0368.4162 Foundations of Cryptography November 21, 2019

Lecture 4: The Goldreich-Levin Theorem and Pseudorandom Functions

Lecturer: Nir Bitansky

1 Previously on Foundations of Crypto

We started talking about pseudorandmness, a key concept in cryptography. We stated the theorem by HILL [HILL99] saying that one-way functions (OWFs) imply pseudorandom generators (PRG). A central part of this theorem is the Goldreich-Levin theorem [GL89] showing that every OWF implies has a randomized hardcore bit. We saw that such hardcore bits easily imply PRGs, if we have a one-way permutation. The general case of OWFs is outside the scope of this course.

We'll now recall the definition of randomized hardcore bits and prove the GL theorem.

Definition 1.1 (Randomized Hardcore Bit (RHCB))**.** *A poly-time computable function $B : \{0,1\}^* \to \{0,1\}$ is randomized hardcore bit of f if for every n.u. PPT A there exists a negligible μ such that:*

$$\Pr\left[A(f(U_n), U'_n) = B(U_n; U'_n)\right] \leq \frac{1}{2} + \mu(n) \ .$$

2 The Goldreich-Levin Theorem

In '89 Oded Goldreich and Leonid Levin proved:

Theorem 2.1 ([GL89])**.** *The function*

$$B(x;r) := \langle x, r\rangle \mod 2 = \sum_i x_i r_i \mod 2 = \bigoplus_{i:r_i=1} x_i$$

is hardcore for any OWF.

As we've already seen, such a theorem is proven by a reduction (from an attacker that predicts to one that inverts). At the heart of this reduction, lies a beautiful algorithm that has subsequently had different important interpretations, and has given rise to foundational concepts in TOC (beyond crypto per se).

Proof. Fix any (w.l.o.g deterministic) adversary A that given $(f(x), r)$ predicts $B(x, r)$, with advantage $\varepsilon = \varepsilon(n)$:

$$\Pr_{x,r}\left[A(f(x), r) = \langle x, r\rangle \mod 2\right] \geq \frac{1}{2} + \varepsilon(n) \ .$$

We will construct an inverter A' for f relying on the following two claims.

Claim 2.2. *With probability $\varepsilon/2$ over a random $x \leftarrow \{0,1\}^n$:*

$$p_x := \Pr_r\left[A(f(x), r) = \langle x, r\rangle \mod 2\right] \geq \frac{1}{2} + \varepsilon/2 \ ,$$

that is, fixing x, A can predict $B(x, r)$ for a random r, with good advantage.

Claim 2.3 (The GL Decoding Algorithm)**.** *There exists an algorithm GL such that for any $x \in \{0,1\}^n$ and function $\tilde{L}_x : \{0,1\}^n \to \{0,1\}$*

$$\text{if } \Pr_r\left[\tilde{L}_x(r) = \langle x, r\rangle \mod 2\right] \geq 1/2 + \delta \ , \text{ then } \Pr\left[GL^{\tilde{L}_x(\cdot)}_{n,\delta} = x\right] \geq \Omega(\delta^2/n) \ .$$

The algorithm runs in time $\mathrm{poly}(n, \delta^{-1})$.

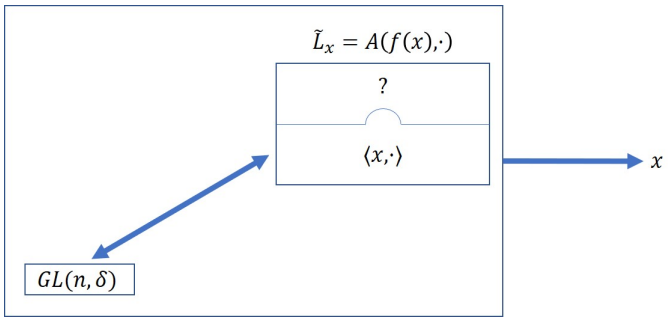


Figure 1: The Goldreich-Levin Algorithm

Indeed, given these two claims, we can easily get an inverter A' as follows:

- Given $f(x)$, "hope x is good" (occurs with probability $\varepsilon/2$).
- Apply the GL decoding procedure, with $\delta = \varepsilon/2$ to retrieve x from $\tilde{L}_x := A(f(x),\cdot)$ (succeed with probability $\Omega(\varepsilon^2/n)$).

Overall, this inverter succeeds with probability $\Omega(\varepsilon^3/n)$.

The core of the proof is the second claim, but for completeness, let us first prove the first claim.

Proof of Claim 2.2. By an averaging argument:

$$\begin{aligned}\frac{1}{2}+\varepsilon \le \mathbb{E}_x p_x &\overset{(1)}{=} \Pr_x\left[p_x < \frac{1+\varepsilon}{2}\right]\cdot \mathbb{E}\left[p_x \;\middle|\; p_x < \frac{1+\varepsilon}{2}\right] + \Pr_x\left[p_x \ge \frac{1+\varepsilon}{2}\right]\cdot \mathbb{E}\left[p_x \;\middle|\; p_x \ge \frac{1+\varepsilon}{2}\right] \\ &\overset{(2)}{\le} \Pr_x\left[p_x < \frac{1+\varepsilon}{2}\right]\cdot \frac{1+\varepsilon}{2} + \Pr_x\left[p_x \ge \frac{1+\varepsilon}{2}\right]\cdot 1 \le \frac{1+\varepsilon}{2} + \Pr_x\left[p_x \ge \frac{1+\varepsilon}{2}\right] \ .\end{aligned}$$

where (1) is by the law of total expectation and (2) is by monotonicity of expectation. □

We now move to the GL algorithm

Proof of Claim 2.3. Rather than giving the algorithm directly, we'll work our way toward it through easier warmup steps. In what follows, inner products, additions, and multiplications will be $\bmod 2$ (and we will not write this explicitly every time).

Warmup 1: Imagine that $\tilde{L}_x(\cdot) \equiv \langle x,\cdot\rangle$. How would we learn say x_1 from the oracle? we can simply make the query $e_1 = (1,0,\ldots,0)$ and obtain exactly $\langle x, e_1\rangle = x_1$. We can learn any other x_i similarly.

Warmup 2: Imagine that

$$\Pr_r\left[\tilde{L}_x(r) = \langle x, r\rangle\right] \ge \frac{3}{4} + \delta \qquad \text{for } \delta \ge 0.01.$$

How would we learn x_1 now? We cannot simply apply the previous approach because the oracle may error on the specific query e_1. Still, the oracle is correct at many random points, and we'd like to take advantage of that. The idea is that we can embed the query e_1 into random queries. Specifically, we can choose a random $r \leftarrow \{0,1\}^n$, and query both r and $r + e_1$. Note that if the oracle is correct on both then we can use the linearity of the inner product to learn:

$$\langle x, r + e_1\rangle - \langle x, r\rangle = \langle x, e_1\rangle = x_1 \ .$$

Now, what is the chance that it answers both correctly? Note, that each query *individually* is distributed uniformly at random. Thus, each one will be answered with probability at least $3/4 + \delta$ and by a union bound both will be answered correctly with probability at least $1/2 + 2\delta$.

We can now amplify this probability by repeating the experiment some number m times, and taking the majority. That is sample $r_1, \ldots, r_m$. For each i get a candidate

$$\tilde{L}_x(r_i + e_1) - \tilde{L}_x(r_i) = x_{1,i} \ ,$$

and output the majority bit as our x_1. How many times m we need to repeat?

Lemma 2.4 (Chernoff-Hoefding Bound)**.** *Let $X_1, \ldots, X_m \in \{0,1\}$ be independent experiments such that $X_i = 1$ with probability p, then*

$$\Pr_{X_1,\ldots,X_m}\left[\left|\frac{1}{m}\sum_{i\in[m]} X_i - p\right| \geq \Delta\right] \leq 2^{-\Delta^2 m} \ .$$

In our case, $p \geq 1/2 + 2\delta$, and we'd like to know how many times m to repeat to guarantee that the majority succeeded with high probability. Using the above tail bound, for $\Delta = 2\delta$, we can set $m = \delta^{-2}\log n$, which isn't too big since δ is constant. We will get an error with probability at most

$$2^{-(2\delta)^2\delta^{-2}\log n} = n^{-4} \ .$$

Thus, we can learn all n coordinates x_i w.p. at least $1 - n^{-3}$ (this follows by a union bound).

Warmup 3: Imagine that

$$\Pr_r\left[\tilde{L}_x(r) = \langle x, r\rangle\right] \geq \frac{1}{2} + \delta \qquad \text{for } \delta \geq 0.01.$$

We cannot apply the previous strategy, each individual answer may be incorrect w.p. 49%, and the probability that both are correct could be as low as 2%. The problem is that for each r, we make two correlated queries and have to absorb the error twice. Let's try to avoid it. For the time being, we'll use our imagination even further, and assume that we're given the following additional help for free. We're given samples $(r_1, \langle x, r_1\rangle), \ldots, (r_m, \langle x, r_m\rangle)$, where $r_1, \ldots, r_m$ are chosen independently at random, and as before $m = \delta^{-2}\log n = 10^4 \log n$. Can we solve the problem now?

Now, to learn x_1, we just need to ask our oracle one query $r_i + e_1$ for each i, and we know it's correct with probability 51%. Again, we can take majority of all the samples

$$\tilde{L}_x(r_i + e_1) - \langle x, r_i\rangle = x_{1,i}$$

and recover x_1 with probability $1 - n^{-4}$. In fact, we can use the same "helper samples" $(r_1, \langle x, r_1\rangle), \ldots, (r_m, \langle x, r_m\rangle)$ to recover each of the coordinates x_i, and as before take a union bound (the recovery procedure at different coordinates is not independent, but it doesn't have to be, it's enough that it succeeds with high enough probability).

But where will the helper samples come from? Well, how about we just guess them? That is, we'll sample the random r_i ourselves and guess the values $\langle x, r_i\rangle$. Our guess will hit the right value w.p. $2^{-m} = 2^{\delta^{-2}\log n} = 2^{-10^4 \log n} = n^{-10^4}$. This is still an inverse polynomial probability (admittedly, a ridiculously small one), which is enough for us.

The Actual Algorithm: We've imagined enough for one day, and now need to deal with the actual setting where the advantage δ is not constant but rather an arbitrarily small polynomial $1/p(n)$. In this case, we cannot just guess the helper values, as there are $n^{p^2(n)}$ such values.

The main idea is to choose $r_1, \ldots, r_m$ not completely at random, but in a correlated manner. On one-hand, they'll be correlated enough so that we can guess all $\langle x, r_i \rangle$ with reasonable probability. On the other hand, they will still be pseudorandom in some sufficient sense.

The algorithm $GL_{n,\delta}^{\tilde{L}_x(\cdot)}$ proceeds as follows:

1. Let $k = \log(2\delta^{-2}n + 1)$.
2. Sample $s_1, \ldots, s_k \leftarrow \{0,1\}^n$ (think about these as a seed for pseudorandomly generating the r's).
3. Sample $\sigma_1, \ldots, \sigma_k \leftarrow \{0,1\}$ (think about these as guesses for $\langle x, s_1 \rangle \ldots \langle x, s_k \rangle$).
4. For any non-empty subset $I \subseteq [k]$, let $r_I = \sum_{j \in I} s_j$ (these will be our pseudorandom r's).
5. Also, for each such I, let $\rho_I = \sum_{j \in I} \sigma_j$ (think about this as a derived guess for $\langle x, r_I \rangle$).
6. For each i, to learn x_i, for every I, obtain from the oracle $\tilde{L}_x(r_I + e_i)$, compute $x_{i,I} = \tilde{L}_x(r_I + e_i) - \rho_I$, and output the majority $maj\{x_{i,I}\}_I$.

Let's analyze the algorithm. First note that we guess all σ_j as $\langle x, s_j \rangle$, with probability $2^{-k} = \Omega(\delta^2 n^{-1})$. Whenever this happens, we also derive correct guesses ρ_I. That is

$$\rho_I = \sum_{j \in I} \sigma_j = \sum_{j \in I} \langle x, s_j \rangle = \langle x, \sum_{j \in I} s_j \rangle = \langle x, r_I \rangle \ .$$

From hereon, let us assume that we indeed guessed correctly, and examine the probability that we recover x. Let us focus on x_1 (the analysis for any other coordinate is the same). We need to make sure that the majority of answers $\tilde{L}_x(r_I + e_1)$ is indeed $\langle x, r_I + e_1 \rangle$.

First, note that each r_I is individually distributed uniformly at random, and thus also $r_I + e_1$. This means that the oracle answers correctly w.p. $1/2 + \delta$. However, now we cannot apply a Chernoff bound because the queries across different I's are not independent. Nevertheless, they are *pairwise independent*, meaning that for each two I, I' the variables $r_I, r_{I'}$ are independent.

Claim 2.5. *$\{r_I\}_{\emptyset \neq I \subseteq [k]}$ are pairwise independent.*

Proof Sketch. Let $I, I' \subseteq [k]$ s.t $I \neq I'$. Then there exists $m \in [k]$ such that $m \in I \triangle I'$. Assume w.l.o.g $m \in I$. Now, s_m and s_i are independent for any $i \in I'$, and thus s_m and $\sum_{i \in I'} s_i = r_{I'}$ are independent. Since s_m is included in sum representing r_I, we have that $r_I, r_{I'}$ are independent. □

We can now use a weaker tail bound for pairwise independent variables.

Lemma 2.6 (Chebyshev Bound (Special Case))**.** *Let $X_1, \ldots, X_m \in \{0,1\}$ be pairwise independent experiments such that $X_i = 1$ with probability p, then*

$$\Pr_{X_1,\ldots,X_m}\left[\left|\frac{1}{m}\sum_{i \in [m]} X_i - p\right| \geq \Delta\right] \leq \frac{1}{\Delta^2 m} \ .$$

In our setting, $p \geq 1/2 + \delta$, and we can set $\Delta = \delta$ and $m = 2^k - 1 = 2\delta^{-2}n$. Applying the inequality, we deduce that $\tilde{L}_x(r_I + e_1)$ will agree with $\langle x, r_I + e_1 \rangle$ for a majority of the sets I, except with probability $1/2n$. By a union bound, this will happen for all coordinates i, with probability at least $1/2$. If in addition, we guessed the right values $\sigma_1, \ldots, \sigma_k$, which happens w.p. $\Omega(\delta^2/n)$, then the algorithm succeeds.

Overall, the algorithm succeeds with probability $\frac{1}{2} \cdot \Omega(\delta^2/n)$. □

This concludes the proof that B is a (randomized) hardcore bit for any OWF. □

Reinterpreting The GL Algorithm. The GL algorithm has been given different interpretations and had a major impact on TOC, beyond cryptography. We mention two such central interpretations:

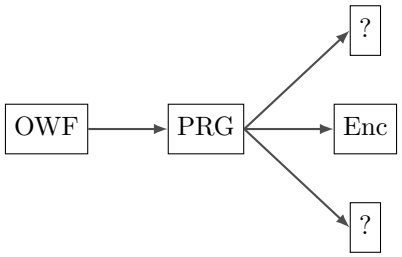
1. **List Decoding:** A central concept in coding theory, and more generally in TOC, is that of *list decoding*. Here we want to give a meaningful decoding guarantee even if the number of errors exceed the unique decoding bound. GL is exactly such an algorithm for the Hadamard code (where each codeword has the form $(\langle x, r\rangle \mid r \in \{0,1\}^n)$). The distance of this code is $1/2$, meaning that we can uniquely decode within a ball of radius $1/4$. The algorithm shows that we can even decode in a ball of radius $1/2 - \delta$, but rather than a unique codeword, we get a list of at most $O(\delta^{-2}n)$ codewords (The Johnson bound actually says that there will only be $O(\delta^{-2})$ such words).

2. **Learning:** GL solves a variant of learning parity with noise problem we discussed earlier, where instead of getting random noisy equations, the learner has the freedom to choose the equations. In particular, it could choose them to be correlated, which as we see makes a huge difference.

   Another interpretation is that the GL algorithm allows for any boolean function to learn a list of its δ-heavy Fourier coefficients in the Hadamard Fourier basis.

3 Beyond PRGs: Pseudorandom Functions

So far we've seen that starting from OWFs, we can obtain PRGs and encryption for messages of arbitrary polynomial length. We now aim to achieve additional cryptographic goals based on these tools, and see how far they could get us. Our first step will be to construct a much stronger form of PRG, which we will call a



pseudorandom function (PRF). We start by a motivating example, within the familiar context of encryption.

A Motivating Question: Encrypting Multiple Messages. We know that assuming OWFs, we can encrypt messages of arbitrary polynomial length $\ell(n) = n^{O(1)}$ (using a key of fixed size n). Our security definitions, however, only dealt with attackers that see a single (perhaps long) ciphertext.

Can we securely encrypt multiple messages?

There are several ways of defining what security for multiple messages actually means. For now, we will consider the following natural definition known as security against *known plaintext attacks* (KPA).[1] Also, for simplicity we will restrict attention to single bit messages. This is w.l.o.g in the sense that to encrypt longer messages, we can encrypt their bits separately (although this may cost us in efficiency).

Definition 3.1 (Multi-Message Encryption (against Known Plaintext Attacks))**.** *A bit encryption scheme (E, D) is multi-message secure if for any polynomial $\ell(\cdot)$,*

$$\{E_{sk}(x_1), \ldots, E_{sk}(x_\ell)\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell(n)} \\ y\in\{0,1\}^{\ell(n)}}} \approx_c \{E_{sk}(y_1), \ldots, E_{sk}(y_\ell)\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell(n)} \\ y\in\{0,1\}^{\ell(n)}}},$$

[1] In a nutshell, in a known plaintext attack, the encrypted messages are known ahead of time. Later, we shall define the notion of chosen plaintext attacks, where the attacker may adaptively request to see different encrypted messages, depending on all encryptions it had seen so far.

where in the above $sk \leftarrow \{0,1\}^n$.

Apriori we may hope that any encryption scheme that's secure for a single message would also be secure if we reuse it for two messages. However, we've already seen that this is not true in general (for instance, in one-time pad). Still, we may hope to construct a designated scheme that will be secure for multiple messages. In fact, in some sense, we've already did.

Stateful Encryption. Our construction of encryption for long messages from one-extra-bit encryption actually already supports many messages to some extent. Specifically, it had a chain structure where we could always encrypt (and decrypt) an additional bit m_{i+1} provided that the encryptor (and decryptor) has the current secret key sk_i. Such *stateful* encryption schemes are commonly known as *stream ciphers* or *state ciphers*. The obvious caveat of such encryption schemes is that they require the parties to remain synchronized. In particular, if some messages are dropped, then we can no longer guarantee consistent decryption. There are solutions that can *resynchronize* after a few failures, but in general this is a problem.

Publicly Synchronizing. One naïve way to solve the synchronization problem is for the encryptor to send his state online. This of course cannot work in general — the state may contain secret randomness (e.g., the next secret key) that will compromise security. However, using PRGs it seems that we can actually construct an encryption scheme that only requires keeping a public state, and in fact, a pretty short one.

The construction is simple: The encryptor would keep a counter of the number of messages it encrypted so far. When it wants to encrypt the ith bit, it will first expand its n-bit random secret key s using a PRG G and use the ith output bit $G(s)_i$ as a one-time pad. Specifically, it will send $i, G(s)_i \oplus m_i$ as the encryption. (Recall that we can indeed extend the output of G as much as we want to any polynomial number of bits.)

In this scheme, there is no issue of synchronization, but the scheme still has some undesired properties. First, the encryptor has to maintain a state (the counter i). Furthermore, the complexity of encryption/decryption potentially grows with the number of encrypted messages — this is certainly the case if we use the chain-style PRG extension we've previously seen.

The Solution: PRFs. Both of the above problems can be solved using PRFs, which intuitively can be viewed as a PRG with an *exponential* number of output bits where the ith output bit $G(s)_i$ can be computed in fixed polynomial time (independent of i). There are several conceivable ways to formalize this. We will consider a relatively strong definition which requires that the function $i \longmapsto G(s)_i$ is indistinguishable from a truly random function.

Definition 3.2 (Pseudorandom Function (PRF))**.** *A pseudorandom function is a polynomial-time function F that takes as input a seed $s \in \{0,1\}^n$ and an index $i \in \{0,1\}^n \cong [2^n]$ and outputs a bit, such that for any n.u. PPT $A = \{A_n\}_{n\in\mathbb{N}}$ there exists a negligible μ such that for all $n \in \mathbb{N}$:*

$$\left|\Pr\left[A_n^{F_s(\cdot)} = 1 \;\middle|\; s \leftarrow \{0,1\}^n\right] - \Pr\left[A_n^{R(\cdot)} = 1 \;\middle|\; R \leftarrow \{0,1\}^{\{0,1\}^n}\right]\right| \leq \mu(n) \ .$$

In this definition, A_n is an oracle-aided algorithm (it does not view the function at its entirety, but can make a polynomial number of arbitrary queries). The notation $\{0,1\}^{\{0,1\}^n}$ stands for the set of all functions mapping $\{0,1\}^n$ to $\{0,1\}$. In other words, R is simply a random function, that for each input string returns a random independent bit.[2]

Encrypting Multiple Messages Using PRFs. Given PRFs, we can now follow the same rational as before, but completely lose the state, and in particular, encrypt in fixed time (independently of the number of messages encrypted so far). The scheme is given by:

- $E_{sk}(m)$:

[2]Note that PRFs indeed strengthen the notion of PRGs. In particular, any PRF F can be used to construct a PRG G (of arbitrary polynomial length), where the ith output bit of $G(s)$ is simply $F_s(i)$.

- Interpret sk as a seed for a PRF.
- Sample a random $r \leftarrow \{0,1\}^n$ and output $r, F_{sk}(r) \oplus m$.

- $D_{sk}(r, v)$:
  - Output $v \oplus F_{sk}(r)$.

It's easy to verify the correctness of the scheme. Let's sketch the proof of security.

Multi-Message Security, Sketch. Notice that we can describe our encryption algorithm E_{sk} as an oracle-aided algorithm $\mathcal{E}^{F_{sk}}$ that uses F_{sk} as a black box. Now, we can consider an alternative (mental) experiment, where instead of using F_{sk} we use a truly random oracle R.

Claim 3.3. *For any polynomial ℓ,*

$$\left\{\mathcal{E}^{F_{sk}}(x_1), \ldots, \mathcal{E}^{F_{sk}}(x_\ell) \mid sk \leftarrow \{0,1\}^n\right\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell(n)}}} \approx_c \left\{\mathcal{E}^{R}(x_1), \ldots, \mathcal{E}^{R}(x_\ell) \;\middle|\; R \leftarrow \{0,1\}^{\{0,1\}^n}\right\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell(n)}}}.$$

This claim follows directly from the security of the PRF. Indeed, given a non-uniform PPT A that distinguishes between the above two ensembles (for infinitely many n and $x \in \{0,1\}^{\ell(n)}$), we can construct a distinguisher $B^{(\cdot)}$ for the pseudorandom function. The distinguisher $B^{(\cdot)}$ simply emulates $\mathcal{E}^{(\cdot)}$ to encrypt the bits of x, and answer the oracle calls that $\mathcal{E}^{(\cdot)}$ makes using its own oracle. B distinguishes a PRF from a random function with exactly the same advantage that A distinguishes encryptions relative to F_{sk} from encryptions relative to R.

This means that to finish the proof, we just need to show that the scheme is secure when instantiated with a random function. This is actually true in a statistical (rather than computational sense).

Claim 3.4.

$$\left\{\mathcal{E}^{R}(x_1), \ldots, \mathcal{E}^{R}(x_\ell)\right\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell(n)} \\ y\in\{0,1\}^{\ell(n)}}} \approx_s \left\{\mathcal{E}^{R}(y_1), \ldots, \mathcal{E}^{R}(y_\ell)\right\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell(n)} \\ y\in\{0,1\}^{\ell(n)}}}.$$

To see that this is true, notice that as long as the random values $r_1, \ldots, r_\ell$ chosen by $\mathcal{E}$ are distinct, then the produced encryptions $(r_1, R(r_1) \oplus x_1), \ldots, (r_\ell, R(r_\ell) \oplus x_\ell)$ are uniformly random, independently of the underlying message x. Thus, it suffices to bound the probability that the random strings r_i are not distinct. There are at most $\binom{\ell}{2}$ such potential collisions, each occurring with probability 2^{-n}. Overall, by a union bound, the probability that these strings are not distinct is at most $2^{-n}\ell^2$, which is negligible. □

Note on Randomized Encryption. Note that the constructed encryption scheme is randomized. In fact, any stateless encryption scheme for more than a single message must be randomized (think why).

References

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 25–32, 1989.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.