Lecture 8: February 14, 2002

Last time we discussed unconditional security for private key systems using keys as long as the message. In particular, we showed that the one-time pad was perfectly secure. In this lecture, we will investigate computational security for private key encryption: we wish to build systems that are secure with reasonably high probability against bounded adversaries (typically this means adversaries constrained to use subexponential space and time).

Recall that the building block of our private key schemes was the trapdoor permutation. Since the existence of such permutations depends on the assumption that $\mathcal{P} \neq \mathcal{NP}$, we were unable to assert with full confidence that our systems really were secure. Instead, we assumed (with at least some empirical justification) that our foundations were good and then showed what kinds of security results we could achieve with that assumption.

In private key cryptography, $\mathcal{P} \neq \mathcal{NP}$ is not the only condition that must hold for security to be possible: we must also assume the existence of *one-way* functions. As in the case of trapdoor permutations, the candidate one-way functions we will examine all seem to be difficult to invert, though there is no guarantee (yet) that they truly are. Thus, our results will carry the same aura of slight uncertainty that they did when we were "proving" theorems about public key schemes.

We begin our investigation by studying one-way functions in greater detail.

1 One-way Functions

Definition 1 *A function $f : \{0,1\}^n \to \{0,1\}^n$ is (t, ϵ)-one-way if f is computable in* $\mathrm{Poly}(n)$ *time and, for all adversaries A running in time bounded above by $t(n)$,*

$$\mathop{\mathbf{Pr}}_{x\in\{0,1\}^n}[A(f(x)) \in f^{-1}(x)] \leq \epsilon(n)$$

Roughly speaking, this definition says that a function is one-way if (1) it can easily be computed in the forward direction, and (2) any reasonably efficient adversary has only a small probability of finding *any* preimage of x (see figure 1). "Running time" in this definition is something of a misnomer: we really mean something more like the aggregate of time and space complexity. This qualification is made to rule out adversaries that store large tables and perform the inversion by doing table lookups, for instance.

When f is injective, the condition can be written

$$\mathop{\mathbf{Pr}}_{x\in\{0,1\}^n}[A(f(x)) = x] \leq \epsilon(n),$$

but we formulated the definition as we did because the general case is still interesting. Quite obviously, one-way functions bear some similarity to trapdoor permutations, though there are also some important differences.

If we assume that trapdoor permutations (TDP) do exist, then it is clear that every TDP defines a one-way function (we merely neglect the trapdoor information). The converse, however, is not necessarily true: there are one way functions for which we know of no information that makes the inversion easy. We will now examine two candidates for one-way functions that cannot be massaged into trapdoor permutations.

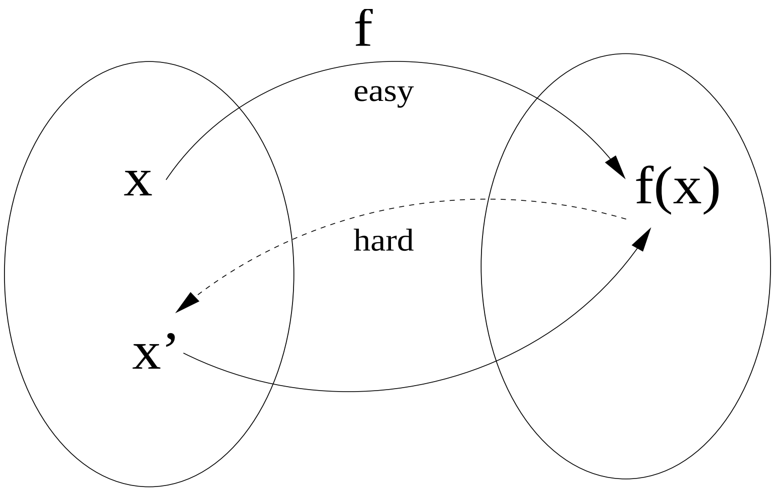

Figure 1: Illustration of a one-way function. Computing f in the forward direction (solid line) is easy, but finding any preimage of $f(x)$ (dashed line) is hard.

1.1 Discrete Logarithm

Our first example of a candidate one-way function is based on the difficulty of computing the *discrete logarithm* of an element in a cyclic group. We construct our function in the following manner:

1. Pick a random prime p such that $2^{n/2} \leq p \leq 2^n$ and a generator g for $\mathbb{Z}_p^*$. A large fraction of the elements in $\mathbb{Z}_p^*$ are generators, and there is an efficient test to determine if an element in the group is a generator (see below).

2. Define the function $E_{g,p}(x) = g^x \pmod p$ for $x \in \{1, 2, \ldots, p-1\}$. Since g is a generator, this function is a permutation, and it is conjectured that it is difficult to invert with high probability. We call x the discrete logarithm of $g^x \pmod p$ to the base g modulo n.

When our prime p is large, it is usually not possible to do an exhaustive check to see if a particular $g \in \mathbb{Z}_p^*$ is a generator. Fortunately, given the factorization of $p-1$, there is an efficient test to make this determination. Let $p_1, p_2, \ldots, p_r$ be the prime factors of $p-1$. Put $m_i = (p-1)/p_i$. Then sequentially check that $g^{m_i} \not\equiv 1 \pmod p$ for $i = 1, 2, \ldots, r$. This test works since the order of the subgroup generated by g must divide $p-1$. (Recall that the order of $\mathbb{Z}_p^*$ is $\varphi(p)$, where φ is the Euler φ-function, the number of positive integers less than p that are relatively prime to p. For prime p, $\varphi(p) = p-1$.) Now if the order of the subgroup is strictly less than $p-1$, then clearly $g^{m_i} \equiv 1 \pmod p$ for some i. Hence if none of the powers is congruent to one modulo p, then we can conclude that the order of the subgroup generated by g is $p-1$—in other words, the subgroup is all of $\mathbb{Z}_p^*$.

This process might seem long and arduous, but fear not: there are algorithms that facilitate the tasks we must perform. First, there exist algorithms that randomly pick numbers along with their factorizations; given the random choice n of such an algorithm, we can see if $n+1$ is prime. We can also look for primes of the form $p = 2q + 1$ where q is prime. Primes of this form are still dense, and since $\varphi(2q+1) = 2q$, the test to check if a given $g \in \mathbb{Z}_p^*$ is a generator will require only two modular exponentiations.

1.2 Linear Codes

Taking an entirely different approach, we can construct a linear-code based one-way function in the following way: we pick a random matrix $C \in \{0,1\}^{k \times n}$, a random vector $x \in \{0,1\}^k$, and another random vector $e \in \{0,1\}^n$ such that e has no more than $d/2$ ones.

Now, consider the following inequality relating k, n, and d:

$$\frac{k}{n} < H(\frac{d}{n}),$$

where H is the binary entropy defined by

$$H(\delta) = \delta \log \frac{1}{\delta} + (1-\delta) \log \frac{1}{1-\delta}.$$

Roughly, this says that k is a small fraction of n and d is an even smaller constant fraction. If this inequality holds, then with high probability over the random choice of the matrix C, for all x and x', $x \neq x'$, Cx and Cx' will differ in more than d positions. Thus, after adding the results of these transformations to the vector e (modulo 2), we will still have encodings of x and x' that differ significantly.

In principle, it is usually possible to reconstruct x from the output $Cx + e$. To see this, suppose that the mapping $x \mapsto Cx + e$ fails to be injective. Then for two inputs x and x', we have

$$Cx + e = y = Cx' + e.$$

Now this means that the output y differs from Cx (and Cx') by a vector that contains at most $d/2$ ones. Thus Cx and Cx' differ in no more than d positions, which happens only with very low probability given the conditions we imposed on C.

To summarize, we can describe the construction in this way: fix α and δ such that $\alpha < H(\delta)$. Now put $k = \alpha n$ and $d = \delta n$. Then randomly choosing an n-vector e with no more than $d/2$ ones, we can define a function LC by

$$LC(C, x, e) = (Cx + e, C),$$

where C is a randomly chosen $n \times k$ matrix, and the input x is a k-vector. With high probability the output of this function has a unique preimage.

One immediate "application" of the linear-code based function is in communication over noisy channels (see figure 2). Suppose Alice wants to send Bob a message x over a noisy channel, and both know the matrix C. Then Alice can send Cx, and we can view the noise-induced corruption of the message as a vector e that is added to Cx. In theory x can be reconstructed given $Cx + e$ and C, but it turns out that this inversion is difficult for all but certain classes of matrices. When C represents a random linear mapping, there is no known subexponential algorithm to reconstruct x from the outputs of LC, and we have our one-way function.

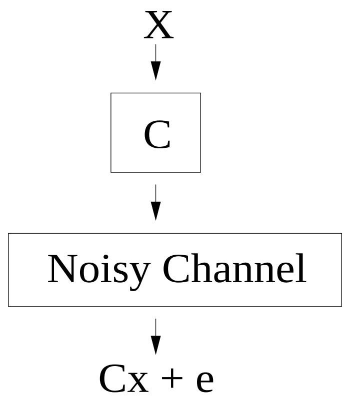

Figure 2: The use of linear-code based functions in communication over noisy channels. The noise is represented by the vector e that is added to the transmitted message Cx. Both parties in the communication know the matrix C.

Incidentally, there is no trapdoor information that would make the computation of a preimage easy.

Now, believing that one-way functions exist, we can construct private key systems with security equivalent to CCA2—that is, schemes that (probabalistically) do not leak information even when the adversary is given a decryption oracle. We build our system up by steps, adding more security as we go along.

2 Pseudorandom Generators

In the case of public-key encryption, trapdoor permutations didn't seem too different from our final objective—all the "components" of the public key scheme (i.e. the key generation, the decryption and encryption procedures, etc.) were latent in the permutation. Granted, using the permutations themselves as the public key system is weak, but it is still workable, especially if the messages are random. This is quite different from the case with private key systems, for which we have no intuition as to how we might generate keys, encrypt, or decrypt messages. With this in mind, we start out constructions with the idea of the pseudorandom generator.

Informally, a pseudorandom generator is a function $G : \{0,1\}^n \to \{0,1\}^m$, with $m > n$, such that for random input $s \in \{0,1\}^n$, the output $G(s)$ "looks random." Of course, what we desire from G is not possible in the strict information-theoretical sense, but we still have use for such generators—we wish to be able to create strings that cannot be distinguished from truly random strings by a bounded adversary. We now formalize this definition:

DEFINITION 2 *A function $G : \{0,1\}^n \to \{0,1\}^m$ is (t,ϵ)-pseudorandom if, for all adversaries A running in time bounded above by $t(n)$,*

$$\left| \mathop{\mathbf{Pr}}_{s\in\{0,1\}^n}[A(G(s)) = 1] - \mathop{\mathbf{Pr}}_{r\in\{0,1\}^n}[A(r) = 1] \right| < \epsilon.$$

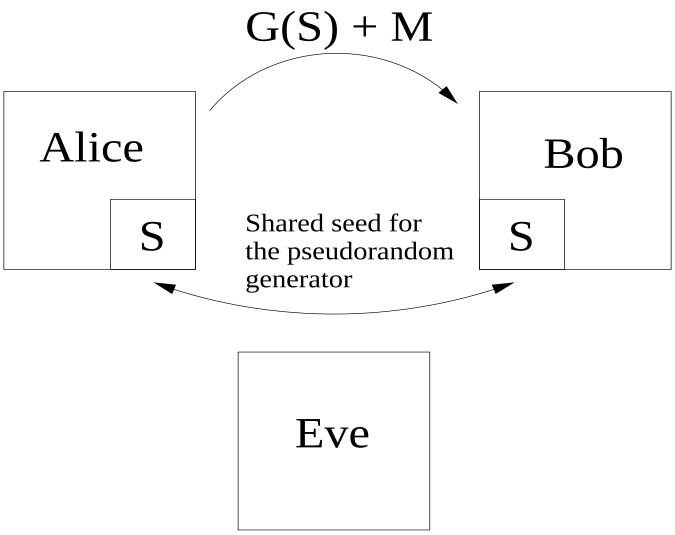

Figure 3: The pseudorandom generator "improvement" to the one-time pad. Instead of sharing a key as long as the message M, Alice and Bob share a shorter seed to the pseudorandom generator. The generator G then "expands" this to a random string as long as the message.

Now consider the following experiment. Pick a bit b at random. If it comes up 0, pick a random $x \in \{0,1\}^n$; otherwise, set $x = G(s)$. If, for all adversaries A that run in time bounded above by $t(n)$,

$$\mathbf{Pr}[A(x) = b] \leq \frac{1}{2} + \epsilon,$$

then G is (t, ϵ)-pseudorandom. In other words, a bounded adversary should not be able to distinguish the output of our generator from a random string any better than an adversary that makes its guesses based on the flip of a coin.

2.1 Improvement on the One-time Pad

Recall that the one-time pad required that the parties exchanging information shared a secret-key that was as long as the message that they wished to exchange. We can improve on this with pseudorandom generators in the following way.

Suppose that Alice and Bob share a secret key s of length n, and Alice wants to send Bob a message of length m, $m > n$ (see figure 3). Then Alice can send $c = G(s) + M$, and Bob can find M by computing $c + G(s)$. By using the generator output as the "key," Alice and Bob no longer needed to share a secret key as long as the message. Since the output of the pseudorandom generator is indistinguishable (by a bounded adversary) from the uniform distribution, the same is true of the messages.

Having introduced the definition and an application of pseudorandom generators, we now state some results concerning their construction.

2.2 Construction of generators

Both of the following results state that it is possible to construct a pseudorandom generator given a one-way (and, in the first case, bijective) function.

THEOREM 1
(**Blum-Micali-Yao**) *Given a (t, ϵ)-one-way permutation $f_n : \{0,1\}^n \to \{0,1\}^n$, there is an efficient construction of a pseudorandom generator $G : \{0,1\}^{2n} \to \{0,1\}^{2n+k}$ such that G is computable with k computations of f_n. G is then $(t^{\Omega(1)} - \mathrm{Poly}(n,k)$, $\epsilon^{\Omega(1)}k)$-pseudorandom.*

We will do the proof of the Blum-Micali-Yao result in the next lecture, showing exactly how we can make use of the one-way permutation along with a hardcore for the function to create a pseudorandom generator.

THEOREM 2
(**H.I.L.L.**) *Given a (t, ϵ)-one-way function $f : \{0,1\}^n \to \{0,1\}^n$, we can construct a pseudorandom generator $G : \{0,1\}^{n^c} \to \{0,1\}^{n^c+k}$ using $O(n^c + k)$ computations of f.*

This second result is substantially more difficult to prove than the first, and from a practical point of view it isn't very feasible. It is important because it establishes half of the claim that the existence of one-way functions (observe that the hypothesis of bijectivity in the first result is no longer used) is necessary and sufficient for the existence of pseudorandom generators. As it is now, the best known value of c is 2 (quadratic). A big open question is whether or not this construction can be made to work linearly (i.e. for $c = 1$).

We now investigate one-way functions in slightly greater depth.

3 Weak and Strong One-way Functions

Recall that in our definition of one-way functions, we stated that every bounded adversary A could invert the function only for a small fraction of the input. We can modify this condition to state that the function should be hard to invert *somewhere* in particular. This yields the concept of the *weak* one-way function.

DEFINITION 3 *A function f is a weak one-way function if it is $(t, 1 - 1/p(n))$-one-way and $p(n)$ is a polynomial.*

A one-way function is *strong* if both t and ϵ are super-polynomial. What we would like to know is if it is possible to construct strong one-way functions from weak one-way functions.

Now consider this intuitive construction: suppose $f : \{0,1\}^n \to \{0,1\}^n$ is a $(t, 1/2)$-one-way function. Define $g : \{0,1\}^{nk} \to \{0,1\}^{nk}$ by $g(x_1, x_2, \ldots, x_k) = f(x_1)f(x_2)\cdots f(x_k)$. Our intuition tells us that g can be inverted only if $x_1, \ldots, x_k$ are inputs for which f can be inverted. This would lead us to say that g is $(t, 1/2^k)$-one-way. Unfortunately, this line of reasoning is misguided. That f is $(t,\ 1/2)$-one-way does not mean that the inversion is "half-easy" and "half-hard" for all inputs but rather that the inversion difficulty is balanced for each specific algorithm A. It is very possible that the adversary that inverts g will apply an algorithm uniquely customized to the input x_i when inverting $f(x_i)$. In this case, the computation of the "strength" of g was incorrect.

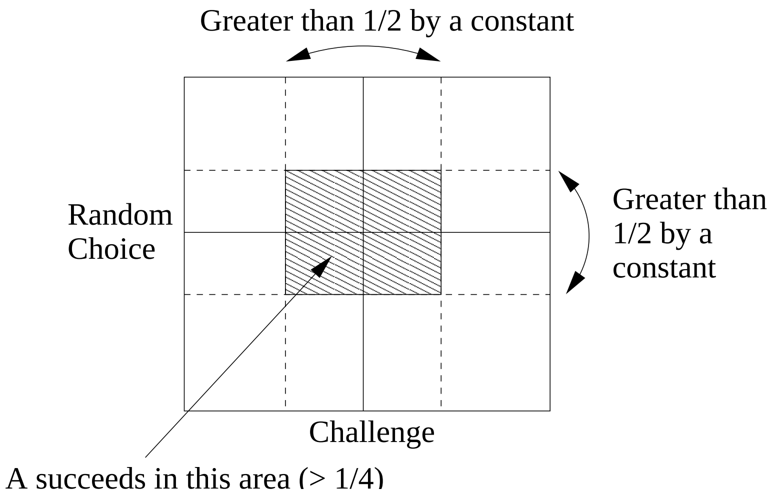

Figure 4: Relationship between inversion probabilities for f and g.

We'll now show a correct argument based on a reduction in the following way. We construct a new one-way function g from our one-way function f, and we show that if we can invert g with a certain probability, then we can invert f with a probability that is greater than what we originally allowed for f. Here is the construction.

Suppose f is a $(t, 1/2)$-one-way function. Define the function g as $g(x_1, x_2) = f(x_1)f(x_2)$. Let us first start with the intuition before making the most general argument. Suppose we have an adversary A that, given $f(x_1)f(x_2)$, can invert its inputs with probability greater than 0.26. For now assume that the distribution is in the shape of a "box" (see figure 4). Now we can define a new adversary A' to invert f:

```
A'(y)
  Repeat t times
    Pick x1 at random
    Run A(f(x1), y)
    If a preimage of y is found, output preimage and halt with success
```

Notice that in figure 4, the probability that the "challenge" y falls within the range that is "good" for A is greater than $1/2$ (since A succeeds with probability greater than $1/4$). Thus, the probability that A' succeeds in inverting y is

$$\begin{aligned} &\sqrt{0.26} \times \mathbf{Pr}[\text{At least one random choice falls within the "good" area}] \\ =&\sqrt{0.26} \times \left[1 - \left(\frac{1}{2}\right)^t\right] \end{aligned}$$

which can be made arbitrarily close to $\sqrt{0.26}$ by taking t sufficiently large. Hence we have succeeded in inverting f with probability greater than $1/2$, contradicting the fact that f is $(t, 1/2)$-one-way.

The keen reader will notice that this analysis has "cheated": we assumed that the area of success was a square. As an alternative, we can consider another geometry such as

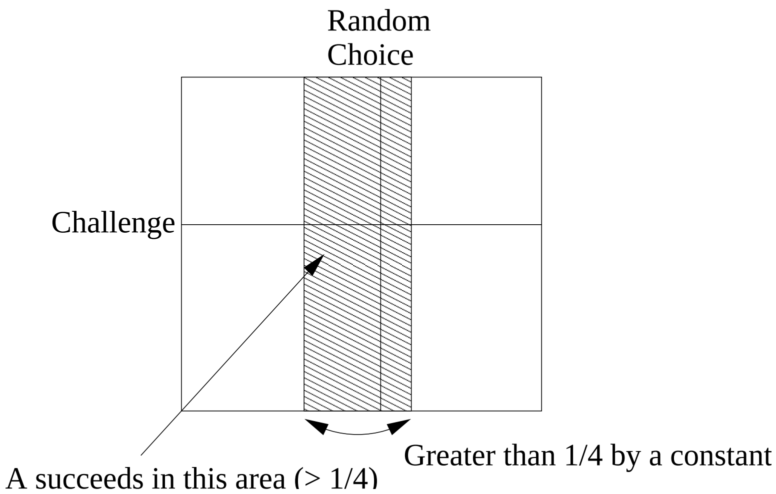

Figure 5: Another possibility

that shown in figure 5. Notice that we can switch the roles of our random choice and the challenge so that the probability that the challenge is good is unity and the probability that our random choices are good is $1/4$. Then we can invert with probability $1-(3/4)^t$, which we can make arbitrarily close to unity by taking t sufficiently large.

Now consider the situation in which our success region is arbitrary in shape (figure 6). Our claim in this general setting is that if the area of the region is greater than $1/4$, then we can invert f with probability greater than $1/2$—a contradiction. We start by assuming the existence of an algorithm A that inverts g with probability $0.26 > 1/4$. Using A we can construct A' that will invert f with probability greater than $1/2$:

```
A'(y)
  Repeat t times
    Pick x1 at random
    Run A(f(x1), y)
    Pick x2 at random
    Run A(y, f(x2))
    If A is successful in either invocation, output preimage
      and halt with success; otherwise continue the loop
```

Now to analyze the success of this algorithm, we define a set S as follows: $S = \{x : \mathbf{Pr}[\text{Internal loop inverts} f(x)] > \gamma\}$. We leave γ unspecified for now. Then we can write

$$\mathbf{Pr}[\text{Algorithm inverts } f] \geq \mathbf{Pr}[S] \times (1-(1-\gamma)^t).$$

If we can show that $\mathbf{Pr}[S] \geq 1/2$, then by taking t sufficiently large we will be able to invert f with probability greater than $1/2$, and that contradiction will establish the claim. Now since A succeeds with probability at least 0.26, we can write

$$\begin{aligned} 0.26 \leq \mathbf{Pr}[A \text{ succeeds}] = & \mathbf{Pr}[A(f(x_1), f(x_2)) \text{ succeeds } \wedge x_1 \in S \wedge x_2 \in s] + \\ & \mathbf{Pr}[A(f(x_1), f(x_2)) \text{ succeeds } \wedge x_1 \notin S] + \\ & \mathbf{Pr}[A(f(x_1), f(x_2)) \text{ succeeds } \wedge x_2 \notin S] \end{aligned}$$

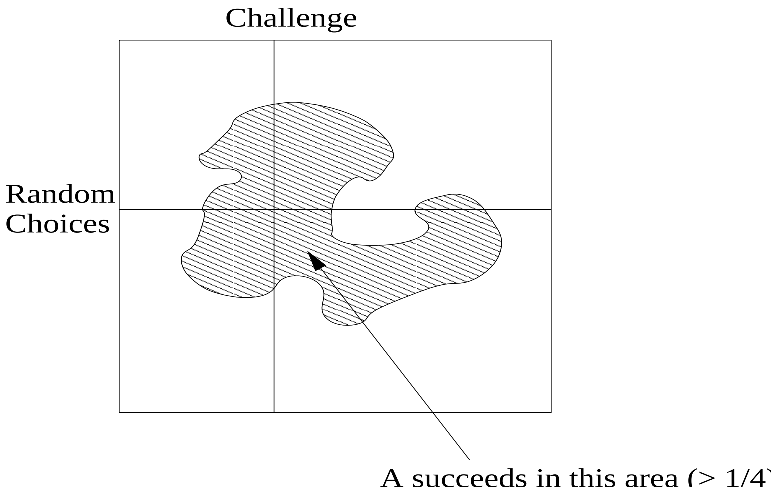

Figure 6: An arbitrarily shaped region. We make random choices along one side while the challenge is on the other (see lines 3-4 of the algorithm A').

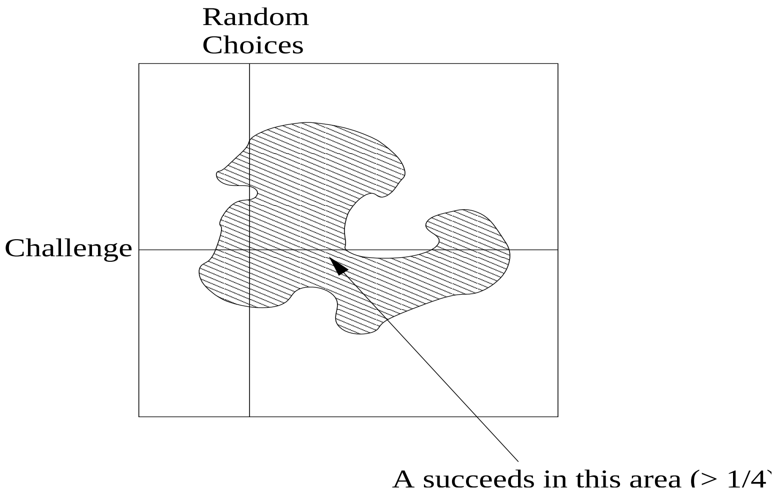

Figure 7: The same arbitrary region with the random choices and the challenge reversed (lines 5-6 of A').

The summation is rough, but we can get away with that by bounding it from above. The first term is bounded by

$$(\mathbf{Pr}[x \in S])^2,$$

the second term by

$$\sum_{x_1 \notin S} \mathbf{Pr}_{x_2}[A(f(x_1), f(x_2)) \text{ succeeds }] \times \mathbf{Pr}[x_1],$$

and the third term by

$$\sum_{x_2 \notin S} \mathbf{Pr}_{x_1}[A(f(x_1), f(x_2)) \text{ succeeds }] \times \mathbf{Pr}[x_2].$$

These last two terms are equivalent to the internal loop in the algorithm (lines 3-8) succeeding, so we can bound them above by γ. Hence,

$$0.26 \leq (\mathbf{Pr}[x \in S])^2 + 2\gamma.$$

Now we can take γ arbitrarily small: put $2\gamma < 0.05$ for example. Then

$$\mathbf{Pr}[x \in S] \geq \sqrt{0.255} > 1/2.$$

Hence,

$$\mathbf{Pr}[\text{Algorithm inverts } f] \geq \mathbf{Pr}[S] \times (1 - (1-\gamma)^t) > 1/2$$

for sufficiently large t. This contradicts the claim that f is $(t, 1/2)$-one-way, so we conclude that indeed our function g is $(t, 0.26)$-one-way.