INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Towards Mobile Cryptography

Tomas Sander and Christian F. Tschudin
{sander,tschudin}@icsi.berkeley.edu

TR-97-049

Nov. 22, 1997

Abstract

Mobile code technology has become a driving force for recent advances in distributed systems. The concept of mobility of executable code raises major security problems. In this paper we deal with the protection of mobile code from possibly malicious hosts. We conceptualize on the specific cryptographic problems posed by mobile code. We are able to provide a solution for some of these problems: We present techniques how to achieve "non–interactive computing with encrypted programs" in certain cases and give a complete solution for this problem in important instances. We further present a way how an agent might securely perform a cryptographic primitive, digital signing, in an untrusted execution environment. Our results are based on the use of homomorphic encryption schemes and function composition techniques.

1 Introduction

The security of the execution environment is a basic cornerstone of cryptographic systems: the parties which perform a cryptographic protocol require a trusted computing base where they can securely store data, can process them and can carry out the protocol. For example, it is not adviced to compute an RSA signature on a foreign computer: even if the private key is encrypted before transfer, it has to be decrypted and the remote computer may spy at it during the program's execution. We suggest that it is possible to discharge the requirement of a trusted computing base such that cryptographic primitives can be realized on untrusted hardware and data can be stored and processed in arbitrary places. This finding relies on the ability to ship programs between computers i.e., requires a mobile code infrastructure.

Mobile code technology has become a driving force for recent advances in distributed systems: **Java applets** are small mobile code programs that can be downloaded into a WEB browser (active content). **Mobile agent systems** allow programs to independently roam the net for resource discovery: they are studied for military applications as well as for civil E-commerce transactions. **Active networks** enable the user to reprogram the communication network at a packet level. In all these mobile code applications there is the inherent danger that code is either not duly executed or data carried within mobile programs are disclosed and are tampered with.

There exists a folklore in the mobile code research community saying that mobile code can not be effectively protected against the executing system because the host has full access to the program's code and data. This seems to imply that mobile code requires strong guarantees on the trustworthiness of the executing computer, thus mobile code technology would underline the necessity of a trusted computing base. Contrarily we believe that the opposite is true: mobile code holds the key to uncouple the secure execution of programs from the trustworthiness of the underlying execution support. The central argument is that we can obtain a system where a host can *execute an encrypted function* without having to decrypt it. Thus, functions would be encrypted such that the resulting transformation can be implemented as a (mobile) program that will be executed on a remote host. The executing computer will see the program's cleartext instructions but will not be able to understand the function that the program implements. The analogue at the communication level is a ciphermessage that should be freely forwardable without jeopardizing the content's privacy. Having function and execution privacy immediately yields execution integrity: an adversary can not modify a program in a goal-oriented way. Modifying single bits of the encrypted program would disturb its correct execution, but similar to changing single bits in a ciphermessage it is very hard to produce a desired outcome.

1.1 Our Results and Outline of the Paper

In section 2 we analyze security aspects of mobility from a cryptographic point of view. We briefly review basic concepts of mobile code, point to its potential applications and discuss some security threats for mobile code. In section 3 we further conceptualize on the specific cryptographic problems posed by mobile code and present prototypical mobile code problems that require cryptographic solutions. In section 4 we show briefly that existing approaches for code privacy are not satisfactory for mobile code. We then present a protocol that realizes mobile code privacy for programs that compute polynomials over rings and give an exact analysis of its information leakage. The protocol relies on the use of a public key encryption scheme that has certain additional properties. Lipton and Sander made recently an important contribution [6] in succeeding to construct such a scheme for rings of type $\mathbb{Z}/N\mathbb{Z}$, N smooth. Thus for this case we can present a first complete solution for the "non–interactive evaluation of encrypted functions". In section 5 we show how to attack the difficult problem of mobile agents who want to remotely sign their output. This approach is based on composition techniques for rational functions.

2 Mobile Code

Mobile code is regarded as an important new networking technology which, however, suffers from considerable security problems. Because it is based on the execution of mobile programs on remote and possibly untrusted computers, many observers question its fitness e.g., for E-commerce. In this section we briefly describe the mobile code concept for discussing related security threats. This also enables to identify the specific constraints that mobile code imposes on cryptographic solutions.

2.1 Applications

Making code mobile means that complete programs or program fragments can be exchanged between computers. The heterogeneity of execution support is hidden by a common language in which the mobile programs are expressed. Java, for example, is a mobile code system: Web browsers can download *Java applets* from a server – these applets will be executed inside the brower's Java virtual machine and may display fancy graphics or implement highly responsive interactive applications. PostScript is another example for a portability technology that is used for typesetting by explicitly programming the printer. Based on such a technology it is possible to create distributed environments where programs can autonomously move from one computer to the other: unlike Java applets which are downloaded at a user's request, *mobile software agents* decide for themselves when and where to go. The computations themselves have become mobile. Various mobile agent systems have been built and are subject to ongoing research, each targeting at another application field (see e.g., [10, 15]). In the following we describe three of them.

Mobile Computing: Computing with mobile devices poses problems to networking because the changing physical location requires a continuous reconfiguration of the data links. If connectivity can not always be maintained it also requires applications to handle extended off-line periods. Mobile software agents are very useful in this context because they can encapsulate longlasting transactions. They carry a request (e.g., browse through a user's main mailbox) to a server, cause its execution and bring back the result as soon as connectivity is reestablished. Because mobile agents can also preprocess the result (e.g., extract only the required text sections), they make better usage of the usually slow communication link between the mobile device and the network.

Information retrieval is also the basic scenario for military applications where mobile agents would roam the slow battle field network for finding suspicious incidents that endanger a large scale operation [3]. Furthermore, mobile agents can be used to deploy new information or software in an autonomous way. The detached nature of a mobile agent's processing makes them more robust against partial network failures (which was one of the basic requirements in the development of the ARPA-net that eventually became the Internet).

Active Networks: Current computer networks are based on the exchange of passive data packets. In active networks, each data packet is replaced by an active mobile code packet that carries data but also the instructions telling a network node how to process this packet. Although it induces some processing overhead, this instructional mode of communications gives the most freedom for realizing customized communication architectures [13].

Several experimental systems are under construction (MIT, University of Pennsylvania, etc), many of them under a special DARPA research program [12]. A primary goal of active networks is to shift information processing tasks from the end nodes to internal nodes. Information fusion inside the network reduces latency and bandwidth. Performing information diffusion inside the network also saves bandwidth (e.g., data filtering) and enables customizable data delivery protocols.

E-Commerce: Mobile agents are an enabling technology for the automatization of electronic commerce. Beyond simple information gathering tasks, mobile agents can take over all tasks of commercial transactions, namely price negotiation, contract signing and delivery of (electronic) goods and services. A typical scenario is a "stock watch agent" that is programmed by its owner to continuously observe selected stocks and to immediately react on changing situations. In another case, a mobile shopping agent is sent to the servers of various airlines for checking the availability and finding out the best price for a given itinerary and date. Both scenarios heavily rely on the flexibility provided by mobile code technology: They take place in an open and competitive market environment where new products, new providers and new customers can join the system at any time. Turning the network into a huge market place is also a research direction that reaches deep into active networks where means have to be found to allocate the network's limited resources like bandwidth, CPU time and memory.

2.2 Security Problems

The most evident security concern for mobile software agents is host security: hosts must be protected from the effect of foreign code which in many respect resembles network worms or computer viruses. The other concern is the security of the mobile agents themselves: the mobile agent's code and data is at the full mercy of the executing host.

So far little research was done on protecting a mobile agent from malicious hosts: the main focus was on making the execution of mobile code efficient and safe for the host. One reason for this disequilibrium may be the folklore saying that mobile code is impossible to protect without resort to special hardware, simply because the code has to be executed by the hosting system. Intuitively this makes sense because the following points certainly apply:

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

This list also points to the problems of using mobile code in security sensitive applications. In the case of E-commerce, for example, a shopping agent could be "brainwashed" by a malicious server so it forgets the best prices collected to far. Furthermore, it would be unwise to let the mobile agent digitally sign an order form because this implies that the agent carries the user's private key. Spying at the agent's data also permits stealing attacks either on electronic money or on other electronic credentials (passwords, capabilities). Similar threats with perhaps even more catastrophic consequences can be produced for active networks and battlefield agents too. Thus, the challenge for cryptography is to find answers to the following problems:

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

This list could easily be extended, for example by examining the problems of secure routing, or studying denial of service attacks. What seems important to us, as we will argue in the following section, is to understand that protecting a mobile agent against malicious hosts is not a "nice-to-have" feature but is essential for an agent system's usefulness.

2.3 Constraints on Cryptographic Solutions

The obvious approach of solving some of the problems of mobile agent protection is to setup restricted (and thus more controllable) mobile code environments, to deploy cryptographic protocols between the mobile agent and a cryptographically "safe haven", and to frustrate potential adversaries by making mobile agent tampering difficult. However, these "standard" approaches are insufficient because they are either too restrictive or too unreliable.

Network of Untrusted Nodes: Creating a network of mutually trusted nodes alleviates many mobile agent protection problems: users can trust the executing computers not to tamper with their agents. However, this approach disputes the goal of mobile agents which aim at an open and evolvable system: a new node can only be integrated into the network if all other nodes trust the newcomer. Furthermore, this approach is problematic because some trusted nodes might still misbehave. In the latter case solutions have to be found that allow to detect and prove which node misbehaved. However, the costs to do so may be higher than the damage that has to be recovered. This also applies to other mechanisms like "social control" where a server can be banned from further business if too much agents detect a misbehavior. Cryptographic solutions should presume a network of untrusted nodes.

Non-Interactive Protocols: Ideally a security solution for mobile agents does not rely on a interactive protocol between the agent and its originating site: otherwise truly detached operations become difficult and result in a quite limited form of task delegation. In the example of the shopping agent, the user Alice would like to go off-line instead of keeping in touch with the shopping mobile agent she sent off. Cryptographic solutions therefore should conceive protocols requiring minimal interaction between an originator and its mobile agents.

Provable Mobile Code Security: We expect that mobile code based applications will not be used in security sensitive fields if provably security can not be provided. Making tampering of agents just "difficult" seems to be too vague of a solution. Consider, for example, E-commerce transactions: Because an agent's buying actions should become legally binding operations, customers as well as providers require security guarantees. The protection offered to mobile agents must therefore be provable and truly cryptographic by linking the difficulty to alter an agent or spy it out to mathematically hard problems.

We are convinced that mobile code can not exploit its potential if networks are closed, mobile agents require heavy interactions, and if there are only weak protection guarantees. Cryptographic solutions for the mobile agent protection problem therefore are subject to the following constraints:

1. mobile agents should be allowed to execute in untrusted hosts but still have guarantees for their correct execution.

2. mobile agents should not require interactive protocols with their originator.

3. protection mechanisms should be provably secure.

The twist that shows ways to protect mobile agents according to these requirements is to move away from the assumption that a mobile agent consists of cleartext code and data. 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. This leads to the concept of mobile cryptography.

3 Mobile Cryptography

Performing cryptographic protocols requires that critical elements remain protected and are processed inside a secure execution environment. It is crucial that the key used for generating signatures is kept secret: A company would not sign contracts on the computer system of its strongest competitor. The secure execution environment usually includes the computing base that provides the physical storage and the execution support. Using a graphical notation similar to [9], the left side of figure 1 shows the involved elements, the boundary denotes the secure execution environment.

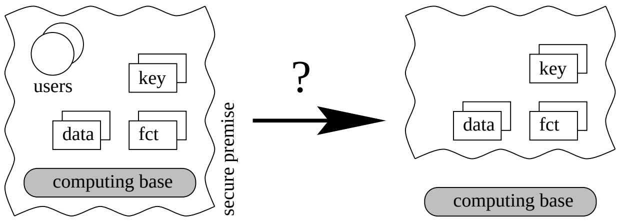

Figure 1: Making the secure premise independent of the execution support.

The trustworthiness of the computing base is crucial. Even in secured multiparty computations it is assumed that the participating parties are able to perform their computations on their own trusted platform. The question then is: Can a secure premise be thought of that does *not* include the trusted computing base? This would mean that all interactions and computations inside the secure execution environment are carried out in an unsecured space (right side of figure 1).

By varying a definition of general cryptography given in [7], we may define mobile cryptography as "the study of mathematical techniques related to aspects of information security of mobile executable code in a network". The distinctive aspect of mobile cryptography is that the border conditions for realizing cryptographic protocols are different. These are namely the constraints we identified in section 2.3 which are the assumption on an untrusted computing base and the requirement of non-interactive protocols.

3.1 The Linking Problem

Mobile code is about function mobility, so a first task is to secure the outsourcing of function evaluation. We argued that programs should not be exported in cleartext. This identifies an important challenge for mobile agent security for which we present in section 4 a solution: *How can we compute with encrypted mobile programs?* However, securing single functions is not sufficient, as we show with a small example on implementing a digital signing primitive for mobile agents.

Let us assume for the moment that we have a way to conceal a function s that produces a digital signature. The problem is that even if the real signature routine can 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. In real life a human user decides which documents to sign i.e., which secret key, which signing function and which document are and should be linked together. Outside the trusted execution environment, a mobile agent has to assure this linkage by itself. The general problem thus is that not only the building blocks – keys, data, functions, are in the open but also the way they interact. Therefor it is not sufficient to secure these items independently: their interactions have to be protected in a holistic way.

This observation outrules a sequential programming approach to let an agent sign some program P's output with a signing routine S:

Execute $y = P(x)$;
Compute the signature $z = S(y)$;
Output the pair (y, z);

An adversary just separates the signing routine S and signs arbitrary documents. This leads us to the *linking problem*:

> How can cryptographic primitives be unremovably attached to the data to which the are supposed to be applied?

Using the signing example we sketch a possible solutions for this problem that is explained in more detail in section 5.2. The general idea: Assume that the output of the function f is supposed to be signed with a signature generating function s. Then Alice computes their composition $h := s \circ f$ on her trusted platform and creates a mobile agent with the following structure (an uppercase letter denotes the program that realizes the lowercase function):

Execute $y = F(x)$;
Compute the signature $z = H(x)$;
Output the pair (y, z);

Crucial for the security of this scheme is the difficulty of an adversary to recover s i.e., the difficulty of decomposing the final function h into its elements s and f. At a more general level, the formidable question is whether it is possible to hide secrets in software and that one can enable this software to do useful things with the secrets in the public without requiring a hardware based "safe haven".

3.2 Other Aspects Relevant to Mobile Cryptography

There are many other interesting and relevant aspects of mobile code security we have not even mentioned here but that should also be treated as part of mobile cryptography. Further information on such topics can be found e.g., in [4, 14].

- The whole area of *protecting the executing host* from malicious actions of mobile code.
- The *protection of the network* as a whole (e.g., spamming agents or hosts).
- The *secure routing* of mobile code.
- The *detection of tampering* by and the *identification* of a malicious host.
- The protection of mobile code against *input/output analysis.*

Furthermore, the potential of techniques from distributed cryptography, secure multiparty computations and secret sharing schemes have not been exploited yet to protect an agent's mission (in contrast to the protection of an individual agent).

4 Computing with Encrypted Functions

In this section we present a protocol that allows non–interactive evaluation of encrypted polynomials over rings $\mathbb{Z}/N\mathbb{Z}$. Considering only polynomial/rational functions may be a restriction. This is motivated by the requirement for rigorous security proofs. So it will be interesting to extend these results to algebraic circuits and finally to Boolean circuits. We regard our results as a first important step in this direction. If one succeeds with non–interactive evaluation of encrypted functions (EEF) for Boolean circuits the problem of EEF for general programs would – at least theoretically – be solved. Note for this that every Turing machine program can be efficiently simulated by Boolean circuits [16]

4.1 Non–Interactive Evaluation of Encrypted Functions

The ability to hide a function f inside an executable program is of central importance for mobile agent protection because the obtained code privacy immediately yields code integrity as explained earlier. For this we have to encrypt a function such that its encrypted form remains executable. The general problem of *non–interactive evaluation of encrypted functions* (EEF) can be described as follows:

> Alice (the originating host) has an algorithm to compute a function f. Bob (the remote host) 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 simple protocol for EEF is shown in figure 2 and looks like this:

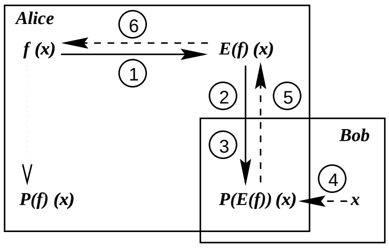

Figure 2: Computing with encrypted functions.

(1) Alice encrypts f.

(2) Alice creates a program $P(E(f))$ which implements $E(f)$.

(3) Alice sends $P(E(f))$ to Bob.

(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 $f(x)$.

This protocol has no further interactions than the exchange of the program and the resulting value at the end, so it has an *optimal number of communication rounds* wherefore we call this protocol non–interactive.

The unspecified word "substantial" used in the problem description depends on the application. Further remarks on the amount of information leaked by letting Bob execute the encrypted function f are given in section 4.3.1.

4.2 EEF via Homomorphic Functions

Subsequently we show how homomorphic encryption schemes enable the realization of EEF. We start by discussing related work on the problem of processing encrypted data (PED).

Abadi and Feigenbaum [1] described a protocol how to securely evaluate a Boolean circuit in encrypted data. They further reduced EEF (for Boolean circuits) to PED by representing the Boolean circuit that is to be hidden as data fed to a universal Boolean circuit. For mobile code there are two problems with their solution. First the use of universal Boolean circuits per se is unfeasible as they have exponential size. Even after repairing this by using cleverly chosen partial universal Boolean circuits (i.e., Boolean circuits that are universal for a certain subclass of the set of Boolean functions) their protocol for EEF requires a high amount of communication between Alice and Bob and thus violates our non–interactiveness requirement.

Also motivated by the problem of processing encrypted data, Rivest et al. [9] asked in 1978 for encryption schemes having certain additional "homomorphic" properties. In 1991 Feigenbaum and Merritt asked more specifically [5]:

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

It is easy to see that an encryption function having these properties allows to evaluate polynomial expressions in the encrypted data without revealing the input and the result. However, an encryption function that supports both addition and multiplication on the ciphervalues has not been found till today and the general problem of evaluating non–interactively a polynomial for encrypted data is still open.

An important observation we make is that for computing with encrypted polynomials it is not necessary to have both the additive and multiplicative property of an encryption function: it is sufficient that the encryption supports addition and "mixed multiplication". These notions are explained in the following

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,*
- *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.*

Using an additive and mixed multiplicative encryption function we can give a protocol for solving EEF for encrypted polynomials:

Proposition 2 *Let $E : R \to S$ be an additively and mixed multiplicatively homomorphic encryption scheme. Then we can implement non–interactive EEF for polynomials $p \in R[X_1, \ldots, X_s]$ with E.*

Proof: Let p be the polynomial $\sum a_{i_1 \ldots i_s} X_1^{i_1} \ldots X_s^{i_s}$

(1) Alice creates a program $P(X)$ that implements p in the following way:

- each coefficient $a_{i_1 \ldots i_s}$ of p is replaced by $E(a_{i_1 \ldots i_s})$,
- the monomials of p are evaluated on the input $x_1, \ldots, x_s$ and stored in a list $L := [\ldots, (x_1^{i_1} \ldots x_s^{i_s}), \ldots]$,
- the list $M := [\ldots, E(a_{i_1 \ldots i_s} x_1^{i_1} \ldots x_s^{i_s}), \ldots]$ is produced by calling MIXED-MULT for the elements of L and the coefficients $E(a_{i_1 \ldots i_s})$
- 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 $x_1, \ldots, x_s$ and obtains $P(x_1, \ldots, x_s)$

(5) Bob sends the result $P(x_1, \ldots, x_s) = E(p(x))$ back to Alice.

(6) Alice decrypts the result by applying E^{-1} and obtains $p(x)$.

$\blacksquare$

4.3 $\mathbb{Z}/N\mathbb{Z}$ Cryptosystems

For the important class of rings $\mathbb{Z}/N\mathbb{Z}$ we can weaken the requirements on the encryption scheme even further:

Corollary 3 *Let $E : \mathbb{Z}/N\mathbb{Z} \to R$ be an additively homomorphic encryption scheme. Then we can implement non–interactive EEF for polynomials $p \in \mathbb{Z}/N\mathbb{Z}[X_1, \ldots, X_s]$ with E.*

Proof: We claim that every additively homomorphic encryption scheme on $\mathbb{Z}/N\mathbb{Z}$ is also mixed-multiplicative. For this 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 $y_i \neq 0$. Now apply proposition 2. Observe that the algorithm Mixed-Mult makes at most $\mathcal{O}(\log N)^2$ calls to Plus. $\blacksquare$

Until recently there were no additively homomorphic encryption schemes on rings $\mathbb{Z}/N\mathbb{Z}$ known. In [6] however, Lipton and Sander describe an additive homomorphic scheme on rings $\mathbb{Z}/N\mathbb{Z}$ for smooth integers N (a positive integer is called smooth if it has only small prime factors). This makes it possible to implement non–interactive EEF for polynomials in $\mathbb{Z}/N\mathbb{Z}[X_1, \ldots, X_s]$ whose security is discussed in the following paragraphs.

4.3.1 Information Leakage

The proposed protocol leaks information about the coefficients of the unencrypted polynomials. Depending on the application, this leakage may be unacceptable. In this section we analyze the amount of information that is leaked and study the security of the protocol.

$E(f)$ can be easily reconstructed from the program P. So a set of monomials $\mathcal{U}$ containing the monomials with non–zero coefficients of f is inherently revealed by the protocol. We call such a set $\mathcal{U}$ a *skeleton* of f. If we think of the monomials as given in a certain order we may identify f with the list of coefficients of f that we call m. In the same way we can identify $E(f)$ with its list of coefficients that we call $E(m)$. So the message spaces M_n consist of n-tuples of elements from $\mathbb{Z}/N\mathbb{Z}$ on which the encryption function E operates componentwise. The statement that our protocol does not leak any information about f except its skeleton is now equivalent to the fact that our encryption function does not leak any information when applied to elements of the message spaces. Encryption schemes that have this strong property are called polynomial time indistinguishable (see [8] for further details). With this background we can summarize the results as follows:

Theorem 4 *Let E be an additively homomorphic encryption scheme on $\mathbb{Z}/N\mathbb{Z}$. Then the protocol of proposition 2 realizes non–interactive EEF for polynomials $f \in \mathbb{Z}/N\mathbb{Z}[X_1, \ldots, X_s]$. Assume further that the used encryption scheme is polynomial time indistinguishable. Then no information is leaked about f except it's skeleton.*

The additive scheme of Lipton and Sander [6] is polynomial time indistinguishable under the (reasonable) assumption of the hardness of the Power Residue Hypothesis (which is a generalization of the well-known Quadratic Residue Hypothesis to residues of higher degree). So the problem of non–interactive EEF for polynomials in $\mathbb{Z}/N\mathbb{Z}[X_1, \ldots, X_s]$, N smooth, is completely solved if one allows the leakage of the skeleton.

4.3.2 Limitations

There exist applications where the information leakage of the skeleton – even for a polynomial time indistinguishable encryption scheme E – may allow to recover the full polynomial f. Assume an adversary knows that the encrypted function f is an RSA–function, $x \mapsto x^d$ and which therefore has only a single non-zero coefficient. Assume that input/output pairs $(x, f(x))$ are known. Then it is feasible to find f and consequently d by evaluating every element of $\mathcal{U}$ on the inputs x.

5 Composition Based Approaches

We believe that the technique of composing functions will play a primer role in securing mobile code by linking data and functions in an undetachable way (c.f. section 3). In this section we first sketch an application of the composition idea to EEF and then show how they might be used for obtaining "undetachable digital signatures".

5.1 EEF via Composition Techniques

The composition of functions is another way of realizing the evaluation of encrypted functions. A simple example gives the idea: 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, computes $B := SA$ and 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.

To increase the security of this simple scheme and to catch a broader class of functions we generalize the matrix example. Assume f is a rational function (the quotient of two polynomials) and s is also a rational function such that Alice is able to invert s efficiently. Let $E(f) := s \circ f$. Then Alice and Bob can proceed as in the protocol just described. The security of this method is based on the difficulty of decomposing $E(f)$, i.e., the difficulty of reconstructing f from $E(f)$:

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

Interestingly enough there are already results on the hardness of decomposing rational functions. Decomposition problems have been studied in various forms, but no polynomial time algorithm for decomposing multivariate rational functions is known (see [17] for further references).

Shamir [11] has described ways to obtain *birational permutation schemes* $s : K^n \to K^n$, where the components are given by rational functions and such that s is easy to invert. It is easy to transform the above problem with one function f into this setting. As we deal with such a transformation in the following section we omit here further details.

5.2 Undetachable Signatures

In section 3 we pointed out that it is not sufficient to protect e.g., a signing function as an independent building block. What we need is a way to glue the signature routine s to the function f that produces the output to be signed.

We give now the outline of an idea how one can 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 Alice's function f, he would have to construct $s(n)$. So the security of the method lies in Bob's 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 low degree, because else the size of $s \circ f$ in a dense representation will explode. So the 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, i.e., $f(x) = n$, 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:

1. 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. (To be more precise, s is defined as $s : U \to R^k$ where U is a Zariski–open subset of R^k).

2. Let $v = (v_1, \ldots, v_k) : R^k \to R^k$ be the inverse function of s, i.e., $s \circ v = v \circ s = id_{R^k}$.

3. Let $f : R^l \to R^t$ be the function whose output Alice wants to be signed.

4. We further assume that rational functions $G_2, \ldots G_k : R^t \to R$ 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 Alice's 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 tuple 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.

1. 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.

2. Because an adversary does not know v_1 he can not compute input/output pairs for the interpolation of the s_i.

3. 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.

4. 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 of $(r, G_2 \circ f, \ldots, G_k \circ f) : R^l \to R^k$. (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 the second part of [11]. However, the schemes resulting from those constructions have been successfully attacked by Coppersmith, Stern and Vaudenay [2]. So there is a need to find new constructions for secure birational functions to put our ideas to work. 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.

A detailed analysis of the security of the proposed scheme remains to be given.

References

[1] M. Abadi and J. Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(1):1–12, 1990.

[2] 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.

[3] DARPA "Agent Based Systems" Program, September 1997. http://abs.wwwhome.com/.

[4] Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code, Monterey CA, USA, March 1997. http://www.cs.nps.navy.mil/research/languages/wkshp.html.

[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] Richard Lipton and Tomas Sander. An additively homomorphic encryption scheme or how to introduce a partial trapdoor in the discrete log, November 1997. Submitted for publication.

[7] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[8] S. Micali, C. Rackoff, and B. Sloan. The notion of security for probabilistic cryptosystems. *SIAM Journal on Computing*, 17(2):412–426, 1988.

[9] 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.

[10] K. Rothermel and R. Popescu-Zeletin, editors. *Mobile Agents'97 - Proceedings of the 1st International Workshop*. LNCS 1219. Springer, April 1997.

[11] 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.

[12] D. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35:80–86, January 1997.

[13] Christian F. Tschudin. *On the Structuring of Computer Communications*. PhD thesis No. 2632, Université de Genève, 1993. http://cui.unige.ch/pub/tschudin.

[14] Giovanni Vigna, editor. *Mobile Agents and Security*. LNCS. Springer, 1998. Forthcoming.

[15] J. Vitek and Chr. Tschudin, editors. *Mobile Object Systems – Towards the Programmable Internet*. LNCS 1222. Springer, April 1997.

[16] Ingo Wegener. *The Complexity of Boolean Functions*. Eiley-Teubner, 1987.

[17] Richard E. Zippel. Rational function decomposition. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 1–6. ACM Press, July 1991.