.
(5) Alice decrypts it i.e., computes $E^{-1}(E(p(x)))$ and obtains $p(x)$.

The program P when run on the value $E(x)$ in fact returns $E(p(x))$ because PLUS$(E(a), E(b)) = E(a+b)$ and MULT$(E(a), E(b)) = E(ab)$. The algorithms for PLUS and MULT (which are dependent on E) are given to Bob e.g., in form of mobile code.

4.2 Homomorphic Schemes for Computing With Encrypted Polynomials

Along the same line as above we can also use algebraic homomorphic trapdoor functions for hiding the *function* instead of the data.

Proposition 3. *An algebraic homomorphic one-way trapdoor function $E : R \to S$ allows non-interactive CEF for polynomials (resp. algebraic circuits) $p \in R[X_1, \ldots, X_s]$.*

Proof. Similar to the case of CED we use the encryption function E to encrypt the coefficients of p. But this time it is Alice who performs the encryption: thus it is Alice who creates the program P and who sends it to Bob together with E. Sending E to Bob is necessary so he can compute $E(x)$ on his input value x before feeding it to the program P. We could imagine that E is also included in the program P as are the PLUS and MULT procedures. Bob then simply runs this fully selfcontained program P_E for the input x and returns to Alice the program's output.

Remark. (Limits of applicability due to information leakage) This protocol has the property that it reveals the non-zero coefficients of the cleartext polynomial p because Bob can compute $E(0)$ and compare it with the coefficients provided by Alice. Partial information about the polynomial is leaked: by comparing the coefficients of the encrypted polynomial an adversary gets to know which coefficients of the cleartext polynomial are equal and which are different.
For classes of polynomials whose coefficients belong to a small subset of R our scheme (like any deterministic public key cryptosystem) is vulnerable to low entropy attacks where an adversary computes a table of the encrypted values of the expected coefficients and then decrypts by looking up the coefficients of $E(f)$ in his table. Probabilistic encryption schemes secure against this type of attack. However, even with probabilistic encryption schemes can not every family of polynomials be safely encrypted by our

protocol. The family of RSA–encryption functions, for example, would still be vulnerable under this protocol because there is only *one* non-zero exponent to look for. As the encrypted polynomial of the RSA-function has only a polynomially bounded number of monomials an adversary might find the secret key by exhaustive search.

Unfortunately there is no secure general *algebraic* homomorphic encryption scheme known so far. So it is desirable to find other HES to perform CEF:

Proposition 4. *Let $E : R \to R$ be an additively and mixed multiplicatively homomorphic encryption scheme. Then Alice needs to publish only* PLUS, MIXED-MULT *and $E(f)$ to realize CEF for f.*

Proof. Let again $p = \sum a_{i_1 \dots i_s} X_1^{i_1} \dots X_s^{i_s}$

(1) Alice creates a program $P(X)$ that implements p in the following way:
- each coefficient $a_{i_1 \dots i_s}$ of p is replaced by $E(a_{i_1 \dots i_s})$
- the monomials of p are evaluated on the input $x_1, \dots, x_s$ and stored in a list $L := [\dots, x_1^{i_1} \dots x_s^{i_s}, \dots]$
- the list $M := [\dots E(a_{i_1 \dots i_s} x_1^{i_1} \dots x_s^{i_s}) \dots]$ is produced by calling MIXED-MULT for L and the coefficients $E(a_{i_1 \dots i_s})$
- the elements of M are added up by calling PLUS.

(2) Alice sends the program P to Bob.
(3) Bob runs P on his private input $x_1, \dots, x_s$ and obtains $P(x_1, \dots, x_s)$
(5) Bob sends the result $P(x_1, \dots, x_s) = E(p(x))$ back to Alice.
(6) Alice decrypts the result by applying E^{-1} and obtains $p(x)$.

Remark. In this case E needs not necessarily to be a one-way function anymore because f needs not to be disclosed by Alice at all (PLUS and MIXED-MULT are sufficient). This further weakening of the requirements on the encryption scheme makes it easier to come up with such a scheme. On the other hand the protection of E against disclosure is rather weak: once an adversary gets to know $E(1)$ he may compute $E(x)$ for arbitrary x by applying MIXED-MULT to the pair $(E(1), x)$.

4.3 $\mathbb{Z}/n\mathbb{Z}$-Cryptosystems

We have seen that it is very useful for CEF to have encryption functions that are additively and mixed multiplicatively homomorphic. A simple but important observation is that for $\mathbb{Z}/nZ$-cryptosystems the first property (additively homomorphic) implies the second one. This makes it possible to describe encryption schemes that can be used to realize computing with encrypted polynomials. In this section we discuss HES for the ring $\mathbb{Z}/nZ$.

Lemma 5. *An additive homomorphic encryption function on $\mathbb{Z}/nZ$ is also mixed multiplicative homomorphic.*

Proof. We have to construct an algorithm MIXED-MULT using PLUS only, such that MIXED-MULT$(E(x), y) = E(xy)$. Assume $y = \sum_{i=0}^{\log n} y_i 2^i$. First compute the list $E(x2^i), 1 \leq i \leq \log n$ by repeated addition using PLUS. To obtain $E(xy)$ add up those $E(2^i x)$ for which $y_i \neq 0$.

Corollary 6. *An additively homomorphic encryption scheme* $E : \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ *allows CEF for polynomials.*

Proof. Combine proposition 4 and lemma 5.

4.4 Algebraic Schemes That are Additively Homomorphic

There are already schemes on $\mathbb{Z}/n\mathbb{Z}$ existing that enable to compute $E(x+y)$ from $E(x)$ and $E(y)$ directly. Examples are the Naccache-Stern public key encryption function [10] and Ferrer's "privacy homomorphism" [6]. However, they can not be used for CEF. The Naccache-Stern approach is computationally infeasible because to guarantee the correctness of the results the polynomial many calls to PLUS that in general are needed to perform CEF require the system parameter p to be chosen exponentially large. The Ferrer privacy homomorphism is multiplicatively homomorphic but not mixed-multiplicatively homomorphic. So the encryption function E would have to be published in order to perform CEF. However, the security of his scheme (used for processing encrypted data) relies on the secrecy of E. We propose a scheme that does not have these problems.

A Scheme Based on Exponentiation
The exponentiation map $E : \mathbb{Z}/(p-1)\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}, x \mapsto g^x$, for a prime p and g a generator of $(\mathbb{Z}/p\mathbb{Z})^\times$ is additively homomorphic: the function PLUS is the simple multiplication because $E(x+y) = E(x)E(y)$. To recover x from $y := E(x)$ one has to solve the discrete logarithm problem $y = g^x$ which is believed to be hard. Alice chooses a prime p such that the discrete log problem is easy to solve (e.g., if $p-1$ has only small prime factors she can use the Pohlig-Hellmann algorithm for computing discrete logarithms efficiently). She further chooses a generator g of $(\mathbb{Z}/p\mathbb{Z})^\times$ which she keeps secret. Alice can use corollary 6 to realize CEF.

Remark. The security of the scheme relies on the secrecy of g. An additively homomorphic encryption scheme based on discrete logarithms which does not have this shortcoming i.e., which can be published, is developed by Lipton and Sander in [8]. Their scheme is furthermore probabilistic which significantly reduces the information leakage about the original polynomial.

4.5 Preliminary Conclusions

Non-interactive computing with encrypted functions is a challenge to cryptography. Although some theoretical results related to CEF were produced in the vicinity of computing with encrypted data, these findings seem to be impractical with respect to their computational feasibility as well as their interactiveness. Our approach of studying algebraic homomorphic encryption schemes (HES) yields a first and surprisingly simple scheme for CEF. Thus, by changing the conditions (looking at polynomials instead of Boolean circuits) we obtained a feasible and non-interactive solution. A more advanced scrutiny will have to show whether and under which conditions this scheme is secure. We should keep looking for other (and hopefully secure) additively homomorphic encryption schemes.

5 Offline Digital Signing in a Malicious Host

Can an agent carry and keep a secret? The answer is certainly yes as any secret data can be given to the agent as long as it remains encrypted. The important point is, however, that an agent would like to *use* the secret in public e.g., to compute the digital signature of an order form but without disclosing the secret needed to do so. In this section we introduce the concept of *undetachable digital signatures* and a possible realization of it that allows a mobile agent to effectively produce a digital signature inside a remote and possibly malicious host without the host being able to deduce the agent's secret or to reuse the signature routine for arbitrary documents.

5.1 Making Digital Signatures "Undetachable"

Let us assume for the moment that we have a way to conceal a function s that produces a digital signature e.g., using the CEF approach. The problem then is that even if the real signature routine could be kept secret, still the whole (encrypted but operational) routine might be abused to sign arbitrary documents which simply would make the signing process worthless. We therefore need a way to glue the signature routine to the function f that produces the output that is to be signed. How could a practically useful remotely signed document look like? The output producing routine f could for example add to each document a prefix saying that the following digitally signed order form is valid only for a single airline ticket issued for a specific date and costing less than a certain amount of dollars. Thus, our intention is to "cast" into the general purpose signing routine some task specific information which is enforced to be part of the signed document.

We give now an outline of the idea how one could sign the output of a function f securely, where f is given by a rational function. Let s be a rational function used by Alice to produce the digital signature $s(m)$ of an arbitrary message m. Furthermore we want the message m to be the result of a rational function f applied to some input data x. Finally we need a function v that Alice publishes in order to let others check the validity of the digital signature i.e., z is regarded to be a valid signature of m if and only if $v(z) = m$. For letting her mobile agent create "undetachable" signatures, Alice computes the dense representation of $f_{signed} := s \circ f$. She sends f and f_{signed} to Bob who evaluates both $f(x)$ and $z = f_{signed}(x)$.

A valid output of this algorithm is the pair $(f(x), z := f_{signed}(x))$. Applying the verification function v to z every user can check that the message $f(x)$ indeed is a valid output of the function f. If Bob wants to pretend that a message n is a valid output of Alices function f, he would have to construct $s(n)$. So the security of the method lies in Bobs inability to construct $s(n)$ for a given n.

Unfortunately there are at least four attacks against this scheme:

- *Left decomposition attack:*
  Given the rational functions $h := s \circ f$ and f determine s.
- *Interpolation attack I:*
  The function v is public. So an adversary can produce a list of pairs $(z, v(z))$. Because s is a rational function it is feasible to reconstruct s by interpolation

techniques. (Observe that s has to be a function of not too high degree, because else the size of $s \circ f$ in a dense representation will explode. So the described interpolation attack is in fact a realistic threat for the secrecy of s.)

- *Interpolation attack II:*
  An adversary can produce pairs $(l, s(l))$ where he obtains l by using the output function f. Again, a rational function s could be reconstructed.
- *Inversion attack:*
  If Bob is able to find a preimage x of n under f he can produce a valid signature z for n by computing $z := f_{signed}(x))$.

The *Interpolation attack I* applies to every signature scheme based on low degree rational functions used for the signing routine. Shamir, who also introduced the signature schemes where the signature routine is given by rational functions, had an interesting idea how to overcome this difficulty that we are also going to use below. To secure our scheme against these attacks we first switch to a multivariate context using the following notation:

(i) Let $s = (s_1, \ldots, s_k) : R^k \to R^k$ be a bijective function whose components are given by rational functions (R is a ring or a field). s is a so called birational map.
(ii) Let $v = (v_1, \ldots, v_k) : R^k \to R^k$ be the inverse function of s, i.e. $s \circ v = v \circ s = id_{R^k}$.
(iii) Let $f : R^l \to R^k$ be the function whose output Alice wants to be signed.
(iv) We further assume that rational functions $G_2, \ldots G_k$ are known to the public. (These functions can be used by many users like Alice to obtain signing routines for their mobile programs).

Now we are ready to describe how the modified scheme works:

Public key Alices public key for the signature verification is given by the function $v_2, \ldots v_k$. (Observe that Alice does not publish v_1).

Construction of the signed program Alice chooses a random rational function $r : R^l \to R$. She constructs the map $f_{signed} : R^l \to R^k$ with components given by $f_{signed,i} := s_i(r, G_2 \circ f, \ldots, G_k \circ f), 1 \leq i \leq k$. Alice computes the dense representation of f_{signed}. She sends the tupel of functions (f, f_{signed}) to Bob.

Execution of the signed program Bob computes $y := f(x)$ and $z := f_{signed}(x)$. (y, z) is then a signed output of the program.

Verification of the signature Compute $G_i(y), 2 \leq i \leq k$ and $v_i(z), 2 \leq i \leq k$. z is a signature of y if and only if $v_i(z) = G_i(y)$ for $2 \leq i \leq k$.

Let us briefly describe how these modifications yield a scheme that is strengthened against the attacks mentioned above. The key point is that an adversary does not know the functions r and v_1.

(i) Because an adversary does not know r the left decomposition attack to obtain s_i from the i'th component of f_{signed} has become even harder.
(ii) Because an adversary does not know v_1 he can not compute input/output pairs for the interpolation of the s_i.

(iii) Because an adversary does not know r he can not compute input/output pairs for the interpolation of the s_i as described in the second interpolation attack.

(iv) Even if an adversary is able to invert f the scheme is not broken. Because an adversary does not know r he does not know how to compute preimages. (Note that already to invert the rational function f an adversary has to solve a multivariate system of algebraic equations. This problem is known to be very hard. So an adversary will not be able to invert f except for some very simple choices of f.)

Ways to construct birational functions s that are easy to invert have been described by Shamir in [13]. However, the schemes proposed there have been successfully attacked by Coppersmith, Stern and Vaudenay [3]. We expect that other ways to construct secure birational maps can be found e.g., based on concepts from Algebraic Geometry where birational maps are extensively studied.

6 Conclusions

Although we do not claim to have a general solution for the main problem of mobile agent protection, we produced evidences that code can at least partially be protected against a malicious host. We identified a special class of functions – polynomials and rational functions – together with encryption schemes that lead to a first non-trivial example of cryptographically hiding a function such that it can nevertheless be executed with a non-interactive protocol. We also described an approach to hide a signing function and to make digital signatures "undetachable" which prevents that the signing procedure can be abused for signing arbitrary documents. While a more thorough analysis of the security properties of our schemes remains to be done, we hope that they represent a first step towards fully software based mobile code protection. We expect interesting and surprising results from a field that might be called "mobile cryptography".

References

1. M. Abadi and J. Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(1):1–12, 1990.
2. D. Chess, B. Grosof, C. Harrison, D. Levine, and C. Parris. Itinerant agents for mobile computing. Technical Report RC 20010, IBM, March 1995.
3. Don Coppersmith, Jacques Stern, and Serge Vaudenay. Attacks on the birational permutation signature schemes. In Douglas R. Stinson, editor, *Proceedings of CRYPTO'93*, number 773 in LNCS, pages 435–443, 1993.
4. Dave Dyer. Java decompilers compared. http://www.javaworld.com/javaworld/jw-07-1997/jw-07-decompilers.html, June 1997.
5. J. Feigenbaum and M. Merritt. Open questions, talk abstracts, and summary of discussions. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 2:1–45, 1991.
6. Josep Domingo i Ferrer. A new privacy homomorphism and applications. *Information Processing Letters*, 60:277–282, 1996.
7. Don Libes. *Obfuscated C and other mysteries*. Wiley, 1993.
8. Richard Lipton and Tomas Sander. An additively homomorphic encyrption scheme or how to introduce a partial trapdoor in the discrete log, November 1997. In preparation.

9. Catherine Meadows. Detecting attacks on mobile agents. In *Proceedings of the DARPA workshop on foundations for secure mobile code, Monterey CA, USA*, March 1997.
10. David Naccache and Jacques Stern. A new public-key cryptosystem. In *Advances in Cryptology - EUROCRYPT'97*, LNCS, pages 27–36, 1997.
11. Franco P. Preparata. Generation of near-optimal universal boolean functions. *Journal of Computer and System Sciences*, 4:93–102, 1970.
12. Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 169–179. Academic Press, 1978.
13. Adi Shamir. Efficient signature schemes based on birational permutations. In Douglas R. Stinson, editor, *Proceedings of CRYPTO'93*, number 773 in LNCS, pages 1–12, 1993.
14. K. B. Sriram. Hashjava - a java applet obfuscator. http://www.sbktech.org/hashjava.html, July 1997.
15. Giovanni Vigna. Protecting mobile agents through tracing. In *Proceedings of the Third ECOOP Workshop on Mobile Object Systems, Jyväskylä Finnland*, June 1997.
16. Joachim von zur Gathen and Gadiel Seroussi. Boolean circuits versus arithmetic circuits. *Information and Computation*, 91:142–154, 1991.
17. Ingo Wegener. *The Complexity of Boolean Functions*. Eiley-Teubner, 1987.
18. Bennet S. Yee. A sanctuary for mobile agents. In *Proceedings of the DARPA workshop on foundations for secure mobile code, Monterey CA, USA*, March 1997.
19. Richard E. Zippel. Rational function decomposition. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 1–6. ACM Press, July 1991.