Mobile Code Protection with Smartcards

Position Paper

Sergio Loureiro[1], Refik Molva
Institut Eurecom, Sophia Antipolis - France
{loureiro, molva}@eurecom.fr

Abstract: This paper addresses mobile code security with respect to potential integrity and privacy violations originating from the runtime environment. The suggested solution requires a trusted hardware with limited capacity like a smartcard and assures the security of a program executed on untrusted runtime environments by means of some interactions between the program and the trusted hardware. The security of this scheme is based on an extension of function hiding using error correcting codes. Unlike prior function hiding schemes, the proposed technique allows multi-step execution and the delivery of cleartext output at the remote site.

Keywords: mobile code security, integrity, privacy, cryptography.

1 Introduction

With the advent of new computing paradigms like mobile code and ubiquitous computing, the privacy and integrity of software programs become a major concern beyond classical data security considerations. Running a program in a potentially hostile environment may raise various security requirements, as follows:

- a company might need to prevent the disclosure of certain sensitive algorithms implemented in its software products despite extensive code analysis and reverse engineering by potential intruders including its customers;

- a mobile software agent acting on behalf of a person might need to assure the integrity of some critical operation performed on an untrusted remote host;

- a data collection agent might need to assure both the confidentiality and the integrity of the results computed at various competing sites.

Security with mobile code has long been viewed only as the problem of protecting local resources against potential misuse and denial of service by the mobile code. This problem received much attention from software manufacturers as well as researchers, resulting in various solutions included in recent releases of products like JAVA.

1. Supported by the Portuguese FCT grant: PRAXIS XXI/BD/13875/97

Conversely, research on mobile code protection against potential attacks from the runtime environment could not come up with practical solutions yet, hence the limited if non-existing coverage of the problem by the industry. Assuring mobile code's security in an untrusted execution environment without tamper proof hardware has been conjectured as impossible in the literature [CHK97]; a breakthrough was achieved by [ST98a] and later by [LM99a] in terms of a "pure algorithmic" solution without recourse to tamper proof hardware, but unfortunately the results were not suitable for practical applications.

This paper presents a solution for the integrity and privacy of mobile code execution. Privacy of execution aims at preventing the disclosure of a program's semantics during its execution in a potentially hostile runtime environment. Integrity of execution assures that a program executed in a potentially hostile environment actually complies with its original semantics. A solution for the verification of integrity of execution may consist of verifying that the results of an execution on a potentially hostile environment actually belong to the set of possible results of the original mobile program.

The suggested solution requires a trusted hardware with limited capacity like a smartcard and assures the security of a program executed on untrusted runtime environments by means of some interactions between the program and the trusted hardware. The security of this scheme is based on an extension of function hiding using error correcting codes [LM99a]. Unlike prior function hiding schemes, the proposed technique allows multi-step execution and the delivery of cleartext output at the remote site.

The existing approaches to the problem of privacy of execution are referred in section two. In section three the problem of integrity of execution is discussed. The focus of these two sections is mainly to identify the limitations of the existing approaches and to justify the need for integrated approaches. Section four is dedicated to the definition of the model and discusses the conditions and benefits of such a model. In section five our original solution is described and in section six an analysis of the protocol is provided.

2 Privacy of Execution

One of the main advantages of mobile code is its ability of executing tasks locally in an autonomous way. Autonomy means that the code is able to perform its tasks with no interaction with the originator of the code. Therefore, this section focuses on approaches that accomplish the requirement of privacy of execution in a non-interactive way.

Sander and Tschudin [ST98b], [ST98a] defined a function hiding scheme based on a non-interactive protocol as depicted in Figure 1. This protocol is non-interactive in so far as the interactions between the owner of the function (Alice) and the remote party that evaluates the function (Bob), consist only of the transmission of the function by Alice to Bob and the transmission of the result back to Alice by Bob.

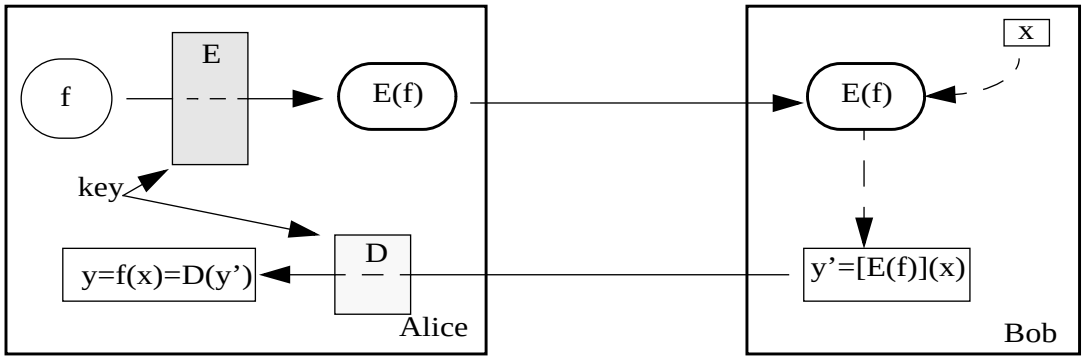

FIGURE 1. Non-interactive protocol for privacy of execution.

In a non-interactive protocol, a function *f* owned by Alice is evaluated by Bob on the input data *x* (provided by Bob), while preventing the disclosure of *f* to Bob. The privacy of *f* is assured by the transformation *E* that satisfies the following properties:

- it is infeasible under the intractability of a computational problem to derive *f* from *E(f)*;

- the cleartext result *f(x)* can be derived from the encrypted result *[E(f)](x)* in polynomial time using a secret trapdoor (algorithm *D*).

Sander and Tschudin [ST98b] illustrated the autonomous protocol concept with a method that allows to encrypt polynomials, based on the Goldwasser Micali [GM84] encryption scheme. Later, Loureiro and Molva [LM99a] proposed an efficient method that allows to encrypt combinational boolean circuits with several inputs and outputs, based on the composition of functions. Briefly, the novelty of the approach consisted of exploring the properties of functions used on coding theory. The technique described in the present paper is an extension of this work.

The models where the solutions for privacy of execution apply are very limited. Furthermore, the result is only meaningful for the originator of the code (Alice), therefore not allowing others to use this result in subsequent computations. One can see this limitation as the possibility of encrypting combinational but not sequential circuits. Therefore, the solutions are limited to scenarios of one time executions. The limitation comes from the fact that the originator cannot give the decryption algorithm to the remote host without compromising the privacy of the function. Another limitation is the absence of integrity of execution, which relies on the trust assumption that if the host cannot understand the code, then it cannot tamper with it in a meaningful way.

3 Integrity of Execution

The aim of integrity of execution is to provide the possibility to the code owner to verify the correctness of execution of his code. This problem has been extensively studied for achieving reliability (see for example [BW97] for a survey) but security requirements taking into

account possible malicious behaviour from the execution environment were not considered. Once more, we focus on non-interactive solutions.

Vigna in [Vig97] proposed a natural way of achieving integrity of execution where the originator receives and checks the trace of execution of the code. Due to the fact that the traces can be very cumbersome in terms of communication complexity the author suggested the use of hashes of the trace in order to oblige the remote host (Bob) to commit to the trace. The overall trace will be sent if the originator is suspicious about the behaviour of the remote host.

Yee [Yee97] suggested the use of proof based techniques. The host has to forward a proof of the correctness of the execution with the result. Complexity theory shows how to build proofs for NP-languages and recently how to build Probabilistic Checkable Proofs (PCP) [ALM+91] [AS98]. PCP proofs assure the correctness of a statement while checking only a subset of the proof. However, the subset has to be randomly determined by the checker, so the problem of using PCP proofs in our non interactive scenario is that the prover has to commit to the overall PCP proof (this proof is bigger than a non PCP proof).

In [BMW98], the authors presented an interesting model for mobile computing and a solution that overcomes the problem of using PCP proofs. The agent is modelled as a probabilistic Turing machine, and the set of all possible states of this machine constitutes an NP language, so there is a verification process for the membership of the language, that is, it is possible to check if an obtained state belongs to the language. To avoid the transmission of the overall PCP proof of the specified language, the randomly chosen queries from the checker are encrypted using non interactive Private Information Retrieval (PIR) techniques (see for example [CMS99]).

Integrity of execution only ensures that the obtained result is a possible output of the function, however the remote host is able to identify which are the possible outputs. Therefore it does not prevent re-execution of the code and selection of the best result or the reverse engineering of the code.

4 Model

To achieve mobile code protection, it is fundamental to tackle both privacy and integrity of execution. The originality of the solution presented here is the delegation of the originator's functionalities to tamper proof hardware (TPH) available on the remote host. We are thinking on smartcards when we refer to tamper proof hardware, even if there is extensive work that has questioned this definition.

The existence of tamper proof hardware on the host acting on behalf of the code's originator, even if limited in terms of storage and computational power, allows to retrieve the cleartext result at the remote host, therefore extending the model where the solution can be applied (without interaction with the code owner). Even if the new solution is based on the original technique described in [LM99a], it does not consist of a straightforward substitution of the code owner of the original scheme with a trusted party located in the remote site. The limitations in terms of storage and computational power are taken into account, where before this

was not an important requirement for the code owner. Furthermore, integrity of execution is achieved in a completely original way.

The code is modelled as a set of combinational circuits F_i as depicted in the left side of figure 2. The computation of each individual circuit F_i depends on a set of inputs $\{x_i\}$ received from the host and a set of outputs $\{y_i\}$ received from the circuits that were previously evaluated. As in the original scheme of [LM99a], the goal of the transformation E is to achieve the privacy of each circuit F_i. For each output of the transformed circuit F'_i the tamper proof hardware should be able to check the integrity of the result and to get the cleartext result y_{i+1} in an efficient way. On the other hand, the verification process should be efficient and less complex than recalculating the function itself.

The integrity verification process uses data that has to be transmitted over a private channel to the TPH. This data can be seen as some random queries related to the transformation applied to each circuit. For the sake of simplicity, the figure shows a sequential set of circuits, but the technique can be applied to parallel executions or circuits executed on other execution environments. The processes of integrity verification and retrieving the cleartext result do not depend on the sequence of execution of the circuits. The TPH has to handle the different sets of random queries and to relate these sets to the corresponding circuits.

5 Solution

Each combinational circuit is converted into a matrix format as described in [LM99b] (an example is presented in the appendix). Each resulting matrix is then encoded through the algorithm E. The result can be seen as another circuit even if referred as a matrix during the description. Starting from a set of equations, our technique creates a set of dependent equations using an error correcting code (namely a Goppa code). This transformation is hidden by a permutation and an addition with an error matrix. This process is similar to the construction of the public key in cryptosystems based on coding theory and its security relies heavily on the underlying class of codes. More information about the security of this kind of construction can be found in the appendix.

We will simplify the description by using binary matrices, because it is suitable for representing boolean functions or circuits, but the technique can be used for matrices with elements from Z_q. This allows the representation of algebraic circuits and implies the use of q-ary Goppa codes which were also proven to be secure [JM96].

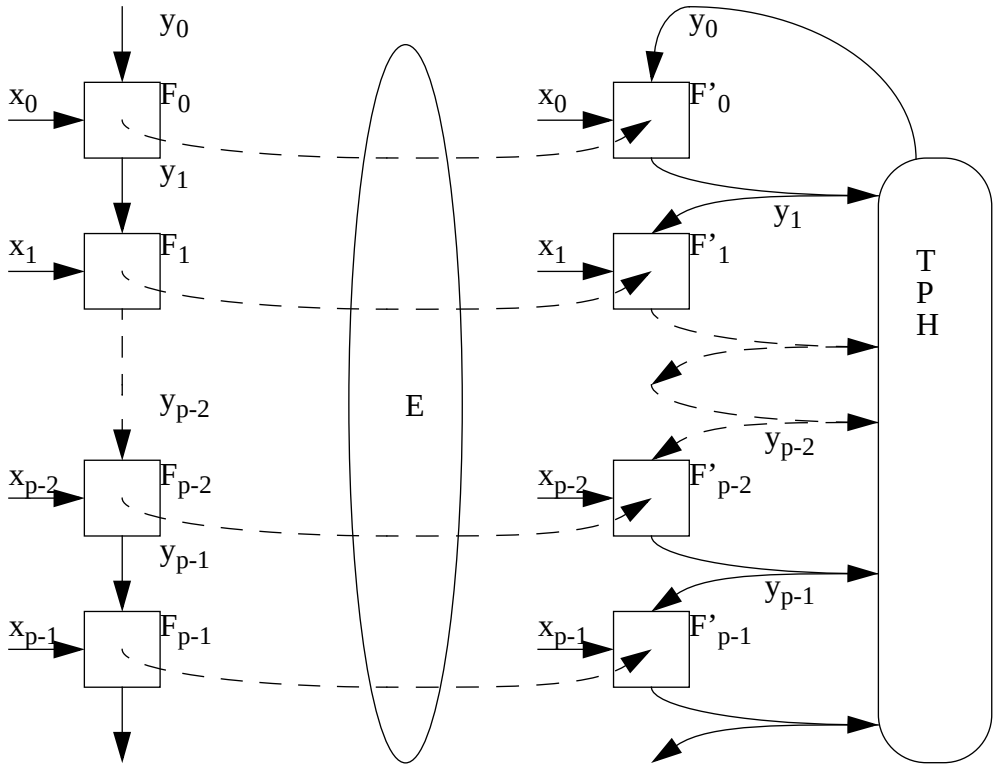

FIGURE 2. Transformation performed by algorithm E and role of the TPH

5.1 Algorithm E

Let E_i be an $k \times n$ random matrix satisfying $(n-k) \geq w(E_i) \geq t$ where *w(E$_i$)* represents the number of non null columns of the matrix *E*. *G, P,* and E_i are kept secret by Alice. Let F_i be an $l \times k$ matrix over Z_2 representing a boolean circuit. Alice computes the matrix F'_i by multiplying and adding the matrices referred over Z_2 as follows:

$$F'_i = F_i GP + E_i, 0 \leq i \leq p-1$$

and sends $\{F'_i | 0 \leq i \leq p-1\}$ to Bob.

Over a private channel, Alice transmits the error matrices $\{E_i | 0 \leq i \leq p-1\}$ to the smart-card.

These matrices can consist of only *t* non-null columns and their positions so the communication complexity is low. We assumed that every individual circuit has the same number of inputs and outputs for the sake of simplicity. If not, it is possible to add bogus equations and

variables (a subset of the variables is given by the smartcard so there is no transformation on the inputs given by Bob).

5.2 Remote Circuit Evaluation

Bob evaluates the circuit F'_i on the inputs $(x_i|y_i) \in (Z_2)^l$ and sends back the result y'_i and the input data x_i to the smartcard (| means concatenation). Compared with the case without privacy, there is no modification concerning the inputs x_i given by the host (Bob). There is an increase on the number of outputs relatively to the original circuit F_i.

5.3 Integrity Verification

The smartcard has to know the code C and the permutation P. The smartcard starts by computing consecutively $e_i = (x_i|y_i)E_i$, $y_a = y'_i + e_i$ and $y_b = y_a P^{-1}$. The integrity check consists of evaluating the syndrome of y_b using the secret code C, which is an easy operation in terms of computational complexity. The syndrome should be zero, or stated in a different way y_b should be a codeword of the code C.

5.4 Retrieving Cleartext Result

Decoding a word of null syndrome is equivalent to the problem of solving a set of linear equations, which is much easier in terms of computational complexity. Basically, an inversion of a $k \times k$ matrix is needed to retrieve the cleartext result.

5.5 Example

For the sake of simplicity, only one circuit is considered. Let F_i be a circuit with 524 boolean inputs and 524 boolean outputs. Using a Goppa code [1024, 524, 101] (same code as proposed by McEliece in his original scheme), the resultant circuit will be of size 524 inputs and 1024 outputs (evaluated on the host) and the checking circuit (error matrix E) evaluated on the smartcard will be of size 524 inputs and 50 outputs.

The information rate of a linear code is given by the ratio k/n. The function expansion as well as the increase in the computational complexity at the remote host (Bob) are inversely proportional to the information rate of the code.

6 Discussion

6.1 Privacy of Execution

The security evaluation of the privacy property is described in [LM99a] and a sketch of the proof is presented in the appendix. Nevertheless, the security evaluation was done for the case of only one circuit and no result given back, so we will focus on the impact of these modifications in an informal way:

-re-using the same code with several circuits does not significantly impact the security of the overall scheme since the number of combinations between all possible circuits, codes, permutations and error matrices is still very high.

-the fact that the result is given back is much more crucial. With sufficient cleartext pairs of inputs/outputs the remote host is able to interpolate the circuit. Supposing that *l*>*k*, at most *l* independent inputs /outputs pairs would be sufficient for completely determining an $l \times k$ circuit.

-moreover, the cleartext result gives information about the code *GP*, which has to be kept secret because it is invertible. However, the use of different error matrices prevents the disclosure of the code.

6.2 Integrity of Execution

The integrity verification (IV) process can be stated using the same terminology as [BK89] as follows:

if $x'F' \neq xF'$ then $\quad P(IV(x', xF') = Accept) < \delta \quad$ (1)

In words, the integrity of execution relies on the difficult of creating valid pairs *(x', xF')*.

Proof: The IV accepts if y_b is a codeword. This implies that y_a should also be a codeword of the permuted code *GP*. The decomposition of y_a yields:

$y_a = y' + x'E$

We have $w(x'E) \leq t$. In order for y_a to be a codeword, *y'* has to be at distance *x'E* of a codeword of the code *GP*. If the host randomly picks a word then the probability is the inverse of the number of codewords (2^{-k}).

A better attack is to pick an output of *F'*. The attack would take advantage of the probability that a valid output *xF'*, along with an input $x' \neq x$ be accepted by the integrity verification process as a valid pair. Therefore, the statement (1) can be reduced to:

$P(xE = x'E) < \delta$ because y_a=*xFGP*, y_a is an output of *F'*.

Realizing that *E* does a random mapping between a space of size 2^k and a space 2^t, this probability is approximately equal to 2^{-t} (we considered *E* of rank *t*) that is negligible with a reasonable code size (*t*=50 as considered before).

However, some of the inputs are provided by the smartcard. So, the verification consists of:

If $y' + (x'|y)E = codeword$ *then accept*

As mentioned before, the best attack is to choose an output of the circuit $y' = (x_a|x_b)F'$. So, statement (1) is equivalent to:

$$P[(x_aE_a|x_bE_b) = (x'E_a|yE_b)] = P(x_aE_a = x'E_a) \cdot P(x_bE_b = yE_b) < \delta$$

Both probabilities are equal to 2^{-t} but the attacker can choose not to cheat on one of them, therefore raising δ to 2^{-t}.

6.3 Related Work

The use of tamper-proof hardware aiming at mobile code protection was already proposed in [WSB98]. However, the approach suggested in that article requires entire code fragments to be executed in the trusted environment. In addition, providing privacy with that approach requires all the code to be transmitted over a private channel. In our case, only a small part piece of code used in the verification process has to be transmitted over a private channel and executed in the tamper proof hardware, while achieving the same security properties as [WSB98], privacy and integrity of execution. Therefore, our solution is more efficient in that the capacity of the smartcard does not depend directly on the size and complexity of the mobile code. Hence, our scheme would be more scalable to large mobile programs.

6.4 Applications

With a secure execution environment, built from a tamper proof hardware, several scenarios can be addressed:

- A mobile user wanting to download mobile code into untrusted public terminals and being able to execute it on a secure way. In this case the tamper proof hardware acts on behalf of the mobile user.

- An organisation (or a group of organisations) willing to implement remote services based on mobile code running in untrusted terminals. So, the organisation will provide the tamper proof hardware and only the terminals possessing it would have access to the services available.

- Copyright protection, in a way that the smartcard acts like an access control agent to protect software or electronic documents distributed over the network. The tamper proof hardware is a fundamental tool to access the data intellectually protected.

7 Conclusion

Our solution provides a fundamental building block for secure computing in potentially hostile environments. The presented solution applies to a more general model of computation when compared with previous solutions for privacy of execution, and tackles the problem of integrity of execution. Integrity of execution assumes an important role but seems hard to achieve without privacy, due to the re-execution and reverse engineering attacks. As opposed to empirical approaches like code obfuscation, the security of our scheme is quantifiable. Further work will focus on the in-depth security evaluation of the scheme and on the extension of the solution to more flexible models of computation.

Acknowledgements

The authors would like to thank the reviewers for their helpful comments and Nicolas Sendrier for many interesting discussions.

References

[ALM+91] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Proc. 33rd IEEE Foundations of Computer Science*, pages 14–23, October 1991.

[AM87] Carlisle M. Adams and Henk Meijer. Security-related comments regarding McEliece's public-key cryptosystem. In Carl Pomerance, editor, *Advances in Cryptology—CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 224–228. Springer-Verlag, 1988, 16–20 August 1987.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM*, 45(1):70–122, 1998.

[BK89] Manuel Blum and Sampath Kannan. Designing programs that check their work. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 86–97, Seattle, Washington, 15–17 May 1989.

[BMW98] Ingrid Biehl, Bernd Meyer, and Susanne Wetzel. Ensuring the integrity of agent-based computations by short proofs. In Kurt Rothermel and Fritz Hohl, editors, *Proc. of the Second International Workshop, Mobile Agents 98*, pages 183–194, 1998. Springer-Verlag Lecture Notes in Computer Science No. 1477.

[Bri84] Ernest F. Brickell. Breaking iterated knapsacks. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 342–358. Springer-Verlag, 1985, 19–22 August 1984.

[BW97] Manuel Blum and Hal Wasserman. Software reliability via run-time result-checking.

Journal of the ACM, 44(6):826–849, November 1997.

[Can96] Anne Canteaut. *Attaques de Cryptosystemes a Mots de Poids Faible et Construction de Functions t-Resilientes*. PhD thesis, L'Universite Paris VI, Octobre 1996.

[CHK97] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile agents: Are they a good idea? – update. In *Mobile Object Systems: Towards the Programmable Internet*, pages 46–48. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology: EUROCRYPT '99*, Lecture Notes in Computer Science. Springer, 1999.

[CS98] A. Canteaut and N. Sendrier. Cryptanalysis of the original McEliece cryptosystem. In *In Advances in Cryptology - ASIACRYPT'98, Lecture Notes in Computer Science. Springer-Verlag*, 1998.

[Gib91] J. K. Gibson. Equivalent Goppa codes and trapdoors to McEliece's public key cryptosystem. In D. W. Davies, editor, *Advances in Cryptology—EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pages 517–521. Springer-Verlag, 8–11 April 1991.

[Gib95] J. K. Gibson. Severely denting the Gabidulin version of the McEliece public key cryptosystem. *Designs, Codes and Cryptography*, 6(1):37–45, July 1995.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[Hei87] R. Heiman. On the security of cryptosystems based on error correcting codes. Master's thesis, Feinberg Graduate School of the Weizman Institute of Science, 1987.

[JM96] Heeralal Janwa and Oscar Moreno. McEliece public key cryptosystems using algebraic-geometric codes. *Designs, Codes and Cryptography*, 8(3):293–307, June 1996.

[LM99a] Sergio Loureiro and Refik Molva. Function hiding based on error correcting codes. In Manuel Blum and C. H. Lee, editors, *Proceedings of Cryptec'99 - International Workshop on Cryptographic Techniques and Electronic Commerce*, pages 92–98. City University of Hong-Kong, July 1999.

[LM99b] Sergio Loureiro and Refik Molva. Privacy for mobile code. In *Proceedings of the Distributed Object Security Workshop - OOPSLA'99*, pages 37–42, Denver, November 1999.

[Loi98] P. Loidreau. Some weak keys in McEliece public-key cryptosystem. In *IEEE International Symposium on Information Theory, ISIT'98, Boston, USA*, 1998.

[McE78] R. McEliece. A public-key cryptosystem based on algebraic coding theory. In *Jet Propulsion Lab. DSN Progress Report*, 1978.

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. In *Probl. Contr. and Information Theory*, 1986.

[Sen94] Nicolas Sendrier. On the structure of a randomly permuted concatenated code. In *EUROCODE 94*, pages 169–173, Abbaye de la BussiËre sur Ouche, France, October 1994.

[SS92] V. Sidelnikov and S. Shestakov. On cryptosystems based on generalized reed-solomon codes. *Diskret. Mat.*, 4:57–63, 1992.

[ST98a] Tomas Sander and Christian Tschudin. On software protection via function hiding. In *Proceedings of the Second Workshop on Information Hiding*, Portland, Oregon, USA,

April 1998.

[ST98b] Tomas Sander and Christian Tschudin. Towards mobile cryptography. In *Proceeding of the 1998 IEEE Symposium on Security and Privacy*, Oakland, California, May 1998.

[Til88] H.C.A. van Tilborg. *An Introduction to Cryptology*. Kluwer Academic Publishers, Boston, 1988.

[Vig97] Giovanni Vigna. Protecting mobile agents through tracing. In *3rd ECOOP Workshop on Mobile Object Systems*, Jyvälskylä, Finland, June 1997.

[WSB98] Uwe G. Wilhelm, Sebastian Staamann, and Levente Butty·n. Protecting the itinerary of mobile agents. In *4th ECOOP Workshop on Mobility: Secure Internet Mobile Computations*, 1998.

[Yee97] Bennet Yee. A sanctuary for mobile agents. Technical Report CS97-537, UC at San Diego, Dept. of Computer Science and Engineering, apr 1997.

Appendix

A Circuits to Matrices

We illustrated the concept of non-interactive secure computing with a method to evaluate a combinatorial circuit with several inputs and outputs on remote data, without disclosing the circuit. The objective is to embed the representation of the circuit into a matrix and to give the inputs to the circuit in a vector format. Here the description of the representation of a circuit into a matrix is given:

A circuit can be seen as a number *k* of Boolean functions on *m* inputs. Consider *k* Boolean functions B_k: $\{0,1\}^m \rightarrow \{0,1\}$ (or a boolean circuit with *m* inputs and *k* outputs). This circuit can be represented by a universal set of *k* boolean equations with $l = 2^m$ terms or *l-1* additions.

Example: For a circuit with 3 outputs and 2 outputs:

$$y_0 = f_{00} \cdot x_0 \cdot x_1 + f_{10} \cdot x_1 + f_{20} \cdot x_0 + f_{30}$$

$$y_1 = f_{01} \cdot x_0 \cdot x_1 + f_{11} \cdot x_1 + f_{21} \cdot x_0 + f_{31}$$

$$y_2 = f_{02} \cdot x_0 \cdot x_1 + f_{12} \cdot x_1 + f_{22} \cdot x_0 + f_{32}$$

This gives the F matrix with l rows and k columns: $\begin{bmatrix} f_{00} & f_{01} & f_{02} \\ f_{10} & f_{11} & f_{12} \\ f_{20} & f_{21} & f_{22} \\ f_{30} & f_{31} & f_{32} \end{bmatrix}$.

This matrix F is transformed according to the operation E previously defined. After this transformation it is possible to perform the inverse operation and to get a homomorphic circuit which has the same number of inputs.

B Proof of Privacy

Claim: It is infeasible to retrieve the private function F from the encrypted function F'.

There are two possible attacks: enumeration attack and trapdoor attack. We will show that our scheme is resilient to these attacks. The construction of the F' matrix is similar to the construction of the public-key of the McEliece [McE78] and Niederreiter [Nie86] cryptosystems, therefore the proof is highly inspired on the cryptoanalysis of these two systems. We assume F as a square matrix $k \times k$, to simplify.

B.1 Enumeration attack

The complexity of the enumeration or brute force attack can be measured by searching exhaustively all combinations of possible permutations ($n!$), Goppa codes ($\sim 2^{mt}/\ t$, with $m=\log n$), circuits ($2^{k\wedge 2}$) and error matrices. Using the parameters proposed by McEliece ([1024, 524, 50]), this attack is obviously infeasible [Til88]. In order to be resilient to this attack the class of codes used must be large enough to avoid any enumeration, which happens with Goppa codes.

B.2 Trapdoor Attack

This attack consists of retrieving the structure of code from the generator or parity-check matrix of the permuted code. In other words, the attack tries to identify an invariant that is not influenced by the homomorphism applied to the original code. These are the best known attacks against coding theory based cryptosystems, consisting of the exploitation of the properties of linear codes to find a trapdoor, usually called a Brickell-like attack [Bri84].

The resilience to this attack is highly dependent on the class of codes used. The initial proposal from Niederreiter used concatenated codes, which were proven to be insecure [Sen94]. Reed-Solomon codes were also proven to be insecure [SS92]. Goppa codes generated by a Goppa polynomial which has binary coefficients are also insecure [Loi98].

Heiman [Hei87] was the first to study the trapdoor attack and showed that the random matrix *S* (replaced by the matrix *F* in our case) used in the original McEliece scheme serves no security purpose concerning the protection of the code, because it does not change the codewords of the original code. However, Canteaut [Can96] showed that the matrix *S* serves the purpose of hiding the systematic structure of the Goppa code therefore having an important security role. Adams and Meijer [AM87] showed that the likelihood of finding a trapdoor for Goppa codes is small and that there is usually only one trapdoor. Later, [Gib91] challenged this result and proved that each permutation applied to the Goppa codes can be regarded as a possible trapdoor and there are at least *m.n.(n-1)* trapdoors. This results from the fact that not equivalent Goppa polynomials can generate equivalent codes. However, this number is still very small when compared with the *n!* possible trapdoors. The number of trapdoors is still open, but calculated lower bounds [Gib95] showed that an exhaustive search remains infeasible.

Recently, there were efforts to find an efficient algorithm to retrieve the characteristic parameters of the code from a permuted code represented by the generator or parity check matrix, with techniques that try to identify invariants among the classes of codes [Sen94]. However, the results were negative relatively to Goppa Codes. Therefore, there is no efficient algorithm to retrieve the characteristic parameters from a permuted generator matrix for Goppa Codes [CS98] and we can conclude that for sufficiently large codes (*n>1024*, the original parameters from McEliece scheme) the composition scheme is secure. The resilience to the trapdoor attack is further enhanced by the error matrices.

B.3 Information Leakage

The transformation *E* does not hide everything about the function *F. F'* gives some information about the rank of the original function *F*, concretely:

$$k \geq rank(F) + rank(E) \geq rank(F') \geq rank(F).$$

For example *rank(F)=k* implies that *rank(F')=k*. This fact will simplify the enumeration of all possible circuits to circuits of a given rank, which is not significant because there are still many possible combinations.