1 One-Way Functions

Recall our formal definition of a one-way function:

Definition 1.1. A function $f : \{0,1\}^* \to \{0,1\}^*$ is *one-way* if it satisfies the following conditions.

- *Easy to compute.* There is an efficient algorithm computing f. Formally, there exists a (uniform, deterministic) poly-time algorithm F such that $\mathsf{F}(x) = f(x)$ for all $x \in \{0,1\}^*$.
- *Hard to invert.* An efficient algorithm inverts f on a random input with only negligible probability. Formally, for any non-uniform PPT algorithm $\mathscr{I}$, the *advantage* of $\mathscr{I}$

$$\mathbf{Adv}_f(\mathscr{I}) := \Pr_{x \leftarrow \{0,1\}^n}\left[\mathscr{I}(1^n, f(x)) \in f^{-1}(f(x))\right] = \mathrm{negl}(n)$$

  is a negligible function (in the security parameter n).

We also allowed a syntactic relaxation where the domain and range (and the input distribution over the domain) may be arbitrary, as long as the input distribution can be sampled efficiently (given the security parameter n).

1.1 Candidates

At the end of last lecture we defined two functions, with the idea of investigating whether they could be one-way functions:

1. Subset-sum: define $f_{\mathrm{ss}} : (\mathbb{Z}_N)^n \times \{0,1\}^n \to (\mathbb{Z}_N)^n \times \mathbb{Z}_N$, where $N = 2^n$, as

$$f_{\mathrm{ss}}(a_1, \ldots, a_n, b_1, \ldots, b_n) = (a_1, \ldots, a_n, S = \sum_{i\,:\,b_i = 1} a_i \bmod N).$$

   Notice that because the a_is are part of the output, inverting f means finding a subset of $\{1, \ldots, n\}$ (corresponding to those b_is equaling 1) that induces the given sum S modulo N.

2. Multiplication: define $f_{\mathrm{mult}} : \mathbb{N}^2 \to \mathbb{N}$ as[1]

$$f_{\mathrm{mult}}(x,y) = \begin{cases} 1 & \text{if } x = 1 \vee y = 1 \\ x \cdot y & \text{otherwise.} \end{cases}$$

First consider f_{ss}; clearly it can be computed in polynomial time. The subset-sum problem is a famous NP-complete problem, hence if $\mathsf{P} \neq \mathsf{NP}$ there is no poly-time algorithm to solve it, and f_{ss} must be one-way, right? Not exactly: subset-sum is NP-complete in the *worst case*, but for f_{ss} to be one-way we need it be hard on the *average*. Nevertheless, many believe that this form of subset-sum is indeed average-case hard, and hence f_{ss} is conjectured to be one-way.

Now consider f_{mult}; again, it can be computed in poly-time. Inverting f_{mult} means factoring xy (into non-trivial factors), and factoring is hard, right? Again, we need to be more careful: consider the following inverter $\mathscr{I}(z)$: if z is even, then output $(2, z/2)$, otherwise fail. Notice that $\mathscr{I}$ outputs a valid preimage of

[1] For security parameter n, we use the domain $[1, 2^n] \times [1, 2^n]$. The cases $x = 1$ and $y = 1$ are special to rule out the trivial preimages $(1, xy)$ and $(xy, 1)$ for the output $xy \neq 1$.

z exactly when $z = xy$ is even, which occurs with probability $3/4$ when x, y are chosen independently and uniformly from $[1, 2^n]$.

The problem here is that while *some* natural numbers seem hard to factor, a large fraction of them (e.g., those that are even) certainly are not. There are two common ways to deal with this. The first is to "tweak" the function (actually, its input distribution) to focus on the instances that we believe to be hard. For the factoring problem, it is believed (but not proved!) that the hardest numbers to factor are those that are the product of exactly two primes of about the same size. More specifically, define

$$\Pi_n = \{p \ : \ p \in [1, 2^n] \text{ is prime}\}$$

to be the set of "n-bit prime numbers." Then we may make the following plausible conjecture:

Conjecture 1.2 (Factoring Assumption)**.** For every non-uniform PPT algorithm $\mathscr{A}$,

$$\Pr_{p,q \leftarrow \Pi_n}[\mathscr{A}(p \cdot q) = (p, q)] = \text{negl}(n).$$

Let's next change the input distribution of f_{mult} so that x, y are chosen uniformly and independently from Π_n. Now, the one-wayness property for f_{mult} is syntactically identical to Conjecture 1.2, so for that property we have a "proof by assumption." The only remaining question is the distribution over inputs: can we efficiently sample a random n-bit prime, i.e., sample uniformly from Π_n? Algorithm 1 gives a potential algorithm, GenPrime, to do so.

Algorithm 1 Algorithm $\mathsf{GenPrime}(1^n)$ for sampling a uniformly random n-bit prime.

1: Choose a fresh uniformly random integer $p \in [1, 2^n]$.
2: If p is prime, output p. Otherwise start over.

Clearly, if GenPrime terminates, it outputs an element of Π_n. Moreover, it outputs a *uniformly random* element of Π_n, because every $\bar{p} \in \Pi_n$ is equally likely to be chosen in the first step. So the only remaining question is the running time: can we test primality efficiently? And can we guarantee that the algorithm terminates quickly, i.e., does not start over too many times?

The first question has many answers. The definitive answer was provided in 2002 by Agrawal, Kayal, and Saxena, who gave a *deterministic* primality test that runs in time polynomial in the bit length of the number (i.e., $\text{poly}(n)$ for a number in $[1, 2^n]$). The running time, while polynomial, is quite large, so the AKS test is not used in practice. Instead, *randomized* tests such as Miller-Rabin are used to quickly test primality; however, such tests have a *very small* probability of returning the wrong answer ("prime" when the number is composite, or vice versa, or both). This introduces a tiny bias into the output distribution of GenPrime, which we can fold into a negligible error term that we won't need to worry about.[2] (Later on we will rigorously define what "negligible error term" means. We may study primality tests in more detail.)

The second question, about the running time of GenPrime, is also very interesting. The probability that a particular iteration terminates is exactly $|\Pi_n|/2^n$ (assuming a perfect primality test). How does $|\Pi_n|$ compare to 2^n, i.e., how "dense" are the n-bit primes? It is easy to show that there are an infinite number of primes, but this is not strong enough for our purposes — primes could be so vanishingly rare that GenPrime effectively runs forever. Fortunately, it turns out that the primes are quite dense. For an integer $N > 1$, let $\pi(N)$ denote the number of primes not exceeding N. We have the following:

[2] Another alternative is to use *deterministic* primality tests which only have proofs of correctness under unproved (but widely-believed) number-theoretic assumptions. These are less common in practice because they aren't so fast.

Lemma 1.3 (Chebyshev)**.** *For every integer $N > 1$, $\pi(N) > \frac{N}{2\log_2 N}$.*

(In fact, this is a weaker form of the "prime number theorem," which says that $\pi(N)$ asymptotically approaches $\frac{N}{\ln N}$. We will not give a proof of Lemma 1.3 now, but there is an elementary one in the Passhelat notes.) Using Lemma 1.3, we have

$$\frac{|\Pi_n|}{2^n} = \frac{\pi(2^n)}{2^n} > \frac{2^n}{2^n \cdot 2\log_2(2^n)} = \frac{1}{2n}.$$

Finally, since all the iterations of GenPrime are independent, the probability that it fails to terminate after (say) $2n^2$ iterations is

$$(1 - \tfrac{1}{2n})^{2n \cdot n} \leq (1/2)^n,$$

which is exponentially small.[3]

1.2 Weak OWFs and Hardness Amplification

The second way of dealing with the above difficulty of "somewhere hard" functions is more general, but also most costly and technically more complex. Suppose we were not confident where the "hard" inputs to a OWF were likely to be, but we did believe that they weren't too rare. We would then have a δ-*weak* one-way function, where the "hard to invert" property from Definition 1.1 is relaxed to the following:

- For any non-uniform PPT algorithm $\mathscr{I}$,

$$\mathbf{Adv}_f(\mathscr{I}) := \Pr_{x \leftarrow \{0,1\}^n}\left[\mathscr{I}(f(x)) \in f^{-1}(f(x))\right] \leq 1 - \delta,$$

  for some $\delta = \delta(n) = 1/\operatorname{poly}(n)$.

The only change is that we have replaced the $\operatorname{negl}(n)$ success probability by $1 - \delta$. This definition allows for f to be easily invertible for a large fraction of inputs (e.g., all the even numbers), but not "almost everywhere:" a δ-fraction of the inputs remain hard.

Suppose that f is a δ-weak OWF. Can we get a (strong) OWF from it? The answer is yes; this kind of result lies within the area called *hardness amplification* (for obvious reasons). Assume that we have a function f; then define its m-wise *direct product*

$$f'(x_1, \ldots, x_m) = (f(x_1), \ldots, f(x_m)),$$

where we parse the input so that $|x_1| = \cdots = |x_m|$. We have the following theorem:

Theorem 1.4. *If f is a δ-weak one-way function, then f' is a (strong) one-way function for any* $\operatorname{poly}(n)$*-bounded $m \geq 2n/\delta$.*

Proof idea: The intuition here is that inverting f' requires inverting f on m independently chosen values. If each component is inverted independently with probability at most $1 - \delta$, then the probability of inverting all of them should be at most

$$(1 - \delta)^{2n/\delta} \approx (1/e)^{2n} = \operatorname{negl}(n).$$

[3] To be completely formal, in order to say that GenPrime is polynomial-time, we should modify it so that if it has not terminated after $2n^2$ iterations, it just outputs some fixed value (say, 1). This introduces only an exponentially small bias in its output distribution.

Rigorously proving that this intuition is correct, however, turns out to be quite subtle. The reason is that an adversary $\mathscr{I}'$ attacking f' does not have to treat each of the inversion sub-tasks $y_1 = f(x_1), \ldots, y_m = f(x_m)$ independently. For example, it might look at all of the y_is at once, and either invert *all* of them (with some non-negligible probability overall), or none of them. So the individual events (that $\mathscr{I}'$ succeeds in the sub-task of inverting $y_i = f(x_i)$) can be highly correlated.

To give a correct proof of the theorem, we need to perform a *reduction* from inverting f with advantage $> 1 - \delta$ to inverting f' with some non-negligible advantage. Unpacking this a bit, we want to prove the contrapositive: suppose there is some (efficient) adversary $\mathscr{I}'$ that manages to break the (strong) one-wayness of f', i.e., it inverts with some non-negligible advantage $\mathbf{Adv}_{f'}(\mathscr{I}') = \alpha = \alpha(n)$. Then we will construct another (efficient) adversary $\mathscr{I}$ that manages to break the δ-weak one-wayness of f, i.e., it inverts f with advantage $\mathbf{Adv}_f(\mathscr{I}) > 1 - \delta$. In doing so, we will use $\mathscr{I}'$ as a "black box," invoking it on inputs of our choice and using its outputs in whatever way may be useful. However, we cannot assume that $\mathscr{I}'$ "works" in any particular way; we can only rely on the hypothesis that it violates the strong one-wayness of f'.

The basic idea of the reduction is this: $\mathscr{I}$ is given $y = f(x)$ and wants to use $\mathscr{I}'$ to invert y. An obvious first attempt is to "plug in" the value y as part of the output of f', i.e., to run $\mathscr{I}'$ on

$$y' = (y_1 = y = f(x), y_2 = f(x_2), \ldots, y_m = f(x_m)) = f'(x_1 = x, x_2, \ldots, x_m)$$

for some random $x_2, \ldots, x_m$ chosen by $\mathscr{I}$. When $\mathscr{I}'$ produces a (candidate) preimage $x' = (x'_1, \ldots, x'_m)$, then $\mathscr{I}$ just outputs x'_1 as its candidate preimage of $y = f(x)$. Notice that if $\mathscr{I}'$ happens to invert y' successfully, then in particular $x_1 \in f^{-1}(y)$ as required.

Unfortunately, this simple idea does not quite work. The problem is that $\mathscr{I}'$ inverts only with non-negligible probability α, whereas we need $\mathscr{I}$ to invert with (typically larger) probability $> 1 - \delta$, to violate the weak one-wayness of f. We might hope to fix this problem by running $\mathscr{I}'$ a large number of times, using the same $y_1 = f(x)$ but *fresh* random $x_2, \ldots, x_m$ each time, expecting $\mathscr{I}'$ to successfully invert at least once. (Note that we can check whether $\mathscr{I}'$ succeeded in inverting f' on a particular tuple, because f' is efficiently computable.) This strategy is better, but still flawed: the problem is that the runs are *not independent*, since we use the same $y_1 = f(x)$ every time. Indeed, $\mathscr{I}'$ might behave in a "sneaky" way to make this very strategy fail. For example, $\mathscr{I}'$ might fix a certain α-fraction of "good" values of x_1 for which it always inverts (ignoring $x_2, \ldots, x_m$), and fail on *all* other values of x_1. Notice that such $\mathscr{I}'$ has overall advantage α, as required. But no matter how many time we repeatedly run $\mathscr{I}'$ with $y_1 = y = f(x)$, we will eventually succeed only if the original x is "good," which happens with probability only α.

A better reduction (which will turn out to work) just applies the above "plug-and-repeat" strategy for *each* position $i = 1, \ldots, m$. As it turns out, we can show that *some* position must have a very large ($> 1 - \delta/2$) fraction of "good" x_i's, for a suitable definition of good. Conditioned on the x from our given instance $y = f(x)$ being good, by repeatedly calling $\mathscr{I}'$ a polynomial number of times (using fresh $x_2, \ldots, x_m$ each time), the probability that $\mathscr{I}'$ inverts successfully in at least one call will be extremely close to 1, so our overall advantage in inverting f will be $\approx 1 - \delta/2 > 1 - \delta$, as desired.

We now give the formal proof.

Proof of Theorem 1.4. Because $m = \text{poly}(n)$ by assumption (and because $2n/\delta = 2n \cdot \text{poly}(n) = \text{poly}(n)$), if f is efficiently computable, then so is f'. Now suppose that some efficient adversary $\mathscr{I}'$ violates the one-wayness of f', with non-negligible advantage $\alpha = \alpha(n) = \mathbf{Adv}_{\mathscr{I}'}(f')$. We wish to construct an efficient adversary $\mathscr{I}$ that uses $\mathscr{I}'$ (efficiently) to violate the δ-weak one-wayness of f.

We first establish a crucial property of $\mathscr{I}'$ that will help us obtain the desired reduction. Define the "good" set for position i to be the set of all x_i such that $\mathscr{I}'$ successfully inverts with "good enough" proba-

bility, over the random choice of all the remaining x_j. Formally,

$$G_i = \left\{ x_i \ : \ \Pr_{x_j \text{ for } j \neq i} \left[\mathscr{I}'(f'(x')) \text{ succeeds} \right] \geq \frac{\alpha}{2m} \right\}.$$

Note that because $\mathscr{I}'$ succeeds with at least α probability overall, it is easy to see that G_i is at least $(\alpha/2)$-dense for every i. However, we will show a very different fact: that G_i is at least $(1-\delta/2)$-dense for *some* i. (Note that typically, $(1-\delta/2)$ is much larger than α.)

Claim 1.5. *For some i, we have $\frac{|G_i|}{2^n} \geq (1-\delta/2)$.*

Proof. Suppose not. Then we have

$$\begin{aligned}
&\Pr_{x'}[\mathscr{I}'(f'(x')) \text{ succeeds}] \\
&\leq \Pr[\mathscr{I}' \text{ succeeds} \wedge \text{every } x_i \in G_i] + \sum_{i=1}^{m} \Pr[\mathscr{I}' \text{ succeeds} \wedge x_i \notin G_i] && \text{(union bound)} \\
&\leq \Pr[\text{every } x_i \in G_i] + \sum_{i=1}^{m} \Pr[\mathscr{I}' \text{ succeeds} \mid x_i \notin G_i] && \text{(probability)} \\
&< (1-\tfrac{\delta}{2})^{2n/\delta} + m \cdot \tfrac{\alpha}{2m} && \text{(def \& size of } G_i) \\
&\leq 2^{-n} + \alpha/2 < \alpha,
\end{aligned}$$

contradicting our assumption on $\mathscr{I}'$. □

The remainder of the proof is a simple exercise, following the "plug-and-repeat" strategy described above, and using Claim 1.5. □