Advanced Topics in Cryptography

Lecture 1: Program Obfuscation

March 24, 2009

Lecturer: S. Goldwasser, M. Naor Scribe by: L. Kor, Y. Tzur Updated: April 17, 2009

1 Introduction

While protecting data is well-studied, few have considered to formalize the issue of protecting *programs*. One would often like to give the ability to execute a program without revealing any information about its implementation, other than its input/output behavior.

In practice, low-level code is often "obfuscated" in a way that makes it hard to analyze and reverse engineer. We take a complexity-theoretic approach and consider the feasibility of such generic processes: intuitively, an Obfuscator is a processor of programs (e.g. Turing Machines or boolean circuits) which outputs a program with the same functionality, but unintelligible code. In particular, an adversary possessing the obfuscated program should be able to learn nothing of the original program other than its functionality on evaluated inputs; I.e., we would like the obfuscated program to simulate a black-box access to the program.

The obfuscation paradigm was first extensively studied by [1], which also showed the impossibility of universal obfuscation.

2 Definition

Definition 1 *A probabilistic algorithm $\mathcal{O}$ is called a* TM/circuit Obfuscator *if the following three conditions hold:*

1. ***Functionality:*** *For every TM/circuit P and for every input x: $P(x) = (\mathcal{O}(P))(x)$.*
2. ***Polynomial Slowdown:*** *There exists a polynomial $q(\cdot)$ such that for every TM/circuit P, $|\mathcal{O}(P)| \leq q(|P|)$. For TM's, we also require that for every input x, if P halts after t steps on x then $\mathcal{O}(P)$ halts within $q(t)$ steps on x.*
3. ***Virtual Black-Box:*** *For any PPT A, there is PPT oracle machine S such that for all TM/circuit P:*

$$\left|\Pr[A(\mathcal{O}(P)) = 1] - \Pr[S^P(1^{|P|}) = 1]\right| < neg(|P|).^{1}$$

[1]In fact, for TM's the oracle provided to S is a time-restricted version of P: given t, x, return $P(x)$ if P halts within t steps on x, and $\perp$ otherwise.

Discussion

1. The Functionality requirement is sometimes relaxed to allow a probabilistic obfuscation: for every program P and input x, the probability over the coins of $\mathcal{O}$ that P and $\mathcal{O}(P)$ behave differently on x is very small. The paper also rules out such "approximate" obfuscators.
2. The Polynomial Slowdown requirement for TM's handles input-specific running time. The impossibility result actually uses this requirement in a way that does not work for other notions, such as a polynomial relation between the worst-case (or average-case) running time. This will be further discussed after the impossibility result is described.
3. An equivalent formulation of the Virtual Block-Box property that is sometimes more convenient, is: for every predicate π and for any PPT A, there is PPT oracle machine S such that for all TM/circuit P:
$$\left|\Pr[A(\mathcal{O}(P)) = \pi(P)] - \Pr[S^P(1^{|P|}) = \pi(P)]\right| < neg(|P|).$$
4. The obfuscator itself need not be efficient.

3 Applications

Other than the straightforward application to software protection, a universal Obfuscator would also have cryptographic and complexity-theoretic implications:

- The Random Oracle Model is an idealized cryptographic setting in which all parties have access to a common truly random function. A generic way of replacing the Random Oracle with a publicly known function would have vast implications. Theoretically, this can be done by obfuscating a family of pseudorandom functions.

  Open Problem 2 *Show that pseudorandom functions cannot be obfuscated.*

- While it is known that public-key encryption schemes are stronger than private-key schemes, the obfuscation of a private-key encryption algorithm would immediately yield a public-key encryption scheme: the obfuscation of the encryption algorithm with the hard-coded secret key can be used as the public encryption algorithm.

  Open Problem 3 *Can private-key encryption algorithms be obfuscated?*

4 Impossibility of Universal Obfuscation

We show that a universal Obfuscator cannot exist by providing a function ensemble $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ of efficiently computable functions that cannot be obfuscated. In particular, any Obfuscator would fail on a random function chosen from $\mathcal{H}_n$, for large enough n. In fact, the result we prove is strong in the sense that all Obfuscators must fail on the same specific adversary A (that does not depend on the Obfuscator). That is, we show a specific PPT A and a specific function z_n such that:

1. For any $f \in \text{support}(\mathcal{H}_n)$, and for any TM/circuit C, that computes f: $A(C) = 1$;
2. For any TM/circuit C that computes z_n, $A(C) = 0$;
3. No PPT oracle machine S can distinguish z_n from a random f chosen from $\mathcal{H}_n$:

$$\forall S: \quad \left| \Pr_{f \leftarrow \mathcal{H}_n}[S^f(1^n) = 1] - \Pr_{f \leftarrow \mathcal{H}_n}[S^{z_n}(1^n) = 1] \right| \leq neg(n)$$

The construction will consist of three steps: we first construct two functions C and D that are not obfuscatable together (i.e., when chosen together from a joint ensemble, and the adversary has access to both programs). Second, we will combine each C and D to a single function, giving an ensemble that is unobfuscatable for Turing Machines. Finally, we will show how to implement this for circuits, for which the same ideas cannot be applied.

4.1 Impossibility of Obfuscating Two Programs

For a given n, we choose two strings $\alpha, \beta \in \{0,1\}^n$ uniformly at random. We then define two programs:

$$C_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x = \alpha \\ 0^n & \text{otherwise} \end{cases}, \qquad D_{\alpha,\beta}(C') = \begin{cases} 1 & \text{if } C'(\alpha) = \beta \\ 0 & \text{otherwise} \end{cases}.$$

Note that both C and D can be easily implemented (either as TM's or circuits). We define the adversary A that given two programs C', D' simply outputs $D'(C')$ (see remark below). Note that for any $\alpha, \beta \in \{0,1\}^n$, $D_{\alpha,\beta}(C_{\alpha,\beta}) = 1$. Thus for any Obfuscator $\mathcal{O}$, we have $A(\mathcal{O}(C_{\alpha,\beta}), \mathcal{O}(D_{\alpha,\beta})) = 1$. On the other hand, for any PPT S with oracle access to $C_{\alpha,\beta}, D_{\alpha,\beta}$, where α, β are chosen at random *after* S, the probability one of the polynomially-many queries S issues to $C_{\alpha,\beta}$ will yield a nonzero output, is negligible. Thus, defining Z_n to always output 0^n, we get

$$\Pr_{\alpha,\beta}[S^{C_{\alpha,\beta}, D_{\alpha,\beta}}(1^n) = 1] \approx \Pr_{\alpha,\beta}[S^{Z_n, D_{\alpha,\beta}}(1^n) = 1],$$

while for any α and $\beta \neq 0$,

$$\Pr[A(\mathcal{O}(Z_n), \mathcal{O}(D_{\alpha,\beta}) = 1] = 0.$$

This means that if S simulates A for $\mathcal{O}(Z_n)$, then for random α, β, S fails to simulate A on $\mathcal{O}(C_{\alpha,\beta})$, giving that $\mathcal{O}$ does not satisfy the virtual black-box property.

Remark In the case of TM's, to make sure D always halts we should add a timeout mechanism with a polynomial that would certainly suffice for running any "valid" C' (that exists due to the polynomial slowdown property). Likewise, A would run D' using a timeout polynomial in $|C'|, |D'|$. In the case of circuits this is not needed, as the Polynomial Slowdown property simply allows D to be constructed with enough input gates, guaranteeing that a valid C' (e.g., $\mathcal{O}(C)$) is not too large.

4.2 Combining C and D to one TM

In the case of TM's, converting the above joint ensemble to a single function ensemble is straightforward: for $\alpha, \beta \in \{0,1\}^n$ define

$$F_{\alpha,\beta}(b,x) = \begin{cases} C(x) & b=0 \\ D(x) & b=1 \end{cases}.$$

The adversary A is then defined, on input F', to generate the code F'_0 of F' with its first input fixed to 0, and run $F'(1, F'_0)$. Clearly, oracle access to $F_{\alpha,\beta}$ is equivalent to oracle access to the previous $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$. By applying the arguments above we get that a universal Obfuscator for Turing Machines does not exist.

Remark This is where the input-specific running time notion of the Polynomial Slowdown property is important. Suppose we required a polynomial relation only between worst-case running time of $F_{\alpha,\beta}$ and $\mathcal{O}(F_{\alpha,\beta})$. The worst-case running time of F is at least the timeout D allows C (e.g., q^3 for q the polynomial bounding the slowdown), for input $b=1$ and a program $x = C'$ that never halts. Thus a legitimate obfuscator may have $\mathcal{O}(F)$ run for longer than this timeout for all inputs, even when $b=0$. This would cause $F'(1, F'_0)$ to always stop F'_0 prematurely. When the slowdown is defined per-input, this does not occur, as the C-part and D-part retain their separate running time bounds.

4.3 Combining C and D to one circuit

In the case of boolean circuits the above technique does not work per se, since F cannot be constructed to accommodate the size of $\mathcal{O}(F)$ in its input gates (see remark at the end of section 4.1). We thus need a method to allow A to check for an arbitrary given circuit C' of any size whether $C'(\alpha) = \beta$ (without knowing α, β). We will construct a circuit $D_{\alpha,\beta}$ that allows this while having a small number of input gates, whereas no other useful information can be learned from access to $D_{\alpha,\beta}$.

To construct such a circuit $D_{\alpha,\beta}$, we would need the assumption that one-way functions exist, yielding secure bit encryption schemes.[2,3] Thus when choosing D (i.e., choosing α, β), we would also choose a secret key K. $D_{\alpha,\beta,K}$ then provides three possible functionalities, chosen by an indicator parameter:

1. For an index $i \in [k]$, $D_{\alpha,\beta,K}(1,i) = \mathrm{Enc}_K(\alpha_i)$. That is, the encryption of the i-th bit of the (hidden) α.

2. For two bit-encryptions and a boolean gate (AND/OR/NOT),

$$D_{\alpha,\beta,K}(2,c,d,\odot) = \mathrm{Enc}_K(\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)).$$

   In fact, this implements a Homomorphic encryption scheme.

[2] Actually, the existence of an *efficient* Obfuscator implies the existence of one-way functions, so the existence of an efficient universal Obfuscator is ruled out unconditionally.

[3] While secure encryption schemes require randomness, we restricted ourselves to deterministic machines. This minor complication is addressed by using a family of pseudorandom functions.

3. Given k bit-encryptions $\gamma_1, \ldots, \gamma_k$:

$$D_{\alpha,\beta,K}(3, (\gamma_1, \ldots, \gamma_k)) = \begin{cases} 1 & \forall 1 \leq i \leq k : \mathrm{Dec}_K(\gamma_i) = \beta_i \\ 0 & \text{otherwise} \end{cases}.$$

   I.e., D checks if we have an encryption of β.

A possible encryption scheme that can be used to construct $D_{\alpha,\beta,K}$ is to encrypt a bit b by $(r, b \oplus f_{SK}(r))$ for $\{f_{sk}\}$ a family of pseudorandom functions and random r.

Indeed, given any circuit C' explicitly (even larger than D), $D_{\alpha,\beta,K}$ can be used to check whether $C'(\alpha) = \beta$ in time $\mathrm{poly}(|C|, |D|)$ in the following way:

1. Retrieve $a_i = \mathrm{Enc}_K(\alpha_i)$ for $1 \leq i \leq k$;
2. Evaluate C' gate-by-gate on the bit encryptions using the Homomorphic encryption functionality;
3. Check whether the resulting output is indeed the bit-by-bit encryption of β.

Finally, we want to combine $C_{\alpha,\beta}$ and $D_{\alpha,\beta,K}$ to a single circuit $F_{\alpha,\beta,K}$ as before. The adversary A, given a circuit F', would compute the description of a circuit F_C whose indicator parameter is fixed to run C, and the use above method to check whether $F_C(\alpha) = \beta$ (again, without A knowing α, β).

Indeed, for every α, β, K and Obfuscator $\mathcal{O}$: $A(\mathcal{O}(F_{\alpha,\beta,K})) = 1$, while replacing $C_{\alpha,\beta}$ with Z_k must give output 0. However, any PPT oracle machine S must fail in simulating A as in the previous setting: The only functionality added to $D_{\alpha,\beta,K}$ provides encrypted values that are meaningless to S (e.g. by using the aforementioned encryption scheme, S only sees pseudorandom strings) or allows to verify an encryption of β. Hence, without knowing α, β or K, the simulator has a negligible probability to distinguish $C_{\alpha,\beta}$ from Z_K.

References

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai and K. Yang, "On the (Im)possibility of Obfuscating Programs", *In Proceedings of the 21st Annual international Cryptology Conference on Advances in Cryptology*, 2001.