A Note on the Non-Uniform Model of Computation

1 Non-Uniform Turing Machines and Circuits

Recall that our goal is to construct cryptographic schemes that are secure against "poly-size" adversaries, as opposed to "poly-time" adversaries.

Definition 1 (Poly-size (non-uniform) Turing machine)**.** *A poly-size (non-uniform) Turing machine is a polynomial-time Turing machine M augmented with an infinite collection of advice strings $\{a_n\}_{n\in\mathbb{N}}$ of polynomial size (i.e. $|a_n| = O(n^d)$ for some constant c). On input $x \in \{0,1\}^n$, the Turing machine is allowed to utilize the advice corresponding to its input length, and produce output $M(x, a_n)$.*

The set of languages decided by poly-size (non-uniform) Turing machines is denoted by $\mathsf{P/poly}$.

An equivalent way to formalize the non-uniform model of computation is using circuits. Circuits by definition operate on inputs of a fixed length, so to decide a language $L \subseteq \{0,1\}^*$, we need a *family* of circuits $\{C_n\}_{n\in\mathbb{N}}$, one circuit C_n for every input length $n \in \mathbb{N}$.

To see why these two models are equivalent, think of of the circuit C_n for input length n as the advice a_n given to the non-uniform Turing machine when operating on inputs of length n.

2 The Power of Non-Uniform Computation

From the non-uniform Turing machine definition, it is easy see that $\mathsf{P} \subseteq \mathsf{P/poly}$; the non-uniform Turing machine to decide $L \in \mathsf{P}$ just runs the uniform Turing machine for L on the empty advice string for all input lengths.

Does non-uniform advice buy us any additional power? Well, for one, unlike the case for P, we actually know that $\mathsf{BPP} \subseteq \mathsf{P/poly}$.

2.1 Randomness is (provably) for free

Theorem 1 (Adleman, Bennett, Gill)**.** $\mathsf{BPP} \subseteq \mathsf{P/poly}$.

Proof. For any language $L \in \mathsf{BPP}$, we will give a $\mathsf{P/poly}$ machine that decides L. Let M be a BPP machine such that for every $x \in L$,

$$\Pr_r[M(x;r) = 1] \geq 2/3$$

and for every $x \notin L$,

$$\Pr_r[M(x;r) = 1] \leq 1/3.$$

Let M' be the Turing machine that samples m independent random strings $r_1, \ldots, r_m$ and outputs the majority vote of $M(x; r_i)$. By the Chernoff bound, for every $x \in \{0,1\}^n$, by choosing $m = \Omega(n)$, we can get that for every $x \in L$

$$\Pr_{r_1,\ldots,r_m}[M'(x; r_1, \ldots, r_m) \neq 1_{x\in L}] < \frac{1}{e^n}.$$

where $1_{x \in L} = 1$ if $x \in L$ and 0 otherwise. By a union bound over all $x \in \{0,1\}^n$,

$$\Pr_{r_1,\ldots,r_m}[\exists x : M'(x; r_1, \ldots, r_m) \neq 1_{x \in L}] < \frac{2^n}{e^n} < 1.$$

Therefore, there must exist some choice of $R_n^* = (r_1^*, \ldots, r_m^*)$ such that $M'(x; R_n^*) = 1_{x \in L}$ for all $x \in \{0,1\}^n$. Thus, M' is a non-uniform Turing machine with advice string R_n^* for input length n that decides L. □

This, on its own, is not a convincing argument that $\mathsf{P/poly}$ is stronger than P, since we believe that $\mathsf{BPP} = \mathsf{P}$. Next, we will show that in fact, $\mathsf{P/poly}$ is strictly larger than P—it contains the unary versions of *undecidable* languages.

2.2 Undecidable problems in $\mathsf{P/poly}$

Define the Unary Halting problem by the language

$\mathsf{UHALT} = \{1^n \mid$ The binary decomposition of n can be interpreted as $\langle M, w \rangle$ such that M halts on input $w\}$.

Theorem 2. $\mathsf{UHALT} \in \mathsf{P/poly}$.

Proof. For $n \in \mathbb{N}$, let $a_n = 1$ if M halts on w and 0 otherwise, where $\langle M, w \rangle = \textsf{binary-decomposition}(n)$. Define the non-uniform Turing machine M' as follows:

$$M'(x) = \begin{cases} a_{|x|} & \text{if } x = 1^{|x|} \\ 0 & \text{otherwise} \end{cases}$$

Therefore M' decides UHALT. □

Remark 1. *Another way to see that non-uniform computation is much more powerful than uniform computation is to observe that the number of uniform Turing machines is countably infinite, but the number of non-uniform Turing machines is uncountably infinite.*

2.3 Wait, is this even reasonable?

The fact that some non-uniform adversaries can solve undecidable problems might lead to some skepticism about considering such a model. What hope can we have of finding assumptions that might reasonably be hard against potentially (much more powerful) adversaries?

It is here that we note that while non-uniform programs are a strictly stronger model of computation than uniform ones, we don't think that such adversaries can necessarily solve all *interesting* hard problems.

For those who may have more complexity theory background, there is some indirect evidence for this viewpoint: a result of Karp and Lipton shows that if $\mathsf{NP} \subseteq \mathsf{P/poly}$, then the polynomial hierarchy collapses (which is widely believed *not* to be the case).

Theorem 3 (Karp-Lipton)**.** $\mathsf{NP} \subseteq \mathsf{P/poly} \implies \Sigma_2^P = \Pi_2^P$.

Thus, we believe that at the very least, not *all* hard problems are necessarily easy for non-uniform adversaries.[1] And of course, heuristically, most of our cryptographic assumptions have withstood decades of cryptanalysis and suggested attacks, even against non-uniform adversaries.

[1]Note that most cryptographic problems are not NP-complete, so Karp-Lipton does not directly apply, but it still gives some justification as to why considering non-uniform adversaries is not wholly unreasonable.

3 Why do we care about non-uniform adversaries in cryptography?

As we have seen above, the non-uniform model of computation is strictly more powerful than the uniform model of computation, so if we can prove a cryptographic scheme is secure against more powerful non-uniform adversaries, we should do it! This is why nearly all the definitions that cryptographers write down allow for non-uniform adversaries.

But as we have also seen, the non-uniform model of computation can at times be unreasonably powerful. So why are cryptographers so diligent about making their schemes secure against non-uniform adversaries? Does it ever matter in practice? A concrete way to phrase this question is as follows: is there a cryptosystem that is secure when considering uniform adversaries that suddenly becomes insecure in when considering non-uniform adversaries?

Non-uniformity captures an important aspect of adversaries in the real-world—namely, *precomputation.* If it was possible for and adversary to spend a lot of resources and time into a precomputation/pre-processing step that then made solving a large set of instances of a problem easy, then a cryptosystem that relied on the hardness of instances from this set would be broken. Non-uniformity tries to capture this kind of precomputation attack when the set of problems made easy is the set of all inputs of a certain length.

For example, suppose a scheme relies on the hardness of computing the discrete log over a given group G. Namely, the assumption is that given g^x it is hard to compute x. Suppose an adversary performs a lot of precomputation on the specific group G, that makes the discrete log problem in that group easy. That is, after this precomputation step on G, given an instance g^x, the adversary is able to quickly find x. This would completely break the security of the scheme. In fact, this is not a hypothetical attack! The Number Field Sieve algorithm was used to do exactly this for Diffie-Hellman key exchange when the group was chosen to be a fixed prime order!

4 A final remark

Lastly, we want to emphasize a distinction between the uniformity of the *security reduction* in our proofs and the uniformity of the adversaries we consider.

Namely, oftentimes when proving that a certain cryptographic protocol or primitive is secure assuming some assumption (or another primitive), our proof goes as follows: we take a (supposed) adversary $\mathcal{A}$ for the construction and use it to build another adversary $\mathcal{B}$ that breaks the underlying primitive or assumption, thereby reducing the security of our construction to the security of the assumption or simpler primitive.

This reduction may have the property that for *uniform* adversaries $\mathcal{A}$, the resulting adversary $\mathcal{B}$ is also uniform; in this case, we say that the *reduction itself is uniform.* **However, although the reduction itself is uniform, it still applies to non-uniform adversaries.** Indeed, it is not hard to see that any uniform reduction (which is proven only with respect to uniform adversaries) will also convert non-uniform adversaries $\mathcal{A}$ to non-uniform adversaries $\mathcal{B}$, so uniform reductions are at least as strong as non-uniform reductions. Moreover, most natural cryptographic reductions (including the ones in this course) will usually be uniform.[2]

Thus, when describing security reductions, it may be preferable to try to construct uniform security reductions when possible (and we may even phrase our reductions in this language if it makes the proof simpler). Still, this is merely an additional property that may be convenient but not necessary for our reductions to have, and may not always even be possible![3]

[2]There are also specific circumstances where we might need uniform security reductions (for example if we are focusing only on uniform adversaries), and sometimes, our impossibility results can only rule out the more restricted category of uniform reductions.

[3]Indeed, there are certain situations where a non-uniform security reduction is necessary or simpler (and a uniform reduction will provably fail or is much more difficult to construct). As an example, working with uniform adversaries when dealing with zero-knowledge proofs (a concept we will encounter later in this course) is often quite tricky.