600.641 Special Topics in Theoretical Cryptography April 3, 2007

Lecture 18: Program Obfuscation

Instructor: Susan Hohenberger *Scribe: Zachary Scott*

1 Administrivia

- The talk schedule for Homework 3 has been posted on the class website. Please check it and let Prof. Hohenberger know if you have any schedule conflicts.
- If you are interested in playing with proxy re-encryption as discussed yesterday, open source libraries are available online at: `http://spar.isi.jhu.edu/~mgreen/prl`

2 Objective

The goal of program obfuscation is to provably complicate the task of reverse engineering. Owners of source code wish to make it as hard as possible for others to extract information from their binaries - information which may reveal trade secrets, allow undesirable modifications, or enable software piracy.

Obfuscation is typically seen as a problem relating to programming languages or compilers. The connection to cryptography is not entirely clear and was only first noticed in 2000. Since then, a few papers have been published every year on the subject.

Programs are generally viewed as black boxes:

$$INPUT \longrightarrow \boxed{\text{a.out}} \longrightarrow OUTPUT$$

This isn't really accurate. The binary code of a program may reveal a lot about how it operates internally. For example, someone may write a signature delegation program which signs messages on behalf of another given certain circumstances. It is very possible that the signer's secret key could be recovered from that program.

Commercial and freely-available obfuscation tools are often not very good - they rely on "security through obscurity." Renaming variables, ROT-13'ing string literals and adding nonsense instructions may deter the more impatient adversaries, but against a dedicated adversary they offer little to no security. Despite these shortcomings sales and investments made to the makers of obfuscation tools amount to millions of dollars; obviously there is a great demand for such technology. So how can we make it better?

Is is *fundamental* that code must leak secrets about its own operation? The goal of program obfuscation research is to develop means to scramble code so that it still works, but <u>provably</u> cannot be reverse engineered. Current products assume that patience for reverse engineering is limited, but this may not be the case against serious opponents. Can obfuscators make it computationally infeasible to reverse engineer?

Unfortunately, theoretical results have shown that in general, provable obfuscation is **impossible**. ([Had00], [BGI+01], [Wee05], [GK05], [HMLS07], [GR07]) That is, under the definitions of obfuscation studied in these works, there does not exist one algorithm that can obfuscate all programs.

Luckily, though impossible in general, obfuscation has been shown to be **possible** for point functions:

$$\begin{aligned} I_x(y) &= 1 \text{ (if } y = x) \\ &= 0 \text{ (if } y \neq x) \end{aligned}$$

([Can97], [CMR98], [LPS04], [DS05], [Wee05])

For example, a point function such as this may be used for authentication, where x is a password. All we want is to hide x... how can we do it?

First attempt: Store only the hash of x.

$$\begin{aligned} I_x(y) &= 1 \text{ (if } Hash(y) = Hash(x)) \\ &= 0 \text{ (otherwise)} \end{aligned}$$

This is a good try, but the adversary can still recover the hash of x, and that might prove useful somehow. Ran Canetti proposed the following obfuscated point function:

$$\begin{aligned} I_x(y) &= 1 \text{ (if } r^y = r^x) \\ &= 0 \text{ (otherwise)} \end{aligned}$$

This obfuscated code can be denoted by the information it reveals: (r, r^x). Under the Discrete Log assumption, no adversary can extract x from r^x. (Under a variant of DDH, this construction provably satisfies a strong definition of obfuscation.) In addition to password authentication, this technique might also be used to hide constants in the source code.

3 Positive Results: Obfuscating Re-encryption

Recently, obfuscation has also been found to be possible for the re-encryption functionality discussed in the last lecture [HRSV07]. In fact, the security definition of obfuscation has also matured as we'll discuss shortly.

3.1 What is Re-encryption?

Recall from Lecture 17 that re-encryption is the process of securely transforming a ciphertext under one secret key to a ciphertext under another secret key. (See [BBS98], [DI03] and [AFGH06] for a background on re-encryption and constructions that are *almost* obfuscations.)

$$C_a \longrightarrow \boxed{\text{Re-encryption}} \longrightarrow C_b$$

For example, the ciphertext C_a may be an encrypted song available in the iTunes music store. When the song is purchased and transferred to the user's iPod, the song is re-encrypted under a key specific to that device. [Hoh06] It is clearly in Apple's interest to make this process difficult or impossible to reverse engineer, in order to prevent the spread of unencrypted songs. In fact, Apple used an insecure re-encryption scheme which was not sufficiently obfuscated and thus before long its iTunes DRM was cracked.

3.2 What is Obfuscation (formally)?

First we must establish what properties an obfuscator should have.

1. Preserving the program's functionality: The target program should behave exactly the same after the obfuscation as before.
2. The obfuscated code can suffer at most a polynomial slow-down compared to the original program.
3. The obfuscated code is a "virtual black box": it does not leak any information to would-be reverse engineers.

We use the following notation to describe obfuscation:

- $\mathbf{C}$ is a family of circuits
- $\mathbf{C}_n$ are those circuits which take inputs of length n
- $C \in \mathbf{C}_n$ is a circuit
- C' is the obfuscated version of C
- Obf or $O(\cdot)$ is the obfuscation function

In the realm of cryptography, it is very likely that C, C' and Obf will be probabilistic functions. Because of the randomness involved, even on identical inputs C and C' will almost certainly produce differing outputs. Therefore in order to define what it means for an obfuscated program to behave identically to a non-obfuscated program, we must address *average case* secure obfuscation. Note that this modification does not help us with general obfuscation; impossibility results have been shown even for the average case [GK05].

3.3 Defining Obfuscation

- **Preserving Functionality**
  We wish to say, "With high probability, C' behaves identically to C." Again, C, C', and Obf are probabalistic circuits, so we must define equivalence in relation to the overall distribution of outputs.

  We begin by saying that the statistical difference between the outputs of C and the outputs of C' must be low:
  $\forall x \Delta(C(x), C'(x)) < \mu(n)$

  This alone is not sufficient, because Obf itself may be a probabilistic circuit. So we must take into account the probability of statistical difference between C and C', over all of the random coins of Obf:
  $\Pr_{Obf}[\forall x \Delta(C(x), C'(x)) < \mu(n)] > 1 - \mu(n)$

- **Virtual Black Box**:
  Adversaries in possession of the obfuscated program should be able to extract no more information from it than they could gather from oracle access to the original program.

$\forall A\ \exists S\ \forall D\ \forall z$:
$\Pr[D^C(A(Obf(C), z), z) = 1] \approx_c \Pr[D^C(Sim^C(1^n, z), z) = 1]$

This definition (as it is stated in [HRSV07]) looks complicated - what does it mean? Suppose C is a cryptographic scheme. Let us first examine the left half of the equation. $A(Obf(C))$ represents an attack on the obfuscated code by adversary A. If the attack is successful that means that A has learned values of interest from the code. D is a distinguisher; given the outputs from A's attempted attack, D tries to decide whether the resulting values represent the output of a true Obfuscator, or if it is something else (junk, randomness, etc.). If it believes its input to be a valid obfuscated program, it answers with '1'.

On the right side of the definition, D is trying to accomplish the same thing. Instead of outputs from the adversary, however, D is basing its computation on the outputs from a Simulator with oracle access to the unobfuscated program. (z is just an auxiliary input and will not be addressed here.)

If the probabilities that the distinguisher D answers '1' are computationally indistinguishable between the adversary and the simulator, then the obfuscated code offers no advantage over a black-box (oracle) version of the original program. Therefore, $Obf(C)$ must not leak any useful information. In other words, "any attack which A can mount can also be mounted without code."

Why does D have oracle access to C? In order to make sure that the existence of an obfuscated program does not wreck the security of other schemes in its execution environment. In the original paper by Barak et al. [BGI$^+$01], the distinguishers did not have oracle access. However, it was later shown that for certain applications such as obfuscated signature programs it was possible to satisfy the definition but completely break the security of the underlying signature scheme. Granting oracle access to C to the distinguisher in order to avoid this problem makes sense - it gives D a handle on the actual functionality on which it is attempting to operate. In other words, access to a re-encryption oracle mimics the real-world environment that an implementation of such an obfuscated program would find itself in. This allows the definition to model more realistic threats to the security of the cryptosystem which is being obfuscated. In fact, for other applications such as obfuscated signatures, granting D access to additional oracles may be needed to correctly model the "real world". This is an area of ongoing research.

Oracle access for the distinguisher completes the Obfuscation Paradigm for building systems. First, a cryptographic scheme is designed which should be proven secure against adversaries with black-box access. Then, if obfuscated under the above definition, the security of the system is preserved.

At TCC 2007 an alternate definition was proposed [HMLS07]:
$\forall D\ \exists S$:
$\Pr[D^C(Obf(C), z) = 1] \approx_c \Pr[D^C(Sim^C(1^n, z), z) = 1]$

Can you describe the relationship of this definition to the one we've been discussing? Except for the order of the quantifiers, the two definitions are in fact equivalent. It is not necessary to include A in the definition. W.l.o.g., A is just some algorithm performed on the obfuscated code; for simplicity's sake this function can just be integrated into the action of the distinguisher. (A and D are considered to be cooperating.) The order of the quantifiers in [HRSV07] makes it a stronger definition than the more general one of [HMLS07]; as with zero knowledge, it is more common to prove statements under one simulator for all distinguishers, than under an individual simulator for each distinguisher.

Indeed, the definition of [HMLS07] is more simple and therefore probably preferrable.

4 Obfuscating Re-encryption

As we saw on Monday, proxy re-encryption has several applications including secure distributed storage, secure email delegation, and digital rights management (as with iTunes). Although the simplest approach would be to decrypt the input ciphertext and re-encrypt under the public key of the desired recipient, such a solution is a prime target for reverse engineering. It's very likely a skilled reverse engineer would be able to pluck the stored secret key straight out of the code.

Maybe we can replace the simple re-encryption scheme with something similar to the AFGH scheme, which could then be obfuscated securely.

4.1 Background

As with the [BBS98] and [AFGH06] systems, we will base our scheme on bilinear maps.

Let $\mathbb{G}_1, \mathbb{G}_2$ be cyclic groups of prime order q. Then we are interested in a bilinear map e: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ where the following properties hold:

- e is bilinear: $e(g^x, h^y) = e(g,h)^{xy}$ for all $g, h \in \mathbb{G}_1$
- $e(\cdot)$ is effiently computable
- e is non-degenerate: if g, h generate $\mathbb{G}_1$, then $e(g,h)$ generates $\mathbb{G}_2$

4.2 Encryption Scheme

We now begin by describing an encryption scheme due to Boneh, Boyen, and Shacham. We will then see how to devise a suitable re-encryption functionality.

- Key Generation:
  $h \in \mathbb{G}_1, a, b \in \mathbb{Z}_q$
  $PK = (h, h^a, h^b)$
  $SK = (a, b)$
- Encryption of message m:
  $r, s \in \mathbb{Z}_q$
  $C = (h^{ar}, h^{bs}, h^{r+s}m)$

- Decryption:
  $(h^{ar})^{1/a} = h^r \quad (h^{bs})^{1/b} = h^s$
  $h^r h^s = h^{r+s}$
  $m = \frac{h^{r+s}m}{h^{r+s}}$

Theorem: This encryption scheme is semantically secure, even if the adversary is given a re-encryption oracle.

4.3 Re-encryption: First Attempt

$$C_1 \longrightarrow \boxed{\text{Re-encryption}} \longrightarrow C_2$$

$PK_1 = (h, h^a, h^b) \quad PK_2 = (g, g^c, g^d)$
$Z_1 = g^{c/a} \quad Z_2 = g^{d/b}$
$C_1 = (h^{ar}, h^{bs}, h^{r+s}m)$
$C_2 = (e(h^{ar}, Z_1),\, e(h^{bs}, Z_2),\, e(h^{r+s}m, g))$
$= (e(h,g)^{cr},\, e(h,g)^{ds},\, e(h,g)^{r+s}e(m,g))$

There are two issues with this re-encryption scheme. First, as with [AFGH06], the output ciphertext has a different form than the input ciphertext. Secondly, because of the altered form of the output ciphertext, decryption is only possible for small message spaces. The original ciphertext form can be decrypted to reveal m; however, the re-encrypted ciphertext can only be decrypted to reveal $e(m, g)$. With a sufficiently small message space (such as $\{0, 1\}$), the recipient could compute the bilinear mapping of each potential message and compare it to the decrypted value. Otherwise, the message cannot be recovered because of the one-way nature of bilinear maps.

There is a bigger problem, too. Using this system, it is impossible to prove the obfuscation property. After obfuscation, the adversary can extract Z_1 and Z_2 from the code as well as the two public keys. Using those values, the distingisher can simply test $e(h^a, Z_1) \overset{?}{=} e(h, g^c)$. If equal, the code is almost certainly a valid obfuscation and not the output of a simulator, and so the definition will not be met. (Note that if the definition *could* still be met, that would mean the simulator is able to extract the same values from the oracle - i.e., the re-encryption functionality is *learnable* which it should not be.)

4.4 Re-encryption: Second Attempt

$$C_1 \longrightarrow \boxed{\text{Re-encryption}} \longrightarrow C_2$$

$PK_1 = (h, h^a, h^b) \quad PK_2 = (g, g^c, g^d)$
$y \in \mathbb{G}_1$
$Z_1 = y^{c/a} \quad Z_2 = y^{d/b}$
$C_1 = (h^{ar}, h^{bs}, h^{r+s}m)$
$C_2 = (e(h^{ar}, Z_1),\, e(h^{bs}, Z_2),\, e(h^{r+s}m, y),\, y)$
$= (e(h,y)^{cr},\, e(h,y)^{ds},\, e(h,y)^{r+s}e(m,y),\, y)$

This time, given the random y and Z_1, Z_2, there is no easy test for the distinguisher to perform (without calling its oracle) under the Strong Diffie-Hellman Indistinguishability assumption: given $\{g, g^a, g^b, g^c, T\}$, it is computationally infeasible to decide $T \overset{?}{=} g^{abc}$.

But, there is a new problem. The above statement holds true *if* the distinguisher lacks oracle access to C. However, the distinguisher does have such access to the unobfuscated code. That means that it is free to form a query to the oracle using (y, Z_1, Z_2) which might help it test a "funny" property that the simulator's output could not satisfy. Indeed, this is the case here! (To the interested reader: can you figure out how to feed in the obfuscated code to C to help you check if the obfuscated code is valid?) How can we fix this? By adding more randomness to the inputs and outputs; not only does C have to re-encrypt, it also has to re-randomize.

4.5 Re-encryption: Final Attempt

$$C_1 \longrightarrow \boxed{\text{Re-encryption}} \longrightarrow C_2$$

$PK_1 = (h, h^a, h^b) \quad PK_2 = (g, g^c, g^d)$
$y \in \mathbb{G}_1$
$u, v, z \in \mathbb{Z}_q$
$Z_1 = y^{c/a} \quad Z_2 = y^{d/b}$
$C_1 = (h^{ar}, h^{bs}, h^{r+s}m)$
$C_2 = (e(h^{ar}h^{au}, Z_1)^z,\ e(h^{bs}h^{bv}, Z_2)^z,\ e(h^{r+s}m \cdot h^{u+v}, y)^z,\ y^z)$
$= (e(h, y)^{c(r+u)z},\ e(h, y)^{d(s+v)z},\ e(h, y)^{(s+u+r+v)z}e(m, y^z), y^z)$

Decryption of a re-encrypted ciphertext C_2 with $SK = (c, d)$ works as follows:
$C_2 = (e(h, y)^{c(r+u)z},\ e(h, y)^{d(s+v)z},\ e(h, y)^{(s+u+r+v)z}e(m, y^z), y^z)$

1. Compute $(e(h, y)^{c(r+u)z})^{1/c} \cdot (e(h, y)^{d(s+v)z})^{1/d} = e(h, y)^{(s+u+r+v)z}$
2. Compute $\frac{e(h,y)^{(s+u+r+v)z}e(m,y^z)}{e(h,y)^{(s+u+r+v)z}} = e(m, y^z)$
3. Test $e(m, y^z)$ against $e(m_i, y^z)$ for all messages m_i in the message space.

This definition includes the necessary randomness (from u, v, z) to complicate the task of the distinguisher and satisfy the obfuscation property.

5 Summary and Open Problems

The current state of research indicates that general obfuscators are likely impossible, and even provable obfuscation for some common functions is likely to be impossible. However, positive results have been shown for point functions and re-encryption, and may exist for other cryptographic functions. When considering obfuscation, it is crucial to use a definition which preserves the security of the underlying cryptosystem.

Many obfuscation-related problems remain unsolved. In particular,

- Can be devise meaningful relaxations of the obfuscation definitions that are generally realizable?

- Can we make additional assumptions (e.g., physical assumptions) that allow us to obfuscate interesting classes of functionalities?
- How can we ensure obfuscated (re-encryption or other) security when adversaries have auxiliary inputs?
- Can we design re-encryption schemes with identically-formatted input and output ciphertexts?
- What other functions can be obfuscated? (Signatures, DRM...?)
- How efficient can these obfuscations be? (Note, for example, that the point function and re-encryption obfuscations run slower than their unobfuscated counterparts. How much of a slow-down can be tolerated in practice?)

References

[AFGH06] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. In *NDSS*, 2006.

[BBS98] M. Blaze, G. Bleumer, and M. Strauss. Divertable protocols and atomic proxy cryptography. In *EUROCRYPT*, 1998.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (Im)Possibility of Obfuscating Programs. In *CRYPTO*, 2001.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hid all partial information. In *CRYPTO*, 1997.

[CMR98] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions. In *STOC*, 1998.

[DI03] Y. Dodis and A. Ivan. Proxy Cryptography Revisited. In *NDSS*, 2003.

[DS05] Yevgeniy Dodis and Adam Smith. Correcting errors without leaking partial information. In *STOC*, 2005.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the Impossibility of Obfuscation with Auxiliary Inputs. In *FOCS*, 2005.

[GR07] Shafi Goldwasser and Guy Rothblum. On Best-Possible Obfuscation. In *TCC*, 2007.

[Had00] Satoshi Hada. Zero-Knowledge and Code Obfuscation. In *ASIACRYPT*, 2000.

[HMLS07] Dennis Hofheinz, John Malone-Lee, and Martijn Stam. Obfuscation for Cryptographic Purposes. In *TCC*, 2007.

[Hoh06] Susan Hohenberger. *Advances in Signatures, Encryption, and E-Cash from Bilinear Groups*. PhD thesis, MIT, 2006.

[HRSV07] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely Obfuscating Re-Encryption. In *TCC*, 2007.

[LPS04] Benjamin Lynn, Manoj Prabhakaran, and Amit Sahai. Positive Results and Techniques for Obfuscation. In *EUROCRYPT*, 2004.

[Wee05] Hoeteck Wee. On Obfuscating Point Functions. In *STOC*, 2005.