Levin's Universal OWF!

Noah Stephens-Davidowitz

May 29, 2023

1 Universality and combiners

We've now seen a few examples of candidate OWFs—i.e., functions f that are one way if some problem or another is hard. Specifically, we saw multiple one-way functions that are secure if factoring is hard, a one-way function that is secure if the discrete logarithm over $\mathbb{Z}_q^*$ is hard, and a one-way function that is secure if Ajtai's SIS problem is hard. (In all of these cases, we had to be very careful by what we meant by our hardness assumption.) There are many many more examples that we could go through. In particular, I think it's pretty hard to imagine that we live in a world in which OWFs simply don't exist. (Of course, we do not know to *prove* that one-way functions exist, since we do not even know how to prove that $\mathsf{P} \neq \mathsf{NP}$. But, we think they probably do.)

So, I'm pretty comfortable assuming that there exists *some* one-way function. But, it's much easier for me to question whether some *particular* function f actually is one way—even functions f that are commonly assumed to be one-way functions. E.g., maybe factoring is not hard (or maybe factoring *is* hard, but not on the input distribution that we chose). Or, maybe the discrete logarithm is not hard (or maybe it *is* sometimes hard, but not over the group $\mathbb{Z}_q^*$ that we used). This is concerning because if we want to actually use one-way functions for things (e.g., to build a hard puzzle or to build secret-key encryption), we presumably need to pick one. And, what if we choose wrong?

One can defend against this a little bit by using a *combiner*. Combiners come up in different contexts in cryptography and can be applied to different primitives.[1] E.g., a combiner for secret-key encryption is a procedure for converting two or more secret-key encryption schemes $(\mathsf{Gen}_1, \mathsf{Enc}_1, \mathsf{Dec}_1), \ldots, (\mathsf{Gen}_\ell, \mathsf{Enc}_\ell, \mathsf{Dec}_\ell)$ into a single encryption scheme $(\mathsf{Gen}^*.\mathsf{Enc}^*, \mathsf{Dec}^*)$ with the property that $(\mathsf{Gen}^*.\mathsf{Enc}^*, \mathsf{Dec}^*)$ is secure if *at least one* of the schemes $(\mathsf{Gen}_i, \mathsf{Enc}_i, \mathsf{Dec}_i)$ is secure. (Maybe you can see how to do this? There's a rather ugly way called onion encryption, but there's also a much more elegant way that's related to the one-time pad.)

Similarly, a combiner for one-way functions is just a way of combining many one-way functions $f_1, \ldots, f_\ell$ into a single function f^* so that f^* is one way if at least one of the f_i is one way. In other words, if we have some long list of candidate one-way functions $f_1, \ldots, f_\ell$ (and we do!), then we can combine them all together into a single one-way function that will be secure as long as at least one of our candidates is actually one way.

In fact, combining one-way functions is quite easy. Just define $f^*(x_1, \ldots, x_\ell) := (f_1(x_1), \ldots, f_\ell(x_\ell))$ (where $|x_i| = |x_j| = n/\ell$ and we assume for simplicity that the input length is divisible by ℓ—

[1]By the way, I use this word "primitive" a lot. I'm not sure if there's a truly formal definition of a "cryptographic primitive," but it basically just means a "cryptographic thing" that has a formal definition. So, secret-key encryption is a cryptographic primitive. One-way functions are a cryptographic primitive. Etc.

remember that we justified this assumption when we studied efficiently sampleable one-way functions in the previous lecture). In other words, we divide our input up into ℓ equal pieces and run each function on one piece. The fact that this is one way if at least one of the f_i is one way is relatively straightforward. (Much more straightforward than, e.g., our proof that weak one-way functions imply strong one-way functions.) In particular, if we had an adversary that inverted f^* on uniformly random input of length n, then we could use this to invert f_i on uniformly random input of length n/ℓ with essentially the same running time and the same probability. (A formal proof would work via a reduction, where the reduction $\mathcal{A}'$ samples uniformly random $x_j \sim \{0,1\}^{n/\ell}$ for $j \neq i$, and uses these to produce input for a hypothetical adversary $\mathcal{A}$ that breaks f^*, etc. I show such a formal proof in the supplementary notes showing examples of proofs by reduction. We will also see a version of this reduction below as part of the proof of our main theorem. There are also many other details that I hand-waved above. For example, what happens if some of the f_i are not efficiently computable?)

But, in order for this to be useful, we still need to find a list of functions $f_1, \ldots, f_\ell$ with the guarantee that at least one of them is actually one way. One can imagine a world in which one-way functions exist, but no function in our list $f_1, \ldots, f_\ell$ (e.g., no function discussed in this course) is one way.

Of course, for most of this course, we will be perfectly happy assuming things like "factoring is hard," and even making far more audacious claims. But, the purpose of this lecture is to show how to build a *universal* one-way function. That is, a function f_U with the property that *if any function at all is one way, then f_U is one way*!! Over the next few lectures, we will show how to use any one-way function f to build many other cryptographic primitives (and how to build one-way functions from these primitives), so that all of these primitives will also have universal constructions. (Our universal construction will be quite artificial. Levin later showed a universal one-way function that is much more natural [Lev03], but the proof that it is universal is more difficult.)

2 Levin's universal one-way function

The idea behind Levin's universal one-way function f_U [Lev87] is essentially to use the above "combining" idea with "the list of all possible (efficiently computable) functions." As you might guess, making this precise is a bit delicate.

Here's a first try. You first fix an ordering $T_1, T_2, T_3, \ldots,$ of *all* possible computer programs—written as Turing machines or Python programs, or C programs, or in whatever model of computation you like. E.g., you might first iterate through all possible one-character ASCII strings in order, interpreting them as a program written in your favorite programming language, then all two-character strings, and so on. You can view strings that do not compile as corresponding to the computer program that always outputs 0 on any input. (See how artificial this construction is?!) We will assume for simplicity that the T_i are deterministic, and in fact we will only show that f_U is a one-way function if there exists a one-way function that is computable in *deterministic* polynomial time. (This issue is easily fixed.)

We can then set, e.g., $f^*(x_1, \ldots, x_{\sqrt{n}}) = (T_1(x_1), T_2(x_2), \ldots, T_{\sqrt{n}}(x_{\sqrt{n}}))$, which is in fact a function because the T_i are deterministic, where $|x_i| = \sqrt{n}$ and we have assumed without loss of generality that the input length n is a perfect square so that $\sqrt{n}$ is an integer.[2] Then, at least

[2]We can do this without loss of generality by recalling that efficiently sampleable one-way functions are equivalent

intuitively, f^* should be hard to invert if and only if at least one of the T_i is hard to invert. (There is nothing particularly special about the choice of $\sqrt{n}$ here. We could use $T_1, \ldots, T_\ell$ with inputs of length n/ℓ for, e.g., $\ell = n^{1/3}$, or $\ell = n^{99/100}$, or whatever. The important thing to notice is that n and n/ℓ are polynomially related so that a running time is polynomial in n/ℓ if and only if it is polynomial in n, and similarly a success probability ε is negligible in n/ℓ if and only if it is negligible in n.) In particular, if there is *some* machine T_{i^*} that computes some hard-to-invert function f, then eventually $\sqrt{n} \geq i^*$, and T_{i^*} will therefore be included in $T_1, \ldots, T_{\sqrt{n}}$. And, we should expect that f^* is hard to invert. (Here, we are of course relying heavily on asymptotics. In particular, our definition of "hard to invert" is an asymptotic definition, and therefore only makes sense as $n \to \infty$. We therefore do not care about the behavior of the function f^* for inputs of length $n < i^*$, since i^* is some fixed constant.)

The only issue with the above construction is that it won't produce an efficiently computable function f^*. (In fact, in the uniform model of computation, it also might not be secure.) Some of the T_i will *not* run in polynomial time. In fact, some of the T_i will never halt at all! (Actually, it's not even clear what I mean when I write $T_i(x)$ if T_i does not halt on input x! Most ways of making this precise would yield an f^* that is not computable *at all*, let alone *efficiently* computable!)

The solution to this issue is to define the function

$$T_i^{t(n)}(x) := \begin{cases} T_i(x) & T_i \text{ halts in time at most } t(|x|) \text{ on input } x \\ 0 & \text{otherwise} \end{cases}$$

In other words $T_i^{t(n)}$ outputs whatever T_i outputs, unless it takes too long, in which case it outputs 0. Notice that by definition $T_i^{t(n)}$ is computable in roughly $t(n)$ time (maybe a little more, like $O(t(n)\log t(n))$ to account for the overhead of keeping track of the running time). In particular, if $t(n)$ is a polynomial, then $T_i^{t(n)}$ is computable in polynomial time. Notice also that the value taken by $T_i^{t(n)}$ is very sensitive to our particular model of computation and our particular definition of running time. However, our proof that this construction works will not care about the model (as long as certain simple functions are computable in a reasonable amount of time in this model).

We will just take $t(n) = n^{10}$ from now on for convenience, and we will define our actual universal one-way function to be

$$f_U(x_1, \ldots, x_{\sqrt{n}}) := (T_1^{n^{10}}(x_1), \ldots, T_{\sqrt{n}}^{n^{10}}(x_{\sqrt{n}})) .$$

for $|x_i| = \sqrt{n}$. (If one is very careful, then one can replace $T_i^{n^{10}}$ by $T_i^{n^2}$, but we do not want to be very careful.) Then, clearly f_U can be computed in time at most, say, $O(n^{11})$.

Let's first prove a theorem that is *almost* as good as what we want. Indeed, this is exactly the theorem that we want except that it only guarantees that f_U is one way if there is a one-way function that is computable in some *fixed* polynomial time, $O(n^9)$. It does not immediately tell us anything about what happens if, e.g., there is a one-way function that is computable in time $\Theta(n^{11})$ or $\Theta(n^{100})$.

Theorem 2.1. *f_U is a one-way function if there exists* any *one-way function f that is computable in time $O(n^9)$ on inputs of length n (in whatever model of computation we're using to define T_i^t).*

to one-way functions. In particular, we can run f^* with the sampling algorithm that takes as input 1^n and outputs a uniformly random bit string of length $\lfloor\sqrt{n}\rfloor^2$—i.e., "n rounded down to the nearest perfect square." Essentially equivalently, we can always replace f^* by f^{**}, which takes strings of arbitrary length and simply ignores any input bits after the first $\lfloor\sqrt{n}\rfloor^2$ bits.

Proof. The proof is just a more formal version of the high-level discussion above. We have already shown that f_U is efficiently computable, so we need only prove that it is hard to invert. We may assume that there is a one-way function f that is computable in time $O(n^9)$. Then, there exists some index i^* such that T_{i^*} is a machine that (1) runs in time $O(n^9)$ on inputs of length n; and (2) computes f, i.e., $T_{i^*}(x) = f(x)$.

Let n_0 be large enough so that $n_0 \geq i^*$ and so that the running time of T_{i^*} on input $n \geq n_0$ is less than n^{10}. Therefore, so that $T_{i^*}^{n^{10}}(x) = f(x)$ for all $x \in \{0,1\}^n$. (Since T_{i^*} runs in time $O(n^9)$, there must exist such an n_0.)

Then, for $x \in \{0,1\}^n$ with $n \geq n_0$, we have $f_U(x_1, \ldots, x_{\sqrt{n}}) = (\ldots, f(x_{i^*}), \ldots)$.

Now, suppose for contradiction that there exists a polynomial-time adversary $\mathcal{A}$ such that

$$\varepsilon(n) := \Pr_{x_1,\ldots,x_{\sqrt{n}} \sim \{0,1\}^{\sqrt{n}}}[(x'_1, \ldots, x'_{\sqrt{n}}) \leftarrow \mathcal{A}(1^n, f_U(x_1, \ldots, x_{\sqrt{n}})) \ : \ f_U(x') = f_U(x)]$$

is non-negligible.

We construct $\mathcal{A}'$, an adversary in the one-way function game against f as follows. $\mathcal{A}'$ takes as input 1^m and $y^* = f(x^*)$, where $x^* \sim \{0,1\}^m$. If $m < n_0$, $\mathcal{A}'$ simply gives up (e.g., outputs 0). Otherwise, for $j = 1, \ldots, m$ with $j \neq i^*$, $\mathcal{A}'$ samples $x_j \sim \{0,1\}^m$ uniformly at random and sets $y_j := T_j^{n^{10}}(x_j)$. It also sets $y_{i^*} := y^*$. It then calls $\mathcal{A}$ on input 1^{m^2} and $(y_1, \ldots, y_m)$, receiving as output $x'_1, \ldots, x'_m$. Finally, it outputs x'_{i^*}.

Clearly $\mathcal{A}'$ is PPT. Notice that

$$\Pr[f(x'_i) = y^*] \geq \Pr[f(x'_1) = y_1, f(x'_2) = y_2, \ldots, f(x'_m) = y_m] = \varepsilon(m^2) ,$$

where the last equality holds because the y_j are independent and distributed as $f(x_j)$ for $x_j \sim \{0,1\}^m$ (including the case $j = i^*$). Therefore,

$$\varepsilon'(m) := \Pr_{x^* \sim \{0,1\}^n}[x' \leftarrow \mathcal{A}(1^m, x^*) \ : \ f(x') = f(x^*)] \geq \varepsilon(m^2) ,$$

which is non-negligible in m. This contradicts the assumption that f is one way. Therefore, no such $\mathcal{A}$ can exist, i.e., f_U is one way. □

Finally, it remains to remove the restriction that our one-way function f must run in time $O(n^9)$. To do so, we use a simple trick. In particular, we make our function more efficient by ignoring most of the input.

Theorem 2.2. *If there exists a one-way function, then there exists a one-way function that is computable in time $O(n^9)$ (for any "reasonable" model of computation).*

Proof. Let f be a one-way function. By assumption, f is computable in polynomial time (in our model of computation). So, f is computable in time $O(n^C)$ for some fixed $C \geq 1$. Let $f'(x)$ be defined as follows for $x \in \{0,1\}^m$. Let $n := \lfloor m^{1/C} \rfloor$, and let $x = (x_1, \ldots, x_m)$ be the individual bits of x. Then, $f'(x) = f(x_1, \ldots, x_n)$.

Notice that f' is computable in $O(m^9)$ time on inputs of length m (in any reasonable model of computation, e.g., any model of computation in which $\lfloor m^{1/C} \rfloor$ can be computed in $O(m^9)$ time for any constant $C \geq 1$).

So, we only need to show that f' is one way (assuming that f is one way). To that end, we suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ such that

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

is non-negligible. We then construct an adversary $\mathcal{A}'$ in the one-way function game against f as follows. $\mathcal{A}'$ takes as input 1^n and $y^* := f(x^*)$ where $x^* \sim \{0,1\}^n$. It then runs $\mathcal{A}$ on input 1^{n^C} and y^*, receiving as output $x' \in \{0,1\}^{m'}$. Finally, $\mathcal{A}'$ outputs the first $\lfloor (m')^{1/C} \rfloor$ bits of x'. $\square$

References

[Lev87] Leonid A. Levin. One way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, December 1987. 2

[Lev03] L. A. Levin. The tale of one-way functions. *Problems of Information Transmission*, 39(1):92–103, January 2003. 2