6.875: Solutions for Homework 6

Yevgeniy Dodis

Problem 1: One-Way Functions and Hardcore Bits

Recall that the *hardcore bit* of a function f is a predicate of x that is easy to compute given x, but very hard to compute given $f(x)$. For *any* one-way function f, Goldreich-Levin theorem says that a parity of a *random* subset of x's bits is a hardcore bit for f. For *specific* one-way function candidates, we often have a that a particular bit of x is a hardcore bit, e.g. $lsb(x)$ for RSA and $msb(x)$ for exponentiation modulo a prime. This raises a question of whether it is the case that for any one-way function f, some particular bit of x is a hardcore bit.

Show that the general answer is "no". In other words, assuming one-way functions exist, construct a one-way function g such that *no single bit of* x is a hardcore bit for this function. In fact, every bit of x can be predicted with probability at least $\frac{3}{4}$ given $g(x)$. Hence, Goldreich-Levin result is in some sense the best we can hope for.

Solution: Let f be any OWF. For a string $x = x_1 \ldots x_k$ of length k and a subset $I = \{i_1 \ldots i_j\} \subseteq [k]$ (here $[k] := \{1 \ldots k\}$), let $x_I = x_{i_1} \ldots x_{i_j}$ be the concatenation of x's bits with index in I. Here is a function g taking a string x and a subset $I \in \{1 \ldots |x|\}$.

$$g(x, I) := \langle f(x_I), x_{\bar{I}}, I \rangle$$

Hence, we "give away" I and x's bits *not* in I. For x's bits in I, we evaluate our OWF f. Intuitively, we give out some of the bits of x for free so that the prediction of each bit is easy. However, we do not give too many (on average), so that g is still hard to invert. In other words, the result follows from the following 2 claims. Here $y = \langle x, I \rangle$ is the input to g.

Claim 1 *Every bit of y can be predicted from $g(y)$ with probability at least $\frac{3}{4}$.*

Claim 2 *If f is a OWF, then so is g.*

Proof of Claim 1: Let $|y| = n = 2k$, where $|x| = |I| = k$. We claim that every bit y_i is easy to predict from $g(y)$. This is immediate for $i > k$ (i.e. the bits of subset I) since $g(y)$ contains every bit of I "in the clear". Thus, we only need to establish the claim for $i \le k$, i.e. the bits of x. Here is the predictor P for x_i on input $g(y) = \langle f(x_I), x_{\bar{I}}, I \rangle$.

- If $i \notin I$, then output x_i from $x_{\bar{I}}$.
- Otherwise, output a random bit.

We claim that for every $1 \le i \le k$, $\Pr[P(g(y)) = x_i] = \frac{3}{4}$ where the probability is taken over the random choice of g's input y, i.e. a random $x \in \{0,1\}^k$ and a *random subset* $I \in [k]$. Since I is random, $\Pr[i \notin I] = \frac{1}{2}$. In this case, we are correct with probability 1. Otherwise, we are correct with probability $\frac{1}{2}$ (we flip a coin), a combined of $\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$. ■

Proof of Claim 2: Assume for a contradiction that g is not a OWF. Since the only non-trivial thing in the invention of g is to find x_I from $f(x_I)$, we assume (to make the notation shorter) that our hypothetical PPT inverter A, on input $\langle w, x_{\bar{I}}, I\rangle$, tries to find $z \in \{0,1\}^{|I|}$ such that $f(z) = w$. Then, for some polynomial p and for infinitely many n (say, $n \in S$, where S is some infinite set) we have that:

$$\Pr[\ f(z) = w \mid x \leftarrow \{0,1\}^k,\ I \subseteq [k],\ w = f(x_I),\ z \leftarrow A(\langle w, x_{\bar{I}}, I\rangle)\ ] \geq \frac{1}{p(n)}$$

Since I is chosen at random, we expect $|I| = k/2$. By Chernoff's inequality, with overwhelming probability $|I| \geq k/3 = n/6$. More specifically, $\Pr[|I| < \frac{n}{6}] \leq 2^{-\Omega(n)}$ which is much less than $1/p(n)$. Hence, A succeeds inverting with probability at least $(1/p(n) - 2^{-\Omega(n)})$, given that I is a *random subset of cardinality at least* $\frac{n}{6}$. Averaging over all the possible lengths of I from $n/6$ to $k = n/2$, we get that there exists some particular length $l = l(n)$ such that A succeeds inverting g (with good probability) when I is a *random subset of cardinality exactly* $l(n)$. In other words, A must be good for some particular length $l(n) \in \{\frac{n}{6} \dots \frac{n}{2}\}$, so that

$$\Pr[\ f(z) = w \mid x \leftarrow \{0,1\}^k,\ I \subseteq [k],\ |I| = l(n),\ w = f(x_I),\ z \leftarrow A(\langle w, x_{\bar{I}}, I\rangle)\ ] \geq \frac{1}{p(n)} - 2^{-\Omega(n)} \quad (1)$$

Recall that n came from an infinite set S. Define $S' = \{l(n) \mid n \in S\}$. Since $l(n) \in \{\frac{n}{6} \dots \frac{n}{2}\}$, S' is also infinite. We now construct an inverter B for our OWF f that will succeed inverting $f(z)$ with non-negligible probability for all $|z| \in S'$. Here is B on input w trying to invert f on inputs of length m.

1. Try for even $n = 2m \dots 6m$: (let $k = n/2$)
2. Pick a random subset I of $[k]$ of cardinality exactly m. Pick a random $x' \in \{0,1\}^{k-m}$.
3. Let $z \leftarrow A(w, x', I)$.
4. If $f(z) = w$ output z and halt, else go to the next n.
5. Output "fail" if all iterations fail.

Clearly, B is PPT since A is. Let $m \in S'$, i.e. there is some $2m \leq n \leq 6m$ such that $m = l(n)$. This n will be tried in one of the iterations above. For this particular n, we call A with a random I of size $m = l(n)$, a random $x' \in \{0,1\}^{k-l(n)}$ (that should be viewed as $x_{\bar{I}}$) and our challenge w, where w is $f(z')$ for some $z' \in \{0,1\}^{l(n)}$. As we average the behavior of B by picking a random z' (that should be viewed as x_I), we call A on *exactly the same* probability distribution as in (1)! Hence, A will succeed inverting f with probability at least $1/p(n) - 2^{-\Omega(n)} \geq 1/q(m)$ (for some polynomial q, since $n \leq 6m$), which is non-negligible. Since we try even more values of n (including the one s.t. $m = l(n)$) and halt only when we succeed inverting f, our overall probability of success for $m \in S'$ is at least $1/q(m)$, contradicting the fact that f is a OWF. ■

Note: We can extend this construction by letting for $t \geq 1$

$$g_t(X, I_1, \dots, I_t) := \langle f(X_{I_1 \cap \dots \cap I_t}), X_{\overline{I_1 \cap \dots \cap I_t}}, I_1, \dots, I_t\rangle$$

Here $I_j \subseteq [k]$ and $|X| = k$ (for $t = 1$ this is just the function we defined). It is easy to see that every bit of $y = \langle X, I_1, \dots, I_t\rangle$ is predictable with probability at least $1 - 2^{-t-1}$. This is because we make an error only when X_i is not given in the "clear" (i.e. $i \notin I_1 \cap \dots \cap I_t$, which happens w/pr. 2^{-t}) and, in addition, we incorrectly guessed X_i, a combined $2^{-t} \cdot \frac{1}{2} = 2^{-t-1}$ probability of error. Moreover, similar to the argument we had, we can show that g_t is a OWF as long at $t < (1 - \epsilon) \log k$, for any $\epsilon > 0$ (otherwise, we can no longer apply the Chernoff bound effectively). This means that we can construct a OWF such that *each of its individual bits is predictable with probability* $1 - \frac{1}{k^{1-\epsilon}}$, for any $\epsilon > 0$. This is the strongest possible negative result we can expect, since we can never predict every bit of an input to a OWF with probability any better than $1 - \frac{1}{k} + negl(k)$. Indeed, if every bit is predicted incorrectly with probability at most $\frac{1}{k} - \frac{1}{poly(k)}$, then by the union bound we predict *at least one of* k *bits* incorrectly with probability at most $1 - \frac{k}{poly(k)}$, i.e. we correctly invert the function with non-negligible probability at least $\frac{k}{poly(k)}$, a contradiction. Therefore, it is not surprising that we could not push the argument beyond $t = (1-\epsilon)\log k$. See the attached handout for a very different elegant solution to this problem.

Problem 2: Claw-Free Permutation Pairs

Let $\mathcal{C} = \{(f_i, g_i) : D_i \mapsto D_i \mid i \in I\}$, $F(\mathcal{C}) = \{f_i \mid i \in I\}$, where each f_i, g_i are permutation on some domain D_i. Here I is some infinite index set that is samplable, i.e. there is PPT $G(1^k)$ outputting a random $i \in \{0,1\}^k \cap I$, the number of bits to represent $x \in D_i$ is polynomial in $|i|$, and both g_i and f_i are polynomial-time computable.

Recall that $\mathcal{C}$ is a collection of *claw-free permutation pairs* if no PPT A can find $x, y \in D_i$ s.t. $f_i(x) = g_i(y)$, i.e.

$$\Pr(A(i) = (x,y) \text{ such that } f_i(x) = g_i(y) \mid i \leftarrow G(1^k)) = negl(k)$$

(a) Show that if $\mathcal{C}$ is a collection of claw-free permutation pairs, then $F(\mathcal{C})$ is a collection of one-way permutations.

Solution: Assume that $F(\mathcal{C})$ is not a family of trapdoor permutations, i.e. there is a PPT inverter A that for a random $i \in I \cap \{0,1\}^k$ and a random $x \in D_i$ has a reasonable chance of inverting $y = f_i(x)$. Here is a PPT B that will find claws for (f_i, g_i) for a random $i \in I \cap \{0,1\}^k$. Pick random $z \in D_i$. Let $y = g_i(z)$. Let $x \leftarrow A(y)$. Output (x, z). Since z is random and g_i is a permutation, y is random. Since f_i is also a permutation, we can think of y as $f_i(x)$ for a random $x \in D_i$. By assumption, A has non-negligible chance to find x, but then $f_i(x) = y = g_i(z)$, so we indeed found a claw (x, z).

(b) Under the RSA assumption, construct a family of permutation pairs $\mathcal{C}$ such that $\mathcal{C}$ is *not* claw-free, but $F(\mathcal{C})$ is a collection of one-way permutations.

Solution: This is such a tautology that it is not even worse stating, there are plenty of answers varying in their level of triviality (the reason for asking you this question was to show that claw-freeness is a stronger property than one-wayness). For example, for any family F of trapdoor permutations, define $\mathcal{C} = \{(f_i, f_i) \mid i \in I\}$. Clearly, $\mathcal{C}$ is not claw-free. Or let us augment each D_i with some special element which we denote by 0. Augment each f_i in F by $f_i(0) = 0$. This clearly does not change the fact that F is a family of trapdoor permutations. Let g_i be any function such that $g_i(0) = 0$ (e.g. $g_i \equiv 0$). Again, $\mathcal{C}$ is not claw-free as $(0,0)$ is a claw.

For RSA, anything you do will work since any 2 RSA functions $(x^{e_1} \bmod n)$ and $(x^{e_2} \bmod n)$ have a claw $(1,1)$, or more generally, (z^{e_2}, z^{e_1}), for any $z \in \mathbb{Z}_n^*$. For example, $\mathcal{C} = \{(x^{e_1} \bmod n, x^{e_2} \bmod n) \mid n \text{ - RSA modulus}, e_1, e_2 \text{ - RSA exponents}\}$.

(c) Recall that a family $\mathcal{H} = \{h_i : \{0,1\}^* \mapsto D_i \mid i \in I\}$ is a family of *collision-free hash functions* if no PPT B can find distinct strings $a, b \in \{0,1\}^*$ mapping to the same value, i.e.

$$\Pr(B(i) = (a,b) \text{ such that } h_i(a) = h_i(b) \mid i \leftarrow G(1^k)) = negl(k)$$

Construct a collision-free family $\mathcal{H}$ of hash functions based on any family $\mathcal{C}$ of claw-free permutation pairs.

Solution: We use the same construction as we did in class. First, we can assume w.l.o.g. that no two messages are suffixes of each other by using some suffix-free encoding, for example. The index set for $\mathcal{H}$ is $I' = \{\langle i, R\rangle \mid i \in I, R \in D_i\}$. Since I and D_i are samplable, I' is samplable as well (the generator first finds $i \in I \cap \{0,1\}^k$, then random $R \in D_i$).

Here is how we define $h_{i,R}(a)$ where $i \in I$, $R \in D_i$ and $a = a_1 \dots a_n$ for some $n > 0$. Let $c_0(a) = R$. Define $c_j(a) = f_i(c_{j-1}(a))$ if $a_j = 0$, and $c_j(a) = g_i(c_{j-1}(a))$ if $a_j = 1$. Let $h_{i,R}(a) = c_n$. We claim that $\mathcal{H}$ is collision-free. More specifically, whenever we find a collision (a, b) in $h_{i,R}$, we also find a claw (x, y) for (f_i, g_i) somewhere along the way, since we assumed that all messages are suffix-free. Indeed, take the smallest j such that $a_{|a|-j} \neq b_{|b|-j}$ ($j > 0$ since $h_{i,R}(a) = c_{|a|}(a) = c_{|b|}(b) = h_{i,R}(b)$ and it exists because of suffix-freeness). W.l.o.g. assume that $a_{|a|-j} = 0$, $b_{|b|-j} = 1$. Let $z = c_{|a|-(j-1)}(a)$ ($= c_{|b|-(j-1)}(b)$), $x = c_{|a|-j}(a)$, $y = c_{|b|-j}(b)$. Then we have by our construction that $f_i(x) = z = g_i(y)$, i.e. (x, y) is a claw. Thus, assuming that $\mathcal{H}$ is not collision-free, here is the algorithm for finding claws given $i \in I$: pick random $R \in D_i$ and run the assumed collision algorithm on $i' = \langle i, R\rangle$. If find a collision, use it to find a claw.

Note: How important was it to put R in the index set I' and to pick it random? Conceptually, it is not important at all. We can make $I' = I$. The choice of R to start with is unimportant, except it has to be *constructive*. Since we do not know anything about D_i except that we can sample a random $x \in D_i$, we enforced this constructibility by placing a *random* R as part of the index. But any R we can construct will do. E.g., if we assume that each D_i has a special element 0, we can make $R = 0$. But random R is the cleanest way out without making extra-assumptions. However, it is OK if you ignored this issue as it is minor.

Problem 3: Claw-Free Permutations based on Discrete Log

Construct a family of claw-free permutation pairs based of the Discrete Log assumption (remember, your permutations need not be trapdoor). Either directly, or using the previous problem, define a collision-free family of hash functions based on the Discrete Log assumption.

Solution: Here is the family. The index set $I = \{\langle p, g, R\rangle \mid p$ - prime, g - generator for $\mathbb{Z}_p^*, R \in \mathbb{Z}_p^*\}$. Define $f_{p,g,R}(x) = g^x \bmod p$, $h_{p,g,R} = R \cdot g^x \bmod p$. The index generator $G(1^{3k})$ outputs a random prime p of length k, random generator g (we know how to do it using factorization of $p-1$) and $R \in \mathbb{Z}_p^*$. The domain D_i is $\mathbb{Z}_p^*$ that is easily samplable as well. It remains to show that it impossible to have a PPT A s.t. for infinitely many k

$$\Pr(A(p, g, R) = (x, y) \text{ such that } g^x = Rg^y \bmod n \mid \langle p, g, R\rangle \leftarrow G(1^{3k})) = \frac{1}{poly(k)} \tag{2}$$

But if $g^x = Rg^y$, then $R = g^{x-y} \bmod p$, and thus $[(x - y) \bmod (p-1)]$ is a discrete log of R. This suggests the following trivial algorithm to solve the discrete log problem. Given p, g and R, simply run $A(p, g, R)$ and then get the discrete log of R if A succeeds. Since R is part of the index i (this is important!), we find a discrete log of a *random* R, directly contradicting the discrete log assumption.

Note: As I pointed out, it is crucial that you make R part of the index, a *fixed* R is not contradicting anything since DLP might be easy for a particular R. Also, a lot of you defined different and *incorrect* families. A typical one is $f = g_1^x \bmod p$, $h = g_2^x \bmod p$, where g_1 and g_2 are two (random) generators of $\mathbb{Z}_p^*$. This seems to be a good candidate, but I do not see how we can *prove* it based on the discrete log assumption. I'm not going to try to describe the pseudo-proof that this is a good claw-free family, since I do not even know how to start! Some of you made informal arguments that "it is hard to find discrete log of g_1 base g_2". First of all, this is probably true, but it does not seem to follow from the discrete log assumption. Plus, even if it did, I do not see how it applies, as I don't see why it is necessary (while it is clearly sufficient) to find the discrete log of g_1 base g_2 in order to obtain a claw.

The family of collision-free hash functions follows from the previous problem.

Problem 4: Signature Scheme Secure Under the Random Oracle

Assume the signer, the verifier and the adversary all have access to a *random oracle*, i.e. to a shared random function $\mathcal{O}$, where $\mathcal{O}(x)$ is a truly random string of the same length as x, $\forall x \in \{0,1\}^*$. Assuming trapdoor permutations exist, design a simple signature scheme that is existentially secure against adaptive chosen message attack (under the random oracle). Prove the claimed security of your signature scheme remembering that the adversary has access to the random oracle as well!

Solution: Assume that we don't have a random oracle. The simplistic "signature scheme" to use based on a trapdoor permutation f is to make a signature $s = f^{-1}(m)$; verification is $f(s) = m$. However, the security of such a signature is quite bad. For example, it is existentially forgeable by key only attack since $m = f(s)$. For some f it is totally insecure against chosen message attacks (e.g. Rabin's scheme, since extracting square roots is equivalent to factoring). The source of the problem is that f^{-1} is applied to the message m itself. The availability of a random oracle (RO) allows us to overcome all these problems and obtain a very simple "hash and sign" signature. Here it is.

- Key generation algorithm $G(1^k)$. Pick a random trapdoor permutation f of index length k together with a trapdoor for computing f^{-1}. Let the description of f be the public key, the description of f^{-1} (i.e. the trapdoor) be the secret key.
- Signature of m: $s = f^{-1}(\mathcal{O}(m))$.
- Verification of (s, m): $f(s) \stackrel{?}{=} \mathcal{O}(m)$.

The intuition is that since $\mathcal{O}(m)$ is totally random, the signature s is random as well no matter what m we try. Hence, to break it, we need to be able to invert f at a random point which we know is hard. Also, we can not effectively run the adaptive chosen message attack since we cannot control where to invert f (the RO controls it). However, this intuition has to be converted to a formal proof, as detailed below.

Assume the signature is existentially forgeable against adaptive chosen message attack (ACMA). This means that there exists a PPT forger F that *has free access to the oracle* $\mathcal{O}$. He is given the description of f (chosen by $G(1^k)$) and is allowed to (adaptively) get signatures of messages $m_1, \ldots, m_N$ of his choice. This means that he obtains $f^{-1}(\mathcal{O}(m_i))$. After getting all the signatures and all the oracle values that he wishes, F has to output a pair $\langle m, s\rangle$, where $m \notin \{m_1, \ldots, m_N\}$ and $f(s) = \mathcal{O}(m)$. We assume for the sake of contradiction that F has non-negligible probability of success for infinitely many k. Formally,

$$\Pr(f(s) = \mathcal{O}(m), F \text{ never asked } f^{-1}(O(m)) \mid (f, f^{-1}) \leftarrow G(1^k), (s, m) \leftarrow F^{\mathcal{O}, f^{-1}(\mathcal{O})}(f)) > \frac{1}{p_1(k)} \tag{3}$$

Here $F^{\mathcal{O}, f^{-1}(\mathcal{O})}$ means that F has oracle access to functions $\mathcal{O}$ and $f^{-1}(\mathcal{O})$ and we require that F did not ask the signature $f^{-1}(\mathcal{O}(m))$ of m. We claim, on the other hand, that F should better ask $\mathcal{O}(m)$ if he indeed wants some non-negligible success probability. Indeed, no matter what is m, if F did not asks for $\mathcal{O}(m)$ then $y = \mathcal{O}(m)$ is a *totally random value* (unknown to F). So no matter what s he outputs, the probability that $f(s)$ equals to a random value y is exctly 2^{-k}. Hence, we can strengthen (3) to (we omit the run of G to shorten the notation):

$$\Pr(f(s) = \mathcal{O}(m), F \text{ never asked } f^{-1}(\mathcal{O}(m)), \text{ but did ask } \mathcal{O}(m) \mid (s, m) \leftarrow F^{\mathcal{O}, f^{-1}(\mathcal{O})}(f)) > \frac{1}{p_2(k)} \tag{4}$$

Based on such an F, we have to obtain some contradiction. The only possible thing is to try to invert a trapdoor permutation f. And that is what we will do by constructing a successful PPT inverter A. Notice that *A does not have access to the oracle $\mathcal{O}$*. Indeed, the definition of a trapdoor permutation has nothing to do with the random oracle. But F expects a random oracle; how will we use F if we don't have a random oracle? The answer is that *A will simulate it himself!* But he better do it honestly because F really expects a random oracle. This part alone is easy to do, since A can just flip coins by himself. But this is not good enough since F is also able to request $f^{-1}(\mathcal{O}(m_i))$. To get around it, whenever A needs to provide f with either $\mathcal{O}(m_i)$ or $f^{-1}(\mathcal{O}(m_i))$, A creates a random x_i and computes $y_i = f(x_i)$. Since f is a permutation, y_i is indeed random. A pretends that $\mathcal{O}(m_i) = y_i$, which is OK since y_i is random. If F wants $\mathcal{O}(m_i)$, A only gives y_i; if F wants the signature $f^{-1}(\mathcal{O}(m_i))$, A replies with x_i. This way A can make F run correctly to the completion and F would not notice that the answers for $\mathcal{O}$ and $f^{-1}(\mathcal{O})$ were in fact simulated by A!

But how do we invert f on our challenge y? The answer is that when F succeeds getting $f^{-1}(\mathcal{O}(m)) = s$, we know from (4) that he is likely to ask $\mathcal{O}(m)$. Let us pretend that $\mathcal{O}(m) = y$! This is OK, since $y = f(x)$ where x is chosen at random, i.e. y is random as well. The only problem is that we do not know in advance when F will ask $\mathcal{O}(m)$, where m is the message that he wants to invert. But we can guess it! Since F makes at most some polynomial $q(k)$ number of calls to $\mathcal{O}$ and $f^{-1}(\mathcal{O})$, we have $1/q(k)$ chance of guessing it right. Now all the pieces come together in the following PPT inverter A that tries to compute $f^{-1}(y)$ for a random $y = f(x)$.

1. Pick a random $r \in \{1 \dots q(k)\}$. Initialize *counter* $= 0$ and start simulating $F(f)$.
2. If F asks $\mathcal{O}(\tilde{m})$ then:
   - If we have a record $(\tilde{x}, \tilde{y}, \tilde{m})$ (or $(y, \tilde{m})$), reply with $\tilde{y}$ (or y).
   - If *counter* $= r$, then reply with our challenge y and record $(y, \tilde{m})$.
   - Otherwise, pick a random $\tilde{x}$ and let $\tilde{y} = f(\tilde{x})$. Reply with $\tilde{y}$ and record $(\tilde{x}, \tilde{y}, \tilde{m})$.
3. If F asks $f^{-1}(\mathcal{O}(m_i))$ then:
   - If we have a record (x_i, y_i, m_i) reply with x_i.
   - If we have a record (y, m_i), then *fail* since we guessed the round r incorrectly.
   - Otherwise, pick a random x_i and let $y_i = f(x_i)$. Reply with x_i and record (x_i, y_i, m_i).
4. Output s if the following is true (else, fail).
   - *counter* got up to r.
   - we did not fail giving F all the answers to $\mathcal{O}$ and $f^{-1}(\mathcal{O})$.
   - F completed and produced (s, m).
   - F asked about $\mathcal{O}(m)$ and we replied with y (i.e. *counter* was r at the moment).
   - $f(s) = y$ $(= \mathcal{O}(m))$.

Our previous comments and (4) show that A inverts f for a random $y = f(x)$ with probability at least $\frac{1}{q(k)p_2(k)}$, which is non-negligible. Note, it is important that F indeed asks $\mathcal{O}(m)$ in (4), so our transformation from (3) to (4) was necessary. Hence, we strongly use the random oracle in these two places: 1) F is likely to call $\mathcal{O}(m)$, and 2) we can pretend to answer the signatures by picking a random x_i and pretending that $\mathcal{O}(m_i) = f(x_i)$ (which is indeed random since f is a permutation). The second use was crucial for the simulation of $\mathcal{O}$ and $f^{-1}(\mathcal{O})$ by A, and it justified the introduction of RO (without RO there is no way we can invert f^{-1} on any point of F's choice, so we are enforcing F to ask us f^{-1} of a *random point*). The first use is a necessary consequence of the introduction of RO, and we were lucky to manage to deal with it "at no extra-cost".