Notes on Computer Theory Last updated: November, 2015

$\mathcal{IP} = \mathsf{PSPACE}$, zero knowledge

Notes by Jonathan Katz, lightly edited by Dov Gordon.

1 $\mathcal{IP} = \mathsf{PSPACE}$

A small modification of the previous protocol gives an interactive proof for any language in PSPACE, and hence $\mathsf{PSPACE} \subseteq \mathcal{IP}$. In our homework we will prove that $\mathcal{IP} \subseteq \mathsf{PSPACE}$, and here we only show the the more interesting direction, namely showing that $\mathsf{PSPACE} \subseteq \mathcal{IP}$. We will now work with the PSPACE-complete language TQBF, which (recall) consists of true quantified boolean formulas of the form:

$$\forall x_1 \exists x_2 \cdots Q_n x_n \quad \phi(x_1, \ldots, x_n),$$

where ϕ is a 3CNF formula. We begin by arithmetizing ϕ as we did in the case of $\#\mathcal{P}$; recall, if ϕ has m clauses this results in a degree-$3m$ polynomial Φ such that, for $x_1, \ldots, x_n \in \{0,1\}$, we have $\Phi(x_1, \ldots, x_n) = 1$ if $\phi(x_1, \ldots, x_n)$ is true, and $\Phi(x_1, \ldots, x_n) = 0$ if $\phi(x_1, \ldots, x_n)$ is false.

We next must arithmetize the quantifiers. Let Φ be an arithmetization of ϕ as above. The arithmetization of an expression of the form $\forall x_n \, \phi(x_1, \ldots, x_n)$ is

$$\prod_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \Phi(x_1, \ldots, x_{n-1}, 0) \cdot \Phi(x_1, \ldots, x_{n-1}, 1).$$

If we fix values for $x_1, \ldots, x_{n-1}$, then the above evaluates to 1 if the expression $\forall x_n \, \phi(x_1, \ldots, x_n)$ is true, and evaluates to 0 if this expression is false. The arithmetization of an expression of the form $\exists x_n \, \phi(x_1, \ldots, x_n)$ is

$$\coprod_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n) \stackrel{\text{def}}{=} 1 - \big(1 - \Phi(x_1, \ldots, x_{n-1}, 0)\big) \cdot \big(1 - \Phi(x_1, \ldots, x_{n-1}, 1)\big).$$

Note again that if we fix values for $x_1, \ldots, x_{n-1}$ then the above evaluates to 1 if the expression $\exists x_n \, \phi(x_1, \ldots, x_n)$ is true, and evaluates to 0 if this expression is false. Proceeding in this way, a quantified boolean formula $\exists x_1 \forall x_2 \cdots \forall x_n \phi(x_1, \ldots, x_n)$ is true iff

$$1 = \coprod_{x_1 \in \{0,1\}} \prod_{x_2 \in \{0,1\}} \cdots \prod_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n). \tag{1}$$

A natural idea is to use Eq. (1) in the protocols we have seen for $\mathsf{co}\mathcal{NP}$ and $\#\mathcal{P}$, and to have the prover convince the verifier that the above holds by "stripping off" operators one-by-one. While this works in principle, the problem is that the *degrees* of the intermediate results are too large. For example, the polynomial

$$P(x_1) = \prod_{x_2 \in \{0,1\}} \cdots \prod_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n)$$

may have degree as high as $2^n \cdot 3m$ (note that the degree of x_1 doubles each time a $\prod$ or $\coprod$ operator is applied). Besides whatever effect this will have on soundness, this is even a problem for completeness since a polynomially bounded verifier cannot read an exponentially large polynomial (i.e., with exponentially many terms).

To address the above issue, we use a simple[1] trick. In Eq. (1) the $\{x_i\}$ only take on *boolean* values. But for any $k > 0$ we have $x_i^k = x_i$ when $x_i \in \{0, 1\}$. So we can in fact reduce the degree of every variable in any intermediate polynomial to (at most) 1. (For example, the polynomial $x_1^5 x_2^4 + x_1^6 + x_1^7 x_2$ would become $2x_1x_2 + x_1$.) Let R_{x_i} be an operator denoting this "degree reduction" operation applied to variable x_i. Then the prover needs to convince the verifier that

$$1 = \coprod_{x_1 \in \{0,1\}} R_{x_1} \prod_{x_2 \in \{0,1\}} R_{x_1} R_{x_2} \coprod_{x_3 \in \{0,1\}} \cdots R_{x_1} \cdots R_{x_{n-1}} \prod_{x_n \in \{0,1\}} R_{x_1} \cdots R_{x_n} \Phi(x_1, \ldots, x_n).$$

As in the previous protocols, we will actually evaluate the above modulo some prime q. Since the above evaluates to either 0 or 1, we can take q any size we like (though soundness will depend inversely on q as before).

We can now apply the same basic idea from the previous protocols to construct a new protocol in which, in each round, the prover helps the verifier "strip" one operator from the above expression. Denote the above expression abstractly by:

$$F_\phi = \mathcal{O}_1 \mathcal{O}_2 \cdots \mathcal{O}_\ell \; \Phi(x_1, \ldots, x_n) \bmod q \,,$$

where $\ell = \sum_{i=1}^{n} (i+1)$ and each O_j is one of $\prod_{x_i}$, $\coprod_{x_i}$, or R_{x_i} (for some i). At every round k the verifier holds some value v_k and the prover wants to convince the verifier that

$$v_k = \mathcal{O}_{k+1} \cdots \mathcal{O}_\ell \; \Phi_k \bmod q,$$

where Φ_k is some polynomial. At the end of the round the verifier will compute some v_{k+1} and the prover then needs to convince the verifier that

$$v_{k+1} = \mathcal{O}_{k+2} \cdots \mathcal{O}_\ell \; \Phi_{k+1} \bmod q,$$

for some Φ_{k+1}. We explain how this is done below. At the beginning of the protocol we start with $v_0 = 1$ and $\Phi_0 = \Phi$ (so that the prover wants to convince the verifier that the given quantified formula is true); at the end of the protocol the verifier will be able to compute Φ_ℓ itself and check whether this is equal to v_ℓ.

It only remains to describe each of the individual rounds. There are three cases corresponding to the three types of operators (we omit the " mod q" from our expressions from now on, for simplicity):

Case 1: $\mathcal{O}_{k+1} = \prod_{x_i}$ (for some i). Here, the prover wants to convince the verifier that

$$v_k = \prod_{x_i} R_{x_1} \cdots \coprod_{x_{i+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, x_i, \ldots, x_n). \tag{2}$$

(*Technical note*: when we write an expression like the above, we really mean

$$\left(\prod_{x_i} R_{x_1} \cdots \coprod_{x_{i+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(x_1, \ldots, x_{i-1}, x_i, \ldots, x_n) \right) [r_1, \ldots, r_{i-1}].$$

[1] Of course, it seems simple in retrospect...

That is, first the expression is computed symbolically, and then the resulting expression is evaluated by setting $x_1 = r_1, \ldots, x_{i-1} = r_{i-1}$.) This is done in the following way:

- The prover sends a degree-1 polynomial $\hat{P}(x_i)$.
- The verifier checks that $v_k = \prod_{x_i} \hat{P}(x_i)$. If not, reject. Otherwise, choose random $r_i \in \mathbb{F}_q$, set $v_{k+1} = \hat{P}(r_i)$, and enter the next round with the prover trying to convince the verifier that:

$$v_{k+1} = R_{x_1} \cdots \coprod_{x_{i+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, r_i, x_{i+1}, \ldots, x_n). \tag{3}$$

To see completeness, assume Eq. (2) is true. Then the prover can send

$$\hat{P}(x_i) = P(x_i) \stackrel{\text{def}}{=} R_{x_1} \cdots \coprod_{x_{i+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, x_i, \ldots, x_n);$$

the verifier will not reject and Eq. (3) will hold for any choice of r_i. As for soundness, if Eq. (2) does *not* hold then the prover must send $\hat{P}(x_i) \neq P(x_i)$ (or else the verifier rejects right away); but then Eq. (3) will not hold except with probability $1/q$.

Case 2: $\mathcal{O}_{k+1} = \coprod_{x_i}$ (for some i). This case and its analysis are similar to the above and are therefore omitted.

Case 3: $\mathcal{O}_{k+1} = R_{x_i}$ (for some i). Here, the prover wants to convince the verifier that

$$v_k = R_{x_i} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_j, x_{j+1}, \ldots, x_n), \tag{4}$$

where $j \geq i$. This case is a little different from anything we have seen before. Now:

- The prover sends a polynomial $\hat{P}(x_i)$ of appropriate degree (see below).
- The verifier checks that $\left(R_{x_i} \hat{P}(x_i)\right)[r_i] = v_k$. If not, reject. Otherwise, choose a **new** random $r_i \in \mathbb{F}_q$, set $v_{k+1} = \hat{P}(r_i)$, and enter the next round with the prover trying to convince the verifier that:

$$v_{k+1} = \mathcal{O}_{k+2} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_i, \ldots, r_j, x_{j+1}, \ldots, x_n). \tag{5}$$

Completeness is again easy to see: assuming Eq. (4) is true, the prover can simply send

$$\hat{P}(x_i) = P(x_i) \stackrel{\text{def}}{=} \mathcal{O}_{k+2} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, x_i, r_{i+1}, \ldots, r_j, x_{j+1}, \ldots, x_n)$$

and then the verifier will not reject and also Eq. (5) will hold for any (new) choice of r_i. As for soundness, if Eq. (4) does *not* hold then the prover must send $\hat{P}(x_i) \neq P(x_i)$; but then Eq. (5) will not hold except with probability d/q where d is the degree of $\hat{P}$.

This brings us to the last point, which is what the degree of $\hat{P}$ should be. Except for the innermost n reduce operators, the degree of the intermediate polynomial is at most 2; for the innermost n reduce operators, the degree can be up to $3m$.

We may now compute the soundness error of the entire protocol. There is error $1/q$ for each of the n operators of type $\prod$ or $\coprod$, error $3m/q$ for each of the final n reduce operators, and error $2/q$ for all other reduce operators. Applying a union bound, we see that the soundness error is:

$$\frac{n}{q} + \frac{3mn}{q} + \frac{2}{q} \cdot \sum_{i=1}^{n-1} i = \frac{3mn + n^2}{q}.$$

Thus, a polynomial-length q suffices to obtain negligible soundness error.

2 Zero Knowledge

An interactive proof for a language L ensures that the verifier accept if and only if $x \in L$. Proofs for languages in $\mathcal{NP}$ are extremely simple to demonstrate: the prover can simply send the witness w that is guaranteed to exist (by the definition of $\mathcal{NP}$), and the verification algorithm is $M(x, w)$, where, again by the definition of $\mathcal{NP}$, M is guaranteed to output 1 if and only if $x \in L$. However, what if the prover would like to convince the verifier that x is in L, *without revealing* w*?*. Although this might sound impossible, it turns out that if one-way functions exist, then there are zero knowledge proof systems for every language in $\mathcal{NP}$.

We will demonstrate a zero knowledge proof system for Graph Isomorphism, but first we have to define what we mean when we say that a proof system does not reveal anything about the witness. Formally capturing the notion of knowledge is non-trivial. Intuitively, the idea is to consider the view of the verifier during the course of its interaction with the prover – that is, we consider the sequence of messages received by the verifier from the prover – and to treat this view as a long string drawn from some distribution. The support of this distribution is defined by the set of all messages that a verify might send (according to the protocol description, if he is honest, or, arbitrary messages if we allow him to be malicious), and the distribution is determined by the choice of random coins used by both parties. We wish to claim that the verifier learns nothing from his view beyond the fact that x is (or is not) in L. To capture this, we demonstrate that this same distribution (i.e. over the possible views of the verifier) can be *simulated* by a machine that has no access to a prover, and whose only input is x. Since this simulator has no access to w, certainly nothing he can compute leaks any information about w. Furthermore, since his output has the same distribution as the view of the verifier, it follows that the view of the verifier must leak nothing more than the output of S, which is to say, it must reveal nothing about w at all! Formally,

Definition 1 *$L \in \mathcal{ZK}$ if there exist a pair of interactive algorithms $(\mathbf{P}, \mathbf{V})$, with $\mathbf{V}$ running in probabilistic polynomial time (in the length of the common input x), such that*

1. *If $x \in L$, then* $\Pr[\langle \mathbf{P}, \mathbf{V} \rangle (x) = 1] = 1$.
2. *If $x \notin L$, then for* any $\mathbf{P}^*$ *we have* $\Pr[\langle \mathbf{P}^*, \mathbf{V} \rangle (x) = 1] \leq 1/2$.
3. *If $x \in L$, then for any PPT V^*, there exists a simulator S running in* expected *PPT, such that*
   $\mathsf{view}(\langle \mathbf{P}, \mathbf{V}^* \rangle (x)) \equiv S(x)$,

where $\mathsf{view}(\langle \mathbf{P}, \mathbf{V}^* \rangle (x))$ *denotes the distribution of $\mathbf{V}^*$'s view while interacting with prover $\mathbf{P}$ on input x.*

At first glance, it seems that the very existence of such a simulator provides a method for deciding whether $x \in L$, suggesting an expected PPT algorithm for $\mathcal{NP}$. There are two important reasons why this is not the case. First, note that while the simulator does not have the benefit of interacting with a prover that has knowledge of w, he also is not restricted by such an engagement. In particular, when simulating the view of the verifier, the simulator can choose to restart his simulation whenever it is not "going well" (i.e. if he realizes that he has chosen bad random coins); the real verifier has no such opportunity, since a prover will simply refuse to answer his messages if he attempts to do so. As we will see, this strategy will be crucial in our proofs, and it is crucial in most proofs of security for zero knowledge protocols. The other point to make is that the above definition only requires the simulator to output a good view if $x \in L$. When $x \notin L$, the view that the simulator generates might be quite different from the view of the real verifier. Indeed,

because we do not know how to decide languages in $\mathcal{NP}$, it must be the case that the output of the simulator when $x \notin L$ is indistinguishable from the its output when $x \in L$, and, in particular, views output by the simulator must *all* result in output 1 by the (simulated) verifier.

2.1 A Zero Knowledge Proof for Graph Isomorphism

The prover and verifier each have input (G_0, G_1), and the prover additionally holds a witness (i.e. a permutation) π such that $\pi(G_0) = G_1$. The protocol proceeds as follows.

- **P** chooses a random bit $b \leftarrow \{0,1\}$, and a random permutation σ. He computes $G = \sigma(G_b)$ and sends the resulting graph to **V**.
- **V** chooses $b' \leftarrow \{0,1\}$ and sends this to **P**. (Intuitively, he is asking for a permutation mapping $G_{b'}$ to G.)
- **P** sends permutation $\tau = \begin{cases} \sigma & \text{if } b' = b \\ \sigma\pi^{-1} & \text{if } b = 0 \text{ and } b' = 1 \\ \sigma\pi & \text{if } b = 1 \text{ and } b' = 0 \end{cases}$
- **V** accepts iff $\tau(G_{b'}) = G$.

Completeness is trivial to verify. Soundness holds because if the graphs are non-isomorphic, then with probability $1/2$, $b' \neq b$, and there does not exist any permutation mapping $G_{b'}$ to $G = \sigma(G_b)$. To prove zero knowledge, we have to describe a simulator S, running in expected polynomial time and outputting a view of the verifier. Note that such a view consists of two messages: a graph, sent from the prover in the 1st round, and a permutation, sent from the prover in the 3rd round. S simulates the first message by choosing $b \leftarrow \{0,1\}$ and a random permutation σ. He outputs $G = \sigma(G_b)$, just as an honest prover would do. Note that the verifier $\mathbf{V}^*$ might be malicious, and so S cannot now assume that the reply of $\mathbf{V}^*$ is a uniformly random bit, as it is supposed to be. However, we can describe $\mathbf{V}^*$'s choice of b' using some polynomial-time computable function $f(G, G_0, G_b; r)$. S chooses random coins r and runs f himself[2] to determine the bit b'. If $b' \neq b$, then S restarts the simulation from the beginning. Otherwise, he outputs σ as the prover's second message to $\mathbf{V}^*$.

To prove that the output of S has the correct distribution, we first note that for any pair $(0, \sigma)$ that S might randomly choose when computing $G = \sigma(G_0)$, there is a pair $(1, \sigma\pi^{-1})$ that is chosen with the same probability. (While S does not know π, realize that he does not need to. We argue only that, when a choosing a permutation at random, he is as likely to choose $\sigma\pi^{-1}$ as he is any other permutation.) Notice that $G = \sigma\pi^{-1}(G_1) = \sigma(G_0)$, so there are 2 ways of generating message G, each with a different value b.[3] It follows that $\Pr[b = 0 \mid G] = 1/2$, and, in particular, regardless of which function f is used by $\mathbf{V}^*$ to compute b', $\Pr[b = b'] = 1/2$. Therefore, the expected number of executions that S must make is 2, and, since the probability of S re-starting is always $1/2$, independent of the value of G, it also follows that G was sampled from the appropriate distribution. Once G and b' are chosen, there is only one possible value to be sent in the 3rd round (regardless of the value of b); since G and b' have the appropriate distributions, it follows that the pair (G, σ) has the appropriate distribution as well.

[2] Note that the way we've defined zero knowledge requires that for every $\mathbf{V}^*$ there exists some simulator S, so, in particular, S's description can depend on the description of $\mathbf{V}^*$. There are alternative ways of defining zero knowledge. See [1] for further discussion.

[3] In fact, there may be even more than 2 ways of generating G, if we consider automorphisms and compose those with σ. But it is easy to see that there are exactly the same number of permutations mapping G_0 to G as there are mapping G_1 to G.

References

[1] O. Goldreich. A Short Tutorial of Zero Knowledge. Available here