Guaranteed Slowdown, Generalized Encryption Scheme, and Function Sharing

Yury Lifshits*

July 10, 2005

Abstract

The goal of the paper is to construct mathematical abstractions of different aspects of real life software protection. We introduce three following notions: program slowdown, generalized encryption scheme and function sharing. These schemes allowed to discover new applications of such known ideas as trap-door functions and self-correcting programs.

1 Introduction

Software protection is very active research area now. It involves both practical approaches and theoretical investigation. In contrast to cryptography there is a lack of strict models for almost all aspects of program protection. In this paper we consider three notions motivated various intuitive protection ideas. We give a series of definitions examples and open questions around them.

There are a lot of practical tools, but the level of protection provided by them is unclear. In many cases we require an evidence that program is protected. So the main problem is to construct a *proof* of security. As soon as you start think about proof you immediately ask for formal model and definition of security.

In 2001 the general formal model for program protection (namely, *black-box security*) [2] was presented. However there are only few positive results

*This work is supported by grants NSh-2203.2003.1 and INTAS-???

[14, 9] in this framework. An informal conclusion of this study is that threats are different and in each case you need specific protection. Hence we ask for new formal models describing program protection and attacks on programs.

Our new approaches are the following.

First idea is *guaranteed slowdown* scheme. Guaranteed slowdown is a slow version of some algorithm which is difficult to speed up to the original level. One application is slowdown of encryption algorithms in public key cryptosystems. It helps against brute force attacks. In this work we introduce two examples of slowdown transformation. We show that idea of guaranteed slowdown is closely related to trap-door functions. The most interesting open question is how to get a formal proof that slowdown is difficult to reverse.

Second protection method we study is *generalized encryption scheme* (GES). This notion was already presented in work of Abadi et al. [1]. The problem is the following: how to use untrusted computational resource with information-theoretic security of your own data? Here we present one new example of such a scheme and discuss utilization of this notion in software protection. The main result of Abadi et al. [1] is that GES is impossible for NP-complete problems. Still it is very perspective to find such schemes for polynomial tasks.

Third notion is *function sharing*. This is a natural generalization of secret sharing schemes. The question is how to divide a computational task among several parties in the way such that subgroups can't say anything about the original task. This scheme are interesting for protecting mobile agents from malicious hosts [6, 8] development and as a basic block for new obfuscating transformations [5]. We present here a scheme for function sharing in the most simple model. For several additional restrictions existence of such schemes remains an open question.

Next three sections devoted to these models. For each of them we introduce some motivation, several examples, then formalization and conclude with related work and open questions.

2 Guaranteed slowdown

Informally, guaranteed slowdown is a pair of programs P_1 and P_2 with three following properties: P_1 and P_2 are functionally equivalent, P_1 is faster than P_2, and there is some "evidence" that given only source code of P_2 to construct a functionally equivalent program comparable (in speed) with P_1 is

difficult.

The task of slowdown seems quite unusual so we need to explain our motivation. There are four points here:

A Author of some program can distribute it in the slow version in order to protect his copyright. To prove the authorship of his code he (and only he) can show speedup version of the program.

B We now explain how slowdown of encryption algorithm might be useful against brute force attacks on public key cryptosystem. Let Bob is sending message to Alice. Then to obtain the maximum level of security they choose the largest length of the key such that decryption and encryption algorithms are still feasible. Consider the case when decryption (using this long key) requires full resources from Alice while Bob has some reserve of computational power during encryption. In such circumstances instead of public key we can distribute slowdown version of encryption algorithm. Thus Bob still can encrypt what he want but any brute force attack that uses a lot of encryptions becomes harder.

C It is popular to bound some functionalities in the trial versions of software. We can apply guaranteed slowdown scheme to obtain provable crack resistance for trial version. If we use slow version as a trial one, then by our informal definition recovering of original speed for the program is difficult.

D We can divide all attacks on a program in the two classes: understanding (gaining knowledge) and modification. When we distribute a slow version program a potential attack is a speedup transformation. Hence guaranteed slowdown scheme is a partial case of modification protection. Therefore there is a hope to extend some ideas of slowdown scheme to protection against other modification threats.

We start with two examples of slowdown.

Example 1. Consider a function $f(x) = x^a \mod N$, where N is a production of two prime numbers as in RSA cryptosystem. We can slow down computation of f in the following way. We keep a in secret but announce $b = a + k\varphi(N)$ for some integer k instead. By Euler's theorem $x^a \equiv x^b(\mod N)$. To proof that $g(x) = x^b \mod N$ is guaranteed slowdown for f we

should check to things: a) g is slower than f and b) g is difficult to speed up. We have only informal evidence for this statements. Firstly, $b > a$ and, moreover, we are free to choose k as large as we want; it seems very natural that raising to the larger power requires more computation. Secondly, up to now there is no known polynomial algorithms to compute $\varphi(N)$ so we believe that given b to compute lesser degree c such that for any x the equality $x^c \equiv x^b(\mod N)$ holds is also hard. However, we still need rigorous proofs here.

Example 2. Take any trap-door function f_k (see [7]). Knowing the secret key k there is a polynomial algorithm A to compute f_k^{-1}. As a slowdown version of A we can take the following algorithm A':

```
for y=1 to 2^n do
    if f_k(y)=x then return y;
```

Let us discuss the idea of guaranteed slowdown. The first question to answer is how to measure the speed of algorithm. On any particular input we can achieve even a constant time for computation. Hence for the guaranteed slowdown definition speed should be an integral characteristic. We use average speed but other approaches might be reliable as well.

Another point of discussion is whether to study slowdown for functions defined on finite or infinite domains? Below we choose first answer (hence we can speak about circuit slowdown). But slowdown for functions on infinite domains also might be investigated. So the average speed of finite function is just a number. If we will speak in terms of circuits than we would use circuit size as a measure for speed.

We now give a slightly more formal description of our model. A pair of algorithms A_f (fast), A_s (slow) is called a *guaranteed slowdown* for function f if both algorithms compute f, A_f works on time t and given only A_s it is "difficult" to construct an algorithm computing f faster than t_1.

How can we prove difficulty of speed up? To have a complexity bound we should specify a computational problem. That is a family of questions and answers. The above description is a *single slowdown scheme* but security proof might exist only for *family* of such schemes. The way out is the following. We specify a family of functions F. Security proof should be a statement like: given A_s for some $f \in F$ it is difficult to construct a (fast) algorithm computing f.

The notion of guaranteed slowdown is very close to the idea of trap-door function. The common point is that in both cases we have pair of algorithms and knowing only the public one it is difficult to get the secret algorithm. The difference is that we do not require dramatic difference between efficiency of algorithms in the pair. Another point is that in trap-door function scheme there are three algorithms (straightforward computation, obvious slow inversion and secret fast inversion) while in guaranteed slowdown there are only two.

What are further questions about guaranteed slowdown? It might be interesting to know:

- Whether it is possible to make any level slowdown? That is whether for any function s there is a function f that could be slowdown from t to $s(t)$?
- Whether it is possible to organize multilevel slowdown? We mean here a series of algorithms $A_1, \ldots, A_k$ computing f such that A_i is slower than A_{i+1} and knowing any A_i it is difficult to get an algorithm comparable with A_{i+1}.
- How to prove that speed up is difficult?
- Function that is nonzero only on one input value is called *point function*. It seems quite natural that such functions have slowdown schemes. The question is to make a full study of guaranteed slowdown schemes for point functions.

3 Generalized encryption schemes

In this section we investigate the following model [1]. There are two participants, Alice and Bob. Alice has some computational task, that is function $f \in F$ and input x, Bob has nothing. Alice want to get a result $f(x)$ faster than just doing all computation by herself. Alice can communicate with Bob and ask him to do some computation for her. We study only *semihonest* model here that is Bob send back correct results. The main restriction is information-theoretic security of x and f (Bob knows only that $f \in F$). In contrast to [1] below we also consider generalized encryption scheme with cryptographic security.

We present generalized encryption scheme for two functions. In these examples f is not a secret, but x is.

Example 3. [1] Discrete logarithm. Let prime number p be fixed and g be generator for Z_p^*. For every integer u such that $(u, p) = 1$ the value of discrete logarithm function $f(u)$ is the unique integer $e \in [1, p-1]$ for which $g^e \equiv u \mod p$. Generalized encryption scheme for discrete log is as follows. Alice send to Bob $u' = ug^r$ (for random r), Bob send back discrete log e' for u', and Alice get the answer $e = e' - r$. Value of ug^r is uniformly distributed on the set $[1, p-1]$, therefore Bob get no information about u. Discrete log in usual computation is believed to be a hard problem.

Example 4. Inverse operation in a group. Here we present a schema that could be applied for many functions. Let G be any group such that inversion operation as at least twice harder than multiplication. Then there is a fully encrypted computation scheme for inversion. Alice asks Bob to compute inversion of rx, receives $x^{-1}r^{-1}$ and multiply result by r. So Alice uses two multiplications instead of one inversion. Bob get no knowledge about x. Multiplication group of matrices over the finite field is a natural candidate for "inversion is twice harder than multiplication" property.

Use of auxiliary untrusted computational source arise in several practical areas. First is *market-based parallel computing*. That is we "encrypt" our hard computational tasks and (for money) ask outside computers to perform it for us. Second relevant framework is mobile agents technology. We send programs on others computers to make some task for us (usually depending on that host's inputs). Another property of mobile agents is their ability to change the host from time to time. Third application field is development of smart cards. Informally smart card sends some computational task to an auxiliary computer and tries not to reveal any essential information about computation. In all these cases we have two security aspects: to hide internal information from outside host and to check whether it's results are correct. In our study we deal only with first one.

We now recall the definition of generalized encryption scheme. It is a two party (Alice and Bob) protocol. Alice has a computational task that is $x \in X$ and $f \in F$. There are finite number of communication rounds between Alice and Bob. In each round i Alice compute her massage a_i based on her input, all previous Bob answers and random coins and send a_i to Bob. He compute an answer based on all massages received from Alice to the moment. After

the last round Alice compute an $f(x)$ based on all information she has. We have two requirements: efficiency and security. Efficiency means that total amount of computation performed by Alice should be less than that in the case of straightforward computation (without Bob). There are two different approaches two security. In the work [1] information-theoretic security is used. This means that distribution of Alice messages should be the same for all input values.

We now introduce cryptographic security for GES and discuss its relation *black-box security* [2]. If distribution of Alice massages for different pairs (x, f) are *computational indistinguishable* (see [7]), then we call GES to be cryptographically secure. Consider any family of functions F defined on a set of strings X. Let us fix some *pseudorandom function* G (again, see [7]). Then for any string r and any seed k we call

$$\tilde{f} = f(x \text{ XOR } r) + G_k(x)$$

to be a *prepared* form of f.

Proposition 1. *Suppose for every prepared form of every function from F there exists an obfuscation with black-box security. Then there is a GES for F with cryptographic security.*

Proof. Our construction is straightforward. We choose randomly r and k and send to Bob string (x XOR r) and obfuscated $\tilde{f}$. Bob send back $\tilde{f}(x \text{ XOR } r)$ and Alice subtract $G_k(x \text{ XOR } r)$ from it to obtain $f(x)$.

To get a security property we should combine two facts. First is that G_k is computationally indistinguishable with truly random function. Hence $\tilde{f}$ is also indistinguishable with random function. Second point is that it is computationally hard to get any knowledge besides input-output behavior about function obfuscated with black-box security. Thus our GES is cryptographically secure. □

We know [2] that black-box secure obfuscation not possible for every function. Still obfuscation even with smaller level of security leads to a potentially applicable generalized encryption schemes.

We now recall models that are the most relevant to generalized encryption scheme. *Secure function evaluation* is a common computational task for several parties each of which knows only part of data. The security requirement is not to reveal more information about input data than just output

value. *Encrypted computation* [12] is a two parties (Alice and Bob) task. Alice has a function, Bob has an input value, Alice sends her function in encrypted form to Bob, he performs computation on his value and returns an (encrypted) result back. Third problem is acceleration of raising to a power in RSA cryptosystem [3, 10], which is, in fact, a particular case of generalized encryption scheme with cryptographic security. Besides this models there is another relevant notion. As we see in examples, basic idea is to ask Bob to compute the same function but with different input. The same idea was used in *self-correctors* [4]. The difference in our works is that we ask that new input reveal no information about original one.

We conclude this section with two open problems:

- For which functions there exists a generalized encryption scheme with cryptographic security?
- For which computational tasks there is essential polynomial speed up with theoretic-informational security?

4 Function Sharing

Function sharing is a protocol for distributed computation satisfying some security requirements. There are three parties, say Alice, Bob, and Carl. Let family of functions F be fixed and known to everybody. We study the following process of computing a function $f \in F$. Alice, Bob and Carl have predefined secret functions A_f, B_f, and C_f, respectively. Input x goes to Alice and Bob, they compute their functions on it and send results $A_f(x)$ and $B_f(x)$ to Carl, Carl compute his function on these results and outputs the value of f. Thus we have the decomposition formula

$$f(x) \equiv C_f(A_f(x), B_f(x)).$$

The security requirement is that knowing only two of functions A_f, B_f, and C_f it is impossible to get any knowledge about f except that f belongs to F.

Example 5. Let F_N be the class of all functions f that there exists an algorithm Alg_f computing it and that length of description of Alg_f is less than N. Then there is a simple function sharing scheme for F_N. We already have a good developed theory of data secret sharing. Hence our idea is to

treat a function as a sort of data. So we write down a description string S of an algorithm computing f. Than we generate a random string R of the same length. We send R to left party, $S\text{XOR}R$ to the right one. They wouldn't compute anything and just resend x and their part of f coding to Third party. Third party recover S end evaluate a universal circuit (universal Turing machine) on S and x.

In program obfuscation [5] the idea of usual secret sharing is used in the following obfuscating transformation: boolean variable v is splitted into two x, y such that on any point in time $v = x\text{XOR}y$. In the same way any function sharing scheme will lead to a procedure splitting transformation. Notion of function sharing is applicable wherever program is decomposed into several parts. Now two such approaches are obfuscation on interpretation level [11] and dividing mobile agent into set of agentlets [6]. Notice that in all these applications function sharing will be useful mostly against static attacks.

It is a natural idea to decompose a secret into several parts such that knowing only some of them does not help to recover this secret. When the secret is a string of bits a lot of beautiful secret sharing schemes were constructed since the seminal paper [13] was published. Function sharing is an extension of that idea. There is another framework, namely minimal model for secure computation [**?**], close to function sharing. The common is three parties, result should be computed by third (Carl), inputs goes only to first two (Alice and Bob). But these models have a significant difference. Namely, in our case input circuit is a secret and data is not while in multiparty communication complexity circuit is public, but data is a secret.

Now we present two modifications of our main task. One additional requirement is to ask C_f be as simple as possible. Second is to restrict the size of output of A_f and B_f. More precisely an interesting questions are:

- For which function families it is possible to construct a function sharing scheme with C_f equal to XOR?
- For which function families it is possible to construct a function sharing scheme with output size of A_f and B_f at most constantly larger then output size of f?

5 Conclusion and future work

Theoretical research in the area of software protection goes in two ways: investigation of existed models and introduction of new ones. We present three frameworks: guaranteed slowdown, generalized encryption scheme and function sharing. These models allow us to connect problems of program protection with such good investigated notions as trap-door functions (for guaranteed slowdown) and random self-reducibility for (generalized encryption scheme).

A series of new theoretical problems naturally arise in our study. Probably the most interesting are to find a large class of functions having generalized encryption schemes and to construct a formal proof of irreversibility of slowdown transformation.

To define an adequate theoretical framework for software protection is not easy. The main difficulty is pure understanding of what are protected programs and what are threats. One of the unanswered question in modelling is: how to define what is known to adversary and what is a secret in a program. In cryptography there is a Kerckhoff law saying that an algorithm is public and a key is a secret. In practice of software protection real secrets have different nature.

The target of modelling is to get some theoretical result and bring it back to practice. Here we introduce several ideas with potential for applications, e.g. RSA encryption slowdown. Still for further theoretic investigation our formalizations should be more rigorous.

6 Acknowledgements

I thank Peter Holgerson for idea of program slowdown.

References

[1] Martín Abadi, Joan Feigenbaum, and Joe Kilian. On hiding information from an oracle. *J. Comput. Syst. Sci.*, 39(1):21–50, 1989.

[2] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO '01: Proceedings of the 21st Annual*

International Cryptology Conference on Advances in Cryptology, pages 1–18, London, UK, 2001. Springer-Verlag.

[3] Philippe Begiun and Jean-Jacques Quisquater. Fast server-aided rsa signatures secure against active attacks. In *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 57–69, London, UK, 1995. Springer-Verlag.

[4] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[5] Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical Report 148, Dept. Computer Science, University of Auckland, July 09 1997.

[6] L. D'Anna, B. Matt, A. Reisse, T. Van Vleck, S. Schwab, and P. LeBlanc. Self-protecting mobile agents obfuscation report. Technical Report 03-015, Network Associates Labs, June 2003.

[7] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.

[8] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. *Lecture Notes in Computer Science*, 1419:92–113, 1998.

[9] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT*, pages 20–39, 2004.

[10] Tsutomu Matsumoto, Koki Kato, and Hideki Imai. Speeding up secret computations with insecure auxiliary devices. In *CRYPTO '88: Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, pages 497–506, London, UK, 1990. Springer-Verlag.

[11] Akito Monden, Antoine Monsifrot, and Clark Thomborson. A framework for obfuscated interpretation. In *CRPIT '04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 7–16, Darlinghurst, Australia, 2004. Australian Computer Society, Inc.

[12] Tomas Sander and Christian F. Tschudin. On software protection via function hiding. In *Information Hiding*, pages 111–123, 1998.

[13] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[14] Nikolay P. Varnovsky and Vladimir A. Zakharov. On the possibility of provably secure obfuscating programs. In *Ershov Memorial Conference*, volume 2890 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 2003.