Lecture 21: Software Obfuscation

Boaz Barak

Let us stop and think of the notions we have seen in cryptography. We have seen that under reasonable computational assumptions (such as LWE) we can achieve the following:

- CPA secure *private key encryption* and *Message Authentication codes* (which can be combined to get CCA security or authenticated encryption)- this means that two parties that share a key can have virtual secure channel between them. An adversary cannot get *any additional information* beyond whatever is her prior knowledge given an encryption of a message sent from Alice to Bob. Even if Moreover, she cannot modify this message by even a single bit. It's lucky we only discovered these results from the 1970's onwards— if the Germans have used such an encryption instead of ENIGMA in World War II there's no telling how many more lives would have been lost.
- *Public key encryption* and *digital signatures* that enable Alice and Bob to set up such a virtually secure channel without *sharing a prior key*. This enables our "information economy" and protects virtually every financial transaction over the web. Moreover, it is the crucial mechanism for supplying "over the air" software updates which smart devices whether its phones, cars, thermostats or anything else. Some had predicted that this invention will change the nature of our form of government to crypto anarchy and while this may be hyperbole, governments everywhere are worried about this invention.
- *Hash functions* and *pseudorandom function* enable us to create authentication tokens for deriving one-time passwords out of shared keys, or deriving long keys from short passwords. They are also useful as a tool in *password based key exchange*, which enables two parties to communicate securely (with fairly good is not overwhelming probability) when they share a 6 digit PIN, even if the adversary can easily afford much much more than 10^6 computational cycles.
- *Fully homomorphic encryption* allows computing over encrypted data. Bob could prepare Alice's taxes without knowing what her income is, and more generally store all her data and perform computations on it, without knowing what the data is.

- *Zero knowledge proofs* can be used to prove a statement is true without revealing *why* its true. In particular since you can use zero knowledge proofs to prove that you posses X bitcoins without giving any information about their identity, they have been used to obtain fully anonymous electronic currency.
- *Multiparty secure computation* are a fully general tool that enable Alice and Bob (and Charlie, David, Elana, Fran,..) to perform any computation on their private inputs, whether it is to compute the result of a vote, a second-price auction, privacy-preserving data mining, perform a cryptographic operation in a distributed manner (without any party ever learning the secret key) or simply play poker online without needing to trust any central server.

(BTW all of the above points are notions that you should be familiar and be able to explain what are their security guarantees if you ever need to use them, for example, in the unlikely event that you ever find yourself needing to take a cryptography final exam. . .)

While clearly there are issues of efficiency, is there anything more in terms of *functionality* we could ask for? Given all these riches, can we be even more greedy?

It turns out that the answer is *yes*. Here are some scenarios that are still not covered by the above tools:

Witness encryption

Suppose that you have uncovered a conspiracy that involves very powerful people, and you are afraid that something might happen to you. You would like an "insurance policy" in the form of writing down everything you know and making sure it is published in the case of your untimely death, but are afraid these powerful people could find an attack any trusted agent. Ideally you would to publish an encrypted form of your manuscript far and wide, and make sure the decryption key is automatically revealed if anything happens to you, but how could you do that? A *UA-secure encryption* (which stands for secure against an Underwood attack) gives an ability to create an encryption c of a message m that is CPA secure but such that there is an algorithm D such that on input c and any string w which is a (digitally signed) New York Times obituary for Janine Skorsky will output m.

The technical term for this notion is *witness encryption* by which we mean that for every circuit F we have an algorithm E that on input F and a message m creates a ciphertext c that in a CPA secure, and there is an algorithm D that on input c and some string w, outputs m if $F(w) = 1$. Witness encryption can be used for other applications. For example, you could encrypt a message to

future humanity, that can be decrypted only with a valid proof of the Reimann Hypothesis.

Deniable encryption

Here is another scenario that is seemingly not covered by our current tools. Suppose that Alice uses a public key system (G, E, D) to encrypt a message m by computing $c = E_e(m; r)$ and sending c to Bob that will compute $m = D_d(c)$. The ciphertext is intercepted by Bob's archenemy Freddie Baskerville Ignatius (or FBI for short) who has the means to force Alice to reveal the message and as proof reveal the randomness used in encryption as well. Could Alice find, for any choice of m', some string r' that is pseudorandom and still c equals $E_e(m', r')$? An encryption scheme with this property is called *deniable* , since we Alice can deny she sent m and claim she sent m' instead.[1]

Functional encryption

It's not just individuals that don't have all their needs met by our current tools. Think of a large enterprise that uses a public key encryption (G, E, D). When a ciphertext $c = E_e(m)$ is received by the enterprise's servers, it needs to be decrypted using the secret key d. But this creates a single point of failure. It would be much better if we could create a "weakened key" d_1 that, for example, can only decrypt messages related to sales that were sent in the date range X-Y, a key d_2 that can only decrypt messages that contain certain keywords, or maybe a key d_3 that only allows to detect whether the message encoded by a particular ciphertext satisfies a certain regular expression. This will allow us to give the key d_1 to the manages or the sales department (and not worry about her taking the key with her if she leaves the company), or more generally give every employee a key that corresponds to his or her role. Furthermore, if the company receives a subpoena for all emails relating to a particular topic, it could give out a cryptographic key that reveals precisely these emails and nothing else. It could also run a spam filter on encrypted messages without needing to give the server performing this filter access to the full contents of the messages (and so perhaps even outsource spam filtering to a different company).

The general form of this is called a *functional encryption*. The idea is that for every function $f : \{0,1\}^* \to \{0,1\}^*$ we can create a decryption key d_f such that on input $c = E_e(m)$, $D_{d_f}(c) = f(m)$ but d_f cannot be used to gain any other information on the message except for $f(m)$, and even if several parties holding $d_{f_1}, \ldots, d_{f_k}$ collude together, they can't learn more than simply

[1] One could also think of a deniable witness encryption, and so if Janine in the scenario above is forced to reveal her randomness, she will claim that she didn't encrypt her knowledge of the conspiracy, but merely wanted to make sure that her family secret recipe for pumpkin pie is not lost when she passes away.

$f_1(m), \ldots, f_k(m)$. Note that using fully homomorphic encryption we can easily transform an encryption of m to an encryption of $f(m)$ but what we want here is the ability to *selectively decrypt* only some information about the message.

The formal definition of functional encryption is the following:

Definition (Functional encryption): A tuple $(G, E, D, KeyDist)$ is a *functional encryption scheme* if:

- For every function $f : \{0,1\}^\ell \to \{0,1\}$, if $(d, e) = G(1^n)$ and $d_f = KeyDist(d, f)$, then for every message m, $D_{d_f}(E_e(m)) = f(m)$.
- Every efficient adversary Eve wins the following game with probability at most $1/2 + negl(n)$:
  1. We generate $(d, e) \leftarrow_R G(1^n)$.
  2. Eve is given e and for $i = 1, \ldots, T = poly(n)$ repeatedly chooses f_i and receives d_{f_i}.
  3. Eve chooses two messages m_0, m_1 such that $f_i(m_0) = f_i(m_1)$ for all $i = 1, \ldots, T$.
  4. For $b \leftarrow_R \{0,1\}$, Eve receives $c^* = E_e(m_b)$ and outputs b'.
  5. Eve *wins* if $b' = b$.

The software patch problem

It's not only exotic forms of encryption that we're missing. Here is another application that is not yet solved by the above tools. From time to time software companies discover a vulnerability in their products. For example, they might discover that if fed an input x of some particular form (e.g., satisfying a regular expression R) to a server running their software could give an adversary unlimited access to it. In such a case, you might want to release a patch that modifies the software to check if $R(x) = 1$ and if so rejects the input. However the fear is that hackers who didn't know about the vulnerability before could discover it by examining the patch and then use it to attack the customers who are slow to update their software. Could we come up for a regular expression R with a program P such that $P(x) = 1$ if and only if $R(x) = 1$ but examining the code of P doesn't make it any easier to find some x satisfying R?

Software obfuscation

All these applications and more could in principle be solved by a single general tool known as *virtual black-box (VBB) secure software obfuscation*. In fact, such an obfuscation is a general tool that can also be directly used to yield public key encryption, fully homomorphic encryption, zero knowledge proofs, secure function evaluation, and many more applications.

We will now give the definition of VBB secure obfuscation and prove the central result about it, which is unfortunately that it does not exist. We will then talk about the relaxed notion of *indistinguishablity obfuscators* (IO) - this object turn out to be good enough for many of the above applications and whether it exists is one of the most exciting open questions in cryptography at the moment. We will survey some of the research on this front.

Lets define a *compiler* to be an efficient (i.e., polynomial time) possibly probabilistic map $\mathcal{O}$ that takes a Boolean circuit C on n bits of input and outputs a Boolean circuit C' that also takes n input bits and computes the same function; i.e., $C(x) = C'(x)$ for every $x \in \{0,1\}^n$. (If $\mathcal{O}$ is probabilistic then this should happen for every choice of its coins.) This might seem a strange definition, since it even allows the trivial compiler $\mathcal{O}(C) = C$. That is OK, since later we will require additional properties such as the following:

Definition (VBB secure obfuscation): A compiler $\mathcal{O}$ is a *virtual black box (VBB) secure obfuscator* if it satisfies the following property: for every efficient adversary A mapping $\{0,1\}^*$ to $\{0,1\}$, there exists an efficient simulator S such that for every circuit C the following random variables are computationally indistinguishable:

- $A(\mathcal{O}(C))$
- $S^C(1^{|C|})$ where by this we mean the output of S when it is given the length of C and access to the function $x \mapsto C(x)$ as a black box (aka oracle access).

(Note that the distributions above are of a single bit, and so being indistinguishable simply means that the probability of outputing 1 is equal in both cases up to a negligible additive factor.)

Applications of obfuscation

The writings of Diffie and Hellman, James Ellis, and others that thought of public key encryption, shows that one of the first approaches they considered was to use obfuscation to transform a private-key encryption scheme into a public key one. That is, given a private key encryption scheme (E, D) we can transform it to a public key encryption scheme (G, E', D) by having the key generation algorithm select a private key $k \leftarrow_R \{0,1\}^n$ that will serve as the decryption key, and let the encryption key e be the circuit $\mathcal{O}(C)$ where $\mathcal{O}$ is an obfuscator and C is a circuit mapping c to $D_k(d)$. The new encryption algorithm E' takes e and c and simply outputs $e(c)$.

These days we know other approaches for obtaining public key encryption, but the obfuscation-based approach has significant additional flexibility. To turn this into a fully homomorphic encryption, we simply publish the obfuscation of $c, c' \mapsto D_k(c) NAND D_k(c')$. To turn this into a functional encryption, for every function f we can define d_f as the obfuscation of $c \mapsto f(D_k(c))$.

We can also use obfuscation to get a witness encryption, to encrypt a message m to be opened using any w such that $F(w) = 1$, we can obfuscate the function that maps w to m if $F(w) = 1$ and outputs `error` otherwise. To solve the patch problem, for a given regular expression we can obfuscate the function that maps x to $R(x)$.

Impossiblity of obfuscation

So far, we've learned that in cryptography no concept is too fantastic to be realized. Unfortunately, VBB secure obfuscation is an exception:

Theorem (Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang ,2001): Under the PRG assumption, there does not exist a VBB secure obfuscating compiler.

We will now show the proof. For starters, note that obfuscation is trivial for *learnable* functions. That is, if F is a function such that given black-box access to F we can recover a circuit that computes it, then we can obfuscate it. Given a circuit C, the obfuscator $\mathcal{O}$ will simply use it as a black box to learn a circuit C' that computes the same function and output it. Since $\mathcal{O}$ itself only uses black box access to C, it can be trivially simulated perfectly. (Verifying that this is indeed the case is a good way to make sure you followed the definition.)

However, this is not so useful, since it's not hard to see that all the examples above where we wanted to use obfuscation involved functions that were unlearnable. But it already suggests that we should use an unlearnable function for our negative result. Here is an extremely simple unlearnable function. For every $\alpha, \beta \in \{0,1\}^n$, we define $F_{\alpha,\beta} : \{0,1\}^n \to \{0,1\}^n$ to be the function that on input x outputs β if $x = \alpha$ and otherwise outputs 0^n. Given black box access for this function for a random α, β, it's extremely unlikely that we would hit α with a polynomial number of queries and hence will not be able to recover β and so in particular will not be able to learn a circuit that computes $F_{\alpha,\beta}$.[2]

This function already yields a counterexample for a stronger version of the VBB definition. We define a *strong VBB obfuscator* to be a compiler $\mathcal{O}$ that satisfies the above definition for adversaries that can output not just one bit but an arbitrary long string. We can now prove the following:

Theorem: There does not exist a strong VBB obfuscator.

Proof: Suppose towards a contradiction that there exists a strong VBB obfuscator $\mathcal{O}$. Let $F_{\alpha,\beta}$ be defined as above, and let A be the adversary that on input a circuit C' simply outputs C'. We claim that for every S there exists some α, β and an efficient algorithm $D_{\alpha,\beta}$

[2] Pseudorandom functions can be used to construct examples of functions that are unlearnable in the much stronger sense that we cannot achieve the machine learning goal of outputing some circuit that *approximately predicts* the function.)

$$\left|\Pr[D_{\alpha,\beta}(A(\mathcal{O}(F_{\alpha,\beta}))=1]-\Pr[D_{\alpha,beta}(S^{F_{\alpha,\beta}}(1^{10n}))=1]\right|>0.9 \quad (*)$$

these probabilities are over the coins of $\mathcal{O}$ and the simulator S. Note that we identify the function $F_{\alpha,\beta}$ with the obvious circuit of size at most $10n$ that computes it.

Clearly $(*)$ implies that that these two distributions are not indistinguishable, and so proving $(*)$ will finish the proof. The algorithm $D_{\alpha,\beta}$ on input a circuit C' will simply output 1 iff $C'(\alpha)=\beta$. By the definition of a compiler and the algorithm A, for every α,β, $\Pr[D_{\alpha,\beta}(A(\mathcal{O}(F_{\alpha,\beta}))=1]=1$.

On the other hand, for $D_{\alpha,\beta}$ to output 1 on $C'=S^{F_{\alpha,\beta}}(1^{10n})$, it must be the case that $C'(\alpha)=\beta$. We claim that there exists some α,β such that this will happen with negligible probability. Indeed, assume S makes $T=poly(n)$ queries and pick α,β independently and uniformly at random from $\{0,1\}^n$. For every $i=1,\ldots,T$, let E_i be the event that the i^{th} query of S is the first in which it gets a response other than 0^n. The probability of E_i is at most 2^{-n} because as long as S got all responses to be 0^n, it got no information about α and so the choice of S's i^{th} query is indepednent of α which is chosen at random in $\{0,1\}^n$. By a union bouns, the probability that S got any response other than 0^n is negligible. In which case if we let C' be the output of S and let $\beta'=C'(\alpha)$, then β' is independent of β and so the probability that they are equal is at most 2^{-n}. QED

The adversary above does not seem very impressive. After all, it merely printed out its input. Indeed, the definition of strong VBB security might simply be an overkill, and indeed "plain" VBB is enough for almost all applications. However, as mentioned above, plain VBB is impossible to achieve as well. We'll prove a slightly weaker version of the theorem above:

Theorem: If fully homomorphic encryption exists then there is no VBB secure obfuscating compiler.

(To get the original theorem from this, note that if VBB obfuscation exist then we can transform any private key encryption into a fully homomorphic public key encryption.)

Proof: Let $(G,E,D,EVAL)$ be a fully homomorpphic encryption scheme. For strings $d,e,c,\alpha,\beta,\gamma$, we will define the function $F_{e,c,\alpha,\beta,\gamma}$ as follows: for inputs of the form $00x$, it will output β if and only if $x=\alpha$, and otherwise output 0^n. For inputs of the form $01c'$, it will output γ iff $D_d(c')=\beta$ and otherwise output 0^n. And for the input 1^n, it will output c. For all other inputs it will output 0^n.

We will use this function family where d,e are the keys of the FHE, and $c=E_e(\alpha)$. We now define our adversary A. On input some circuit C', A will compute $c=C'(1^n)$ and let C'' be the circuit that on input x outputs $C'(00x)$. It will then let $c''=EVAL_e(C'',c)$. Note that if c is an encryption of α and C' computes $F=F_{d,e,c,\alpha,\beta,\gamma}$ then c'' will be an encryption of $F(00\alpha)=\beta$. The adversary A will then compute $\gamma'=C'(01c')$ and output γ_1.

We claim that for every simulator S, there exist some tuple $(d, e, c, \alpha, \beta, \gamma)$ and a distinguisher D such that

$$\left|\Pr[D(A(\mathcal{O}(F_{d,e,c,\alpha,\beta,\gamma}))) = 1] - \Pr[S^{F_{d,e,c,\alpha,\beta,\gamma}}(1^{|F_{d,e,c,\alpha,\beta,\gamma}|})]\right| \geq 0.1$$

Indeed, the distinguisher D will depend on γ and on input a bit b will simply output 1 iff $b = \gamma_1$. Clearly, if (d, e) are keys of the FHE and $c = E_e(\alpha)$ then no matter what circuit C' the obfuscator $\mathcal{O}$ outputs on input $F_{d,e,c,\alpha,\beta,\gamma}$, the adversary A will output γ_1 on C' and hence D will output 1 with probability one on A's output.

In contrast if we let S be a simulator and generate $(d, e) = G(1^n)$, pick α, β, γ independently at random in $\{0,1\}^n$ and let $c = E_e(\alpha)$, we claim that the probability that S will output γ_1 will be equal to $1/2 \pm negl(n)$. Indeed, suppose otherwise, and define the event E_i to be that the i^{th} query is the first query (apart from the query 1^n whose answer is c) on which S receives an answer other than 0^n. Now there are two cases:

Case 1: The query is equal to 00α.

Case 2: The query is equal to $01c'$ for some c' such that $D_d(c') = \beta$.

Case 2 only happens with negligible probability because if S only received the value e (which is independent of β) and did not receive any other non 0^n response up to the i^{th} point then it did not learn any information about β. Therefore the value β is independent of the i^{th} query and the probability that it decrypts to β is at most 2^{-n}.

Case 1 only happens with negligible probability because otherwise S is an algorithm that on input an encryption of α (and a bunch of answers of the form 0^n, which are of course not helpful) manages to output α with non-negligible probability, hence violating the CPA security of the encryption scheme.

Now if neither case happens, then S does not receive any information about γ, and hence the probability that its output is γ_1 is at most $1/2$. QED

This proof is simple but deserves a second read. A crcucial point here is to use FHE to allow the adversary to essentially "feed C' to itself" so it can obtain from an encryption of α an encryption of β, even though that would not be possible using black box access only.

Indistinguishability obfuscation

The proof can be generalized to give private key encryption for which the transformation to public key encryption would be insecure, and many other such constructions. So, this result might (and indeed to a large extent did) seem like a death blow to general-purpose obfuscation. However, already in that paper we noticed that there was a variant of obfuscation that we could not rule out, and this is the following:

Definition: We say a compiler $\mathcal{O}$ is an *indistinguishability obfuscator (IO)* if for every two circuits C, C' that have the same size and compute the same function, the random variables $\mathcal{O}(C)$ and $\mathcal{O}(C')$ are computationally indistinguishable.

It is a good exercise to understand why the proof of the impossiblity result above does not apply to rule out IO. Nevertheless, a reasonable guess would be that:

1. IO is impossible to achieve.
2. Even if it was possible to achieve, it is not good enough for most of the interesting applications of obfuscation.

However, it turns out that this guess is (most likely) wrong. New results have shown that IO is extremely useful for many applications, including those outlined above. They also gave some evidence that it might be possible to achieve. We'll talk about those works in the next lecture.