Random oracles and oracle worlds!

Noah Stephens-Davidowitz

November 26, 2024

1 Oracles

1.1 Crypto is hard

Cryptography is hard :(. In particular, it is extremely hard to prove things in cryptography (at least for humans)—so hard that we rarely ever actually prove anything *unconditionally*. Instead, we usually simply make some "reasonable assumption"—that factoring is hard or that one-way functions exist or whatever—and prove whatever we want to prove *under this assumption*.

One can argue that this simply comes with the territory—that there is something inherently difficult about cryptography relative to other similar fields. When we study, say, algorithms, we usually only wish to find some specific algorithm and show that it solves some specific problem in some specific running time. This can already be quite difficult! In cryptography, we typically want to prove that *all* efficient algorithms fail to solve some problem! Such proofs are typically far beyond our current technology, so we cheat and use computational assumptions.

1.2 So let's do something else instead—oracles

But, there is an alternative! What if we change our model of computation? In particular, we have so far been interested in probabilistic polynomial-time algorithms. And, what we mean by this are Turing machines or Python programs that may flip coins and that run in polynomial time. But, why limit ourselves to models of computation that actually exist in the real world?

What if we imagine that Python has a new built-in function $\mathcal{O} : \{0,1\}^* \to \{0,1\}^*$? For the purpose of calculating running time, we think of this function as costing just one time step—just as fast as, e.g., taking the XOR of two bits. Will PPT algorithms with this new additional functionality be more powerful than PPT algorithms without it?

Of course, if $\mathcal{O}$ is efficiently computable in our original model, then this new functionality will not buy us much. E.g., if $\mathcal{O}$ multiplies two large numbers together or determines whether an integer is prime or verifies that a graph has been properly 3-colored, then the set of problems that are solvable by a PPT algorithm will not change if we give them access to $\mathcal{O}$.

But, what if $\mathcal{O}$ does something more interesting? There are lots of interesting things that we could imagine $\mathcal{O}$ doing. E.g., maybe $\mathcal{O}$ solves the halting problem! (Of course, we could never actually design a Python function that solves the halting problem, but we can certainly imagine one and prove things about it.) Or, less ambitiously, maybe $\mathcal{O}$ factors large integers instantly, or instantly solves some NP-complete problem.

We call such a function $\mathcal{O}$ an *oracle*—I suppose because we think of $\mathcal{O}$ as some sort of divine entity that answers our questions, like the oracles in ancient Greece. When a program runs $\mathcal{O}$,

we call it an *oracle query*. An algorithm with access to an oracle $\mathcal{O}$ is typically written with a superscript $\mathcal{A}^{\mathcal{O}}$. And, we call a world in which all algorithms have access to an oracle an *oracle world*. The boring world without oracles (the world in which we live) is called *the standard model*.

We have, of course, already seen this notation in the context of oracle-based security definitions. But, we can go even further if we want. We can ask about whole complexity classes. E.g., $\mathsf{P}^{\mathcal{O}}$ is the set of all problems solvable by any polynomial-time algorithm $\mathcal{A}^{\mathcal{O}}$ *with access to the oracle* $\mathcal{O}$. And, $\mathsf{NP}^{\mathcal{O}}$ is the set of all problems with solutions that are verifiable in polynomial time. And, we can ask, does $\mathsf{P}^{\mathcal{O}} = \mathsf{NP}^{\mathcal{O}}$? Of course, in the standard model, this is a question that we famously do not know how to answer. But, as it turns out, this question is *sometimes* easy to answer in oracle worlds! It also turns out that the answer depends on $\mathcal{O}$! In other words, there are some oracle worlds in which polynomial-time algorithms can solve all polynomial-time verifiable problems, $\mathsf{P}^{\mathcal{O}} = \mathsf{NP}^{\mathcal{O}}$, and some oracle worlds in which they cannot, $\mathsf{P}^{\mathcal{O}} \neq \mathsf{NP}^{\mathcal{O}}$! And, we can even prove this!

As cryptographers, we are of course interested in cryptographic questions in oracle worlds. For example, we can define a one-way function in an oracle world as follows.

Definition 1.1. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a* one-way function under the oracle $\mathcal{O} : \{0,1\}^* \to \{0,1\}^*$ *if it satisfies the following two criteria.*

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

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

Notice that in the above definition, we have augmented *all* PPT algorithms with the oracle $\mathcal{O}$. I.e., both the adversary $\mathcal{A}$ and the algorithm computing the function $\mathcal{B}$ get access to the oracle $\mathcal{O}$.

Now, we can ask whether one-way functions exist in the world with oracle $\mathcal{O}$. And, just like with the $\mathsf{P}^{\mathcal{O}} = \mathsf{NP}^{\mathcal{O}}$ question, the answer depends on the oracle. Sometimes the answer is yes, sometimes it is no, and sometimes we do not know the answer.

In fact, essentially all of the questions that originally interested us in the standard model are interesting in oracle worlds as well. We can define commitment schemes, coin flipping, hardcore predicates, PRGs, PRFs, secret-key encryption, collision-resistant hash functions, MACs, signatures, public-key encryption, zero-knowledge proofs, two-party computation, etc., all in our oracle world. And we can ask which of these primitives exist and which don't.

1.3 Random oracles

We will focus primarily on *random oracles*. In particular, let $\mathcal{H} := \{f : \{0,1\}^* \to \{0,1\}^* : \forall x,\ |f(x)| = |x|\}$ be the set of all length-preserving functions. (The fact that we restrict our attention to length-preserving functions is not very important. It is just a convenient choice. We could also take functions that only output a single bit, or that double or halve the length of their input, or whatever.) Then, we can consider a *random* function $H \sim \mathcal{H}$. In other words, we imagine that we live in a world in which at the beginning of time someone sampled a uniformly random function $H \sim \mathcal{H}$, and then all algorithms were given oracle access to H. (In particular, notice that

the oracle H is sampled once and then fixed permanently.) Proving something in the *random oracle model* amounts to saying that it is true with high probability over the choice of H.

We use the letter H to represent random oracles (rather than $\mathcal{O}$) because we often think of them as *idealized cryptographic hash functions*. In particular, "a good cryptographic hash function should have output that has nothing at all to do with the input," and a random oracle models this property. (Here, I am describing the practitioner's intuitive view of what a hash function should do. As theorists, we prefer to use formal definitions—like one-wayness or collision resistance.) In particular, it is quite common to prove the security of some cryptographic scheme in the random oracle model and then to implement the random oracle using some hash function like SHA-3. This is a heuristic, which is often quite reasonable.

Notice, however, that random oracles are not real, and SHA-3 certainly is not actually a random oracle. For example, a simple argument shows that $\mathcal{H}$ is an *uncountable set*, so that one cannot even write down a typical element H in $\mathcal{H}$ with finitely many bits. I.e., it's not just that most functions in $\mathcal{H}$ are not efficiently computable. Most are not even describable!

So, while a construction of some cryptographic primitive (like, say, semantically secure encryption) in the random oracle model provides some evidence that such a primitive might exist in the real world, it is not ironclad. In fact, there are (rather artificial) cryptographic constructions that are provably secure in the random oracle model but provably insecure when the random oracle H is replaced with *any* efficiently computable function [CGH04]!

So, a proof in the random oracle model yields a heuristic argument for security in the standard model, which is very useful for arguing that something might be secure, but not nearly as good as a true security proof in the standard model. It is common for cryptographer to first construct some primitive in the random oracle model and to later show a (typically more complicated) construction that is provably secure in the standard model (under appropriate cryptographic assumptions). It is also common for practitioners to work with the random-oracle-based construction, since they are usually simpler and more efficient.

2 Let's prove that one-way functions exist! (in the random oracle model...)

Let's get our feet wet by showing that one-way functions exist in the random oracle model.

Let's be clear about what the actual statement is, though. We will prove that the oracle function H is itself a one-way function, which is captured by the following theorem.[1]

Theorem 2.1. *For all PPT oracle algorithms $\mathcal{A}^H$, there exists negligible ε such that*

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

[1] One can prove stronger theorems. For example, one can prove that with probability one over the choice of H, we have that for all oracle algorithms $\mathcal{A}^H$ there exists negligible $\varepsilon(n)$ such that

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

This is a stronger statement because the order of quantifiers is different. (Notice that since $\mathcal{H}$ is an uncountable set "with probability 1" is not the same as "always." E.g., if one samples x uniformly at random from the real numbers between 0 and 1, $\Pr[x \neq 1/2] = 1$, but this does not mean that x is *never* $1/2$.

Proof. Suppose that the algorithm $\mathcal{A}^H$ makes q queries $x_1, \ldots, x_q$ to the oracle. By possibly increasing q by one, we may assume without loss of generality that $x' \in \{x_1, \ldots, x_q\}$. Then, we have[2]

$$\Pr[H(x') = H(x)] \leq \Pr[\exists i \mid H(x_i) = H(x)] \leq \sum_{i=1}^{q} \Pr[H(x_i) = H(x) \mid \forall j < i,\ H(x_j) \neq H(x)] \ .$$

Now,

$$\begin{aligned}
&\Pr[H(x_i) = H(x) \mid \forall j < i,\ H(x_j) \neq H(x)] \\
&\quad \leq \Pr[x_i = x \mid \forall j < i,\ H(x_j) \neq H(x)] + \Pr[H(x_i) = H(x) \mid x_i \neq x \text{ and } \forall j < i,\ H(x_j) \neq H(x)] \\
&\quad \leq \frac{1}{2^n - q} + 2^{-n} \ ,
\end{aligned}$$

where the probability of $\frac{1}{2^n - q}$ comes from the fact that, conditioned on $H(x_1) \neq H(x), \ldots, H(x_{i-1}) \neq H(x)$, x is a uniformly random element from the set $\{x \in \{0,1\}^n \ : \ \forall j < i,\ x \neq x_j\}$, which has size $2^n - i + 1 > 2^n - q$. It follows that $\Pr[H(x') = H(x)] \leq 2q/(2^n - q)$. Since $\mathcal{A}^H$ runs in polynomial time, $q \leq \text{poly}(n)$, and therefore $2q/(2^n - q)$ is negligible, as needed. □

2.1 Also, commitment schemes, coin flipping, PRGs, PRFs, secret-key encryption, MACs, signatures, zero-knowledge proofs, etc.

We could now work to prove that all of our favorite Minicrypt primitives exist in the random oracle model. None of the proofs are *very* hard. But, actually, we don't need to.

We already proved in the standard model that the existence of one-way functions implies the existence of commitment schemes, coin flipping, PRGs, PRFs, secret-key encryption, MACs, signatures, zero-knowledge proofs, etc. As it turns out, all of these proofs work in oracle worlds as well. Formally, we say that these proofs *relativize* which means that they work *relative to an oracle*.

Indeed, one can take any of the proofs in which we showed that one primitive implies another (e.g., that PRGs imply PRFs), and simply put some superscripts $\cdot^H$ in appropriate places to prove the same result in the random oracle model. So, in particular, all of the Minicrypt primitives that we have seen in this class *provably* exist in the random oracle model. (Of course, many of these primitives are much easier to construct in the random oracle model. E.g., to build a PRG, just define $G(x) := H(x|0)$.)

However, there do exist certain proofs that *do not relativize*. These are typically proofs that are "non-black box" because they use something other than just the underlying functionality of an algorithm. For example, 3-SAT is *not* NP-complete in the random oracle model. In the standard model, we can prove that 3-SAT is NP-complete by writing any polynomial-time verifier as a circuit, defining the satisfiability problem over that circuit, and performing simple operations to convert our arbitrary circuit into the form of 3-SAT. However, in the random oracle model, we cannot write an arbitrary polynomial-time verifier as a standard circuit because the verifier might call the oracle.

[2]Notice that the query x_i might depend on the responses $H(x_1), \ldots, H(x_{i-1})$ that the adversary has received previously. This is the only subtlety here. In particular, we would like to argue that $\Pr[H(x_i) = H(x)] = \Pr[x_i = x] + \Pr[H(x_i) = H(x) \text{ and } x_i \neq x] \leq 2 \cdot 2^{-n}$, but this is not quite true. E.g., imagine that the adversary queries randomly until it finds x_j such that $H(x_j) = H(x)$. It then just queries x_j repeatedly. This adversary will have $\Pr[H(x_i) = H(x)] = \Pr[\exists j \leq i,\ x_j = x] + \Pr[\exists j \leq i,\ H(x_j) = H(x) \text{ and } \nexists j \leq i,\ x_j = x] \approx 2j2^{-n}$. This is why we must make this slightly tedious argument conditioning on $H(x_j) \neq H(x)$ for $j < i$.

So, we cannot write a polynomial-time verifier as a circuit. (Instead, we must write it as an *oracle* circuit.)

A famous example of a proof that works in the standard model but does not relativize is Adi Shamir's proof that $\mathsf{IP} = \mathsf{PSPACE}$ [Sha92]. In other words, while Shamir proved that $\mathsf{IP} = \mathsf{PSPACE}$, we also know that there exist oracles $\mathcal{O}$ such that $\mathsf{IP}^{\mathcal{O}} \neq \mathsf{PSPACE}^{\mathcal{O}}$—and indeed this holds with probability 1 when $\mathcal{O} = H$ is a random oracle.

Generally, a proof that relativizes is considered simpler than a proof that does not. And, sometimes when we are unable to prove something, we show that any proof would not relativize in order to argue that it would be difficult to prove. E.g., one "reason" that it is difficult to prove that $\mathsf{P} \neq \mathsf{NP}$ is the fact that any such proof cannot relativize, since there are oracles $\mathcal{O}$ such that $\mathsf{P}^{\mathcal{O}} = \mathsf{NP}^{\mathcal{O}}$. We will discuss more ideas like this below.

2.2 Two stronger primitives: collision-resistant hash functions and *non-interactive* zero-knowledge proofs

In fact, there are some primitives that exist in the random oracle model that are *not* known to be implied just by one-way functions (and are not thought to be implied just by one-way functions, as we will discuss more below).

We'll do two examples: collision-resistant hash functions and non-interactive zero-knowledge proofs.

Collision-resistant hash functions are pretty easy actually. We will simply use the function $H_k(x) := H(k,x)_{[0:|k|]}$ (i.e., the function that outputs the first $|k|$ bits of $H(k,x)$).

Theorem 2.2. *For any PPT oracle adversary $\mathcal{A}^H$, there exists negligible $\varepsilon(n)$ such that*

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

Proof. Let $y_1,\ldots,y_q$ be the queries that $\mathcal{A}^H$ makes to H. We may assume without loss of generality that (1) the x_i are distinct; and (2) there exists some i such that $y_i = k|x$ and some j such that $y_j = k|x'$. (In other words, we may assume that the adversary $\mathcal{A}^H$ queries the oracle on $k|x$ and $k|x'$ before outputting x,x'.) Then,

$$\begin{aligned}
&\Pr_{k\sim\{0,1\}^n}[H_k(x) = H_k(x'),\ x' \neq x] \\
&\quad\leq \Pr[\exists i \neq j,\ |y_i|,|y_j| \geq |k|,\ H(y_i)_{[0:|k|]} = H(y_j)_{[0:|k|]}] \\
&\quad\leq \sum_{i\neq j} 2^{-n} \leq q^2 2^{-n}\ .
\end{aligned}$$

Since $q(n) \leq \mathrm{poly}(n)$, this probability is negligible, as needed. □

A *non-interactive zero-knowledge proof* is a zero-knowledge proof with only a single message sent from the prover to the verifier. Actually, this definition is *slightly* too strong to achieve. Instead, we have to modify the definition of soundness to be *computational*. That is, we only ask that a PPT adversary $\mathcal{P}$ should not be able to convince us of a false statement with non-negligible probability.

To build a NIZK, the rough idea is to take your favorite *interactive* zero-knowledge proof and "replace the verifier with the random oracle." In particular, suppose that we have a zero-knowledge

proof with the property that (1) all of the verifier's messages are simply sampled uniformly at random, and (2) the validity of the proof can be verified by simply looking at the transcript. Such protocols are called *public-coin* protocols, and many of the protocols that we have seen satisfy this property.

For example, our protocol for 3-colorability had this property. In this protocol, the prover sent a single message, which was a uniformly random edge in a graph. To verify correctness, the verifier simply had to check that some commitments were opened correctly and that they opened to two different colors. More concretely, in such a protocol, the prover sends some message m_1, the verifier responds with a uniformly random message $m_2 \sim \{0,1\}^\ell$, the prover sends another message m_3 (which might be a function of m_1 and m_2), etc. Finally, the verifier runs some efficiently computable verification function $\mathsf{Ver}(m_1, \ldots, m_\ell) \in \{0,1\}$ and outputs the result.

To construct a non-interactive variant of this proof in the random oracle model, we simply replace m_2 by $H(m_1)$, and m_4 by $H(m_1, m_2, m_3)$, etc. I.e., our full non-interactive proof is just the odd messages $m_1, m_3, \ldots, m_\ell$, and to verify it we run $\mathsf{Ver}(m_1, m_2 := H(m_1), m_3, m_4 := H(m_1, m_2, m_3), \ldots, m_{\ell-1} := H(m_1, \ldots, m_{\ell-2}), m_\ell)$.

This is called the Fiat-Shamir transform [FS87]. It is immediate that it preserves both completeness and zero knowledge. Furthermore, if the original protocol had soundness error s, then one can show that the NIZK will have soundness error at most qs against adversaries that make at most q queries to the oracle.

NIZKs *do* exist in the standard model (under various computational assumptions), but they require some additional setup—namely, a fixed common reference string that must be sampled by a trusted party. And, they are quite difficult to build. In contrast, the Fiat-Shamir transform gives a relatively simple NIZK. Indeed, the Fiat-Shamir transform is used in *many* places, particularly now in blockchains. Of course, in practice, we do not use a random oracle (because random oracles don't exist), but instead replace the random oracle with a specific cryptographic hash function like SHA-3 or whatever, and we hope that the result remains secure. There are also by now many theoretical constructions of NIZKs that work by instantiating the Fiat-Shamir heuristic with a carefully designed hash function.

3 Oracle separations

A long-standing open problem in cryptography is to construct public-key encryption from one-way functions. This would be an absolutely amazing result!

However, most cryptographers don't really think it's possible to do so. Public-key encryption seems like a much more powerful primitive than one-way functions, so we don't expect to be able to build public-key encryption from one-way functions alone.

But, what do we mean when we say that "public-key encryption cannot be constructed from one-way functions alone"? Secure public-key encryption either exists or it doesn't. We don't know how to prove that it exists (or that it doesn't exist), but it still either exists or it doesn't, and we're pretty sure that it does. Assuming that a secure public-key encryption scheme exists, then what would it mean to say that "you can't build public-key encryption from one-way functions"? The statement seems vacuous.

We actually *can* give such a statement a precise meaning by showing that there is an oracle relative to which one-way functions exist but public-key encryption does *not*. Indeed, Impagliazzo and Rudich proved exactly this [IR90]. In fact, they proved much more—They showed an oracle

world in which one-way *permutations* exist but key agreement does not.

We call such results oracle separations or *black-box separations*, and by now *many* such results have been proven. One-way permutations are black-box separated from one-way functions; collision-resistant hash functions are black-box separated from one-way permutations; etc. In fact, there are oracle worlds representing each of Russell's five worlds—Minicrypt, Algorithmica, Heuristica, Pessiland, and Cryptomania (as well as many oracle worlds that Russell did not envision). E.g., there is an oracle world in which $\mathsf{P} = \mathsf{NP}$ (Algorithmica), an oracle world in which one-way functions exist but public-key encryption does not (Minicrypt), an oracle world in which one-way functions do not exist but there are problems in NP that are hard on average (Pessiland), etc.

It's not entirely clear how to interpret such results. We often say that they imply that a black-box separation between A and B implies that "there is no black-box construction of A from B." In particular, everything that we have seen in this class was a black box construction—all of our proofs work relative to any oracle (arguably with the exception of our use of NP-completeness to get zero-knowledge proofs for all of NP). This means that if we want to prove something like "one-way functions imply public-key encryption," we would need to use very different proof techniques from the ones that we used in this class.

On the other hand, there are by now a decent number of examples of constructions that are *not* black box—i.e., proofs that A implies B when we already know an oracle relative to which A exists but B does not. The details are rather technical, but they all typically involve explicitly looking at *how* a function is computed. E.g., a non-black-box construction using a one-way function f might use a circuit representing f in some way—perhaps by evaluating f homomorphically or by running a zero-knowledge proof that some y lies in the image of f.

Let me try to convey some of the flavor of these black-box separation proofs, without getting into the details—which are technical and messy. Say we're trying to separate public-key encryption from one-way functions—like Impagliazzo and Rudich did. To do this, we construct two oracles H and $\mathcal{O}$.[3] H is going to be our one-way function—in fact, it will simply be a random oracle, as this notation suggests—and $\mathcal{O}$ is going to be "an oracle that breaks any public-key encryption scheme." If we can show that H is still a one-way function even when our adversary has access to both H and $\mathcal{O}$, then we will have proven our result.

But, making this formal takes some work. First of all, what does it mean that $\mathcal{O}$ breaks *any* public-key encryption scheme? It could, for example, take as input circuits computing $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *and* a public key pk, and output a secret key matching the public key. (Notice that $\mathcal{O}$ must take as input circuits like this, since we want it to break *every* public-key encryption scheme. This means we have to have some way to tell it which encryption scheme it's meant to break.)

One can make this idea formal, but it is not enough. We also need to show that H cannot somehow be used to construct a public-key encryption scheme. This means that we must allow for the possibility of *oracle circuits*, i.e., descriptions of public-key encryption schemes $(\mathsf{Gen}^H, \mathsf{Enc}^H, \mathsf{Dec}^H)$ that use H. Even worse, we need to worry that $\mathcal{O}$ *itself* can be used to construct a public-key encryption scheme. Indeed, $\mathcal{O}$ is clearly quite a powerful oracle, so maybe there is some crazy

[3] If you are bothered by our use of two oracles, rather than just one, notice that you can always collapse two oracles into one by defining

$$H^*(b, x) := \begin{cases} H(x) & b = 0 \\ \mathcal{O}(x) & b = 1 \end{cases}.$$

Then, having oracle access to H^* is equivalent to having oracle access to both H and $\mathcal{O}$. This is called *domain separation*, since we are separating the domain of H^* into two parts.

public-key encryption scheme that one can build with $\mathcal{O}$. So, $\mathcal{O}$ must take as input descriptions of public-key encryption schemes $(\mathsf{Gen}^{H,\mathcal{O}}, \mathsf{Enc}^{H,\mathcal{O}}, \mathsf{Dec}^{H,\mathcal{O}})$ that incorporate $\mathcal{O}$ itself! This of course makes such proofs quite delicate, since it is not even entirely clear that such an oracle $\mathcal{O}$ can be implemented in a consistent way. Fortunately (or unfortunately?), it can be. Once you have constructed such an oracle $\mathcal{O}$, you must then show that H is a one-way function, even in the presence of $\mathcal{O}$.

References

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4), 2004.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1987.

[IR90] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *CRYPTO*, 1990.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992.