A first look at computational security definitions, semantically secure secret-key encryption, and one-way functions

Noah Stephens-Davidowitz

May 27, 2023

1 Probabilistic polynomial-time adversaries with negligible advantage

We saw in the last lecture some hints that it might be helpful to place some finite—but still gigantic—bound on the computational power of our adversary Eve. If we really were trying to model a *very* powerful but finite adversary in the physical world directly, we would say something like "Eve can build the most powerful computer possible using the resources (like planets and the sun and asteroids and whatever) available in our solar system, and she gets one trillion years to correctly guess whether we encrypted m_0 or m_1 with probability at most $1/2 + 2^{-256}$." (Notice that it is trivial to succeed with probability $1/2$, since a random bit will be correct with probability $1/2$. The difference between Eve's success probability and $1/2$ is often called her *advantage*.) If we had some encryption scheme that could not be broken by this kind of adversary, we'd be pretty comfortable saying that it's "unbreakable for all practical purposes." (And, in practice, this is not an absurd request either. I.e., we really do *frequently* use cryptographic schemes that we think would be secure against such colossal adversaries.) But, as theorists, we'd rather not bring the messiness of the real world into our beautiful mathematical world.

A more elegant solution is to model Eve by a (randomized) Turing machine and to bound her running time by, say, 2^{256} bit operations and her success probability by $1/2 + 2^{-256}$. This definition is perfectly reasonable (provided that one is careful with the specific model of computation chosen) and is morally equivalent to our astronomical definition, but it turns out to still be very difficult to work with (at least with the very primitive techniques that humanity has discovered thus far).

It turns out to be much easier to work with an *asymptotic* definition. So, instead of trying to design schemes that have some concrete level of security (i.e., "no adversary running in time $2^{256}\ldots$"), we instead design schemes whose security level is governed by a parameter n known as the *security parameter*. The security parameter is a bit of a tricky concept, but the rough idea is that the security of the scheme should increase with larger security parameter. For example, the length of the secret key in an encryption scheme *might* equal the security parameter (but it also might not—and it is important not to conflate the two). We will require that our adversary Eve runs in time that is at most *some* polynomial in the security parameter. E.g., Eve may run in time n^2 or n^{100} or $2^{256}n$ or $n^{2^{128}}$, but not in time 2^n or time $2^{\sqrt{n}/7}$ or time $n^{\log\log\log n/2^{100}}$. This definition only makes sense if we leave the security parameter as a variable n and consider the asymptotic behavior of the adversary as $n \to \infty$, i.e., it only makes sense *asymptotically*. (E.g., if I tell you that $n = 128$ and ask you whether an adversary that breaks our scheme in $1,000,000,000$ bit operations

runs in time that is polynomial in n or superpolynomial in n, there's simply no reasonable answer. The question doesn't make sense.)

I won't attempt to justify this definition here except to say that it leads to a very rich and beautiful theory, as we will see throughout this class. This definition also happens to lead to ideas that are useful in practice. (In practice, one necessarily needs to fix some specific security parameter, say $n = 128$, and one would therefore probably not be happy if an adversary can break your encryption scheme in time $n^{\log\log\log n/2^{100}}$, nor is one concerned if your scheme can be broken in time $2^{256}n$. So, asymptotic security and practical security are *not* the same thing. And, many practical schemes don't even have an asymptotic definition, in the sense that the scheme is only defined for some fixed security parameter. But, it's rare that we come up with natural cryptographic schemes that are asymptotically secure but not practically secure.)

To make this a fair competition, we will only build cryptographic schemes that run in time polynomial in the security parameter n as well. For example, we will only consider encryption schemes in which the algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ run in polynomial time. So, nearly every algorithm that we will consider in this class runs in time that is polynomial in n.[1] And in general, we are adopting the philosophy that polynomial-time algorithms are "efficient" while superpolynomial-time algorithms are not. (Again, this is a super duper questionable philosophy, but it's very convenient and will turn out to be quite fruitful.)

We will always allow our adversaries to be randomized, and we call the class of adversaries that we consider *probabilistic polynomial-time* adversaries, or just PPT adversaries. "Probabilistic" to emphasize that they might be randomized, and "polynomial-time" because, well, they run in polynomial time. Sometimes, we will vaguely say "efficient," when we mean PPT.

Of course, if our adversaries are probabilistic, then their success or failure (where our definition of success or failure will depend on the context—e.g., below "success" will be distinguishing between an encryption of m_0 and an encryption of m_1) might depend on the random coinflips of the adversary, as well as the random coins of, e.g., the key-generation algorithm and the encryption algorithm. In particular, our adversaries will typically have *some* chance of successfully breaking our scheme. E.g., if they sample a random bit string, there is some small probability that the resulting string will be our secret key, in which case they will probably be able to break our scheme. But, intuitively, if the probability that this happens is very low, then we are presumably fine with that. We therefore need a notion of "probabilities that are so small that we don't care about them." To handle this, it will be useful to define a *negligible* function $\varepsilon(n)$.

Definition 1.1. *A function $\varepsilon : \mathbb{N} \to \mathbb{R}$ is* negligible *if for every positive integer c, there exists an n_c such that for all $n \geq n_c$, $|\varepsilon(n)| < 1/n^c$.*

(This definition isn't totally consistent across different sources. E.g., some sources will require negligible $\varepsilon(n)$ to be positive or to be non-negative. But, these differences don't actually matter. It's also common to see formally different definitions of negligibility that are provably equivalent. E.g., if you've seen asymptotic notation before, then you can simply say that $\varepsilon(n)$ is negligible if $|\varepsilon(n)| = n^{-\omega(1)}$.)

In other words "$\varepsilon(n)$ eventually decays faster than any inverse polynomial $1/n^c$." For example, the functions $\varepsilon(n) = 2^{-n}$, $\varepsilon(n) = n^{-\log n}$, and $\varepsilon(n) = 2^{-\sqrt{n}/1000}$ are all negligible functions.

[1] Notice that our adversary still has a huge advantage over us here. Specifically, while our algorithms like $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ will be fixed algorithms running in some fixed polynomial time—say $O(n^2)$ time—we will consider adversaries that run in time $\mathrm{poly}(n)$ for an arbitrarily large polynomial n. So, we must be secure against adversaries running in time n^3 but also n^4 and n^{10} and n^{100}, etc.

However, the functions $\varepsilon(n) = 1/2$, $\varepsilon(n) = 1/n$, and

$$\varepsilon(n) = \begin{cases} 1/n & n \text{ is prime} \\ 2^{-n} & \text{otherwise.} \end{cases}$$

are *non-negligible*.

A good way to think of this definition in this context is the following heuristic: "A polynomial-time algorithm will never observe anything that happens with negligible probability." This heuristic is true in the sense that if I think of T-time algorithms loosely as "observers that can do something roughly T times," then presumably such observers don't really care about events that happen with probability much less than $1/T$. Our definition of negligible guarantees that for any polynomial running time $T(n) = \text{poly}(n)$, $\varepsilon(n) \ll 1/T(n)$ for sufficiently large n.

Anyway, the point is that in our security definitions, we won't mind if some bad event (like Eve guessing our secret key) happens with negligible probability.

(Try modifying the definition of negligible to see what happens. E.g., what happens if we remove the condition that $n \geq n_0$? What happens if we say "there exists an n_0 such that for every positive integer c" instead of "for every c there exists an n_c"? In general, whenever you see a definition, you should try things like this. Usually, definitions that you encounter in a course or a textbook or whatever are pretty well thought out, and it helps to see why the definition is as it is by seeing what happens when you modify it.)

2 Secret-key encryption

We are now ready to define the *semantic security* of an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. In fact, we will define *many-message semantic security*.

In order to use our new tools of PPT adversaries and negligible functions, we will need to somehow incorporate the security parameter into our definition. To that end, the Gen algorithm will now take as input 1^n, where n is the security parameter. Here, we have introduced common (rather silly but very convenient) notation, in which we write 1^n to represent the string $1111\cdots1 \in \{0,1\}^n$ consisting of n ones in a row. (This is of course dangerous notation because it looks like exponentiation. Fortunately, in this course, we will never actually want to raise 1 to the nth power, so we'll be okay :).)

In some sense, the purpose of giving the Gen algorithm 1^n as input is just to let Gen "know" the security parameter so that, e.g., Gen can choose the length of the secret key, or, more generally, so that the choice of key can depend on the security parameter). We will also require that the algorithms Gen, Enc, and Dec are PPT as well, since otherwise this is not a fair fight. Notice that Gen takes 1^n as input, and not the number n written in binary. This is a convenient trick to ensure that Gen runs in time $\text{poly}(n)$ if and only if Gen runs in time that is polynomial in the length of its input (since the length of its input 1^n is exactly n by definition).[2] We will do the same for our adversaries. I.e., our adversaries will always receive 1^n as input (in addition to any other input). Before you move on, make sure that you understand why we give the Gen algorithm and our adversary 1^n as input, as we're going to use this idea *a lot* going forward.

[2] We could just as easily give Gen as input any arbitrary bit string of length n, or even any arbitrary bit string of length $\text{poly}(n)$.

Of course, our main issue with the one-time pad was that it can only be used once. (Except for that fatal flaw, the one-time pad is great!) So, in order to resolve this issue, our new definition will necessarily have to provide some security guarantee even when Alice sends *many* plaintexts $m_1, \ldots, m_\ell$ to Bob. This raises the question of how many plaintexts we should imagine Alice sending. Our solution is to let the adversary decide. In fact, we will let the adversary *choose* the plaintexts that Alice sends as well as the number of plaintexts ℓ. Specifically, we will let the adversary choose two *lists* of plaintexts $M_0 = (m_{1,0}, \ldots, m_{\ell,0})$ and $M_1 = (m_{1,1}, \ldots, m_{\ell,1})$, and the adversary's goal will be to guess which one we have encrypted. So, if a scheme is secure under this definition, then no (PPT) adversary can even find two sequences $M_0 = (m_{1,0}, \ldots, m_{\ell,0})$ and $M_1 = (m_{1,1}, \ldots, m_{\ell,1})$ whose encryptions it can distinguish. (This idea of letting the adversary decide is common in cryptography. To make our security definitions as strong as possible, we try to make the adversary's life as easy as we can—and that often means letting the adversary have control of as many things as possible.)

Here's the definition written symbolically, which is a bit hard to understand.

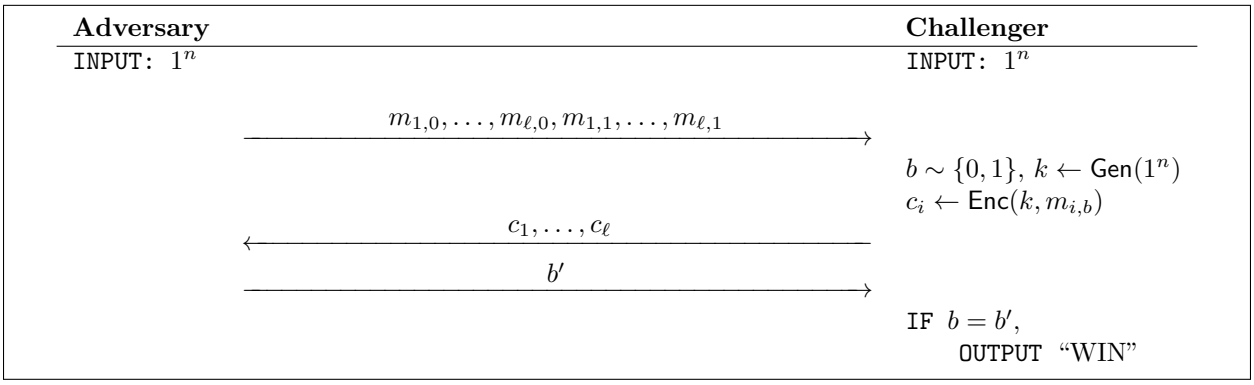
Definition 2.1 (Many message semantic security)**.** *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* many message semantically secure *if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\varepsilon(n)$ *such that for all* n*,*

$$\Pr_{\substack{b\sim\{0,1\},k\leftarrow\mathsf{Gen}(1^n)\\(\sigma,m_{1,0},\ldots,m_{\ell,0},m_{1,1},\ldots,m_{\ell,1})\leftarrow\mathcal{A}(1^n)}} [\mathcal{A}(\sigma, \mathsf{Enc}(k, m_{1,b}), \ldots, \mathsf{Enc}(k, m_{\ell,b})) = b] \leq 1/2 + \varepsilon(n) \ .$$

What this definition is really describing is a *game* that we can imagine the adversary $\mathcal{A}$ playing. In this game, $\mathcal{A}$ chooses two lists of plaintexts, $M_0 := (m_{1,0}, \ldots, m_{\ell,0})$ and $M_1 := (m_{1,1}, \ldots, m_{\ell,1})$, we encrypt either M_0 and M_1 using Enc and a key generated using Gen, and we ask the adversary $\mathcal{A}$ to guess whether we encrypted M_0 or M_1. (Here, M_0 and M_1 are *lists* of plaintexts that are as long as the adversary likes. But, notice that they must both have the same length.) The scheme is secure if no PPT $\mathcal{A}$ can win this game with probability that is non-negligibly larger than $1/2$.

The variable σ is the *state* of $\mathcal{A}$. In other words, when $\mathcal{A}$ chooses M_0, M_1, it also outputs some information σ about its choice, which it later might use to help it distinguish an encryption of M_0 from an encryption of M_1. E.g., σ might consist of any random coins flipped by $\mathcal{A}$ in choosing the plaintexts M_0, M_1, or it might simply be $\sigma = (M_0, M_1)$.

The notation above is pretty clunky, though. It is much cleaner to instead describe this directly as an interactive game between the adversary $\mathcal{A}$ and a *challenger*, as represented below.



It should be clear that a scheme is many message semantically secure if and only if no PPT adversary wins this game with probability non-negligibly larger than 1/2. This game-based way of viewing things is quite convenient. (But, it is sometimes important to remember that these pretty diagrams can be converted into purely symbolic definitions like the one above.)

If we forget for a second about the computationally bounded adversary and the negligible advantage ε, then notice that this really is the "many-time" variant of what we called "perfect security against an adversary" in the previous lecture notes. Of course, we proved in the previous lecture notes that such a many-time definition could not be achieved (in particular, if we take ℓ large enough so that $|\mathcal{M}^\ell| > |\mathcal{K}|$, then the impossibility result from before rules out security). So, we have to add the computational restriction on $\mathcal{A}$ *and* the small ε. We often call the difference between the adversary's success probability and 1/2, the adversary's *advantage*, so that a succinct way of describing this definition is to say that "no polynomial-time adversary has non-negligible advantage in the above game."[3]

3 Complexity theory is hard :(

Now that we have our definition, we would like to show a construction that provably satisfies it. Unfortunately, we won't succeed—at least not in this course, and perhaps not in my lifetime. If we could prove that an efficient many message semantically security encryption scheme exists, then we would immediately prove that $\mathsf{P} \neq \mathsf{NP}$. This seems hard...

Instead, we will only prove that such schemes exist under different computational assumptions—i.e., we will prove *conditional* security. We will consider two broad classes of assumptions, which I will loosely refer to as "specific" and "generic." A specific complexity-theoretic assumption is an assumption about a *specific* computational task, like factoring a number or breaking some specific cryptographic construction. A generic assumption simply posits the existence of a certain kind of computationally difficult task. E.g., "semantically secure secret-key encryption exists" is a generic assumption, while "AES (the most popular secret-key encryption scheme used in practice) is semantically secure" or "factoring is hard" are specific assumptions.

I won't point out this distinction much going forward, but it's worth keeping in mind.

3.1 Reductions

Our main tool for proving these conditional results is a *reduction*. Loosely speaking, for two computational problems A and B, we say that "A reduces to B" if an efficient algorithm for B can be converted into an efficient algorithm for A. I.e., "if B is easy to solve, then so is A." For example, many different problems related to graphs can be reduced to, say, breadth-first search, and many different problems in number theory can be reduced to factoring.

You've likely seen reductions already in the context of NP-hardness. In particular, a problem B is NP-hard if we can reduce SATISFIABILITY to B. The reason that we call this "hardness" is because we take the contrapositive of the statement "if B is easy to solve, then so is A" to conclude that "if A is *hard* to solve, then so is B." So, if we have some problem A (like SATISFIABILITY)

[3] It is interesting to think about what happens if we do one of these things but not the other. E.g., what if we require $\varepsilon = 0$ in the definition of semantic security but keep $\mathcal{A}$ bounded? Or what if we allow $\mathcal{A}$ to be unbounded but keep a non-zero negligible ε? In the first case (when $\varepsilon = 0$), the definition is equivalent to Shannon security. In the second case, the definition is slightly weaker than Shannon security, but a similar impossibility result holds for such schemes as well.

that we believe is hard, and we can show how to reduce A to some other problem B, then we must also believe that B is hard.

More formally, a reduction from A to B is an efficient algorithm that solves problem A *given black-box access to an algorithm that solves* B. I.e., in addition to all the normal things that algorithms can do (e.g., adding two numbers together, sorting lists, etc.), the reduction can can also ask "what's the solution to $B(x)$?" for any valid input x for problem B and get the solution. Alternatively, we imagine that the reduction has access to an *oracle* that answers questions of that form, and we call these questions *oracle queries* or just *queries*. To measure the running time of such a beast, we count each of these queries as a single time step and simply add that number to the running time of all the other normal algorithm-y stuff that the reduction does.

This is a *very* powerful tool for us (i.e., for humans) because we believe that lots of computational problems are hard but we (humans) are unable to actually *prove* that they are hard. So, since it seems to be too difficult for us to prove hardness outright, we often content ourselves with proving that hardness of one problem implies hardness of another.

In cryptography, we take this idea really really far. In particular, nearly every proof that we do starts by assuming that some problem A is hard, and then proving via a reduction that this implies that some crypytographic scheme that we've built must be secure.

However, the reductions in cryptography are a bit different than the ones that you have probably seen so far. In cryptography, we typically need to rule out adversaries that break our schemes with *any* non-negligible advantage. This means that our reduction must work even if the adversary only succeeds with some small (but still non-negligible) advantage.

For example, suppose we imagine some adversary $\mathcal{E}$ that breaks some encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. What this means is that this adversary $\mathcal{E}$ has probability $1/2 + \varepsilon(n)$ of winning the many-message semantic security for some *non-negligible* $\varepsilon(n)$. This is a very specific guarantee. It just guarantees that if we (1) sample a key k using the Gen algorithm, (2) receive two lists of plaintexts from the adversary $\mathcal{A}$, (3) choose (uniformly) a random list to encrypt, (4) send the resulting ciphertexts to the adversary, and (5) ask her which list we encrypted, she will answer correctly with probability non-negligibly larger than $1/2$, maybe probability $1/2 + 1/n^{10}$. And this is the *only* guarantee that we have on $\mathcal{E}$. If our reduction uses $\mathcal{E}$ on input generated in a different way (say, e.g., keys k sampled from a different distribution), then we lose even this tiny guarantee.

E.g., maybe our adversary only has an advantage in distinguishing encryptions of M_0 from encryptions of M_1 when the first twenty bits of $k \in \{0,1\}^n$ are zero, or when k is a prime number, or when k has more ones than zeros in it, or whatever. Then, our reduction must itself at least produce ciphertexts that correspond to such keys with non-negligible probability.

More generally, we must be very very careful about the distribution of the input to the adversary $\mathcal{E}$ in these reductions, and we must be very careful to make sure that our reduction works even if our adversary has just some small (but still non-negligible) advantage. It is surprisingly easy to "prove" something that is false by getting this wrong.

3.2 On (non-)uniformity and who chooses the $m_{i,j}$

Notice that in our definition of semantic security, we have switched from quantifying directly over all plaintexts to having $\mathcal{A}$ output the plaintexts $M \in \mathcal{M}^*$. (Recall that $\mathcal{M}^*$ is the set of all finite sequences of elements from $\mathcal{M}$.) Since we are quantifying over all PPT algorithms $\mathcal{A}$, this is almost equivalent. But, notice that some sequences of plaintexts M_n are not possible. As a trivial example, notice that a PPT algorithm $\mathcal{A}$ cannot output a sequence of plaintexts M_n, if M_n

has superpolynomial length. (Notice that here we are parameterizing by n, since otherwise it does not make sense to ask whether M has length that is "polynomial in n.") More generally, a PPT algorithm can, of course, only output a sequence of plaintexts M_n that is efficiently computable. (Really, here, we should be discussing *families of distributions of sequences of plaintexts* M_n—families because they are parameterized by n and distributions because $\mathcal{A}$ is randomized—but we will avoid this cumbersome terminology.)

The set of possible M_n actually depends quite a bit on our explicit model of computation. In particular, there are two popular notions of PPT adversaries: uniform and non-uniform. A uniform adversary can be modeled as a fixed Turing machine T that takes as input arbitrarily long strings. It is probably the model that you have seen the most. (Of course, we will almost never talk about the Turing machine T directly, and we will certainly not describe our algorithms as Turing machines. One can also think of, e.g., a program written in Python.)

A *non-uniform* adversary can be thought of in multiple ways. One can think of it as a *family* $\{T_n\}_{n\in\mathbb{N}}$ of Turing machines (or programs written in Python), one for each input length n. The "running time" of such a family on input $x \in \{0,1\}^n$ is defined as the sum of the actual running time together with the length of the description of T_n. One can equivalently think of a non-uniform adversary as a fixed Turing machine T together with an *advice function* $s(n)$. When such a Turing machine receives input $x \in \{0,1\}^n$, it is also given as input $s(n)$. The "running time" is again $|s(n)|$ plus the conventional running time. Yet a third model is as a family C_n of *circuits*, which is a representation of an arbitrary function $\{0,1\}^n \to \{0,1\}^*$ as a composition of many two-bit functions called gates, $G : \{0,1\}^2 \to \{0,1\}$. The "running time" of a circuit is the number of gates. All of these non-uniform models lead to equivalent definitions of polynomial-time adversaries.

If our adversary is non-uniform, then asking the adversary to provide the challenge plaintexts M_n is equivalent to quantifying over all $M_n \in \mathcal{M}^*$ with length polynomial in n, since we can always consider the non-uniform PPT adversary that takes M_n as part of its advice string. But, if the adversary is uniform, then some sequences $M_n \in \mathcal{M}^*$ simply cannot be computed by the adversary, even if they are not particularly long.

We try to ignore this distinction as much as possible, and most of our definitions and proofs will work in either model. But, formally the distinction does matter. For example, if we instead defined semantic security by quantifying over all polynomial-length M_n, we would run into trouble later in trying to prove security in the uniform setting. In particular, when we try to prove security via a uniform reduction, we will posit the existence of an adversary that breaks this security and derive a contradiction. But, if we wish to make any use of our hypothetical adversary, we will need to efficiently compute some sequence of messages M_n for which it has an advantage.

4 One-way functions

Instead of trying to prove that $\mathsf{P} \neq \mathsf{NP}$, we will try to find the weakest complexity-theoretic assumption that we can make that will imply the existence of secret-key encryption. Indeed, we will see that the existence of a very simple cryptographic primitive called a *one-way function* is *equivalent* to the existence of (1) secret-key encryption; and (2) many other really beautiful cryptographic primitives. Intuitively, a one-way function is *one way* in the sense that it is easy to compute, but hard to invert. I.e., "you can compute it in one direction, but not the other." Here's the definition.

Definition 4.1. *A function* $f : \{0,1\}^* \to \{0,1\}^*$ *is a* one-way function *if it satisfies the following*

two criteria.

- **Easy to compute.** *There exists a PPT algorithm $\mathcal{B}$ such that $\forall x \in \{0,1\}^*$, $\mathcal{B}(x) = f(x)$.*
- **Hard to invert.** *For any PPT adversary $\mathcal{A}$, there exists a negligible function $\varepsilon(n)$ such that for all $n \geq 1$,*

$$\Pr_{x \sim \{0,1\}^n}[x' \leftarrow \mathcal{A}(1^n, f(x)),\ f(x') = f(x)] \leq \varepsilon(n) .$$

There are many things to notice about this definition. First, notice that this definition has a very different form than our definition of semantic security of an encryption scheme. Our definition of semantic security was an *indistinguishability-based definition*, in which the adversary $\mathcal{A}$ only needed to guess *one* bit b with probability non-negligibly better than a random guess. In contrast, here we ask the adversary $\mathcal{A}$ to output a whole string x'! For example, if f is an injective function, then our adversary must output all n bits of x exactly. (If f is not injective, then there could be many x' with $f(x) = f(x')$, each of which is a valid answer.) In this case, a random guess is successful with probability only 2^{-n}, which is why here we ask that the adversary has negligible probability of success, rather than $1/2 + \text{negl}(n)$. This distinction is similar to the distinction between a computational search problem and a computational decision problem.

So, relative to the notion of semantic security, one-way functions should seem like relatively weak primitives.

It is *always* a good idea to play with a definition to see what happens when you change things. For example, what happens if we say that the adversary wins only if $x' = x$, instead of when $f(x') = f(x)$? Well, then the function $f(x) = 1$ would be "a one-way function," since no adversary (PPT or otherwise) can possibly win the corresponding game with probability better than 2^{-n}. (Make sure you see this.) So, this definition would be rather silly.

As another example, let's think about what happens if we do not give the adversary 1^n as input. I.e., what if we replace the inequality in the definition of hardness of inversion by

$$\Pr_{x \sim \{0,1\}^n}[x' \leftarrow \mathcal{A}(f(x)) \ :\ f(x') = f(x)] \leq \varepsilon(n) \ ?$$

(This is the same thing except that I removed the 1^n from the definition.) Then, consider the function $f(x) = |x|$, where $|x|$ is the length of x written in binary. I claim that this function is "hard to invert" under this new definition. (Why?)

On the other hand, the definition as we wrote it (the "right" definition) is non-trivial in the sense that the existence of a one-way function immediately implies that $\mathsf{P} \neq \mathsf{NP}$. To see this, consider the NP (search) problem in which the input is 1^n and $y \in \{0,1\}^*$ and the goal is to find $x \in \{0,1\}^n$ such that $f(x) = y$ (if such an x exists). Notice that this problem is in NP but not in P if f is a one-way function.

However, in some sense the existence of a one-way function implies something that seems qualitatively much stronger than $\mathsf{P} \neq \mathsf{NP}$. In particular, we do not simply ask that it is hard to invert f in the *worst case* (i.e., that there exists x such that, given $f(x)$, it is difficult to find x' with $f(x) = f(x')$), but instead that it is hard to invert f *on the average*, with any non-negligible probability.

4.1 Welcome to Minicrypt!

In Russell Impagliazzo's famous paper from 1995 [Imp95], he defined what are now known as "Russell's five worlds," five different scenarios that are compatible with what we currently know

how to prove. E.g., the world in which $\mathsf{P} = \mathsf{NP}$ is called Algorithmica. This is a world in which algorithms are extremely powerful and useful. Most of what we learn in this course would be useless in Algorithmica, so we won't spend much time talking about it...

The world in which one-way functions exist (but certain fancier cryptographic primitives like public-key encryption do not) is called Minicrypt. We will spend about half of this class in Minicrypt, and we'll see that Minicrypt is a pretty amazing place. For example, we will see that many-message semantically secure secret-key encryption, zero-knowledge proofs, and signatures are in Minicrypt!

Nevertheless, we will leave Minicrypt eventually to enter what Russell called Cryptomania! This is the world in which public-key encryption exists. (There is also a new world that was discovered after Russell's paper, which some people have called Cryptofantasia. In Cryptofantasia, a rather strange primitive called indistinguishability obfuscation exists.)

For completeness, I should tell you about Russell's two remaining worlds: Heuristica and Pessiland. But, I won't because I want to encourage you to read Russell's paper yourself. It is very short and very fun to read [Imp95].

5 Factoring and weak one-way functions

It remains to actually build a one-way function. There's actually a very natural candidate that you might have thought of yourself. Let $f(p,q) := pq$, where $p \in [0, 2^{\lfloor n/2 \rfloor} - 1]$ is a non-negative integer with bit length at most $\lfloor n/2 \rfloor$ and $q \in [0, 2^{\lceil n/2 \rceil} - 1]$ is a non-negative integer with bit length at most $\lceil n/2 \rceil$. Of course, f is efficiently computable. And, inverting f is exactly the problem of finding a factorization of the integer pq. And, "factoring is hard," so this should be a one-way function, right?

Well, no. First, there's a silly problem. First, notice that $p' = 1, q' = N \in [0, 2^{2\lfloor \log_2 N \rfloor + 1} - 1]$ is a valid inverse for any N. (Notice that the bit length of p', q' does not need to be the same as the bit length of the original input p, q to f.) So, we should probably make our function either $f(p,q) = (p+2)(q+2)$ or effectively disallow 1 as an input by defining $f(1,q) = f(p,1) = 0$, and $f(p,q) = pq$ when $p, q \neq 1$. In fact, one-way functions with restricted input (e.g., $p, q \in [2, 2^{n/2} - 1]$ or even $p, q \in \{0, 2, 3, \ldots, 2^{n/2} - 1\}$) are just as good as one-way functions. Below, we simply take $p, q \in [2, 2^{n/2} - 1]$. So, now in order to break our one-way function, an adversary must find factors of N that are greater than 2.

The more serious problem is that, while factoring is thought to be hard in the worst case, it is certainly not hard to find a non-trivial factor of a random integer, or a random product of two integers. For example, if $p \sim [2, 2^{\lfloor n/2 \rfloor} - 1]$, then p is even with probability $1/2$, and the same is of course true for $q \in [2, 2^{\lceil n/2 \rceil} - 1]$. So, with probability $3/4$, $pq = 2 \cdot (pq/2)$ is a valid factorization, and there is therefore a trivial algorithm that inverts f with probability $3/4$.

One can try to fix this by enforcing some restrictions on p and q. Ideally, one would like to restrict to the case when p and q are prime, and we will work with this variant later. But, for now, let's consider the following much weaker definition.

Definition 5.1. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a* weak one-way function *if it satisfies the following two criteria.*

- **Easy to compute.** *There exists a PPT algorithm $\mathcal{B}$ such that $\forall x \in \{0,1\}^*$, $\mathcal{B}(x) = f(x)$.*

- **Weakly hard to invert.** *There exists some positive integer c such that for any PPT adversary $\mathcal{A}$, there exists an n_0 such that for all $n \geq n_0$,*

$$\Pr_{x \sim \{0,1\}^n}[x' \leftarrow \mathcal{A}(1^n, f(x)),\ f(x') = f(x)] \leq 1 - 1/n^c\ .$$

Again, it is always a good idea to play with a definition when you first see it. For example, what happens if we remove the condition that $n \geq n_0$? What if we reverse the quantifiers and say "for every PPT adversary, there exists a c"?

We do not know whether or not the multiplication function $f(p,q) = pq$ is even a *weak* one-way function because we do not even know how to prove unconditionally that *anything* is a weak one-way function. But, we think it probably is. For example, with probability $\Theta(1/n^2)$, $p \sim [2, 2^{\lfloor n/2 \rfloor} - 1]$ and $q \sim [2, 2^{\lceil n/2 \rceil} - 1]$ will be large primes (say, primes larger than $2^{n/2-10}$). And, we think that it is likely difficult to factor the product of two random large primes. More generally, this definition captures the idea that "there is some non-negligible subset of inputs x for which $f(x)$ is hard to invert."

Now, we show that the existence of any weak one-way function implies the existence of a ("strong") one-way function. (Of course, the other direction is obvious, so that the existence of weak one-way functions is equivalent to the existence of strong one-way functions.) This will be our first reduction.

Theorem 5.2. *If a weak one-way function f exists, then there exists a strong one-way function g.*

The idea behind the proof is to simply take $g(x_1, \ldots, x_\ell) := (f(x_1), f(x_2), \ldots, f(x_\ell))$ for some suitably large choice of ℓ with $n := \ell m$ and $x_i \in \{0,1\}^m$. (We ignore the issue that n might not be decomposable into factors ℓ, m of the appropriate size. To fix this, we can simply ignore any "extra" input bits to g.)

The proof will then suppose that there exists a PPT adversary $\mathcal{A}$ that inverts g with non-negligible probability $\varepsilon(n)$, and attempt to prove that the existence of such an adversary implies an $\mathcal{A}'$ that inverts f with probability greater than $1 - 1/m^c$ for infinitely many m. This would contradict the assumption that f is a weak one-way function. (Nearly all of our proofs have this form.)

Here's a first attempt: Our adversary $\mathcal{A}'$ takes as input $y^* := f(x^*)$ for $x^* \sim \{0,1\}^m$. It then samples $x_2, \ldots, x_\ell \sim \{0,1\}^m$, compute $y_i := f(x_i)$, and compute $(x'_1, \ldots, x'_\ell) \leftarrow \mathcal{A}(1^n, y^*, y_2, \ldots, y_\ell)$. If we do this many times, say, $100n^c/\varepsilon(n)$ times, then we would expect to find at least one time when $f(x'_1) = y^*$. The problem with this strategy is that it doesn't work because our adversary $\mathcal{A}$ might be, well, adversarial. For example, suppose that $\mathcal{A}$ refuses to output anything (or always outputs 0 or whatever) when the first bit of x_1 is 0. Such an $\mathcal{A}$ could still have very large advantage, but our new adversary $\mathcal{A}'$ would fail with probability $1/2$ in this case—much higher than the probability of $1/m^c$ that we need.

But, suppose that $\mathcal{A}$ instead refused to output anything if *any* of the x_i had first bit zero. Such an adversary $\mathcal{A}$ could not have advantage larger than $2^{-\ell}$, which is much less than $1/n^c$ (for large enough n). This suggests that we should try putting our input y^* in many different positions.

Ok... here's the real proof.

Proof. It is trivially the case that g is efficiently computable, since f is. So, we "only" need to prove that g is hard to invert.

Suppose that there exists a PPT adversary $\mathcal{A}$ that inverts g with non-negligible probability $\varepsilon(n)$. I.e., for infinitely many n, $\varepsilon(n) \geq 1/n^d$ for positive integer d. We may take $\ell > dm^c \log(n)$ to be large enough that $(1 - 1/(2m^c))^\ell \leq 1/(2n^d)$.[4] Then, we wish to construct an adversary $\mathcal{A}'$ that inverts f on input with length m with probability greater than $1 - 1/m^c$ for infinitely many m.

Our adversary $\mathcal{A}'$ takes as input 1^m and $y^* := f(x^*)$ for $x^* \sim \{0,1\}^m$. It then does the following $10m^c\ell n^{d+1}$ times for each $i \in [1, \ell]$. It sets $y_i := y^*$, and for each $j \in [1, \ell]$ with $j \neq i$, it samples $x_j \sim \{0,1\}^m$ and sets $y_j := f(x_j)$. Finally, it calls $\mathcal{A}$ on input $(1^n, (y_1, \ldots, y_\ell))$, receiving as output $(x'_1, \ldots, x'_\ell)$. If at some point it finds x'_i with $f(x'_i) = y^*$, it outputs x'_i. If it finishes the loop without finding such an x'_i, it simply fails.

It is clear that $\mathcal{A}'$ is a PPT algorithm. We wish to show that it succeeds with probability larger than $1 - 1/m^c$ for infinitely many m. To that end, for $x \in \{0,1\}^m$ let $E_i(x)$ be the random variable obtained as follows. Set $y_i := f(x)$, and for each $j \in [1, \ell]$ with $j \neq i$, sample $x_j \sim \{0,1\}^m$ and set $y_j := f(x_j)$. Then, $E_i(x) = 1$ if $\mathcal{A}(1^n, (y_1, \ldots, y_\ell))$ outputs a valid inverse $(x'_1, \ldots, x'_\ell)$ with $f(x'_i) = y_i$ for all i and $E_i(x) = 0$ otherwise. Let

$$\mathsf{BAD}(x) := \{x \in \{0,1\}^m \ : \ \forall i, \ \Pr[E_i(x) = 1] < 1/(2m^c\ell n^d)\} \ .$$

First, notice that for $x^* \notin \mathsf{BAD}$,

$$\begin{aligned}\Pr[x' \leftarrow A'(1^m, f(x^*)), \ f(x) \neq f(x')] &\leq \max_i \Pr[E_i(x^*) = 0]^{10m^c\ell n^{d+1}} \\ &\leq (1 - 1/(2m^c\ell n^d))^{10m^c\ell n^{d+1}} \\ &\leq e^{-n} \ .\end{aligned}$$

In other words, if x^* is not in the "bad set" then our reduction will find an inverse with high probability.

So, it remains to prove that $\Pr_{x^* \sim \{0,1\}^m}[x^* \in \mathsf{BAD}] \leq 1/m^c - e^{-n}$, since then we would have

$$\begin{aligned}\Pr_{x^* \sim \{0,1\}^m}[x' \leftarrow \mathcal{A}'(1^m, f(x^*)), \ f(x') = f(x^*)] &\geq \Pr[x^* \notin \mathsf{BAD}] \cdot \Pr[x' \leftarrow A'(1^m, f(x^*)), \ f(x) = f(x') \mid x^* \notin \mathsf{BAD}] \\ &\geq (1 - e^{-n})(1 - 1/m^c + e^{-n}) \\ &\geq 1 - 1/m^c \ ,\end{aligned}$$

as needed.

Let $X := (x_1, \ldots, x_\ell)$ and $I(X)$ be the event that $\mathcal{A}$ successfully inverts on input $(1^n, f(x_1), \ldots, f(x_\ell))$. We have

$$\begin{aligned}\Pr_{x_1, \ldots, x_\ell \sim \{0,1\}^m}[I(X)] &= \Pr[I(X) \text{ and } \exists i, \ x_i \in \mathsf{BAD}] + \Pr[I(X) \text{ and } \forall i, \ x_i \notin \mathsf{BAD}] \\ &\leq \sum_{i=1}^{\ell} \Pr[I(X) \text{ and } x_i \in \mathsf{BAD}] + \Pr[\forall i, \ x_i \notin \mathsf{BAD}] \\ &< \frac{1}{2m^c n^d} + \Pr_{x^* \sim \{0,1\}^m}[x^* \notin \mathsf{BAD}]^\ell \ .\end{aligned}$$

Since $\Pr[I(X)] \geq 1/n^d$ by assumption, it follows that $\Pr[x^* \ \notin \mathsf{BAD}] \geq 1/(2n^d)^{1/\ell} \geq 1 - 1/m^c + e^{-n}$, as needed. □

[4] There is a small subtlety here in that d represents the advantage of our adversary $\mathcal{A}$, but this advantage can depend on ℓ. Formally, we should fix some $\ell > \omega(m^c \log(m))$, e.g., $\ell = m^{c+1} \log(m)$. Then, if our adversary has success probability $1/n^d$ for *any* constant d, this inequality will hold for sufficiently large n.

References

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory Conference*, 1995.