Three Types of Randomness

Adam J. Raczkowski

Senior Honors Thesis
Department of Computer Science
Tufts University
April, 2009

Abstract

Three types of randomness are integral to the strength of public-key cryptography. With the techniques of [ABK$^+$06], we present an analysis of how the ability to quickly distinguish Kolmogorov randomness allows for a probabilistic attack on two of the conjectured hard problems underlying public-key cryptography: the discrete logarithm and factoring. Specifically, Kolmogorov random strings are pertinent to inverting one-way functions and distinguishing pseudorandomness from true uniform randomness. This method provides a lens that more sharply categorizes the hardness of the discrete logarithm and factoring in the hierarchy of well-known complexity classes. An overview of public-key cryptosystems is presented alongside the traditional analysis of one-way functions and pseudorandom generators. Using this approach, we establish relationships between provably secure public-key cryptography, Kolmogorov complexity, and the essential cryptographic primitives and put forth conjectures of how these relationships may extend into methods for solving other types of problems.

Acknowledgments

I am very grateful to my thesis committee of Professors Anselm Blumer, Benjamin Hescott, and Steve Homer. I extend extra thanks to Professor Hescott for his tireless enthusiasm for the subject matter. His many hours of help fueled my passion for computational theory.

Dedication

For Irene and Michael Panetta, and Dorothy and Adam C. Raczkowski.

Contents

1 Introduction

Chance is the providence of adventurers.
Napoleon Bonaparte

A complexity theorist is part philosopher and part mathematician. Why do certain problems seem hard, and others have simple and fast algorithms? How do these notions of "easy" and "hard" translate into mathematical relationships underlying such problems?

For the cryptographer, the inherent, abstract hardness of a problem translates into a measure of security of communication between two parties. Unfortunately for cryptography, information regarding the hardness of problems remains difficult to extract from a problem formulation. A problem simply stated as integer factorization contains nuances of number theory, advanced group theory, and computational theory.

Computational methods analyzing the expected scenarios of an attack on cryptographic protocols estimate the chance that an attacker will successfully eavesdrop. This analysis relies on three fundamental types of randomness. The attacker may extend an assault with Monte Carlo or Las Vegas algorithms, the realm of randomized algorithms that complete in a feasible amount of time. However, embedded in any secure cryptosystem is a method to create output that appears random—as far as the output could be statistically distinguished from coin flips. A third, descriptive type of randomness bridges the gap for the attacker to complete a statistical attack.

These three types of randomness provide a toolbox for security analysis of a cryptosystem. The result of this analysis is beyond the simple solution to a public-key cryptosystem in nondeterministic polynomial time, which is abstract and likely physically unrealizable. Rather, it is in terms of a probabilistic algorithm with precomputed tables of data. This latter definition provides a stronger sense of security and stems from the interplay of the three types of randomness.

An overview of present-day public-key cryptosystems is given in section 3. Section 4 contains relationships of these cryptosystems to their hard underlying problems. In section 5, the basic cryptographic primitives, one-way functions and pseudorandom generators, are introduced. These primitives are generalized further in section 7. An overview of Kolmogorov randomness is given in section 6, and the result of combining Kolmogorov random strings with the cryptographic primitives is explored in sections 9 and 10. The main result of [ABK$^+$06] for factoring and the discrete logarithm is proved in section 9. The difficulty of proof of existence of secure public-key cryptography is explored in section 8. Many conjectures relating the constructions of this thesis to quantum problems and randomness generated by more capable computing models are introduced and discussed throughout this paper. All conjectures are delineated in the concluding section.

2 Definitions and Notations

2.1 Functions and Asymptotics

Let $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$ denote the natural numbers, the integers, and the real numbers, respectively. Let S, X, Y denote sets. The *power set* $\mathcal{P}(S)$ is the set of all subsets of S. The *identity function* $Id_S : S \to S$ is the function defined by $Id_S(s) = s$ for all $s \in S$. Given two functions $f : X \to S$ and $g : S \to Y$, the *composition of* g *with* f is the function $(g \circ f) : X \to Y$ defined by $(g \circ f)(x) = g(f(x))$ for all $x \in X$. For subset $B \subseteq S$, the *preimage* of B under f is $f^{-1}(B) = \{x \in X | f(x) \in B\}$. A *multivalued function* $f : X \rightrightarrows Y$ is a single-valued function $f : X \to \mathcal{P}(Y)$.

Given two sets X and Y, a *relation* R between X and Y is a subset $R \subseteq X \times Y$. An *equivalence relation* on a single set X is a relation $R \subseteq X \times X$ such that for all $x, y, z \in X$,

1. (Reflexivity). $(x, x) \in R$.

2. (Symmetry). $(x, y) \in R$ whenever $(y, x) \in R$.

3. (Transitivity). $(x, y) \in R$ and $(y, z) \in R$ implies $(x, z) \in R$.

Given an equivalence relation R on a set X and an element $x \in X$, the *equivalence class* $[x]$ is the set of all $y \in X$ such that $(x, y) \in R$. The set of all equivalence classes forms a disjoint partition of the set X.

Let $f, g : \mathbb{N} \to \mathbb{N}$. f is *big oh* of g if there exists a constant $C \in \mathbb{N}$ such that $f(n) \leq C \cdot g(n)$ for almost all $n \in \mathbb{N}$. *Almost all* $n \in \mathbb{N}$ means for all but finitely many $n \in \mathbb{N}$. Denote $f(n) = O(g(n))$ whenever f is big oh of g.

Let $\lg(n)$ denote the base-2 logarithm of n.

2.2 Probability

A *probability distribution* over a finite set S is a function $\Pr : \mathcal{P}(S) \to \mathbb{R}$ where $\Pr(S) = 1$, $\Pr(\emptyset) = 0$ and $\Pr(A \cup B) = \Pr(A) + \Pr(B)$ for disjoint subsets A and B. For a set T with non-zero probability, the *conditional probability of S given T* is $\Pr(S|T) = \frac{\Pr(S \cap T)}{\Pr(T)}$. Two sets S, T are *independent* if $\Pr(S \cap T) = \Pr(S) \cdot \Pr(T)$. If $T \neq \emptyset$, then independent sets S, T satisfy $\Pr(S|T) = \Pr(S)$.

Let Pr be a probability distribution over a finite set S. A *random variable* X is a function $X : \mathcal{P}(S) \to \mathbb{R}$. For $r \in \mathbb{R}$, denote $\Pr\{X = r\}$ as $\Pr X^{-1}\{r\}$. The *expected value* of X is $\mathsf{E}[X] = \sum r \cdot \Pr\{X = r\}$ where the sum is taken over all the finite values in the range of X. The *variance* of X is $\mathsf{Var}[X] = \mathsf{E}[(X - \mathsf{E}[X])^2]$. Two random variables X, Y are *independent* if $\Pr\{X = r \text{ and } Y = s\} = \Pr\{X = r\} \cdot \Pr\{Y = s\}$ for all $r, s \in \mathbb{R}$. To clarify, $\Pr\{X = r \text{ and } Y = s\} = \Pr(X^{-1}\{r\} \cap Y^{-1}\{s\})$.

Let X be a random variables and let $a > 0$. Markov's inequality states

$$\Pr\{|X| \geq a\} \leq \frac{\mathsf{E}[X]}{a}$$

and Chebyshev's inequality states

$$\Pr\{|X - \mathsf{E}[X]| \geq a\} \leq \frac{\mathsf{Var}[X]}{a^2}$$

And if $\Pr\{X < b\} = 1$, Markov's reverse inequality states for $a < \mathsf{E}[X]$

$$\Pr\{|X| > a\} \geq \frac{\mathsf{E}[X] - a}{b - a}$$

2.3 Number Theory and Groups

For integers $a, b, n \in \mathbb{Z}$, *a is equivalent to b modulo n* if there exists an integer m such that $a - b = mn$. This is denoted $a = b \mod n$ and is an equivalence relation on the integers. Let $\gcd(a, b)$ be the *greatest common divisor* of a and b. a is *relatively prime* to b if $\gcd(a, b) = 1$. $\phi(a)$ is defined as the number of integers between 1 and $a - 1$, inclusive, that are relatively prime to a.

A *group* is a set G equipped with a function $* : G \times G \to G$ written with infix notation $*(a, b) = a * b$. $*$ satisfies three properties:

1. There exists $e \in G$ such that $e * a = a$ for all $a \in G$. e is the *identity element* of G.

2. $*$ is *associative*: $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.

3. For all $a \in G$, there exists an element $a' \in G$ such that $a' * a = e$. a' is called the *inverse* of a and is denoted a^{-1} in multiplicative groups or $-a$ in additive groups.

A *subgroup* H of a group G is a subset of G that is a group when equipped with the group operation on G. In this setting, it is important that the group operation is closed in H. H as a subgroup of G is denoted $H \leq G$. For any subgroup $H \leq G$ and an element $a \in G$, the *(left) coset* aH is the set of products of a with an element of H. For $S \subseteq G$, the *subgroup generated by S* (denoted $\langle S \rangle$) is the set of all products of powers of elements of S.

$\mathbb{Z}$ denotes the group of integers under the operation of addition. For $n \in \mathbb{N}$, $n\mathbb{Z}$ is the subgroup of $\mathbb{Z}$ defined on all integer multiples of n. $\mathbb{Z}_n$ denotes the *quotient group* $\mathbb{Z}/n\mathbb{Z}$, that is the set of equivalence classes inherited from the equivalence relation $a = b \mod n$. $\mathbb{Z}_n^*$ is the subset of equivalence classes $[a]$ of $\mathbb{Z}_n$ for a relatively prime to n. $\mathbb{Z}_n^*$ is a group when equipped with multiplication modulo n.

Let $\oplus : \mathbb{Z}_2^n \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ denote the *exclusive or* function. The i^{th} position of $x \oplus y$ is 0 if both x_i and y_i have the same parity, and it is 1 otherwise. Let $\odot : \mathbb{Z}_2^n \times \mathbb{Z}_2^n \to \mathbb{Z}_2$ denote the *inner product* where $x \odot y$ is $\Sigma_{i=1}^{n} x_i y_i$. Here, x_i and y_i are the i^{th} bit of x and y, respectively.

2.4 Turing Machines and Language Classes

An *alphabet* is a finite set of symbols. A *string* over an alphabet is a finite sequence of symbols from the alphabet. The string of zero symbols is the *empty string* and is denoted λ. The *concatenation* of two strings joins the two sequences of strings. The concatenation of two symbols a and b is denoted ab, and the concatenation of two strings s, t is denoted $s \cdot t$. Let $\textsc{Prefix}(x, i)$ and $\textsc{Suffix}(x, i)$ denote the first and last i characters of a string x, respectively.

For an alphabet Σ and an $n \in \mathbb{N}$, Σ^n is the set of strings of length n built from symbols of Σ. Σ^* is the union of all Σ^n over each natural number n. A bijective and efficiently computable function $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ is called a *pairing function*. For any function $f : \Sigma^* \to \Sigma^*$ with *support* $S \subseteq \mathbb{N}$, we consider only the action of f on Σ^s for each $s \in S$. Let U_n denote the uniform distribution over Σ^n.

A *Turing machine* M is a tuple $(\Sigma, \Gamma, Q, \delta, q_0, q_{accept}, q_{reject})$ satisfying

1. Σ and Γ are alphabets where $\Sigma \subsetneq \Gamma$. There is a blank character $b \in \Gamma \setminus \Sigma$. Σ is called the *input alphabet* and Γ is called the *tape alphabet*.

2. Q is a finite set of states. q_0, q_{accept}, and q_{reject} are distinct states in Q. q_0 is the *initial state* of the machine, and q_{accept}, q_{reject} are *halting states*.

3. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the *transition function*, defining how the machine processes characters on the tape while in a particular state, and how the head of the machine moves after this processing.

A *universal Turing machine* is a Turing machine that takes as input descriptions of other Turing machines and simulates their computation with minimal overhead.

A *probabilistic Turing machine* is a Turing machine augmented with a read-only tape initialized uniformly with symbols 0 and 1 (or, coin flips) at the start of the computation. A *zero-error probabilistic Turing machine* is a probabilistic Turing machine with a special state $q_{unknown}$; that is, the machine can answer I Don't Know in addition to Accept or Reject.

Fix a function $f : \mathbb{N} \to \mathbb{N}$. $\mathsf{DTIME}(f(n))$ is the set of languages decidable by a deterministic Turing machine M in time $O(f(n))$. $\mathsf{DSPACE}(f(n))$ is the set of languages decidable by a deterministic Turing machine in space $O(f(n))$. $\mathsf{NTIME}(f(n))$ and $\mathsf{NSPACE}(f(n))$ are the nondeterministic analogues of $\mathsf{DTIME}(f(n))$ and $\mathsf{DSPACE}(f(n))$, respectively. $\mathsf{BPTIME}(f(n))$ is the set of languages decidable by a probabilistic Turing machine with an additional restriction on a language L. If $x \in L$, then such a probabilistic machine must accept with probability at least 2/3 over all uniform choices of a random tape. Likewise, if $x \notin L$, then the machine much reject with probability at least 2/3. $\mathsf{ZPTIME}(f(n))$ is defined analogously for a zero-error probabilistic machine.

Definition 2.1.

$$\mathsf{P} = \bigcup_{k=1}^{\infty} \mathsf{DTIME}(n^k)$$

Definition 2.2.

$$\mathsf{NP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(n^k)$$

Definition 2.3.

$$\mathsf{BPP} = \bigcup_{k=1}^{\infty} \mathsf{BPTIME}(n^k)$$

Definition 2.4.

$$\mathsf{ZPP} = \bigcup_{k=1}^{\infty} \mathsf{ZPTIME}(n^k)$$

Definition 2.5.

$$\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{DTIME}(2^{n^k})$$

Definition 2.6.

$$\mathsf{SUBEXP} = \bigcap_{\delta>0} \mathsf{DTIME}(2^{n^\delta})$$

Definition 2.7.

$$\mathsf{L} = \bigcup_{k=1}^{\infty} \mathsf{DSPACE}(\lg(n)^k)$$

Definition 2.8.

$$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{DSPACE}(n^k)$$

Definition 2.9. $f : \Sigma^* \to \Sigma^*$ is a *polynomial-time computable function* (or P-*computable function*) if there exists a polynomial-time Turing machine M such that for every input $x \in \Sigma^*$, M halts on input x with $f(x)$ remaining on its tape. A $\mathcal{C}$-time computable function is defined analogously for any complexity class $\mathcal{C}$.

For a computation in NP, the *left set* of an input x and path p is the set of all accepting paths lexicographically smaller than p on the computation on input x. UP is defined like NP where there is at most one accepting path for any computation. SPP is defined like BPP where there is exactly one more accepting path than rejecting path for an $x \in L$ and equal numbers of both when $x \notin L$. BQP is the analogue of BPP when using an appropriate quantum computation model.

2.4.1 Oracles, Advice, and Circuits

Fix A a subset of Σ^*. A *Turing machine with oracle* A is a Turing machine with an added query tape and states q_{query}, q_{yes}, and q_{no}. Whenever the machine enters q_{query}, the computation will next enter state q_{yes} if the string written on the query tape is in L, or q_{no} otherwise. For a complexity class $\mathcal{C}$, the class $\mathcal{C}^L$ is the set of languages decidable by an appropriate machine with oracle access to L.

A *circuit* defined on n bits computing a boolean function is a binary tree with n leaves such that every node with one child represents a NOT gate, an each node with two children is labeled as either an AND gate or an OR gate. The computation of a circuit processes upward to produce a single bit. For any circuit C, $\mathsf{SIZE}(C)$ is equal to the number of nodes in C.

For an alphabet Σ, a *polynomial advice function* is a function $h : \mathbb{N} \to \Sigma^*$ that is not necessarily computable but where $|h(n)| \leq n^k$ for some k. A language L is *decidable with polynomial advice* h if h is a polynomial advice function and $\langle x, h(|x|)\rangle$ is accepted by a Turing machine if and only if $x \in L$. $\mathsf{P/poly}$ is the set of languages decidable in polynomial time with a polynomial advice function. Equivalently, $\mathsf{P/poly}$ is the set of languages decidable by a family of polynomial-size circuits $\{C_n\}_{n=0}^{\infty}$, where C_n decides the strings of L of length n.

2.4.2 Reductions

Fix an alphabet Σ, and let A and B be subsets of Σ^*.

A is *polynomial-time many-one reducible* to B (denoted $A \leq_m^{\mathsf{P}} B$) if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for any $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$. A is *polynomial-time many-one equivalent* to B (denoted $A \equiv_m^{\mathsf{P}}$) if $A \leq_m^{\mathsf{P}} B$ and $B \leq_m^{\mathsf{P}} A$.

A is *polynomial-time Turing reducible* to B (denoted $A \leq_T^{\mathsf{P}}$) if there is a polynomial-time Turing machine with oracle access to B that decides A. A is *polynomial-time Turing equivalent* to B (denoted $A \equiv_T^{\mathsf{P}} B$) if $A \leq_T^{\mathsf{P}} B$ and $B \leq_T^{\mathsf{P}} A$.

A is *logarithmic-space Turing reducible* to B (denoted $A \leq_T^{\mathsf{L}}$) if there is a logarithmic-space Turing machine with oracle access to B that decides A. A is *logarithmic-space truth-table reducible* to B (denoted $A \leq_{tt}^{\mathsf{L}}$) if there is a logarithmic-space Turing machine that decides A with a function $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ that generates a finite set of queries to oracle B for the computation on an input x.

3 A Survey of Public-Key Cryptosystems

Cryptography is in essence the study of secure communications. Whether transmission of credit card numbers or storage of sensitive trade secrets, users depend upon computer systems that work analogously to a lock and key. These systems do not necessarily offer *secrecy* of the act of communication; that is, users cannot plausibly deny their engagement in communication. They also do not ensure the *integrity* of information, a guarantee that the data was not changed in transit. What these cryptographic protocols intend to provide is a shield from eavesdroppers with suitable time and computing resources—a well-equipped digital lockpicker.

Notice that cryptography, engendered by the lack of secure communications channels, seeks to provide secure communications over inherently insecure channels. What entails an insecure channel, especially when cryptography allows secure communication through such a channel? In this paper, a *insecure channel* is a means of transmitting information between two parties in which one or more eavesdroppers may view the information in transit without modifying the information. The last restriction is particularly important as eavesdroppers who have the capability of sending information through the communications channel may veil themselves in a *man-in-the-middle attack*, purporting to be one party to the other. Solving such a computer systems issue is not *per se* the goal of cryptography. Yet cryptographic methods do not apply to just computer systems; they can apply to secure communication through the postal service or phone lines or even smoke signals. In this sense, cryptography describes security independently of the means of communication.

With this definition in mind, how can information be relayed securely over an insecure channel? Imagine a customer of an online retailer submitting a purchase with their credit card information. Suppose their credit card number is 503248. In most computer systems, this number would be represented in binary like so:

$$503248_{10} = 1111010110111010000_2$$

where the $_n$ means "base n." Suppose the customer and the retailer met previously in person to swap a random number. Here, a suitably random binary string can be generated by flipping a coin and writing a 1 for each head and 0 for each tail. Assume the random number is as long as the credit card number, and perform the following operation:

$$\begin{array}{rr} & 1111010110111010000 \\ \oplus & 0011011001110100000 \\ \hline & 1100001111001110000 \end{array}$$

Here, the first number is the credit card number, and the second number is the random number known by both parties. $\oplus$ is the exclusive-or operation defined bit-by-bit: It is 1 when both bits are different and 0 when both bits are the same. The customer then sends this result (1100001111001110000) to the retailer. Upon receiving this number, using the same random number as before, the retailer determines the credit card number using the same operation:

$$\begin{array}{rl} & 1100001111001110000 \\ \oplus & 0011011001110100000 \\ \hline & 1111010110111010000 = 503248_{10} \end{array}$$

This recovers the original information as $\oplus$ is its own inverse operation. How can an attacker who receives the transmitted binary string attempt to recover the credit card number? As long as there are many credit card numbers of the same length as the one protected by the one-time pad, then there are corresponding one-time pads to produce this same output. Therefore, the attacker cannot conclusively determine the original credit card number from the information available. It is important to note that a one-time pad is truly meant to be used only one time; subsequent uses of the same one-time pad can leak information about multiple messages and provide an avenue of attack to recover the original messages.

The one-time pad acts like a key to a digital lock. The standard definition of a cryptosystem is defined precisely like a locking mechanism, with keys, locking procedures, and unlocking procedures. For a one-time pad, the locking procedure and unlocking procedure were identical, which leads to a *symmetric cryptosystem.* In general, these procedures may be different, and whenever they are different, the cryptosystem is called *asymmetric.*

Definition 3.1. A *cryptosystem* is a five-tuple (P, C, K, E, D) satisfying the following:

1. P is a finite set of strings (the "plaintexts").
2. C is a finite set of strings (the "ciphertexts").
3. K is a finite set of strings (the "keys").
4. For every $k \in K$, there is a $e_k \in E$ and a $d_k \in D$. $e_k : P \to C$ and $d_k : C \to P$ are both functions such that $d_k \circ e_k = Id_P$.

A *protocol* uses the cryptosystem to successfully and securely relay a message between two people (Alice and Bob) across an insecure channel. The most common way is to randomly select a key $k \in K$ across a secure channel. Alice sends her message m to Bob by sending each of $e_k(m)$. Bob decrypts the message by processing this string through the decryption function d_k, yielding the original message.

For this system, the key must be chosen in person by Alice and Bob or across a channel deemed secure for their purposes. However, such channels are impractical for communication across computer networks—even the Internet. To eliminate the need of a secure channel to communicate the key initially, a system keeping the key private but the encryption method public would obviate the need to communicate the key. Such a system's security hinges on the ease with which discovering the decryption method d_k when the encryption method e_k is known. These cryptosystems are *public-key cryptosystems.*

Since P is finite, an attacker (usually referred to as Eve) can always break a secure channel through brute force search. The idea behind a public-key cryptosystem is to ensure that no approach is better than a brute force search: testing many possible plaintexts to discover which maps to a given ciphertext.

3.1 RSA

RSA, named for its inventors Rivest, Shamir, and Adleman, exploits the hardness of factoring whole numbers. RSA has become the *de facto* standard in public-key cryptographic protocols. It has widespread applications in digital signatures as well as in secure communication. The public key, which is known to Alice and Eve, consists of a product of two primes. The private key contains the original primes. If Eve can factor the product, then she could recover the plaintext. The underlying security of RSA depends on knowledge of number theory.

Definition 3.2 ([RSA83])**.** Fix $n \in \mathbb{N}$. *RSA* is a cryptosystem defined over $P = C = \mathbb{Z}_n$ and whose keys are $K = \{(n, p, q, a, b) | p, q \text{ are prime}, n = pq, \text{ and } ab = 1 \mod \phi(n)\}$. Each $(n, p, q, a, b) \in K$ defines the following encryption/decryption pair of functions:

$$e_k(x) = x^b \mod n$$

$$d_k(x) = x^a \mod n$$

The private key is (p, q, a), and the public key is (n, b).

It should be clear that P and C are finite. K, too, is finite because for a fixed n, there are only finitely many choices for p, q, a, and b. What remains to be proven is that $d_k(e_k(x)) = x$ for every $x \in \mathbb{Z}_n$. Since *RSA* draws upon number theory for its function, the following lemma will be useful in showing this result.

Lemma 3.3. *$a = b \mod pq$ if and only if $a = b \mod p$ and $a = b \mod q$, where p, q are distinct primes.*

Proof. If $a = b \mod pq$, then the result is straightforward. Assume $a = b \mod p$ and $a = b \mod q$ for p, q distinct primes. Then $a = cp + b$ and $a = dq + b$ for some c, d. Comparing these equations, $cp = dq$. Without loss of generality, assume $p < q$. By Euclid's lemma, q divides c or q divides p. Since $p < q$, it must be true that q divides c. Therefore $c = eq$ for some e and $a = epq + b$, which ultimately implies $a = b \mod pq$. □

Now, *RSA* can be shown to be a true cryptosystem.

Theorem 3.4. *For the RSA cryptosystem, $d_k \circ e_k = Id_P$.*

Proof. There are two cases: $x \in \mathbb{Z}_n^*$ and $x \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$.

Consider $x \in \mathbb{Z}_n^*$. Write $ab = s\phi(n) + 1$ as $ab = 1 \mod \phi(n)$. Then:

$$x^{ba} = x^{s\phi(n)+1} \mod n = x \cdot (x^{\phi(n)})^s \mod n$$

The order of the group $\mathbb{Z}_n^*$ is $\phi(n)$. Therefore, for $x \in \mathbb{Z}_n^*$, $x^{\phi(n)} = 1$, and so $x^{ba} = x \mod n$.

If, on the other hand, $x \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$, then x is an integer multiple of p or q. The $x = 0$ case is trivial. From the above lemma, $x^{ba} = x \cdot (x^{\phi(n)})^s \mod n$ if and only if $x^{ba} = x \cdot (x^{\phi(n)})^s \mod p$ and $x^{ba} = x \cdot (x^{\phi(n)})^s \mod q$. Therefore it suffices to prove each of these two congruences for $x = p$ and $x = q$.

Without loss of generality, let x be a multiple of p. Then the first congruence modulo p is obviously true. Suppose $p \neq q$ and x is not equivalent to 0 modulo pq. Then p is an element of order $q - 1$ in $\mathbb{Z}_q$. As $\phi(n) = \phi(pq) = (p - 1)(q - 1)$, it must follow that $(x^{\phi(n)})^s = (x^{q-1})^{(p-1)s} = 1$. Thus, the congruence modulo q is also true. □

Although the essential security concerns of RSA will be presented in a later section, there are a few considerations to make in light of current prime factorization techniques. To encrypt a message of a specific length requires a public key n of at least the same length as the message. (However, in practice the public-key protocol is used to transmit a one-time pad, which also must be at least the length of the message.) The primes p and q should be chosen so that neither prime is small enough to risk attack from tables of precomputed valued of n. Furthermore, if p and q are too close, then p and q are close to $\sqrt{pq}$ and so their product is vulnerable to trial division methods. On this note, any implementation of RSA intended for multiple users should have a proper random number generator ensuring that no pairs of primes p and q are heavily favored over others.

More modern research concludes that both $p-1$ and $q-1$ should be divisible by large primes or else pq may succumb to Pollard's $p-1$ factorization [Pol74]. Likewise, $p+1$ and $q+1$ should also be divisible by large primes to prevent attack by William's $p+1$ factorization [Wil82]. While considerations abound for choice of p and q, our attention is focused on the relationship between RSA and factoring. This relationship will be made clear in the formalization of RSA and factoring in a formal complexity setting in section 4.

3.2 Rabin

The Rabin cryptosystem, named after its creator Michael Rabin, also uses factoring as its foundation. In contrast to RSA, Rabin has a stronger connection to factoring. This connection will be made apparent in the formalization of Rabin as a cryptographic set in section 4. However, the Rabin system has not achieved widespread usage because each ciphertext corresponds to up to four possible plaintexts.

Definition 3.5 ([Rab79])**.** Fix $n = pq$, where p, q are distinct primes with $p, q = 3 \mod 4$. *Rabin* is a cryptosystem where $P = C = \mathbb{Z}_n$ and whose keys $K = \{(n, p, q, B) | 0 \leq B \leq n-1\}$. Each $(n, p, q, B) \in K$ corresponds to the following encryption/decryption pair:

$$e_k(x) = x(x+B) \mod n$$

$$d_k(x) = \sqrt{\frac{B^2}{4} + x} - \frac{B}{2} \mod n$$

The private key is (p, q), and the public key is (n, B).

Since each ciphertext corresponds to four plaintexts, the Rabin cryptosystem requires that the function d_k is multivalued. To prove this is a cryptosystem under this restriction demands that the original plaintext is one of four possible decrypted plaintexts computed from the ciphertext. In reality, this ambiguity in decryption precludes Rabin from widespread adoption.

Theorem 3.6. *For the Rabin cryptosystem, $x \in (d_k \circ e_k)(x)$ for all $x \in P$.*

Proof. Fix $k = (n, p, q, B)$ satisfying the constraints of the Rabin cryptosystem. Then for all $x \in \mathbb{Z}_n$,

$$d_k(e_k(x)) = \sqrt{\frac{B^2}{4} + x(x+B)} - \frac{B}{2} \mod n$$

The square root may be written $\sqrt{(x + \frac{B}{2})^2}$. Therefore, the composition of d_k and e_k results in a multivalued function. For argument x, the possible values are $\omega(x + \frac{B}{2}) - \frac{B}{2}$ where ω is a square root of 1 in $\mathbb{Z}_n$. ☐

Here, if $\omega = 1$, the original plaintext is recovered. Yet, the knowledge that $\omega = 1$ recovers the original plaintext is useless since the square root in $\mathbb{Z}_n$ needs to be calculated. For example, in $\mathbb{Z}_{15}$, 1 has four square roots: 1, 4, 11, and 14.

In general, there are up to four square roots in $\mathbb{Z}_n$. A polynomial time algorithm can compute the square roots when $p, q = 3 \mod 4$, which justifies why p and q are chosen with this restriction. However, these are not the only class of prime numbers with this property. Many methods are known [KL94, Sho98, Gen07] to factor polynomials in finite fields, and these methods can be extended to factor quadratics to produce the desired square roots.

While Rabin does lack desirable properties for everyday use, it does demonstrate that there is more than one way to formulate a public-key cryptosystem on the same underlying problem.

3.3 ElGamal

The remaining classes of public-key cryptosystems use the discrete logarithm rather than factoring as their core. The discrete logarithm is exactly the extension of the natural logarithm into a finite group. Given a group G and two elements a, b in G, the discrete logarithm of a with base b is an integer n where $a = b^n$ if such an n exists. The ElGamal cryptosystem uses the hardness of the discrete logarithm problem rather than factoring to achieve its security. Common implementations use multiplicative cyclic prime groups $\mathbb{Z}_p^*$.

Definition 3.7 ([Gam85])**.** Let $(G, *)$ be a finite group with $\alpha \in G$ an element where the discrete logarithm problem is intractable in $\langle \alpha \rangle$, the subgroup generated by α. *ElGamal* is a cryptosystem where $P = G$, $C = G \times G$, and whose keys are $K = \{(G, \alpha, a, \beta) | \beta = \alpha^a\}$. For a given $(G, \alpha, a, \beta) \in K$ and for a random $r \in \mathbb{Z}_{|\langle \alpha \rangle|}$, define the following encryption/decryption pair:

$$e_k(x) = (\alpha^r, x * \beta^r)$$

$$d_k(x_1, x_2) = x_2 * x_1^{-a}$$

The public key is $((G, *), \alpha^r, \beta^r)$, and the private key is (a, r).

The random value r is a *seed* that can change arbitrarily between protocol uses so long as it is constant for any single protocol use. The public key includes the group. Therefore, both the sender and receiver have agreed upon a group G and can quickly compute group multiplication $*$ and inverses. For cyclic prime groups $\mathbb{Z}_p^*$, group multiplication is multiplication modulo p, and inverses calculated in d_k are computable in polynomial time using the extended Euclidean algorithm. Again, it remains to show that the decryption function d_k recovers the plaintext encoded by e_k.

Theorem 3.8. *For the ElGamal cryptosystem, $d_k \circ e_k = Id_P$.*

Proof. For any given r, $e_k(d_k(x, r)) = x * \beta^k * (\alpha^k)^{-a}$. Given that $\beta = \alpha^a$, the following substitution can be made:

$$(\alpha^k)^{-a} = (\alpha^a)^{-k} = \beta^{-k}$$

Therefore, $(d_k \circ e_k)(x) = x * \beta^k * \beta^{-k} = x$. □

Due to the closeness of the security of the ElGamal cryptosystem and the hardness of the discrete logarithm, the underlying difficulty in computing discrete logarithms in the group G solely describes the difficulty of an adversary retrieving the plaintext from the ciphertext. For $G = \mathbb{Z}_p^*$, many algorithms exist to compute the discrete logarithm, but none is polynomial time [Sha71, PH78]. Special groups known as elliptic curves form another class of groups in which the discrete logarithm problem is thought to be hard.

3.4 Elliptic Curve

Elliptic curves in cryptography generalize the notion of an elliptic equation over an arbitrary field. Solutions to these curves form groups that have driven much recent research in cryptography. This is a paradigm shift from the factoring-based cryptography, such as RSA, to discrete logarithm-based cryptography. The hardness of the elliptic curve derives from finding solutions to their equations over a particular field.

Definition 3.9 ([Mil86, Kob87])**.** Let $p > 3$ be prime and $a, b \in \mathbb{Z}_p$ where $4a^3 + 27b^2 \neq 0 \mod p$. The *elliptic curve* given by p, a, and b is the set of solutions $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ of the polynomial $y^2 = x^3 + ax + b$ over the field $\mathbb{Z}_p$ plus a point O at infinity.

In fact, the solution set of an elliptic curve over $\mathbb{Z}_p$ forms a group. The details of proving that this formulation is indeed a group are omitted.

Theorem 3.10 ([Mil86, Kob87])**.** *Let E be the set of solutions to the elliptic curve defined by parameters p, a, and b. Consider the following operation $+(\cdot, \cdot)$ on points in E:*

First, define $P + O = O + P = P$ for all $P \in E$.

Second, for (x, y) and $(x, -y)$ in E not equal to O define their sum to be O.

Finally, for points (x_1, y_1), (x_2, y_2) not satisfying the above two conditions, define their sum to be (x_3, y_3) where:

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

$$\lambda = \begin{cases} \frac{3x_1^2 + a}{2y_1} & \text{if } x_1 = x_2 \text{ and } y_1 = y_2 \\ \frac{y_2 - y_1}{x_2 - x_1} & \text{otherwise} \end{cases}$$

Then $(E, +)$ is a group.

As elliptic curves present themselves as intractable for the discrete logarithm, there are disadvantages to using ElGamal encryption based on these groups. First, known points on the curve correspond to plaintexts, but no such efficient algorithm exists to calculate points on an elliptic curve. Second, ciphertexts are about four times as long as their decrypted plaintexts, exacerbating the storage and transport of a large amount of confidential data. However, elliptic curves can be used to encode across a more computationally friendly field. Supposing $H \leq E$ is a subgroup with hard discrete logarithms, then the following cryptosystem can be defined.

Definition 3.11 ([MV93])**.** Let E be an elliptic curve defined over $\mathbb{Z}_p$ for some $p > 3$. Let H be a cyclic subgroup of E. The *Menezes-Vanstone elliptic curve cryptosystem* is a cryptosystem where $P = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, $C = E \times \mathbb{Z}_p^* \times \mathbb{Z}_p^*$. The keyspace $K = \{(E, \alpha, a, \beta) | \beta = a\alpha; \alpha \in E\}$.

For a given $k = (E, \alpha, a, \beta) \in K$ and a random $r \in \mathbb{Z}_{|H|}$, define the following encryption function:

$$e_k(x_1, x_2) = (r\alpha, c_1x_1, c_2x_2)$$

where $(c_1, c_2) = r\beta$. Define the decryption function:

$$d_k(x_0, x_1, x_2) = (x_1 c_1^{-1}, x_2 c_2^{-1})$$

where $(c_1, c_2) = a x_0$.

The private key is (a, r), and the public key is (E, α, β).

The next theorem demonstrates the completeness of the definition of the Menezes-Vanstone cryptosystem.

Theorem 3.12. *For the Menezes-Vanstone elliptic curve cryptosystem, $d_k \circ e_k = Id_P$.*

Proof. Fix an elliptic curve E and a cyclic subgroup H of E. Let $(x_1, x_2) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ and $r \in \mathbb{Z}_{|H|}$. Then,

$$(d_k \circ e_k)(x_1, x_2) = d_k(r\alpha, c_1 x_1, c_2 x_2) = (c_1 x_1 d_1^{-1}, c_2 x_2 d_2^{-1})$$

where $(c_1, c_2) = r\beta$ and $(d_1, d_2) = ar\alpha$. Now since E is abelian, $ar\alpha = ra\alpha = r\beta$. Therefore $c_1 = d_1$ and $c_2 = d_2$. But H is also abelian, giving $(c_1 x_1 d_1^{-1}, c_2 x_2 d_2^{-1}) = (x_1 c_1 c_1^{-1}, x_2 c_2 c_2^{-1}) = (x_1, x_2)$. ☐

We need that the discrete logarithm is intractable in H to ensure that a is not determined from $\beta = a\alpha$.

4 Complexity of Cryptographic Sets

In the many years that public key cryptography has been used for secure communications between computer systems, and with the high rewards for an adversary to break these channels, no sufficient algorithms have been produced to consistently defeat these systems. Intuitively, the security of a public key cryptosystem is inextricably tied to the problem of determining the private key from the information in the public key and the ciphertext. Computational complexity theory, with its formalization of feasible and infeasible problems, proves useful in analyzing the underlying security of cryptographic protocols. This requires the cryptosystems as well as any possibly hard problem used in that cryptosystem to be formulated as a decidable set.

This formalism for now provides relationships between cryptographic protocols and their underlying hard problems. These relationships are established through reductions between sets that define a transformation of one type of problem (What is the plaintext corresponding to this ciphertext?) to another (How can I factor this number into two primes?).

The prevailing philosophy in transforming a protocol into a set is: Give the public-key information directly, but supply a search variable for each of the items comprising the private key. This is a reasonable formulation only if any potential eavesdropper only has as much information as the public key and the ciphertext communicated over the insecure channel. In general, membership to this set of an instance should be based on whether the private key is lexicographically less than the given private key. If such a set is computable in P, then a simple binary search can break the cryptosystem in a feasible amount of time.

4.1 Factoring-Based Cryptographic Sets

As many conjecture factoring to be hard, it is important to see how this problem is expressed in the language of computational complexity.

Definition 4.1. $FACTOR = \{\langle x, b\rangle | x \text{ has a factor } f \text{ where } 1 < f \leq b\}$

Here the pairing function $\langle \cdot, \cdot \rangle$ allows both inputs x and b to a Turing machine. x is understood to be a whole number. This set is related to both the RSA and Rabin cryptosystems because of their reliance on multiplication. Indeed as with most sets underlying a public-key cryptosystem, $FACTOR \in \mathsf{NP}$.

Theorem 4.2. $FACTOR \in \mathsf{NP}$

Proof. On input $\langle x, b\rangle$, nondeterministically choose an integer $1 < f \leq b$. If f divides x, accept; otherwise, reject. □

The search space of integers less than b is exponential and this fact intuitively precludes fast algorithms for factoring. In fact, factoring has a heuristic algorithm running in subexponential time [LL93]. Further computational methods for factoring are discussed in [Rie85]. Next, we formally define the RSA cryptosystem as a decidable set from the perspective of an adversary.

Definition 4.3. $RSA = \{\langle n, b, p', q', a'\rangle | n = pq, ab = 1 \mod \phi(n) \text{ for primes } p, q \leq p', q' \text{ and } a \leq a'\}$

As the next theorem states, determining the private key to RSA is no harder than computing the set $FACTOR$ defined previously. This establishes the connection between an adversary's power to factor integers to the strength of RSA. The type of reduction is a polynomial-time Turing reduction, so if $FACTOR$ were decided as a subroutine, RSA could be decided in polynomial time. Again, this is of utmost importance to analyzing the security of a cryptosystem if an adversary has many precomputed factorizations that can be accessed as a subroutine to decrypting a ciphertext.

Theorem 4.4. $RSA \leq_T^{\mathsf{P}} FACTOR$

Proof. Let $r = \langle n, b, p', q', a'\rangle$ be an input instance for RSA. Using a binary search with queries to $FACTOR$, a prime p can be found less than p'. Then another binary search querying $FACTOR$ will determine a second prime q that divides $\lfloor \frac{n}{p} \rfloor$. A quick multiplication verifies that $pq = n$ and a comparison ensures $p \neq q$. If p, q do not satisfy these conditions or if the binary search was unable to find a prime $p \leq p'$ or $q \leq q'$, then the instance $r \notin RSA$.

With p and q in hand, the last step is to determine whether there is an $a \leq a'$ satisfying $ab = 1 \mod \phi(n)$. This is polynomially verifiable. $\phi(n)$ is computable in polynomial time by calculating the factorization of n, and using the Euclidean algorithm, $a = b^{-1} \mod \phi(n)$ is computable also in polynomial time. At this point of the algorithm, $r \in RSA$ if and only if $a \leq a'$. □

What is not known is whether $FACTOR \leq_T^{\mathsf{P}} RSA$. Such a result would place a lower bound on the security of RSA in terms of factoring, and therefore a proof of the hardness of factoring would imply that RSA is a secure public-key cryptosystem. This underscores the lack of a compelling result in the above theorem: it does not rule out the undesirable result that factoring could be difficult but decrypting ciphertexts of RSA without the private key could be easy. The Rabin cryptosystem, although having the downside of an ambiguous decryption multivalued function, shares what is thought to be a closer relationship to factoring in terms of equivalence of their hardness.

Definition 4.5. $RABIN = \{\langle n, p', q', B\rangle | n = pq; p, q$ distinct primes and $p \leq p'$ and $q \leq q'\}$

Theorem 4.6. $RABIN \leq_T^{\mathsf{P}} FACTOR$

Proof. Let $r = \langle n, p', q', B\rangle$ be a input instance for $RABIN$. Naturally, if $B \geq n$, then $r \notin RABIN$. Using binary search and queries to $FACTOR$, a prime p can be found less than or equal to p'. Another binary search yields a $q \leq q'$ where q divides $\lfloor \frac{n}{p} \rfloor$. A simple check ensures $p \neq q$, $n = pq$, and that $p, q = 3 \mod 4$. □

The relationship of a lower bound of $FACTOR$ to $RABIN$ is currently unknown. Rabin provided a probabilistic algorithm to factor n given a black box to break his cryptosystem [Rab79]. However, this equivalence is made only with integers of the form $n = pq$, those exact integers used in the formulation of the Rabin cryptosystem. In general, this can be extended to factoring arbitrary integers. The following theorem establishes this relationship.

Theorem 4.7 ([Rab79])**.** *If M is an output of $e_k(x)$ for $x \in \mathbb{Z}_n$ and $k \in K$ for the Rabin cryptosystem, and if x can be determined from M and the public key (n, B), then n can be factored into its constituent primes.*

Proof. If an adversary can compute all four possible plaintexts corresponding to M, then the adversary has knowledge of four square roots of $(x + \frac{B}{2})^2$. These square roots come in pairs $\pm\alpha_1$ and $\pm\alpha_2$. Taking $\alpha_1^2 - \alpha_0^2$, this equals $M - M = 0$ in $\mathbb{Z}_n$, but it also equals $(\alpha_1 + \alpha_0)(\alpha_1 - \alpha_0)$. Since we know *a priori* that $n = pq$ for primes p and q, then without loss of generality, p divides $\alpha_1 + \alpha_0$ or $\alpha_1 - \alpha_0$. This follows from the fact that neither term in the product is 0 or 1. Therefore, one factor of n is $p = \gcd(\alpha_1 - \alpha_0, n)$. The other factor q can be computed through a division. □

4.2 Discrete Logarithm-Based Cryptographic Sets

Since discrete log can be defined over a prime field $\mathbb{Z}_p$ or over a general group G, there are two definitions for a computational set representing this hard problem. First, the specific case of a discrete logarithm computed over a cyclic prime group.

Definition 4.8. $DLOG_{prime} = \{\langle p, \alpha, \beta, b\rangle | p$ prime; $\alpha, \beta \in \mathbb{Z}_p^*;$ and $\beta = \alpha^a$ for some $1 \leq a \leq b\}$

Theorem 4.9. $DLOG_{prime} \in \mathsf{NP}$

Proof. On input $\langle p, \alpha, \beta, b\rangle$, nondeterministically choose an integer $1 \leq a \leq b$. Compute α^a, and if it equals β, accept. Otherwise, reject. □

To define the ElGamal cryptosystem over $\mathbb{Z}_p^*$ as a NP-search problem requires a similar formalism as defining the discrete logarithm as a set with the addition of the random number r as a part of the key. This foretells an equivalence between the two sets.

Definition 4.10. $ELGAMAL_{prime} = \{\langle p, \alpha, \beta, a', r'\rangle | p$ prime ; $\beta = \alpha^a$ for some $a \leq a'; r \in \mathbb{Z}_{|\langle\alpha\rangle|}$ for some $r \leq r'\}$

Theorem 4.11. $ELGAMAL_{prime} \equiv_m^{\mathsf{P}} DLOG_{prime}$.

Proof. Let $\langle p, \alpha, \beta, a', r' \rangle$ be an input instance for $ELGAMAL_{prime}$. Then, it is clear that $\langle p, \alpha, \beta, a', r' \rangle \in ELGAMAL_{prime}$ if and only if $\langle p, \alpha, \beta, a' \rangle \in DLOG_{prime}$.

Now let $\langle p, \alpha, \beta, a' \rangle$ be an input instance to $DLOG_{prime}$. After calculating $r = |\langle \alpha \rangle|$ using a binary search, $\langle p, \alpha, \beta, a' \rangle \in DLOG_{prime}$ if and only if $\langle p, \alpha, \beta, a', r \rangle \in ELGAMAL_{prime}$ □

Here, the ElGamal cryptosystem is Turing equivalent to the discrete logarithm, an attribute missing from RSA and Rabin's relationship to factoring. For a general group used in the discrete logarithm, there must be an encoding that allows the group to be given as input to the Turing machine. Yet, groups whose smallest encodings are still too large to give as input to the machine could unfairly give the machine enough time to compute the discrete logarithm. For this purpose, a *functional oracle* provides the answer to group multiplication problems when $\langle a, b \rangle$ is written to a query tape and the machine enters a special state q_{mult} . In the next turn, the result $a * b$ remains on the query tape, the result of the multiplication. For any integer k and group element a, another functional oracle provides the answer to a^k on a query tape the turn after $\langle a, k \rangle$ is written to the query tape and the machine enters a state q_{exp}. For a set of strings S and a group G, denote S^G as the set of strings decidable by a machine equipped with the multiplication and exponentiation functional oracles for G. Note that the group operation $*$ is implied in this definition through the behavior of the functional multiplication oracle, and that the representations of the elements of the group in terms of the tape alphabet Γ is duly implied. This allows the definition of the discrete logarithm over an arbitrary group to be formulated in terms of a set of strings.

Definition 4.12. $DLOG_{group}^G = \{\langle \alpha, \beta, b \rangle | (G, *) \text{ is a group; } \alpha, \beta \in G; \text{ and } \beta = \alpha^a \text{ for some } 1 \leq a \leq b\}$

Allowing E to the the group of solutions of an elliptical curve over a field $\mathbb{Z}_p$, then the Menezes-Vanstone cryptosystem is defined with functional oracles for E.

Definition 4.13. $MVEC^E = \{\langle \alpha, \beta, r, a' \rangle | \alpha, \beta \in E; \beta = a\alpha \text{ for some } 1 < a \leq a'\}$

Theorem 4.14. $MVEC^E \equiv_m^{\mathsf{P}} DLOG_{group}^E$

Proof. Noting that multiplication in E is written additively, there is a direct identification to problem instances of $MVEC^E$ and problem instances of $DLOG_{group}^E$. Using the technique of theorem 4.11, this completes a polynomial-time Turing equivalence between the two sets. □

5 Cryptographic Formalisms

The last section established the connection between the security between various public-key cryptosystems and their hard underlying problems. However, this sense of security was mostly upper bounds, which does not prove strength as much as weakness; RSA, for example, would be broken if a fast factoring algorithm is achieved. To prove the absolute strength of a cryptosystem, the abilities of an adversary need to be delineated and the guarantees of the cryptosystem given in terms of these capabilities. The absolute security of a public-key cryptosystem relies upon strong lower bounds on the difficulty of its underlying problem.

The next step in the analysis is the observation that each public-key cryptosystem, by definition, contains two functions e_k and d_k for a given key k that are efficiently computed, allowing for near on-the-fly encryption and decryption. Moreover, d_k inverts the action of e_k on a given plaintext, or at least produces a possible set of preimages. Since d_k used information of the private key to compute efficiently and because any adversary should not have access to the private key, the desire is for e_k^{-1} to be a hard function with a strong lower bound on its hardness. Formally, e_k^{-1} is distinguished from d_k as the inverse function of e_k, possibly multivalued, and with extra information about the public key but not the private key.

What this definition suggests is that e_k is an easy function, but that e_k^{-1} is hard. This is the idea behind a *one-way function*. With the added condition that some information, such as a private key in public-key encryption, gives the ability to invert easily, then e_k is called a *trapdoor one-way function*. It is clear that any prudent analysis of the security of a public-key cryptosystem must include analysis of e_k as a (trapdoor) one-way function.

5.1 One-Way Functions

Critical to the analysis of the security of any given function is the formalization of the capabilities of the adversary. This principle cannot be emphasized enough. If Alice wishes to send Bob a message over an insecure channel using a public-key cryptosystem, then she must predict the capabilities of any eavesdropper—both in terms of current computing power and future power and knowledge. For instance, if Eve intercepts Alice's credit card information during a secure transaction, Eve might have about four years before that particular credit card information expires. This means that the intended recipient, Bob, should do his best to ensure his public-key cryptography today will remain secure for up to four years. Of course, this becomes a statistical exercise in predicting the abilities of tomorrow's computers. As long as the definition of security of a one-way function is not misconstrued as to imply the abilities of an adversary today but rather of the strength of an adversary over a future period of time in which the message should remain secure, then the results of this section provide a recipe for the level of strength of a public-key cryptosystem to use today.

The extra abilities prescribed to an adversary are various: random coin flipping, advice, and oracles that give instantaneous answers to difficult problems, to name a few. Typically, adversaries are characterized as having random coin flipping (a probabilistic mechanism) and a polynomial amount of advice (a precomputed table of values). In practice, rainbow tables are used in crypt-analysis in attempting to break various cryptosystems using lookups to these tables, which have the answers to various difficult queries an adversary might need in decrypting all sorts of ciphertexts [AJO08]. This is exactly a balance of the time-memory tradeoff between computing all ciphertexts for all possible plaintexts, which takes exponential time, and storing the precomputed ciphertexts, which takes exponential space. However, supplying an adversary with both random coin flipping and advice is redundant because any coin flips giving the information that an adversary seeks can be hard-coded as advice. Therefore, the analysis lends itself to applications once a reasonable expectation of the adversary has been assessed.

An adversary typically receives a polynomial amount of advice for each input length. This is represented by a set of circuits that the adversary employs to attempt to invert a function on different lengths of output. For any given function, a bound on the resources—or a bound on the size of circuits—of an adversary becomes one security parameter of the function, and with this resource bound, the upper bound of the fraction of successes of the adversary is given as the second parameter. This fraction of success can be considered as a *success probability* when taken uniformly over all outputs. However, it is understood that this bound is not necessary tight although the words "upper bound" may be omitted preceding "success probability." Since the security of any one-way function is analyzed in terms of its relation to inputs of the same length, much like in the case of a one-time pad, these functions are defined in terms of an *ensemble* of functions of a fixed input.

Definition 5.1. Let $f : \Sigma^* \to \Sigma^*$. Then $f_l : \Sigma^l \to \Sigma^*$ is the *restriction of f to Σ^l* where $f_l(x) = f(x)$. The set $\{f_l : \Sigma^l \to \Sigma^* | l \in \mathbb{N}\}$ is the *ensemble of functions induced by f*.

The formal definition of a one-way function is thusly.

Definition 5.2. Let $\epsilon : \mathbb{N} \to [0, 1]$ and $S : \mathbb{N} \to \mathbb{N}$. $f : \Sigma^* \to \Sigma^*$ is a *one-way function* with security (ϵ, S) if f is polynomial-time computable and for all circuits C with $\mathsf{SIZE}(C) \leq S(l)$ and for almost all $l \geq 0$,

$$\Pr\{f_l(C(f_l(U_l))) = f_l(U_l)\} < \epsilon(l)$$

A one-way function is a *one-way permutation* if it is a bijection.

There are a few observations concerning the security of one-way functions. First, any one-way function secure against a bound of resources is secure against any smaller bound of resources. Too, any one-way function with a given success probability is a one-way function when given any other larger success probability.

Remark 5.3. Suppose f is a one-way function with security (ϵ, S). If $\epsilon(l) \leq \epsilon'(l)$ and $S'(l) \leq S(l)$ for almost all l, then f is a one-way function with security (ϵ', S').

By the contrapositive of above, if f is one-way with security (ϵ, S) but not one-way with security (ϵ', S'), then it must be true for an infinite set of l that either $\epsilon(l) > \epsilon'(l)$ or $S'(l) > S(l)$.

Next, for any one-way function must have a success probability of at least 2^{-n}. This is because any circuit that outputs a single string of length n can invert that one particular string out of all inputs of length n. This is a minimum security bound on the success probability. Namely, not only is this a statement that perfect secrecy is impossible, but also it is infeasible for the success probability to be a function that decays to 0 faster than 2^{-n}.

Remark 5.4. Let $f : \Sigma^* \to \Sigma^*$ be a polynomial-time computable function. Then f is one-way with security (ϵ, S) where $S = 1$ for all n and $\epsilon = 2^{-n}$. We assume that a circuit that outputs a single string has size 1.

Third, if two functions can be compromised by the same circuit, without changing the size of this circuit, then the success probability of inversion of these two functions are identical under the same resource bound.

Remark 5.5. Suppose f is a one-way function with security (ϵ, S). Assume g is a function where any circuit producing a pre-image of g can be converted into a circuit producing a pre-image of f with no size change to the circuit. Then g is a one-way function with security (ϵ, S).

5.1.1 Length-Regular One-Way Functions

In studying one-way functions, it is helpful for the discussion and the proofs for one-way functions that obey nice properties. For one, it is helpful to consider functions whose output is the same length for inputs of the same length. These functions are called *length regular.*

Definition 5.6. A function f is *length regular* if $|f(x)| = |f(y)|$ whenever $|x| = |y|$.

A length-regular one-way function is called *length preserving* if, conversely, all outputs of the same length are the image of inputs of the same length. Length-preserving functions are ultimately the simplest to analyze in terms of security. However, demanding these extra properties from a one-way function does not diminish its security; one-way functions of sufficient security exist if and only if similar length-preserving one-way functions do.

Definition 5.7. A function f is *length preserving* if f is length regular and $|x| = |y|$ whenever $|f(x)| = |f(y)|$.

The main result relies upon the fact that if a function is secure on only a few input lengths, it can be strengthened by defining the weaker lengths in terms of these stronger lengths.

Lemma 5.8 ([Gol00]). *Let f be a one-way function defined on an infinite support $l(n)$ for $n \in \mathbb{N}$. Suppose f has security (ϵ, S) on these lengths, and that $l(n)$ is a polynomially computable function. Then, there exists a one-way function defined on almost all lengths with security (ϵ', S'). Here, $\epsilon'(n)$ is equal to $\epsilon(l(k))$ for the largest $l(k) \leq n$ and likewise for $S'(n)$.*

Proof. Let f and $l(n)$ satisfy the hypotheses above. For any given length, define $\hat{f}(x)$ as $f(x_j)$, where x_j is the first j characters of x, and $j = l(n)$ is the largest integer less than or equal to $|x|$ for some $n \in \mathbb{N}$. j can be found in polynomial time using a binary search and the polynomial-computability of l.

Fix $S'(|x|) = S(j)$. Consider a circuit that produces pre-images of $\hat{f}$ for length $|x|$. This circuit can be used to produce pre-images of f for length j by ignoring the last $|x| - j$ characters of the input. Therefore, by remark 5.5, $\hat{f}$ has security (ϵ', S') where ϵ' and S' have been extended to be defined over all inputs $n \geq l(0)$. ☐

Next, the equivalence of one-way functions and length-regular one-way functions is established.

Lemma 5.9 ([Gol00]). *A one-way function with security (ϵ, S) exists if and only if a length-regular one-way function with security (ϵ, S) exists.*

Proof. Any length-regular one-way function is a one-way function. Consider $f : \Sigma^* \to \Sigma^*$ a one-way function with security (ϵ, S).

Since f is polynomially-time computable, there is some $k \in \mathbb{N}$ such that $|f(x)| \leq |x|^k$ as f only has a polynomially-bounded amount of time to write its output. Using this fact, construct $\tilde{f} : \Sigma^* \to \Sigma^*$ defined by $\tilde{f}(x) = f(x)10^{|x|^k - |f(x)|}$. $\tilde{f}$ is therefore length-regular where $|\tilde{f}(x)| = |x|^k + 1$ for all $x \in \Sigma^*$.

Next, a circuit breaking $\tilde{f}$ can be converted to a circuit breaking f with the same size by hardcoding the inputs corresponding to the padding. The size of the circuit is no larger or smaller. Therefore, $\tilde{f}$ has security (ϵ, S) by remark 5.5. ☐

Combining these lemmas, it can be shown that one-way functions exist if and only if length-preserving one-way functions exist. It is understood that the security of a one-way function versus a length-preserving one-way function is comparable.

Theorem 5.10 ([Gol00]). *Assume ϵ is a non-increasing function and S is a non-decreasing function. A one-way function with security (ϵ, S) exists if and only if a length-preserving one-way function with security (ξ, T) exists with $\xi(n) = \epsilon(\sqrt[k]{n-1})$ and $T(n) = S(\sqrt[k]{n-1})$ for some $k \in \mathbb{N}$.*

Proof. Let f be a length-preserving one-way function with security (ξ, T). Define a $\hat{f}$ on inputs of length $n^k + 1$ as $\hat{f}(x) = f(x_n)$, where x_n is the first n characters of x and $n^k + 1 = |x|$. $\hat{f}(x)$ has security $(\xi(n^k + 1), T(n^k + 1))$, which is precisely (ϵ, S). Extending $\hat{f}$ to a one-way function on all inputs gives a function with security (ϵ', S') where $\epsilon' < \epsilon$ and $S' > S$ since ϵ and S are non-increasing and non-decreasing, respectively. For reasonable security parameters, ϵ does not decrease faster than exponentially, and likewise S does not grow faster than exponentially. Thus, this extension is a one-way function with security (ϵ, S) by remark 5.3.

For the converse, assume that a one-way function exists with security (ϵ, S). By the previous lemma, a length-regular one-way function exists. Call this length-regular one-way function f, and set $k \in \mathbb{N}$ such that $|f(x)| = |x|^k + 1$.

Transforming the length-regular f into a length-preserving one-way function requires one more step. Define $g(x) = f(x_n)$ where x_n is the first n characters of the input. n is the value satisfying $n^k + 1 = |x|$. This gives a length-preserving g.

The support of g is $\{n^k + 1 | n \in \mathbb{N}\}$. Label the security of g as (ϵ', S'). By the definition of g, $\epsilon'(n^k + 1) = \epsilon(n)$ and $S'(n^k + 1) = S(n)$. Clearly, ϵ' and S' are non-increasing and non-decreasing, respectively. Extending this function to all input lengths $\hat{g}$ gives security parameters (ξ, T). By the lemma 5.8, $\xi(n) = \epsilon'(j)$ and $T(n) = S'(j)$ for largest j with $j^k + 1 \leq n$. Since ϵ' is non-increasing, $\xi(n) \leq \epsilon'(n)$; likewise, S' is non-decreasing, so $T(n) \geq S'(n)$.

Finally, $\epsilon'(n) = \epsilon(\sqrt[k]{n-1})$ by the definition of ϵ' and additionally, $S'(n) = S(\sqrt[k]{n-1})$ by the definition of S'. Thus, $\xi(n) \leq \epsilon(\sqrt[k]{n-1})$ and $T(n) \geq S(\sqrt[k]{n-1})$ for almost all n. So $\hat{g}$ has security $(\epsilon(\sqrt[k]{n-1}), S(\sqrt[k]{n-1}))$. $\square$

5.1.2 Strong and Weak One-Way Functions

In a previous lemma, it was shown that even if a function is not secure on all input lengths, the function can be redefined on weaker lengths in terms of these stronger lengths. Since the existence of reasonably secure one-way functions has not been shown, it is popular to relax various conditions on the functions and show that even if these weaker conditions are satisfied, then a stronger function can be constructed. Although this has not demonstrated a simpler case for the existence of one-way functions, the main conclusion is that the security of the success probability can be amplified. One-way functions that are secure up to a polynomial fraction of successful inversions are *weak*; those secure against an exponential fraction are *strong*.

Definition 5.11 (Weak one-way function)**.** A one-way function is a *weak one-way function* if the security parameter $\epsilon(n)$ is less than n^{-k} for some $k \in \mathbb{N}$ and for almost all $n \in \mathbb{N}$.

Definition 5.12 (Strong one-way function)**.** A one-way function is a *strong one-way function* if the security parameter $\epsilon(n)$ is less than 2^{-n^δ} for some $\delta > 0$ and for almost all $n \in \mathbb{N}$.

A strong one-way function is equivalently one whose success probability is asymptotically less than all polynomial fractions.

Remark 5.13. Let $f(n)$ be a function defined on $\mathbb{N}$. $f(n) \leq 2^{-n^\delta}$ for some $\delta > 0$ and almost all $n \in \mathbb{N}$ if and only if $f(n) \leq n^{-k}$ for all $k \in \mathbb{N}$ and for almost all $n \in \mathbb{N}$.

The next two lemmas prove that weakness and strongness of a one-way function remains unchanged when the function is transformed into a length-preserving type.

Lemma 5.14. *There exists a weak one-way function with security (ϵ, S) if and only if there exists a length-preserving weak one-way function with security (ξ, T).*

Proof. Any length-preserving weak one-way function with security (ϵ, S) is a weak one-way function with security (ϵ, S).

Assume f is a weak one-way function with security (ϵ, S). Then there exists a one-way function $\hat{f}$ that is length-preserving with security (ξ, T) and $\xi(n) = \epsilon(\sqrt[k]{n-1})$ and $T(n) = S(\sqrt[k]{n-1})$. Consider $\epsilon \leq n^{-p}$ for some $p \in \mathbb{N}$. Then $\xi(n) \leq n^{-p/k}$, and so $\hat{f}$ is a length-preserving weak one-way function. $\square$

Lemma 5.15. *There exists a strong one-way function with security (ϵ, S) if and only if there exists a length-preserving strong one-way function with security (ξ, T).*

Proof. Any length-preserving strong one-way function with security (ϵ, S) is a strong one-way function with security (ϵ, S).

Assume f is a strong one-way function with security (ϵ, S). Then there exists a one-way function $\hat{f}$ that is length-preserving with security (ξ, T) and $\xi(n) = \epsilon(\sqrt[k]{n-1})$ and $T(n) = S(\sqrt[k]{n-1})$. Consider $\epsilon \leq n^{-p}$ for all $p \in \mathbb{N}$. Then $\xi(n) \leq n^{-p/k}$ for all $p \in \mathbb{N}$, and so $\hat{f}$ is a length-preserving strong one-way function. $\square$

The amplification theorem of the security a one-way function remains a central result in cryptography—that even weak one-way functions contain enough hard information to be transformed into strong one-way functions. Since any strong one-way function is weak, the converse of this is shown to complete the proof that strong one-way functions exists if and only if weak one-way functions exist.

Theorem 5.16 ([Yao82, Gol00, Gol08])**.** *If there exists a weak one-way function, then there exists a strong one-way function.*

Proof. Let f be a weak one-way function with $\epsilon(n) < n^{-k}$ for almost all n. Without loss of generality, assume f is length preserving. Consider the following function g:

$$g(x_1, x_2, ..., x_{t(n)}) = f(x_1)f(x_2)\cdots f(x_{t(n)})$$

where $|x_i| = n$ and $t(n)$ is the run time of f times n. g is defined on support $nt(n)$ for $n \in \mathbb{N}$. Assume g is not a strong one-way function. Then there exists an $l \in \mathbb{N}$ with $\epsilon_g(n) > n^{-l} \equiv q(n)^{-1}$ for almost all n, the success probability of a family of circuits $\{C_n\}_{n=1}^{\infty}$ on inverting outputs of g.

Let $n \in \mathbb{N}$. Consider a circuit I with advice $\langle x_1, ..., x_{t(n)}\rangle$ attempting to invert f for outputs of length n given by:

There are a finite choices of advice strings $\langle x_1, ..., x_{t(n)}\rangle$, and so there are a finite number of circuits $\{I_j\}_{j=1}^{N}$ computing the above algorithm. If these circuits are distributed uniformly in a probability distribution, and the probability of any such circuit producing a successful preimage is at most $\epsilon(n)$, then there must be one such circuit that deterministically produces a successful preimage on at most $\epsilon(n)$ fraction of inputs. Therefore, the argument proceeds probabilistically, although this shift is nothing more than to simplify a counting argument.

Furthermore, each possible circuit I_j has polynomial size, and thus it is advice $\langle I_1, ..., I_{t'(n)}\rangle$ to the following circuit A. Here, $t'(n) = 2n \cdot t(n) \cdot q(nt(n))$.

Algorithm 1 Inversion of f

```
1: $y \leftarrow$ input to circuit, $n \leftarrow |y|$
2: for $i = 1$ to $t(n)$ do
3:     Simulate $(z_1, ..., z_{t(n)}) \leftarrow C_n(f(x_1), ..., f(x_{i-1}), y, f(x_{i+1}), ..., f(x_{t(n)}))$
4:     if $y == f(z_i)$ then
5:         Output $z_i$ and return
6:     end if
7: end for
```

Algorithm 2 Amplified Inversion of f

```
1: $y \leftarrow$ input to circuit, $n \leftarrow |y|$
2: for $i = 1$ to $t'(n)$ do
3:     Simulate $z \leftarrow I_i(y)$
4:     if $y == f(z)$ then
5:         Output $z$ and return
6:     end if
7: end for
```

Again, consider the expected success of circuit A uniformly over all possible sets of advice circuits $\langle I_1, ..., I_{t'(n)} \rangle$. If this expected success is at most $\epsilon(n)$, then for some choice of advice, the success probability is deterministically at most $\epsilon(n)$. Call $\boldsymbol{A}$ the random variable assigning to each circuit A equal probability for each possible set of advice circuits. Similarly, call $\boldsymbol{I}$ the random variable assigning to each circuit I an equal probability for each possible set of advice strings.

Define S as the set of strings whose images under f are inverted with probability greater than $\frac{n}{t'(n)}$ over $\boldsymbol{I}$. What will be shown is that the expected success of A sampled from $\boldsymbol{A}$ is eventually larger than any polynomial and that the number of strings in S is, too, larger than any n^{-a} for any $a \in \mathbb{N}$. These will be proven as lemmas.

Altogether, this will contradict that f is a weak one-way function with security $\epsilon(n) = n^{-k}$.

Lemma 5.17. *The probability that A successfully inverts $f(x)$ is greater than $1 - 2^{-n}$, where A is sampled from $\boldsymbol{A}$ and $x \in S$.*

Consider the probability that A does not invert $f(x)$. This probability is equal to the probability that I does not successfully invert $f(x)$ on each of the $t'(n)$ runs of A:

$$\left(1 - \frac{n}{t'(n)}\right)^{t'(n)} = \left[\left(1 - \frac{n}{t'(n)}\right)^{t'(n)/n}\right]^n < e^{-n} < 2^{-n}$$

And so the lemma is established by the complementary probability.

Lemma 5.18.

$$|S| > \left(1 - \frac{1}{2n^k}\right) 2^n$$

To begin the proof of this bound, denote the complement of S as $\overline{S} = \{0,1\}^n \setminus S$. Next consider the quantity s defined by

$$s = \Pr\{g(C(g(x_1, ..., x_{t(n)}))) = g(x_1, ..., x_{t(n)}) \text{ and } x_i \in \overline{S} \text{ for some } 1 \leq i \leq t(n)\}$$

where x_i are distributed uniformly over all strings of length n. While this is a probability over a known deterministic circuit, it is a simplified way of writing the expected value of successful inversion. Notice that this probability can be written over a (possibly overlapping) union, and so we get an upper bound given by a sum of these overlapping sets.

$$s \leq \sum_{i=1}^{t(n)} \Pr\{g(C(g(x_1, ..., x_{t(n)}))) = g(x_1, ..., x_{t(n)}) \text{ and } x_i \in \overline{S}\}$$

Furthermore, each term of this sum is the probability of an intersection of sets, and so it follows from $\Pr\{A \cap B\} \leq \Pr\{A|B\}$ that

$$s \leq \sum_{i=1}^{t(n)} \Pr\{g(C(g(x_1, ..., x_{t(n)}))) = g(x_1, ..., x_{t(n)})|x_i \in \overline{S}\}$$

Certainly if C can invert $x_1, ..., x_{t(n)}$ given x_i, then I at least can invert $f(x_i)$ with advice $x_1, ..., x_{t(n)}$ with x_i omitted.

$$s \leq \sum_{i=1}^{t(n)} \Pr\{f(I(f(x_i))) = f(x_i)|x_i \in \overline{S}\}$$

Since each of the x_i are not in $\overline{S}$:

$$s \leq \sum_{i=1}^{t(n)} \frac{n}{t'(n)} = n\frac{t(n)}{t'(n)} = \frac{1}{2q(nt(n))}$$

For an upper bound on s, first recall the definition of s:

$$s = \Pr\{g(C(g(x_1, ..., x_{t(n)}))) = g(x_1, ..., x_{t(n)}) \text{ and } x_i \in \overline{S} \text{ for some } 1 \leq i \leq t(n)\}$$

Tightening the restriction of possible x_i in the probability provides a subset of the previous set over which the probability is calculated. Here, the phrase "for some" changes to "for all."

$$s \geq \Pr\{g(C(g(x_1, ..., x_{t(n)}))) = g(x_1, ..., x_{t(n)}) \text{ and } x_i \in \overline{S} \text{ for all } 1 \leq i \leq t(n)\}$$

However, this is precisely a set difference

$$s \geq \Pr\{g(C(g(x_1, ..., x_{t(n)}))) = g(x_1, ..., x_{t(n)})\} - \Pr\{x_i \in S \text{ for all } 1 \leq i \leq t(n)\}$$

Here, the x_i in the second probability are independently and uniformly distributed over $\{0, 1\}^n$. $\Pr\{g(C(g(x_1, ..., x_{t(n)}))) = g(x_1, ..., x_{t(n)})\}$ is exactly identified with $\epsilon_g(nt(n))$, and $\Pr\{x_i \in S$ for all $1 \leq i \leq t(n)\}$ is identified with $\Pr\{x_i \in S\}^{t(n)}$ because each of the x_i are identically and independently chosen. Therefore,

$$\frac{1}{2q(nt(n))} \geq s \geq \epsilon_g(nt(n)) - \Pr\{x_i \in S\}^{1/t(n)}$$

Substituting that $\epsilon_g(n) \geq q(n)^{-1} = n^{-l}$ and rearranging,

$$\Pr\{x_i \in S\} \geq \left(n^{-l} - \frac{1}{2q(nt(n))}\right)^{1/t(n)} = \left(\frac{1}{2q(nt(n))}\right)^{1/t(n)}$$

For large enough n, $\Pr\{x_i \in S\} \geq 1 - \frac{1}{2n^k}$. Since the x_i are uniformly distributed over all 2^n strings of length n, $|S| \geq \left(1 - \frac{1}{2n^k}\right) 2^n$. □

A note on the circuit size bound: The two algorithms presented in the previous proof use the circuits that break g to create a circuit that breaks f. In these algorithms, the circuit C_n is used $t(n) \cdot t'(n)$ times, which when amalgamated into a single circuit, gives a circuit of size $\mathsf{SIZE}(C_n)^{t(n) \cdot t'(n)}$. Therefore, if f has circuit size security of S, then g must have circuit size security of $S^{\frac{1}{t(n) \cdot t'(n)}}$, which is exponentially smaller than S.

5.2 Pseudorandom Generators

A class of one-way functions, called *pseudorandom generators*, have the ability to stretch short and uniformly random strings into longer strings that look uniformly random to an adversary. This introduces the second type of randomness: a pseudorandomness, that for all intents and purposes, behaves like uniform randomness in the realm of efficient computation. Here, the adversary is given the same capabilities as before with polynomial advice given for each input length.

Using pseudorandom strings, Alice and Bob no longer have to negotiate a one-time pad the same length as the plaintext. This would double the amount of bandwidth required because Bob must first send the one-time pad to Alice before Alice sends the encrypted ciphertext of the same length back to Bob. This protocol also requires a large source of randomness available to Bob, which may be rather unlikely in general circumstances. Rather, Alice and Bob can negotiate a shorter and uniformly random string, called a *seed*, using their public-key cryptosystem. Upon receiving the seed, Alice then expands it to the length of her message using a pseudorandom generator that is known to her, Bob, and any eavesdropper. Because the safety of Alice's message is predicated upon the uniform randomness of the expanded string, the security of any pseudorandom generator is defined in terms of the ability of an adversary to distinguish uniformly random strings from these pseudorandom strings. Since a one-time pad is compromised if it was not uniformly generated, so is a pseudorandom string compromised if an adversary, with a defined amount of resources, can distinguish it from a uniformly random string.

Because a pseudorandom generator induces a random variable when defined uniformly over all seeds, the security of the pseudorandom generator will be defined in terms of a *computational distance* between the uniform distribution and the pseudorandom distribution.

Definition 5.19 (Computational Distance)**.** Let $n, S \in \mathbb{N}$. Let X_n, Y_n be probability distributions over Σ^n. The *computational distance* between X_n and Y_n relative to size S is

$$\Delta_{\mathrm{comp},S}(X_n, Y_n) = \max_{\mathsf{SIZE}(C) \leq S} |\Pr\{C(X_n) = 1\} - \Pr\{C(Y_n) = 1\}|$$

and C has n inputs.

A pseudorandom generator must generate a distribution of binary numbers that are computationally indistinguishable from the uniform distribution to adversaries. Adversaries reserve the same capabilities as in the analysis of one-way functions.

Definition 5.20 (Pseudorandom generator)**.** Let $l, L, S \in \mathbb{N}$ where $L > l$ and let $\epsilon > 0$. A length-regular function $g : \Sigma^l \to \Sigma^L$ is a *pseudorandom generator* with security (ϵ, S) if g is polynomial-time computable and if

$$\Delta_{\text{comp},S}(g(U_l), U_L) \leq \epsilon$$

where $L - l$ is the *extension* of g and U_l is the uniform distribution over Σ^l.

A pseudorandom generator only provides a stretch for seeds of a particular length, but an *ensemble* can produce pseudorandom output for seeds of any length.

Definition 5.21 (Ensemble of pseudorandom generators)**.** Let $\epsilon : \mathbb{N} \to [0,1]$, $S : \mathbb{N} \to \mathbb{N}$ and $L : \mathbb{N} \to \mathbb{N}$. Let $\{g_l\}_{l\in\mathbb{N}}$ be a family of functions $g_l : \Sigma^l \to \Sigma^{L(l)}$. $\{g_l\}_{l\in\mathbb{N}}$ is a *ensemble of pseudorandom generators* with security (ϵ, S) if for every $l \in \mathbb{N}$, g_l is a pseudorandom generator with security $(\epsilon(l), S(l))$.

A pseudorandom generator may either refer to a single generator defined over seeds of a particular length or as a complete ensemble. This distinction will be clear from the context.

5.2.1 Stretching the Pseudorandom Information

Just as with one-way functions, no demonstrably secure pseudorandom generator has been shown to exist. However, the result of the following theorem shows that all that is needed is a pseudorandom generator that can stretch its input seed by a single bit. This extension of one is enough to construct a pseudorandom generator that stretches to any arbitrary polynomial output.

Theorem 5.22. *Suppose $g : \{0,1\}^n \to \{0,1\}^{n+1}$ is a pseudorandom generator with security (ϵ, S). Then $\tilde{g} : \{0,1\}^n \to \{0,1\}^{n^k}$ defined by*

$$\tilde{g}(x) = \alpha_1\alpha_2 \cdots \alpha_{n^k}$$

is a pseudorandom generator with security $(n^{-k}\epsilon, T)$, where α_i is the first bit of the output of $g^i(x)$. Here, $g^1(x)$ is the last n bits of $g(x)$ and, inductively, $g^i(x)$ is g^{i-1} applied to the last n bits of $g(x)$. Additionally, $T(n) + O(C_g) < S(n)$, where C_g is the size of a circuit computing g on inputs of length n.

Proof. $\tilde{g}$ is polynomial-time computable by computing $g(x)$ in polynomial time, then writing the first bit of its result to the output, and using $g(x)$ to further compute $g^2(x) = g(g(x))$. This process loops a polynomial number of times, so $\tilde{g}$ is computed in polynomial time.

Next, assume by way of contradiction that $\tilde{g}$ is not a pseudorandom generator with security $(n^{-k}\epsilon, T)$. Then there exists a circuit C where $\mathsf{SIZE}(C) \leq T$ and

$$|Pr\{C(\tilde{g}(U_n)) = 1\} - Pr\{C(U_{n^k}) = 1\}| > n^{-k}\epsilon$$

First, define a random variable H_n^i as the concatenation of the random variable U_i with the first $n^k - i$ bits of $\tilde{g}(U_n)$. This random variable is called a *hybrid*, and this method proof is called the *hybrid technique*. In this notation, the U_i and U_n are independent uniform random variables. For $i = 0$ to n^k, H_n^i is a finite sequence of random variables varying between $H_n^0 = \tilde{g}(U_n)$ to $H_n^{n^k} = U_{n^k}$. Using C to distinguish H_n^0 from $H_n^{n^k}$, C will be modified to distinguish H_n^i from H_n^{i+1}.

Consider the following family of functions $f_i : \{0,1\}^{n+1} \to \{0,1\}^{n^k - i}$ defined by $f_i(x)$ is the concatenation of the first bit of x with the first $n^k - i - 1$ bits of $\tilde{g}(x_1 \cdots x_{n+1})$.

More formally, in terms of the Prefix and Suffix functions,

$$\tilde{g}(x) = \textsc{Prefix}(g(x), 1) \cdot \tilde{g}(\textsc{Suffix}(g(x), n))$$

$$H_n^i = U_i \cdot \textsc{Prefix}(\tilde{g}(U_n), n^k - i)$$

$$f_i(x) = \textsc{Prefix}(x, 1) \cdot \textsc{Prefix}(\tilde{g}(\textsc{Suffix}(x, n)), n^k - i - 1)$$

Notice that based on the definition of $\tilde{g}$ stated previously,

$$\tilde{g}(U_n) = \textsc{Prefix}(g(U_n), 1) \cdot \tilde{g}(\textsc{Suffix}(g(U_n), n))$$

Here, the U_n appearing above are the same random variable and not independent uniform random variables. The following lemmas help simplify the hybrid family of random variables in terms of the function f.

Lemma 5.23. *H_n^i is identical to $U_i \cdot f_i(g(U_n))$ for $0 \le i \le n^k$.*

Proof. Choose i such that $0 \le i \le n^k$. Then, $H_n^i = U_i \cdot \textsc{Prefix}(\tilde{g}(U_n), n^k - i)$. Taking the prefix of the equation for $\tilde{g}(U_n)$,

$$\textsc{Prefix}(\tilde{g}(U_n)), n^k - i) = \textsc{Prefix}(g(U_n), 1) \cdot \textsc{Prefix}(\tilde{g}(\textsc{Suffix}(g(U_n), n)), n^k - i - 1)$$

The left hand side of this equation appears as the last portion of H_n^i, while the right hand side of this equation exactly corresponds with the definition of $f_i(g(U_n))$. Thus, $H_n^i = U_i \cdot f_i(g(U_n))$. □

Lemma 5.24. *H_n^{i+1} is identical to $U_i \cdot f_i(U_{n+1})$ for $0 \le i \le n^k - 1$.*

Proof. Choose i such that $0 \le i \le n^k - 1$. Then, $H_n^{i+1} = U_{i+1} \cdot \textsc{Prefix}(\tilde{g}(U_n), n^k - i - 1)$. It is clear that $U_{i+1} = U_i \cdot U_1$, where U_i and U_1 are independent and uniform random variables. Moreover, the random variables $U_n = \textsc{Suffix}(U_{n+1}, n)$ and $U_1 = \textsc{Prefix}(U_{n+1}, 1)$ are independent by splitting the variable U_{n+1} into disjoint substrings. Therefore, since U_n and U_i are independent, the following substitution maintains independence:

$$H_n^{i+1} = U_i \cdot \textsc{Prefix}(U_{n+1}, 1) \cdot \textsc{Prefix}(\tilde{g}(\textsc{Suffix}(U_{n+1}, n), n^k - i - 1)$$

This corresponds exactly to $H_n^{i+1} = U_i \cdot f_i(U_{n+1})$. □

Recall that by assumption, C distinguishes $\tilde{g}(U_n)$ from U_{n^k} on at least ϵ fraction of strings. By using the definition of the hybrids H_n^i in terms of f, a circuit can be produced that distinguishes U_{n+1} from $g(U_n)$, contradicting the original assumption. Let $\tilde{C}$ with advice i satisfying $0 \le i \le n^k - 1$ and $a \in \{0,1\}^i$ be defined as $\tilde{C}(x) = C(a \cdot f_i(x))$. We define the random variable $\boldsymbol{C}$ as the set of all $\tilde{C}$ where each advice length i is uniformly chosen, and $a \in \{0,1\}^i$ is uniformly chosen given i. Fixing $x \in \{0,1\}^{n+1}$, then we can express the following probability as an average probability over possible advice lengths:

$$\Pr\{\boldsymbol{C}(x) = 1\} = n^{-k} \sum_{i=0}^{n^k - 1} \Pr\{C(U_k \cdot f_i(x)) = 1\}$$

By the results of the previous lemmas,

$$\Pr\{\boldsymbol{C}(U_{n+1})=1\} = n^{-k}\sum_{i=0}^{n^k-1}\Pr\{C(U_k\cdot f_i(U_{n+1}))=1\} = n^{-k}\sum_{i=0}^{n^k-1}\Pr\{C(H_n^{i+1})=1\}$$

$$\Pr\{\boldsymbol{C}(g(U_n)=1\} = n^{-k}\sum_{i=0}^{n^k-1}\Pr\{C(U_k\cdot f_i(g(U_n))=1\} = n^{-k}\sum_{i=0}^{n^k-1}\Pr\{C(H_n^{i})=1\}$$

Combining these results,

$$|\Pr\{\boldsymbol{C}(g(U_n))=1\}-\Pr\{\boldsymbol{C}(U_{n+1})=1\}|$$

$$= n^{-k}\left|\sum_{i=0}^{n^k-1}\Pr\{C(H_n^{i})=1\}-\sum_{i=0}^{n^k-1}\Pr\{C(H_n^{i+1})=1\}\right|$$

This is a telescoping sum,

$$= n^{-k}\left|\Pr\{C(H_n^{0})=1\}-\Pr\{C(H_n^{n^k})=1\}\right|$$

$$= n^{-k}\left|\Pr\{C(\tilde{g}(U_n))=1\}-\Pr\{C(U_{n^k})=1\}\right| > \epsilon$$

However the previous probabilities can be interpreted as an expected value.

$$|\Pr\{\boldsymbol{C}(g(U_n))=1\}-\Pr\{\boldsymbol{C}(U_{n+1})=1\}| = |\mathsf{E}[\boldsymbol{C}(g(U_n))-\boldsymbol{C}(U_{n+1})]| > \epsilon$$

Applying the linearity of expectation gives the result on the right-hand side. Therefore, for some fixed circuit $\tilde{C}$, the following inequality holds:

$$\left|\Pr\{\tilde{C}(g(U_n))=1\}-\Pr\{\tilde{C}(U_{n+1})=1\}\right| > \epsilon$$

This contradicts that g has security (ϵ, S). □

Another note on circuit sizes: $\tilde{C}$ uses the circuit C plus a circuit used to compute g. Now, $\mathsf{SIZE}(C)\le T$, and $\mathsf{SIZE}(\tilde{C}) = \mathsf{SIZE}(C)+O(C_g)\le T+O(C_g)$. For appropriate choices of constants for the big-oh, $T+O(C_g)<S$ by assumption, meaning that the security of the stretched function $\tilde{g}$ is in terms of a smaller resource bound by a constant related to the time needed to compute g.

5.2.2 On the Equivalence of Pseudorandom Generators and One-Way Functions

While one-way functions are useful in analyzing the underlying security of a cryptosystem, pseudorandom generators are tools to construct a simpler one-time pad protocol. The use of trapdoor one-way functions in a public-key cryptosystem can be used to negotiate a seed to a pseudorandom generator, which stretches to a output the same length as a message. Therefore, both one-way functions and pseudorandom generators are essential to secure cryptographic protocols. Pseudorandom generators are necessarily one-way functions, and this shall be proven first. Thus, pseudorandom generators exist only if one-way functions exist.

However, a more general and somewhat surprising result from [HILL99] is that one-way functions exist if and only if pseudorandom generators exist. The details of the full proof is beyond the scope of this thesis, but we will present a construction of a pseudorandom generator from a one-way permutation.

Theorem 5.25 ([Gol00])**.** *Let $g : \{0,1\}^* \to \{0,1\}^*$ be a pseudorandom generator with stretch $L(n) = 2n$ and security (n^{-k}, S) for some $k \in \mathbb{N}$. Then g is a strong one-way function with security $(n^{-(k+1)}, S - 2S_g)$, where S_g is the size of a circuit computing g.*

Proof. Let g be a pseudorandom generator according to the hypotheses. By definition of pseudorandom generator, for all $l \in \mathbb{N}$,

$$\max_{\mathsf{SIZE}(C) \leq S} |\Pr\{C(g(U_l)) = 1\} - \Pr\{C(U_{2l}) = 1\}| \leq \epsilon$$

Assume that g is not a strong one-way function. Then for some $k \in \mathbb{N}$ and almost all $n \in \mathbb{N}$ there exists a circuit C_n where $\mathsf{SIZE}(C_n) \leq S(n)$ and

$$\Pr\{g(C_n(g(U_n))) = g(U_n)\} > n^{-k}$$

Let D_n be a circuit attempting to distinguish U_{2n} from $G(U_n)$. Define $D_n(x) = 1$ if $g(C_n(g(x))) = g(x)$ and $D_n(x) = 0$ otherwise. Then,

$$\Pr\{D_n(g(U_n)) = 1\} = \Pr\{g(C_n(g(U_n))) = g(U_n)\} > n^{-k}$$

However,

$$\Pr\{D_n(U_{2n}) = 1\} = \Pr\{g(C_n(U_{2n})) = U_{2n}\}$$

Because the extension of g is n, there are at least $2^{n+1} - 2^n = 2^n$ different strings in $\{0,1\}^{2n}$ that C_n cannot invert because the range of f has at most 2^n unique strings. Therefore, $\Pr\{D_n(g(U_n)) = 1\} \leq 2^{-n}$. Putting this together,

$$|\Pr\{D_n(g(U_n)) = 1\} - \Pr\{D_n(U_{2n}) = 1\}| > n^{-k} - 2^{-n}$$

For large enough n, $n^{-k} - 2^{-n} > n^{-(k+1)}$. This contradicts that g is a pseudorandom generator. □

The construction of a pseudorandom generator from a one-way function uses hard bits in the output of the one-way function. Certainly, if the output of a one-way function is often not invertible, then some single bit of the output of the function must be difficult to predict. This property is called a *hard-core predicate*, and they form the foundation for the construction of a pseudorandom generator.

Definition 5.26 ([GL89])**.** Let $f : \Sigma^* \to \Sigma^*$ be a polynomial-time computable function. A function $b : \Sigma^* \to \Sigma$ is a *hard-core predicate* with security (ϵ, S) if b is polynomial-time computable function and for all circuits with $\mathsf{SIZE}(C_n) \leq S(n)$ and almost all $n \in \mathbb{N}$,

$$\Pr\{C_n(f(U_n)) = b(U_n)\} < \frac{1}{2} + \epsilon(n)$$

A *strong hard-core predicate* has security $\epsilon(n) \leq 2^{-n^\delta}$ for some δ, analogous to a strong one-way function.

Theorem 5.27 ([GL89])**.** *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a strong length-regular one-way function. Then $g : \Sigma^* \to \Sigma^*$ defined by $g(x \cdot r) = f(x) \cdot r$ for $|x| = |r|$ is a strong one-way function, and $b(x, r) = x \odot r$ is a strong hard-core predicate for g with security, where $\odot$ denotes the inner product of $\mathbb{Z}_2^{|x|}$.*

Proof. Fix $n \in \mathbb{N}$. Assume by way of contradiction that b is not a hard-core predicate for g. Denote the success probability of determining b as $\frac{1+\epsilon_b}{2}$. Then for some circuit C_b and $k \in \mathbb{N}$ we have

$$\Pr\{C_b(g(U_n)) = b(U_n)\} \geq \frac{1}{2} + n^{-k}$$

Define S as the set of strings in $\{0,1\}^n$ where $g(x,r)$ is equal to $b(x,r)$ with success probability $1/2 + \epsilon_b(n)/2$ over uniform choice of r. That is $x \in S$ if

$$s_p(x) = \Pr\{C_b(f(x), U_n) = b(x, U_n)\} \geq \frac{1+\epsilon_b(n)}{2}$$

First, note that the reverse Markov's inequality gives $\mathsf{E}[s_p(U_n)] - a \leq (1-a)\Pr\{s_p(U_n) \geq a\}$ for any positive constant a because $\Pr\{s_p(U_n) \leq 1\} = 1$. But $\mathsf{E}[s_p(U_n)]$ is precisely $1/2 + \epsilon_b(n)$. If we choose $a = \frac{1+\epsilon_b(n)}{2}$, then

$$\Pr\{s_p(U_n) \geq a\} \geq \frac{\epsilon_b(n)}{2}\frac{1}{1/2 - \epsilon_b(n)} \geq \epsilon_b(n)$$

Therefore, $\Pr\{s_p(U_n) \geq a\} \geq \epsilon_b(n)$, which directly implies that $|S| \geq \epsilon_b(n) \cdot 2^n$.

Consider a circuit C on inputs from $\{0,1\}^n$. Let $l = \lg(2n \cdot n^{-2k} + 1)$. Strings $\langle s_1, ..., s_l \rangle$ where $s_i \in \{0,1\}^n$ and $\langle \sigma_1, ..., \sigma_l \rangle$ where $\sigma_i \in \{0,1\}$ are given as advice to C.

C computes as follows:

Algorithm 3 Computation of C

```
1: $y \leftarrow$ input to circuit, $n \leftarrow |y|$
2: $l \leftarrow \lg(2n \cdot n^{-2k} + 1)$
3: for each nonempty set $J \subseteq \{1, 2, ..., l\}$ do
4:     $r_J \leftarrow \oplus_{j \in J} s_j$
5:     $\rho_J \leftarrow \oplus_{j \in J} \sigma_j$
6: end for
7: for $i = 1$ to $n$ do
8:     for each nonempty set $J \subseteq \{1, 2, ..., l\}$ do
9:         $z_{J,i} \leftarrow \rho_J \oplus C_b(y, r_J \oplus e_i)$. Here, $e_i$ is the $i^{th}$ standard basis vector in $\mathbb{Z}_2^n$.
10:    end for
11: end for
12: for $i = 1$ to $n$ do
13:     $z_i \leftarrow$ the majority value of $z_{J,i}$ over all nonempty $J \subseteq \{1, 2, ..., l\}$
14: end for
15: Output $z_1, ..., z_n$
```

As usual, let $\boldsymbol{C}$ be the uniform random variable assigning equal probability to the set of circuits C over all advice strings.

Fix $x \in S$ and $0 \leq 1 \leq n$. For each non-empty $J \subset \{1, 2, ..., l\}$ and choices of $s_1, ..., s_l$, define an indicator random variable I_J where $I_J = 1$ if $b(x, r_J) \oplus C_b(f(x), r_J \oplus e_i) = x_i$ and 0 otherwise. x_i is the i^{th} bit of x.

Lemma 5.28. *$I_J = 1$ if and only if $C_b(f(x), r_J \oplus e_i) = b(x, r_J \oplus e_i)$.*

Proof. Recall that by the definition of b, $b(x, r_J) \oplus b(x, r_J \oplus e_i) = (x \odot r_J) \oplus (x \odot (r_J \oplus e_i))$. Distributing $\odot$ over $\oplus$, this equals $(x \odot r_J) \oplus (x \odot r_J) \oplus (x \odot e_i)$. Finally, this equals $x \odot e_i$ by the idempotency of $\oplus$. But $x \odot e_i$ is exactly x_i. By this result, $I_J = 1$ if and only if

$$b(x, r_J) \oplus b(x, r_J \oplus e_i) \oplus C_b(f(x), r_J \oplus e_i) = x_i \oplus b(x, r_J \oplus e_i)$$

Simplifying by the previous result, $I_J = 1$ if and only if

$$x_i \oplus C_b(f(x), r_J \oplus e_i) = x_i \oplus b(x, r_J \oplus e_i)$$

which is true if and only if $C_b(f(x), r_J \oplus e_i) = b(x, r_J \oplus e_i)$. □

With this lemma, consider the following probability for fixed non-empty $J \subset \{1, 2, ..., l\}$ taken over all possible choices of s_i. With the s_i uniformly distributed, the r_J are also uniformly distributed.

$$\Pr_{r_J \in \{0,1\}^n} \{I_J = 1\} = \Pr_{r_J \in \{0,1\}^n} \{C_b(f(x), r_J \oplus e_i) = b(x, r_J \oplus e_i)\}$$

By assumption, $x \in S$, and the previous probability is identified with $s_p(x) \geq \frac{1+\epsilon_b(n)}{2}$.

Lemma 5.29. *I_J is independent from I_K for $J \neq K$.*

Proof. It suffices to show that r_J and r_K are independent to prove that I_J is independent from I_K. Because J, K are not empty and not equal, there is some $j \in J \setminus K$ and $k \in K \setminus J$. For any $a, b \in \{0,1\}^n$, $\Pr\{r_K = b | r_J = a\} = \Pr\{s_k = b | s_j = a\}$ where the choices of s_i are uniform from $\{0,1\}^n$. Since the s_i are independent, $\Pr\{r_K = b | r_J = a\} = \Pr\{s_k = b\} = \Pr\{r_K = b\}$, which establishes the independence of r_K from r_J. □

Using this independence of I_J, we wish to determine the number of subsets J that provide sufficient information for predicting the hard-core predicate b. Proceed by taking a probability over uniform choice of J, which is identical to summing over indicators.

$$\Pr\left\{|\{J | I_J = 1\}| > \frac{1}{2}(2^l - 1)\right\} = \Pr\left\{\sum_J I_J > \frac{1}{2}(2^l - 1)\right\}$$

Note that there are precisely $2^l - 1$ choices for non-empty J. Focusing on the complement set,

$$\Pr\left\{\sum_J I_J \leq \frac{1}{2}(2^l - 1)\right\} \leq \Pr\left\{\sum_J I_J \leq \frac{1}{2}(2^l - 1)\right\} + \Pr\left\{\sum_J I_J \geq \left(\frac{1}{2} + \epsilon_b(n)\right)(2^l - 1)\right\}$$

$$\leq \Pr\left\{\sum_J I_J - \frac{1+\epsilon_b(n)}{2}(2^l - 1) \leq -\frac{\epsilon_b(n)}{2}(2^l - 1)\right\} + \Pr\left\{\sum_J I_J - \frac{1+\epsilon_b(n)}{2}(2^l - 1) \geq \frac{\epsilon_b(n)}{2}(2^l - 1)\right\}$$

$$= \Pr\left\{\left|\sum_J I_J - \frac{1+\epsilon_b(n)}{2}(2^l - 1)\right| \geq \frac{\epsilon_b(n)}{2}(2^l - 1)\right\}$$

Now, $\mathsf{E}[X_J] \geq \frac{1+\epsilon_b(n)}{2}$ because x is *a priori* in S. Applying Chebyshev's inequality,

$$\leq \frac{\mathsf{Var}[\sum_J I_J]}{[\frac{\epsilon_b(n)}{2}(2^l - 1)]^2} = \frac{(2^l - 1)\mathsf{Var}[I_J]}{[\frac{\epsilon_b(n)}{2}(2^l - 1)]^2} = \frac{4 \cdot \mathsf{Var}[I_J]}{\epsilon_b(n)^2(2^l - 1)}$$

The variance of any indicator random variable is less than or equal to 1/4. By definition of l, $2^l - 1 = 2n \cdot n^{-2k}$. Therefore,

$$\frac{4 \cdot \mathsf{Var}[I_J]}{\epsilon_b(n)^2(2^l-1)} \leq \frac{1}{\epsilon_b(n)^2(2n \cdot n^{-2k})} = \frac{1}{2n}$$

What this proves is that $\Pr\{\sum_J I_J > \frac{1}{2}(2^l - 1)\} \geq 1 - \frac{1}{2n}$. Moreover, for $n \geq 1$, the probability that $\boldsymbol{C}$ produces $z_i = x_i$ for $0 \leq i \leq n$ is greater than 1/2 when $\sigma_i = b(x, s_j)$. Since this appropriate selection of σ occurs with probability 2^{-l}, the probability that $\boldsymbol{C}$ outputs x where $f(x) = y$ is $2^{-l-1} \geq n^{-2k-3}$. This is under the assumption that $x \in S$. Since $|S| > \epsilon_b(n) \cdot 2^n$, the probability that any such $x \in U_n$ is inverted is $> 1/2n^{-2k-3} > n^{-2k-4}$. This contradicts that f is strongly one-way by choosing an appropriate circuit in $\boldsymbol{C}$ to invert f on at least n^{-2k-4} fraction on inputs. □

By extracting hardcore predicates from a one-way permutation, these bits will look random to any efficient adversary, and thus forms a pseudorandom generator. The use of a one-way permutation is critical for this proof to ensure that the image of the uniform distribution is also uniform. In [HILL99], universal hash functions permit any general one-way function to exhibit the properties desired to conclude that any one-way function can be used to construct a pseudorandom generator and thus prove their equivalence.

Theorem 5.30 ([Yao82, GL89])**.** *If there exists a one-way permutation over $\Sigma = \{0,1\}$ with success probability ϵ then there exists a pseudorandom generator with success probability ϵ.*

Proof. Let $\Sigma = \{0,1\}$, and let $f : \Sigma^n \to \Sigma^n$ be a one-way permutation. The candidate pseudorandom generator is $g(x) = f(x) \cdot b(x)$, where $b(x)$ is a hard-core predicate for f. The extension of g is 1, and g is polynomial-time computable because both f and b are.

Define $\overline{b}(x) = 1 - b(x)$. That is, $\overline{b}(x) = 0$ whenever $b(x) = 1$, and vice versa. Consider the random variable E over $\{0,1\}^{n+1}$ defined by $E = f(U_n) \cdot \overline{b}(U_n)$.

Lemma 5.31. *If some circuit C distinguishes $g(U_n)$ from E with success probability ϵ, then C distinguishes $g(U_n)$ from U_{n+1} with success probability $\epsilon/2$.*

Proof. Note that $g(U_n) = f(U_n) \cdot b(U_n)$ and $E = f(U_n) \cdot \overline{b}(U_n)$. By assumption,

$$|\Pr\{C(g(U_n)) = 1\} - \Pr\{C(E) = 1\}| \leq \epsilon$$

Because f is a permutation, $f(U_n) = U_n$. So, $Pr\{C(E) = 1\} = \Pr\{C(U_n \cdot \overline{b}(U_n)) = 1\}$ and $g(U_n) = U_n \cdot b(U_n)$.

Now, $U_{n+1} = U_n \cdot U_1$. Since U_1 is uniformly 0 or 1, it is then uniformly $b(U_1)$ or $\overline{b}(U_1)$. Thus,

$$\Pr\{C(U_{n+1}) = 1\} = \frac{1}{2}\Pr\{C(U_n \cdot b(U_1)) = 1\} + \frac{1}{2}\Pr\{C(U_n \cdot \overline{b}(U_1)) = 1\}$$

$$= \frac{1}{2}\Pr\{C(f(U_n) \cdot b(U_1)) = 1\} + \frac{1}{2}\Pr\{C(f(U_n) \cdot \overline{b}(U_1)) = 1\}$$

Hence,

$$|\Pr\{C(g(U_n)) = 1\} - \Pr\{C(U_n \cdot \overline{b}(U_n)) = 1\}| \leq \epsilon$$

$$|\Pr\{C(g(U_n)) = 1\} - 2\Pr\{C(U_{n+1}) = 1\} + \Pr\{C(U_n \cdot b(U_1)) = 1\}| \leq \epsilon$$

$$2|\Pr\{C(g(U_n)) = 1\} - \Pr\{C(U_{n+1}) = 1\}| \leq \epsilon$$

Therefore, C distinguishes $g(U_n)$ from U_{n+1} with success probability $\epsilon/2$. $\square$

From the result of the lemma, assume by contradiction that for some circuit C with $\mathsf{SIZE}(C) \leq S$ that

$$|\Pr\{C(g(U_n)) = 1\} - \Pr\{C(E) = 1\}| > 2\epsilon$$

Then consider two circuits C_0, C_1 defined on input length n where C_i for $i = 0, 1$ is defined by:

$$C_i(x) = \begin{cases} i & \text{if } C(x \cdot i) = 1 \\ 1 - i & \text{otherwise} \end{cases}$$

If $\boldsymbol{C}$ is the uniform random variables assigning probability 1/2 to C_0 and C_1, then

$$\Pr\{\boldsymbol{C}(f(U_n)) = b(U_n)\}$$

$$= \frac{1}{2}\Pr\{C_0(f(U_n)) = b(U_n)\} + \frac{1}{2}\Pr\{C_1(f(U_n)) = b(U_n)\}$$

$$= \frac{1}{2}(\Pr\{C(f(U_n) \cdot 0) = 0 \text{ and } b(U_n) = 1\} + \Pr\{C(f(U_n) \cdot 0) = 1 \text{ and } b(U_n) = 0\}$$

$$+ \Pr\{C(f(U_n) \cdot 1) = 0 \text{ and } b(U_n) = 0\} + \Pr\{C(f(U_n) \cdot 1) = 1 \text{ and } b(U_n) = 1\})$$

$$= \frac{1}{2}\big(\Pr\{C(f(U_n) \cdot \bar{b}(U_n)) = 0 \text{ and } b(U_n) = 1\} + \Pr\{C(f(U_n) \cdot b(U_n)) = 1 \text{ and } \bar{b}(U_n) = 1\}$$

$$+ \Pr\{C(f(U_n) \cdot \bar{b}(U_n)) = 0 \text{ and } \bar{b}(U_n) = 1\} + \Pr\{C(f(U_n) \cdot b(U_n)) = 1 \text{ and } b(U_n) = 1\}\big)$$

$$= \frac{1}{2}\big(\Pr\{C(f(U_n) \cdot \bar{b}(U_n)) = 0\} + \Pr\{C(f(U_n) \cdot b(U_n)) = 1\}\big)$$

$$= \frac{1}{2}\big(\big(1 - \Pr\{C(f(U_n) \cdot \bar{b}(U_n)) = 1\}\big) + \Pr\{C(f(U_n) \cdot b(U_n)) = 1\}\big)$$

$$= \frac{1}{2} + \frac{1}{2}(\Pr\{C(g(U_n)) = 1\} - \Pr\{C(E) = 1\})$$

$$> \frac{1}{2} + 2\epsilon$$

This contradicts that b is a hard-core predicate for f. Therefore, g as constructed is a pseudo-random generator with appropriate success probability ϵ. $\square$

6 Kolmogorov Complexity

The previous two types of randomness, uniform randomness and pseudorandomness, described randomness in terms of a method by which strings are generated and denoted these resulting strings as random. For instance, a uniformly random string may be generated by flipping an unbiased coin many times and recording a 1 if the coin flips heads and 0 if the coin flips tails; a pseudorandom string may be generated by flipping a coin a few times to form a short binary string, and then processing this string through an algorithm to generate a longer output that looks uniformly random most of the time to an efficient adversary. The coin flips act as a source of "perfect" randomness.

The third type of randomness differs by looking at the content of individual strings and not how they were generated. It derives from an information theoretic approach that examines the descriptive randomness of strings. Strings with short descriptions provide an algorithm to generate the string from its description and these are *compressible*. Strings with low compressibility are *Kolmogorov random*, and they play a critical role in the defining the security of any present-day public-key cryptosystem. This section provides an introduction to Kolmogorov complexity as a measure of descriptive randomness of a particular string and various computability theorems related to Kolmogorov random strings that help in uncovering the true security of problems like factoring and the discrete logarithm.

Definition 6.1 ([Sol64, Kol68, Cha66, Cha69])**.** Fix $\Sigma = \{0, 1\}$, and a universal machine U. The *Kolmogorov complexity* of x is defined $K(x) = \min\{|d| \mid U(d) = x\}$ or ∞ if no such d exists.

The definition uses the notion of a universal machine U that constructs the string x from its description d. Note that the description d may be formalized as the encoding of a Turing machine that outputs x with no input. In this case, the machine that outputs x in $|x|$ steps by writing each symbol of x one-by-one on the tape is a suitable description of x. Therefore $K(x) \leq |x| + c$, for some overhead constant c and all strings x. In this paper, we assume that U is a given fixed machine.

The standard definition of Kolmogorov complexity is unwieldy in practice because it is not computable.

Theorem 6.2. *$K(x)$ is not computable.*

Proof. Suppose $K(x)$ is computable. Then there is a Turing Machine M with M on input x halting with $K(x)$ remaining on the tape. Consider $R_k = \{x \in \Sigma^* | K(x) \geq |x|\}$, which is the set of Kolmogorov random strings. Next consider the following sequence of programs $\{P_n\}_{n=0}^{\infty}$.

P_n on no input tests each $x \in \Sigma^n$ in lexical order and on first x satisfying $M(x) \geq n$ outputs x and halts (if no such x satisfies, P_n outputs λ). Let r_n be the output of program P_n. Then r_n is describable by n, which requires $\log n$ bits. So $K(r_n) \leq \log n + k$, for a fixed constant k related to the overhead of P_n. But r_n is output by P_n, so $K(r_n) \geq n$. For sufficiently large n, $\log n + k < n$, a contradiction. □

From this theorem, P_n is given as much time as necessary to find the first $x \in \Sigma^n$ satisfying $M(x) \geq n$. The motivation in bounding the resources of the universal machine U gives rise to a computable Kolmogorov measure.

6.1 Resource-Bounded Kolmogorov Complexity

Resource-bounded Kolmogorov complexities maintain computability by involving a penalty for strings that take many resources to produce. Here resources are either in terms of time or space. This notion prevents strings that require a large amount of time or space to decompress from being labeled as compressible when no such efficient machine can decompress them. Two formalizations of this concept have been espoused.

Definition 6.3 ([Har83]). Fix $\Sigma = \{0, 1\}$, a function $t : \mathbb{N} \to \mathbb{N}$, and a universal function $U : \Sigma^* \to \Sigma^*$. The *Hartmanis time-bounded Kolmogorov complexity* is $K^t(x) = min\{|p| \mid U(p) = x \text{ using at most } t(|x|) \text{ steps}\}$

Definition 6.4 ([Lev84]). Fix $\Sigma = \{0, 1\}$ and a universal function $U : \Sigma^* \to \Sigma^*$. The *Levin time-bounded Kolmogorov complexity* is $K_t(x) = min\{|p| + \log t \mid U(p) = x \text{ in } t \text{ steps}\}$

The Levin definition is ultimately the easier definition to use in practice. While the Hartmanis definition requires a time bound $t(n)$ to be defined *a priori*, the Levin definition mixes the time penalty with a penalty in the size of the description, creating a natural combination of hardness and randomness.

Desirably, both these Kolmogorov complexities are computable.

Theorem 6.5. *$K^t(x)$ is computable.*

Proof. Fix our function t, our Universal function U, and $x \in \Sigma^*$. Any possible program that generates x must be of size $\leq t(|x|)$ since the program must be fully read as input to U. By choice of $\Sigma = \{0, 1\}$, we have $2^{t(|x|)+1}$ possible programs satisfying that length, although in general this is $|\Sigma|^{t(|x|)+1}$ programs.

We loop over all possible programs p and simulate $U(p)$ for $t(|x|)$ steps. We keep track of all programs that output x in that given time. We return the minimum size of such a program, and if no program output x, return ∞. $\square$

Theorem 6.6. *$K_t(x)$ is computable.*

Proof. Fix $t \in \mathbb{N}$ and our universal function U. Let $x \in \Sigma^*$. Any possible program that generates x must be of length $\leq t$ since U requires at most t steps to read the whole input. This is $|\Sigma|^{t+1}$ programs, or 2^{t+1} for a binary input alphabet.

Looping over all possible programs p, simulate $U(p)$ for t steps. Record the minimum length program such that $U(p)$ outputs x in t steps, in which case return $|p| + \log t$. If no such program p outputs x, return ∞. $\square$

Certainly the amount of space used in decompressing d into x can be penalized analogously to the penalty given to time.

Definition 6.7. Fix $\Sigma = \{0, 1\}$ and a universal function $U : \Sigma^* \to \Sigma^*$. The *space-bounded Kolmogorov complexity* is $K_s(x) = min\{|p| + s \mid U(p) = x \text{ in } s \text{ space}\}$

Theorem 6.8. *$K_s(x)$ is computable in* PSPACE.

Proof. Fix $s \in \mathbb{N}$ and our universal machine U. Let $x \in \Sigma^*$. A possible description of x cannot be longer than s since we otherwise could not store the full input. Loop over all strings y in lexical order (shortest to longest) of strings with length at most s. Run $U(y)$ with a space bound of s, and stop if we use more than s squares of tape or if we start looping, which is detectable in finite space. If $U(y)$ ever halts and outputs x, then halt and output $|y|$. If no y is a description of x, output ∞.

Note that $K_s(x) \leq |x| + c$ because we can always have a machine on no input that outputs x on the tape directly so long as $|x| \leq s$. Therefore each iteration of the loop needs no more than $|x| + c = O(|x|)$ space. If $|x| > s$, then each iteration uses $s = O(|x|)$ space before outputting ∞. Hence, this algorithm uses linear space with respect to the input length, and so $K_s(x)$ is PSPACE computable. $\square$

Some alternative Kolmogorov complexities could be defined in terms of a mixture of functions of the time and space required for the decompression of d into x. There are caveats with applying any arbitrary penalty for time and space, and this caveat will be addressed subsequently. However, in terms of establishing a relationship to public-key cryptography, the standard complexities are used.

6.2 Relativized Resource-Bounded Bit-by-Bit Kolmogorov Complexity

If definition 6.4 is modified for a time penalty of t rather than $\log t$, then the resulting Kolmogorov measure trivial. For any string x, the description that outputs x in linear fashion satisfies $|d| = 2|x| + c$, for a constant c, because the description contains the string x and needs time to write x as output. The time necessary for the universal machine to output the string x skews the desired measure on the time-bounded randomness of a given string [Ron04]. This inspired a bit-by-bit analysis, whereby a universal machine determines each bit of a string x and is penalized for the length of the description plus the time t of computation.

Definition 6.9 ([Ron04])**.** Let A be a set of strings, $t \in \mathbb{N}$, and U an universal function. Then the *time-bounded Kolmogorov complexity relative to A* is $KT^A(x) = min\{|d| + t \mid \forall b \in \{0, 1, *\} \forall i \leq |x| + 1, U^A(d, i, b)$ accepts in t steps if and only if $x_i = b\}$

Analogously define KS^A, where the space penalty is s, and define Kt^A, where the time penalty is the original $\log t$. Using this definition from [Ron04], a whole class of time and space penalties can be analyzed in terms of their effect on the Kolmogorov measure. Next, $KS^{\emptyset}$, that is giving no oracle in the bit-by-bit case, is equivalent to K_s defined previously.

Theorem 6.10 ([Ron04])**.** $KS^{\emptyset}(x) = K_S(x)$

Proof. Fix $x \in \Sigma^*$. We can compute the bit-by-bit $KS^{\emptyset}(x)$ by using the same description d and universal machine U' used for $KS(x)$. Set $U(d, i, b)$ to accept if $U'(d)$ outputs b on bit i and reject otherwise. Since U' runs in s space, the space usage of $U(d, i, b)$ will also be s. Since we are using the same universal machine, there is no space overhead in interpreting the input d. Therefore $KS^{\emptyset}(x) \leq KS(x)$.

Now, given a description d for $KS^{\emptyset}(x)$ and universal machine U, we can simulate U, but we must be careful not to use more than exactly s space. If Γ is the output alphabet for U, we use $(\Gamma \times \Gamma) \cup \Gamma$ as the output alphabet for U' for $KS(x)$. Looping from $i = 1$ to $|x|$, $U'(d)$ will simulate $U(d, i, b)$ on the first coordinate of the output alphabet in s space for each $b \in \{0, 1, *\}$. If U ever needs to access the oracle, we return a negative answer since we are querying the empty set. If U accepts, then we write the i^{th} character to the second coordinate of the output tape. If U accepts when $b = *$, then we write all second coordinates of the tape to the full block of the tape and halt. It is clear from the definition of $U(d, i, b)$ that we get each bit of x. x is calculated and stored in no more than s space. Hence, $KS(x) \leq KS^{\emptyset}(x)$. $\square$

Remark 6.11. $Kt^{\emptyset}$ and Kt are related within a multiplicative constant using the same proof as above.

6.3 Kolmogorov Random Strings

The Kolmogorov measures provide a numerical rank of the randomness or compressibility of a given string. Isolating strings with large Kolmogorov measure form the basis of specifying these strings as Kolmogorov random. Naturally, longer strings have greater Kolmogorov measure, so this notion of randomness takes into account the Kolmogorov measure relative to the length of the original string.

Definition 6.12. Let $\epsilon > 0$ and let $K\mu$ be a Kolmogorov measure. The set of *$K\mu$-random strings with respect to ϵ* is $R_{K\mu,\epsilon} = \{x | K\mu(x) \geq |x|^{\epsilon}\}$.

Remark 6.13. An alternative definition of $R_{K\mu,\epsilon}$ is $\{x | K\mu(x) \geq \epsilon|x|\}$. This definition does not change the results presented here [ABK$^+$06].

Let $R_{KT} = R_{KT,\frac{1}{2}}$ and $R_{KS} = R_{KS,\frac{1}{2}}$ be the canonical sets of time-bounded and space-bounded Kolmogorov random string, respectively. Certainly, because the measures KT and KS are computable, then these sets are decidable. Curiously, as R_{KT} plays an important role for factoring and the discrete logarithm, both of which are computable in nondeterministic polynomial time, R_{KT} itself resides in coNP.

Theorem 6.14 ([ABK$^+$06]). $R_{KT} \in \mathsf{coNP}$

Proof. Fix $x \in \Sigma^*$ and let $n = |x|$. Nondeterministically choose a description d of length $\leq \sqrt{n}$. Loop through $i = 1$ to $|x|$, and loop through each $b \in \{0, 1, *\}$. Simulate $U(d, i, b)$ for at most $\sqrt{n} - |d|$ steps on each inner loop. If $U(d, i, b)$ accepts if and only if $x_i = b$, then certainly $R_{KT}(x) \leq \sqrt{n}$, so we reject on this branch. Otherwise, we accept. All branches accept if and only if $R_{KT}(x) > \sqrt{n}$. Each branch uses $3|x|$ iterations of the loop to, simulating $U(d, i, b)$ each time in polynomial time, and so each branch runs in polynomial time. Thus $R_{KT} \in \mathsf{coNP}$. $\square$

R_{KS} is certainly in PSPACE as the following theorem demonstrates. However, it is not complete for PSPACE under logarithmic space Turing reductions.

Theorem 6.15 ([ABK$^+$06]). $R_{KS} \in \mathsf{PSPACE}$

Proof. Let $y \in \Sigma^*$. Assume $|y| = n$. This algorithm accepts if $KS(y) \geq \sqrt{n}$. Loop through strings of length less than $\sqrt{n}$. For each description d of length $\leq \sqrt{n}$, we can use at most $s = \sqrt{n} - |d|$ space. Loop within 2^s time and check if the $U(d)$ returns y within s space. If so, we reject. Also reject if U tries to use more than s space. If all strings of length less than $\sqrt{n}$ do not describe y in s space, then we know that $KS(y) \geq \sqrt{n}$ and hence, $y \in R_{KS}$.

Checking each string requires at most s space plus a constant. Therefore, R_{KS} requires polynomial space, and $R_{KS} \in \mathsf{PSPACE}$. $\square$

To show that R_{KS} is not logarithmic-space Turing complete for PSPACE, it suffices to show that R_{KS} is not logarithmic-space truth-table complete for PSPACE because these two types of reductions are equivalent.

Lemma 6.16. *$\leq_T^{\mathsf{L}}$ is equivalent to $\leq_{tt}^{\mathsf{L}}$*

Proof. Let A and B be subsets of Σ^*.

If $A \leq_{tt}^{\mathsf{L}} B$, then the Turing reduction at a minimum can compute the truth table and therefore $A \leq_T^{\mathsf{L}} B$.

Assume $A \leq_T^{\mathsf{L}} B$ and let $a \in A$. The reduction has $c \log n$ space, where $|a| = n$ and c is a constant. Therefore there are $2^c n$ different oracle queries that can be made since the query tape obeys the same space constraint. The queries can be ordered lexically and described by an index of length up to $\log 2^c n = O(\log n)$ Therefore the answers to these queries can be precomputed in a truth table and stored in log space. When simulating the Turing reduction, the index for a query is calculated from the natural lexical order of the query, and the result used from the precomputed truth table. Thus, $A \leq_{tt}^{\mathsf{L}} B$. $\square$

Finally, by the space hierarchy theorem, there is a separation between PSPACE and L. Using a *tally set* is pivotal to achieve the result for R_{KS}.

Definition 6.17. Let L be a language. The *Tally set of* L is the set $\{0^{|x|} | x \in L\}$.

Remark 6.18. Note that if $L \in \mathsf{PSPACE} \setminus \mathsf{L}$, then the Tally set of L is in L.

Theorem 6.19 ([ABK$^+$06])**.** *R_{KS} is not $\leq_{tt}^{\mathsf{L}}$-complete for* PSPACE

Proof. Assume R_{KS} is $\leq_{tt}^{\mathsf{L}}$-complete for PSPACE. Let T be a Tally language in $\mathsf{PSPACE} \setminus \mathsf{L}$. Then $T \leq_{tt}^{\mathsf{L}} R_{KS}$. Since it is a truth table reduction, there is a function $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ that generates a set of queries for R_{KS}. f is a P-computable function running in log space.

We describe the reduction on input 0^n. For any query $q_i \in f(0^n)$, we have $KS(q_i) \leq c + O(\log n)$ where c is the description of f and $O(\log n)$ accounts for the description of 0^n. Now if $|q_i| > \log n$, we know $KS(q_i)$ must be at least 1, and therefore $q_i \notin R_{KS}$. So we only use queries of size less than or equal to $\log n$. These queries use linear space of $\log n$, or log space of n. Thus, the entire reduction runs in log space of n.

But instead of querying the oracle, we can compute $KS(q_i)$ in linear space of $|q_i|$, which is log space of n. Therefore, this describes a L algorithm that decides R_{KS}. So $R_{KS} \in \mathsf{L} \subsetneq \mathsf{PSPACE}$, contradicting the fact that R_{KS} is $\leq_{tt}^{\mathsf{L}}$-complete for PSPACE. $\square$

While the relationships between completeness in terms of various reductions remain relatively unknown, R_{KS} is a natural example of a language that is not PSPACE complete under $\leq_{tt}^{\mathsf{L}}$ reductions, but may be complete under $\leq_{tt}^{\mathsf{P}}$ reductions under the appropriate conditions. In contrast, R_{Kt} is complete for EXP and R_{KS} is complete for EXP both under randomized reductions [ABK$^+$06]. A variant of R_{Kt} using the Hartmanis time-bounded Kolmogorov measure is complete for EXP under randomized reduction but not complete under Turing reductions [BM95]. R_{KS} remains an intermediate problem under these reductions, and its power to distinguish space-bounded Kolmogorov strings is likely not as powerful as deciding all of PSPACE. These results imply that the ultimate construction of a class $\mathsf{BPP}^{R_{KT}}$ may be able to separate factoring and the discrete logarithm from the hardest problems residing within PSPACE.

7 Cryptographic Formalisms with Turing Machines

To this point, the abilities of an adversary have been identified by a success probability ϵ and a resource bound S. This resource bound S identified itself with an upper bound on the size of circuits that an adversary may use either to find a preimage under a one-way function f or to distinguish pseudorandom strings from uniformly random strings. To generalize, the adversary could be equipped with any non-uniform set of polynomial-sized circuits from $\mathsf{P/poly}$[1]. This approach is a double-edged sword. Relaxing the constraint of the resources of the adversary from an upper bound on circuit size to access to computation in a complexity class simplifies the analysis in many cases. However, it also detracts from the ultimate goal of this cryptographic analysis, which is to provide a prescription of the strength of a public-key cryptosystem with definite bounds on the capabilities of adversaries. Nevertheless, this alternate definition of security provides the relationship of public-key cryptography to relationships between well-known complexity classes.

Definition 7.1. Let $\mathcal{C}$ be a class of languages (i.e., P, NP, $\mathsf{DTIME}(2^{n^\epsilon})$). Denote $\mathcal{M}(\mathcal{C})$ as the *machine class*, the set of machines that decide the languages in $\mathcal{C}$ in the appropriate resource bound.

The machine class will be complete set of resources available to an adversary. In the previous analysis, an extra constraint, such as size of the circuit, augmented to the security definition given next. To generalize the security of a pseudorandom generator, the parameter S is replaced by a complexity class $\mathcal{S}$, and the adversary has access to any machine within this class.

Definition 7.2 (Turing computational distance)**.** Let X, Y be probability distributions over Σ^n for some $n \in \mathbb{N}$. The *Turing computational distance* with respect to class $\mathcal{S}$ between X and Y is

$$\Delta_{\mathcal{S}}(X, Y) = \sup_{A \in \mathcal{M}(\mathcal{S})} |\Pr\{A(X) = 1\} - \Pr\{A(Y) = 1\}|$$

For pseudorandom generators and one-way functions, we also generalize the computability of the function. Therefore, in general, any of these cryptographic primitives are identified by a computability parameter $\mathcal{C}$, a success probability ϵ, a resource bound $\mathcal{S}$, and possibly a bound on the size of machines S. Here, $\mathcal{C}$ and $\mathcal{S}$ are classes of functions and not of languages.

Definition 7.3 (Generalized pseudorandom generator)**.** Let $l, L \in \mathbb{N}$ and $\epsilon > 0$. A length-regular function $g : \Sigma^l \to \Sigma^L$ is a *$\mathcal{C}$-pseudorandom generator* with security $(\epsilon, \mathcal{S})$ if g is $\mathcal{C}$-computable and

$$\Delta_{\mathcal{S}}(g(U_l), U_l) \leq \epsilon$$

where $L - l$ is the *extension* of g and U_l is the uniform distribution over Σ^l.

A $\mathcal{C}$-pseudorandom generator ensemble is defined analogously.

Definition 7.4 (Generalized one-way function)**.** Let $\epsilon : \mathbb{N} \to [0, 1]$, and let $\mathcal{C}, \mathcal{S}$ be functional classes. $f : \Sigma^* \to \Sigma^*$ is a *$\mathcal{C}$-one-way function* with security $(\epsilon, \mathcal{S})$ if f is $\mathcal{C}$-time computable and for all machines $M \in \mathcal{M}(\mathcal{S})$ and for almost all $l \geq 0$,

$$\Pr(f_l(M(f_l(U_l))) = f_l(U_l)) < \epsilon(l)$$

[1]These previous definitions are commonly called non-uniform one-way functions and non-uniform pseudorandom generators in the literature.

It should be noted that by this definition of a generalized one-way function, the function $f(x) = \lg^2(x)$ is one-way secure against any polynomial-time complexity class (i.e., P, NP, BPP, etc.). The input to this function is exponential in the size of the output, and therefore any polynomial adversary would require exponential time to write a preimage for a given output, although the algorithm for producing such a preimage is straightforward. The concept to ensure that a function is not pathologically length-decreasing is *honesty.*

Remark 7.5. A function $f : \Sigma^* \to \Sigma^*$ is *polynomially honest* if there exists $k \in \mathbb{N}$ such that for all $x \in \Sigma^*$, $|x| \leq |f(x)|^k$. We implicitly assume that for any one-way function f secure against $\mathcal{S}$ that f has the appropriate honesty condition related to the time bounds presented by $\mathcal{S}$.

If polynomial-time one-way functions exist with security against P and some appreciable success probability, then $\mathsf{P} \neq \mathsf{NP}$. This result again underscores the difficulty of proving the existence of any one-way function because it necessitates a panacea to the difficulty of separating P from NP. This relationship is bidirectional as any proof of $\mathsf{P} \neq \mathsf{NP}$ is sufficient for the existence of a one-way function.

Theorem 7.6 ([HO02])**.** *If there exists a* P*-one-way function with security* $(\epsilon, \mathcal{S})$ *where* $\epsilon(n) < 1$ *for some* $n \in \mathbb{N}$ *and* $\mathsf{P} \subseteq \mathcal{S}$*, then* $\mathsf{P} \neq \mathsf{NP}$.

Proof. Assume $\mathsf{P} = \mathsf{NP}$. Let L be the following language (where $\leq$ denotes lexicographic ordering):

$$L = \{\langle x', y\rangle | f(x) = y \text{ for some } x \leq x'\}$$

$L \in \mathsf{NP}$: On input $\langle x', y\rangle$, nondeterministically choose a $x \leq x'$. If $f(x) = y$, then accept; otherwise, reject. Since f is a P-computable function, this procedure decides L in NP time.

By assumption, $L \in \mathsf{P}$ since $\mathsf{P} = \mathsf{NP}$. Therefore, any polynomial time machine whose language is L can be used as an oracle for queries in a binary search. A machine performing this procedure will run in polynomial time and so it is in the machine class of $\mathcal{S}$ by assumption $\mathsf{P} \subset \mathcal{S}$. Therefore, f is inverted for all strings in its range, and thus $\epsilon(n) = 1$ for all $n \in \mathbb{N}$. ☐

Theorem 7.7 ([HO02])**.** *If* $\mathsf{P} \neq \mathsf{NP}$*, then there exists a* P*-one-way function with security* (ϵ, P) *where* $\epsilon(n) < 1$ *for some* $n \in \mathbb{N}$.

Proof. Let $L \in \mathsf{NP} \setminus \mathsf{P}$. Then there exists $N \in \mathcal{M}(\mathsf{NP})$ deciding L. Consider the function $f : \{0,1\}^* \to \{0,1\}^*$ defined by

$$f(\langle x, w\rangle) = \begin{cases} 1x & \text{if } w \text{ is not an accepting path of } N \text{ on input } x \\ 0x & \text{otherwise} \end{cases}$$

This function is polynomial-time computable because following a path in an NP machine is a polynomial-time procedure. This function is also honest because w is at most the run time of N on x, which is polynomial, and there is at most polynomial overhead for the pairing function. Therefore, $\langle x, w\rangle$ is polynomial in the length of x.

Suppose $\epsilon(n) = 1$ for all $n \in \mathbb{N}$. Then some polynomial-time functional Turing machine M produces $\langle x, w\rangle$ on input $0x$ or $1x$ for all $x \in \{0,1\}^*$.

Using M, define a Turing machine M' on input x, an input instance for L. M' calculates $M(1x)$ to produce a pair $\langle x, w\rangle$. By simulating the computation of $N(x)$ and following the path provided by w, M' accepts x if w is an accepting path and rejects otherwise. Thus, M' is a polynomial-time computable machine deciding L. But $L \in \mathsf{P}$ is a contradiction of $L \in \mathsf{NP} \setminus \mathsf{P}$. Hence, $\epsilon(n) < 1$ for some $n \in \mathbb{N}$. $\square$

By [HILL99], these results are analogous for pseudorandom generators. What is unknown is whether the existence of a secure polynomial-time *trapdoor* one-way function might be stronger than the condition that $\mathsf{P} \neq \mathsf{NP}$–even in terms of security against P. It is generally believed that $\mathsf{P} \neq \mathsf{NP}$ is not strong enough to generate a one-way function secure against BPP, which encompasses the class of efficient algorithms available to classical computation and is the most fundamental basis of security for any one-way function.

8 The Possibility of Proving the Existence of a Pseudorandom Generator

Razborov and Rudich explore the possibilities of finding a natural proof of the existence of pseudorandom generators in [RR94]. If a proof showed a function is not in $\mathsf{P/poly}$, then the proof as written describes a method to distinguish the function from true randomness. Recall that proving the existence of a pseudorandom generator in the case of security against non-uniform circuits requires a proof that any distinguisher is not in $\mathsf{P/poly}$. Therefore, the proof would inherently break the generator. Razborov and Rudich categorized these proofs as *almost self-defeating.*

A *natural proof* uses a *natural combinatorial property* to classify functions as either in $\mathcal{C}$ or not in $\mathcal{C}$, where $\mathcal{C}$ is a complexity class. Here, a *property* is a mapping of boolean functions into $\{0, 1\}$. The important properties in exploring the possibilities of proofs of important cryptographic primitives lies in a special subset of properties.

Definition 8.1. A P-*natural property useful against* $\mathsf{P/poly}$ is a property Φ where:

1. (Constructivity). Φ is polynomial-time computable with respect to the size of the truth table of f.
2. (Largeness). $\Phi(f) = 1$ for at least $\frac{1}{2^n}$ of all functions of input size n.
3. (Usefulness). $\Phi(f) = 1$ implies $f \notin \mathsf{P/poly}$.

Razborov and Rudich explain that any *natural* proof used show the existence of a pseudorandom generator is almost self-defeating. The distinction of natural is critical because most approaches to proving lower time bounds of problems such as factoring and the discrete logarithm assemble a proof that provides exactly a P-natural property useful against $\mathsf{P/poly}$. While the minutiae of a natural proof are omitted, the theorem demonstrating the self-defeating nature of natural proofs in a cryptographic realm is proved.

Theorem 8.2 ([RR94])**.** *The existence of a* P-*natural property useful against* $\mathsf{P/poly}$ *implies there are no strong pseudorandom generators secure against* $\mathcal{C}$ *adversaries where* $\mathsf{P} \subseteq \mathcal{C}$.

Proof. Suppose there is a P-natural property useful against $\mathsf{P/poly}$. Label this property Φ. For the sake of contradiction, suppose g is a strong pseudorandom generator secure against $\mathsf{P/poly}$ adversaries. Assume without loss of generality that g stretches input of length n to a pseudorandom string of length equal to a power of 2. This stretch to the nearest power of 2 can be done in polynomial time from the original pseudorandom generator, and this new pseudorandom generator is also strong.

Fixing $n \in \mathbb{N}$, suppose $|g_n(x)| = 2^c$ for some $c \in \mathbb{N}$ and all $x \in \{0,1\}^n$. Note that g_n is in $\mathsf{P/poly}$ and therefore $\Phi(g_n) = 0$. Since the output of g_n is a power of 2, the output of g_n then becomes the truth table of a *pseudorandom function.* This function is computable in $\mathsf{P/poly}$ time because g_n is. Then Φ distinguishes U_{2^c} from $g_n(U_n)$.

$$|\Pr\{\Phi(U_{2^c}\} = 1) - \Pr\{\Phi(g_n(U_n)) = 1\}|$$

However, $\Phi(g_n(U_n)) = 0$ by the usefulness of Φ.

$$= |\Pr\{\Phi(U_{2^c}\} = 1)\}|$$

Moreover, $\Pr\{\Phi(U_{2^c}\} = 1)\} \geq 2^{-c}$ by the largeness condition of Φ. Therefore,

$$|\Pr\{\Phi(U_{2^c}\} = 1) - \Pr\{\Phi(f_k(U_k)) = 1\}| \geq 2^{-c}$$

Since $c \geq \lg n$ and Φ is polynomial-time computable, this contradicts that g is a pseudorandom generator secure against P adversaries. □

A useful corollary is that if Φ is computable by polynomial size circuits, then the same result applies for pseudorandom generators secure against $\mathsf{P/poly}$. This conclusion applies in the analysis of the relationship between Kolmogorov complexity and pseudorandom generators.

Corollary 8.3. *The existence of a* $\mathsf{P/poly}$*-natural property useful against* $\mathsf{P/poly}$ *implies there are no strong pseudorandom generators secure against* $\mathcal{C}$ *adversaries where* $\mathsf{P/poly} \subseteq \mathcal{C}$*.*

From the analysis in [RR94], any reasonable proof, natural or otherwise, establishing the existence of a pseudorandom generator will require both constructivity and usefulness. Therefore, the techniques required to prove the existence of a pseudorandom generator will forego the largeness requirement. This means that proving that integer multiplication is one-way (and therefore implies the existence of a pseudorandom generator from the factoring) demands analysis beyond inclusion or exclusion of just a small sample of the overall problem instances.

9 Trading Hardness for Randomness

Returning the to problems of factoring and the discrete logarithm, a key result from [ABK+06] connects the three types of randomness with an upper bound on the solutions to these problems. First is a restatement of the result from [HILL99] proving the equivalence of the existence of one-way functions with the existence of pseudorandom generators.

Theorem 9.1 ([HILL99])**.** *Let f be a polynomial-time computable function. If there exists a distinguisher $B \in$ BPP of G_f from the uniform distribution succeeding on at least n^{-l} fraction of inputs of length n for some $l \in \mathbb{N}$, then there exists an adversary $A \in$ BPP of f succeeding on at least n^{-k} fraction of inputs of length n for some $k \in \mathbb{N}$.*

Proof. If f is not strongly one-way against BPP adversaries, then the theorem follows. If f is strongly one-way against BPP adversaries, then construct the pseudorandom generator G_f as in [HILL99]. From this construction in [HILL99], G_f is strong against BPP adversaries. The theorem follows from the contrapositive. □

Corollary 9.2 ([HILL99])**.** *Theorem 9.1 follows for a relativized class* BPP^L*; that is, the class* BPP *with oracle access to the language* L*.*

By using the previous result, the hardness of various problems–including those of interest to cryptographers–can be solved in terms of computation of time-bounded Kolmogorov random strings. These Kolmogorov random strings will not contain pseudorandom strings for a large enough extension of a pseudorandom generator. By the previous theorem, distinguishing these Kolmogorov random strings from compressible strings yields a method of distinguishing pseudorandom strings from uniformly random strings, and the function underlying the pseudorandom generator can be inverted at least on a polynomial fraction of inputs.

Theorem 9.3 ([ABK$^+$06])**.** *Let* L *be a polynomially-dense language where* $KT(x) \geq |x|^\epsilon$ *for some* $\epsilon > 0$ *and all* $x \in L$ *(i.e.,* L *is the set of Kolmogorov random strings for* $\epsilon > 0$*). Let* $f_y : \Sigma^* \to \Sigma^*$ *be computable in polynomially time where* $y \in \Sigma^*$ *is a parameter. Then there exists a* BPP *adversary of* f *succeeding on a polynomial fraction of inputs.*

Proof. Construct pseudorandom generator G_{f_y} as in [HILL99]. Following from theorem 5.22, construct pseudorandom function generator g extending strings of length n to strings of length 2^k. k will be chosen so that g is still polynomial-time computable. Using a formulation in [RR94] similar to theorem 5.22, the i^{th} bit of $g(x)$ can be computed in time $O((k+n)^p)$ for some fixed $p \in \mathbb{N}$ and where $n = |x|$.

Then $KT(g(x)) \leq O(n + (k+n)^p) = O((k+n)^p)$ using x as a description and the time of th procedure needed to procure the i^{th} bit of $g(x)$. Now, $|x| = |g(x)|^{1/c}$ for a constant c depending only on k, so $KT(g(x)) \leq O(|g(x)|^{p/c})$ for appropriate choice of $k = O(\lg n)$. For large enough k, $KT(g(x)) \leq |g(x)|^\epsilon$. L contains no pseudorandom strings generated by g because these strings are not Kolmogorov random.

For a distinguisher B of $g(U_n)$ from the uniform distribution U_{2^k}, on input y do the following: If $KT(y) \in L$, output 0; otherwise, output 1. Note that $B \in \mathsf{P/poly}$ with c the advice given for inputs of length n.

For all strings output by g, B will output 1 since these pseudorandom strings are not time-bounded Kolmogorov random. Over the uniform distribution U_{2^k}, B will output 0 with probability equal to the density of strings in L. By assumption, L is polynomially dense. Therefore, the Turing computational distance between $g(U_n)$ and U_{2^k} is at most n^{-k} for some $k \in \mathbb{N}$. By theorem 9.1, f can be inverted on some polynomial n^{-l} fraction of inputs by a BPP adversary. An explicit construction of the statistic test is given in [RR94]. $\square$

Applying this result to the discrete logarithm in both $\mathbb{Z}_p^*$ and a general group G and to factoring provides an upper bound to the capabilities of an adversary desiring to break modern-day public-key cryptosystems.

Theorem 9.4 ([ABK$^+$06])**.** *The discrete logarithm in* $\mathbb{Z}_p^*$ *for a prime* p *is computable in* $\mathsf{BPP}^{R_{KT}}$*.*

Proof. On input $\langle p, \alpha, \beta \rangle$, we must output an integer a at least $2/3$ of the time such that $\alpha^a = \beta$ in $\mathbb{Z}_p^*$. Ensure that p is prime [AKS02] and that $0 \leq \alpha, \beta < p$. The discrete logarithm is the inverse problem of discrete exponentiation given by $f_\alpha(x) = \alpha^x \mod p$. Using the previous theorem, there is a probabilistic Turing machine with oracle access to R_{KT} producing preimages with probability n^{-k}, where $n = |x|$.

Next, using the random self-reducibility of the discrete logarithm, the success probability of n^{-k} can be amplified to over 2/3 probability required by a BPP algorithm. Randomly choosing greater than n^k amount of integers i between 1 and $|\mathbb{Z}_p^*|$, calculate inverses of $\alpha^i \cdot \beta$. By the random self-reducibility of the discrete logarithm, will probability at least 2/3, for at least one such i will yield a result j where $\alpha^j = \alpha^i \cdot \beta$. Therefore, the algorithm outputs $j - i$, where $\alpha^{j-i} = \beta$. □

Theorem 9.5. *The discrete logarithm in a group G is computable in $\mathsf{BPP}^{R_{KT}}$.*

Proof. We assume we are provided functional oracles for G for group multiplication and exponentiation. On input $\langle \alpha, \beta \rangle$, we output an integer a at least 2/3 of the time such that $\alpha^a = \beta$ in G. As the self-reducibility of the discrete logarithm extends to general groups, the proof in the previous theorem applies in this situation, and the result holds. □

For any arbitrary group, β might not fall within the cyclic subgroup generated by α, and the previous algorithm cannot distinguish when it is giving a wrong answer and when no such integer a satisfies $\alpha^a = \beta$. Therefore, this computation cannot proceed in $\mathsf{ZPP}^{R_{KT}}$. In contrast, since every integer has a factorization and any potential factorization can be checked in polynomial time, factoring can be computed in $\mathsf{ZPP}^{R_{KT}}$.

Theorem 9.6 ([ABK+06]). *Factorization is computable in $\mathsf{ZPP}^{R_{KT}}$.*

Proof. In [Rab79], if the function $f_y(x) = x^2 \mod y$ can be inverted on a polynomial $|y|^{-k}$ fraction of inputs, then any composite integer x can be factored using a BPP algorithm. Using R_{KT} as an oracle, f can be inverted as such using the result of theorem 9.3, and therefore a factorization of x can be computed in $\mathsf{BPP}^{R_{KT}}$. Since the resulting factorization can be checked in polynomial time for correctness, this algorithm can be reduced to a $\mathsf{ZPP}^{R_{KT}}$ algorithm. □

Thus, by the results of the reductions in section 4 and these previous theorems, RSA, Rabin, ElGamal, and Menezes-Vanstone remain susceptible to a $\mathsf{BPP}^{R_{KT}}$ adversary. From this result, if there is a significant divide between the power of the classes $\mathsf{ZPP}^{R_{KT}}$ and $\mathsf{BPP}^{R_{KT}}$ and if the result of theorem 9.5 is tight, then there might be some advantage to a public-key cryptosystem founded upon the discrete logarithm rather than one founded upon factoring.

Due to the power of the random strings R_{KT} as an oracle, we conjecture that any given public-key cryptosystem is susceptible to a $\mathsf{BPP}^{R_{KT}}$ adversary. What would complete this conjecture is the result that any encryption function e of a public-key cryptosystem has an underlying random self-reducible problem, which would amplify the result of theorem 9.3 to at least 2/3.

Conjecture 9.7. *If (P, C, K, E, D) is a public-key cryptosystem, then for every public key k, e_k is invertible in $\mathsf{BPP}^{R_{KT}}$.*

9.1 The Quantum Regime

The discussion of the hardness of factoring and the discrete logarithm has skirted the issue that there are efficient probabilistic quantum algorithms that solve these problems in the class BQP [Sho94]. Although current public-key cryptosystems remain vulnerable to an attack by a large enough quantum computer, no quantum computer has been built yet to tackle large problems. In the quantum analysis of these two fundamental problems, it has been shown that factoring and the discrete logarithm are special cases of the *hidden subgroup problem* [Joz01]. Given a group G, information about a subgroup is given through a function that separates the cosets of G. From this information, the problem is to provide a set of generators for the subgroup.

Definition 9.8. Let G be a group, H a subgroup of G, X an arbitrary set, and $f : G \to X$. f *separates the cosets of* H if for all $a, b \in G$, $f(a) = f(b)$ if and only if $aH = bH$.

The hidden subgroup problem can be defined precisely in terms of separating cosets.

Definition 9.9. Let G be a group, H a subgroup of G, X an arbitrary set, and $f : G \to X$ that separates the cosets of H. The *hidden subgroup problem* is to determine set $S \subseteq G$ whereby $H = \langle S \rangle$ with input of G, f, and X.

As factoring and the discrete logarithm take on special cases of the hidden subgroup problem, we conjecture that this problem is, too, solvable by a $\mathsf{BPP}^{R_{KT}}$ machine. A whole class of lattice problems, too, intersect both $\mathsf{BPP}^{R_{KT}}$ [ABK+06] and the hidden subgroup problem [Reg02].

Conjecture 9.10. *The hidden subgroup problem is computable in* $\mathsf{BPP}^{R_{KT}}$.

Additionally, a stronger conjecture is that all of BQP is contained in $\mathsf{BPP}^{R_{KT}}$. Note that BQP is thought to strictly contain BPP, providing support of the strength of quantum randomness over classical randomness. While the nature of quantum information and quantum randomness as a fourth type of randomness is complex, we conjecture that the power of Kolmogorov random strings are capable of giving a BPP machine with uniform randomness the ability to perform quantum computations.

Conjecture 9.11. $\mathsf{BQP} \subseteq \mathsf{BPP}^{R_{KT}}$.

A proof or counterexample to this conjecture would establish the prominence of Kolmogorov randomness over quantum randomness or vice versa, respectively.

10 Construction of a Nondeterministic One-Way Function

Using the generalized notion of a one-way function presented in section 7, a whole stratum of cryptographic primitives can be theorized. It is widely assumed that there is P-computable pseudorandomness secure against BPP. How many of these general notions of pseudorandomness —or equivalently general one-way functions—are feasible? Under hypotheses explored in [HP08], there is evidence of a NP one-way function that is secure against a class of subexponential adversaries. Using the UP-machine hypothesis, we can explicitly construct such a function. While the UP machine hypothesis implies that the polynomial hierarchy collapses into SPP [HP08], it allows for explicit construction of a UP one-way function. An approach with left-sets leads to a similar result in assuming the NP machine hypothesis. These hypotheses are introduced below.

Hypothesis 10.1 (NP machine hypothesis [HP08])**.** There exists an $\delta > 0$ and a NP machine accepting 0^* such that no $\mathsf{DTIME}(2^{n^\delta})$ machine can find infinitely many accepting computations.

Hypothesis 10.2 (UP machine hypothesis [HP08])**.** There exists an $\delta > 0$ and a UP machine accepting 0^* such that no $\mathsf{DTIME}(2^{n^\delta})$ machine can find infinitely many accepting computations.

Note that a UP machine is an NP machine that accepts on at most one branch for any given input. Therefore, this unique path, if indeterminable by a $\mathsf{DTIME}(2^{n^\delta})$ machine, becomes the output to a NP one-way function. A NP machine may halt with different outputs on different branches. Here, it is important to distinguish the two types of functions that an NP machine can simulate: single-valued and multivalued functions.

Definition 10.3. $f : \Sigma^* \to \Sigma^*$ is a *nondeterministic polynomial-time single-valued computable function* (or a *NPSV*) if there exists a nondeterministic polynomial-time Turing machine M such that for every input $x \in \Sigma^*$, M halts on input x with $f(x)$ remaining on the tape of all of the accepting branches.

Definition 10.4. $f : \Sigma^* \Rightarrow \Sigma^*$ is a *nondeterministic polynomial-time multivalued computable function* (or a *NPMV*) if there exists a nondeterministic polynomial-time Turing machine M such that for every input $x \in \Sigma^*$ and every possible output $y \in \Sigma^*$, $x\ f\ y$ if and only if M halts on input x with y remaining on the tape of one of its accepting branches. Here $\cdot f \cdot$ is the underlying relation between Σ^* and itself.

But first some helpful lemmas regarding balanced NP computation trees.

Lemma 10.5. *If $L \in \mathsf{NP}$, then there exists a machine $N \in \mathcal{M}(\mathsf{NP})$ deciding L whose computation tree is binary.*

Proof. If $L \in \mathsf{NP}$, then there is some nondeterministic polynomial-time machine deciding L. Call this machine M. Define N to simulate M. Whenever M branches into some number m subbranches, N will branch $\lg(m)$ times to produce $2^{\lg(m)} = m$ branches. Since M runs in polynomial time, it can only branch at most a polynomial number of times. Therefore N runs in polynomial time of M plus at most a polynomial in the number of branches of M. Since $\mathsf{NP} \subset \mathsf{EXP}$, each branch of M has at most an exponential number of resulting branches. N simulates this exponential number in time of the logarithm of this exponential number, which is polynomial. Hence $N \in \mathcal{M}(\mathsf{NP})$. $\square$

Lemma 10.6. *If $L \in \mathsf{NP}$, then there exists a machine $N \in \mathcal{M}(\mathsf{NP})$ deciding L whose computation tree is a full and balanced binary tree.*

Proof. By the previous lemma, there is some machine $M \in \mathcal{M}(\mathsf{NP})$ deciding L whose computation tree is binary. Define M' by simulating M. If M branches in two at a point in the computation, then so does M'. However, if M does not branch, then M' shall branch at this point, where the computation of M continues down the right branch the state of the machine down the left branch is immediately thereafter q_{reject}. Thus, the computation tree for M' is full as each node is either a leaf or has two children. M' does not run any longer than M on accepting branches, so $M' \in \mathcal{M}(\mathsf{NP})$, and it still decides L.

Next, let n^k be the polynomial bound on the run time of M' for inputs of length n. Define N to simulate M', but whenever M' halts in state q_{accept} (respectively, q_{reject}), N will enter a distinct state q'_{accept} (resp. q'_{reject}). N will continue branching twice at each time step thereafter, remaining in q'_{accept} or q'_{reject} until time step $n^k - 1$, at which N moves from state q'_{accept} to q_{accept} (resp, q'_{reject} to q_{reject}) in its final time step. N still decides L in polynomial time, and moreover the computation tree for N is full and binary. $\square$

Each accepting path in the construction of the previous lemma is precisely polynomial in the size of the input. An accepting path is constructed that is exactly of length n^k, where n is the length of the input. If an NP multivalued function output this accepting path during a simulation of a language L, then this function would be honest. This honesty condition, combined with the UP machine hypothesis allows for the construction of a NP single-valued one-way function. The security of this one-way function should be optimal since any NP one-way function cannot be secure against all of EXP as $\mathsf{NP} \subseteq \mathsf{EXP}$.

Theorem 10.7. *The* NP*-machine hypothesis implies the existence of a* NP *one-way function with security* $(\epsilon, \mathsf{DTIME}(2^{n^\delta}))$ *for some* $\delta > 0$*.*

Proof. Let M be a machine accepting 0^* in NP for which no $\mathsf{DTIME}(2^{n^\epsilon})$ deterministic machine can compute infinitely many accepting paths. Denote the language of M as L. Note that M runs in nondeterministic polynomial time with respect to the length of the input and M can be chosen where its computation tree is a full and balanced binary tree.

Using the technique of left sets [HO02], a single output can be obtained for this function. Because M necessary accepts 0^n for each $n \in \mathbb{N}$, a rightmost witness can be computed in NP time. A *rightmost witness* is an accepting path a such that all other accepting paths are lexicographically less than a.

The pseudorandom generator is constructed as follows: On input x, determine the rightmost witness of $0^{|x|}$ through the computation tree of M. Denote 0 as "left" and 1 as "right." Output the path through the computation tree. This procedure achieves polynomial honesty as remarked earlier.

By the NP machine hypothesis, for some given length N, no $\mathsf{DTIME}(2^{n^\epsilon})$ machine can find *any* accepting paths. This corresponds to no such machine inverting any of the outputs of the function. Therefore, the security parameter ϵ is eventually 0 for lengths at least N. $\square$

Corollary 10.8. *The* UP*-machine hypothesis implies the existence of a* UP *one-way function with security* $(\epsilon, \mathsf{DTIME}(2^{n^\delta}))$ *for some* $\delta > 0$*.*

There are a few features to note of the previous construction. First, the same output is used for each string of a given length. This would not be of practical use because any adversary could either invert all strings of a given length or none of them. The effect of an adversary breaking all strings of a given length would devastate the security of this function in everyday use. Second, though, is that $\epsilon(n) = 0$ for almost all n. Beyond a given length, this one-way function is perfectly secure against subexponential adversaries. While a probabilistic BPP adversary should be able to guess the input a fraction 2^{-n} of the time through uniform guessing, this does not mean that this function is necessarily secure against BPP. However, this nondeterministic one-way function may provide insight as to whether BPP is a strict superset of $\mathsf{DTIME}(2^{n^\delta})$ under the NP machine hypothesis, or if this hypothesis is not strong enough to resolve this. Yet, we conjecture that this NP one-way function does not provide any strength beyond the fundamental core power of Kolmogorov time-bounded random strings.

Conjecture 10.9. *A* NP *one-way function secure against* $\mathsf{DTIME}(2^{n^\delta})$ *is invertible by a* $\mathsf{BPP}^{R_{KT}}$ *adversary.*

11 Conclusions and Summary of Conjectures

So much of life, it seems to me,
is determined by pure randomness.
Sidney Poitier

The relationships among uniform randomness, pseudorandomness, and Kolmogorov randomness, the title cast, present unique insights into the realm of public-key cryptography and average-case hardness. The combination of uniform randomness and Kolmogorov randomness into $\mathsf{BPP}^{R_{KT}}$ was sufficient to defeat the pseudorandomness generated by factoring and the discrete logarithm, and ultimately to invert these functions. An integral component of this approach was that both factoring and the discrete logarithm are random self-reducible. This approach led to the following conjecture, that all public-key cryptography is insecure against this combination of uniform randomness and Kolmogorov randomness.

Conjecture 11.1. *If (P, C, K, E, D) is a public-key cryptosystem, then for every public key k, e_k is invertible in $\mathsf{BPP}^{R_{KT}}$.*

In the study of quantum algorithms for factoring and the discrete logarithm, these problems have been shown to be subproblems of the hidden subgroup problem. The next conjecture categorizes the hidden subgroup problem as solvable by $\mathsf{BPP}^{R_{KT}}$.

Conjecture 11.2. *The hidden subgroup problem is computable in $\mathsf{BPP}^{R_{KT}}$.*

Other types of randomness are conjectured to fall prey to such an approach. The first suggested victim is the quantum analogue of BPP, the complexity class of BQP.

Conjecture 11.3. $\mathsf{BQP} \subseteq \mathsf{BPP}^{R_{KT}}$.

Finally, the NP machine hypothesis provided a foundation for an NP one-way function. Should such a function exist, its pseudorandomness, generated from the construction in [HILL99], is also conjectured to be distinguishable by a $\mathsf{BPP}^{R_{KT}}$ machine.

Conjecture 11.4. *A NP one-way function secure against $\mathsf{DTIME}(2^{n^\delta})$ is invertible by a $\mathsf{BPP}^{R_{KT}}$ adversary.*

Altogether, these conjectures surmise that $\mathsf{BPP}^{R_{KT}}$ is the upper bound on *all* forms of efficient randomness–whether probabilistic polynomial quantum algorithms or nondeterministically generated pseudorandomness.

A method omitted in this paper for security analysis of a cryptosystem is a measure-theoretic approach. This approach has its roots in game theory and *martingales*. An adversary is given initial capital and places bets on succeeding outputs of the characteristic sequence of a language L when given the time to analyze all previous outputs [Lut97]. Correct guesses earn the adversary even money. Success on the language L is achieved if, in the long run, the adversary can attain infinite capital. Adversaries are bounded in their time or space usage. One popular hypothesis is the measure hypothesis, which states that NP does not have p-measure 0 [HP08]. Informally, any polynomial-time martingale cannot succeed on all languages in NP. This is stronger than $\mathsf{P} \neq \mathsf{NP}$, and directly implies the NP-machine hypothesis. Martingales may prove indispensable in a cryptographer's toolbox.

Finally, the hardness of cryptographic one-way functions has not been entirely understood in terms of NP-complete sets. Factoring and the discrete logarithm are believed to be NP-intermediate problems; that is, they are likely not in P and not NP-complete. Whether an NP-complete problem can be used to formulate a secure cryptographic primitive remains the statements of the encrypted complete set conjecture [BT98]. For now, it is unknown if NP-complete problems, like the satisfiability problem in NP, are too hard to be used in practical cryptosystems.

The power of randomness shall not be understated. Strong pseudorandomness provides for cryptographic primitives whose security is understood in terms of uniform randomness and Kolmogorov randomness. While critical results of cryptography—namely the existence of secure public-key cryptosystems—have not yet been proven, one result is clear: the relationships among the types of randomness will become the touchstone of computational bounds and security.

References

[ABK$^+$06] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006.

[AJO08] Gildas Avoine, Pascal Junod, and Philippe Oechslin. Characterization and improvement of time-memory trade-off based on perfect tables. *ACM Trans. Inf. Syst. Secur.*, 11(4):1–22, 2008.

[AKS02] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Ann. of Math*, 2:781–793, 2002.

[BM95] H. Buhrman and E. Mayordomo. An excursion to the kolmogorov random strings. In *SCT '95: Proceedings of the 10th Annual Structure in Complexity Theory Conference (SCT'95)*, page 197, Washington, DC, USA, 1995. IEEE Computer Society.

[BT98] Harry Buhrman and Leen Torenvliet. Complete sets and structure in subrecursive classes. In *In Proceedings of Logic Colloquium '96*, pages 45–78. Springer-Verlag, 1998.

[Cha66] Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *J. ACM*, 13(4):547–569, 1966.

[Cha69] Gregory J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *J. ACM*, 16(1):145–159, 1969.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Gen07] Giulio Genovese. Improving the algorithms of berlekamp and niederreiter for factoring polynomials over finite fields. *J. Symb. Comput.*, 42(1-2):159–177, 2007.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32, New York, NY, USA, 1989. ACM.

[Gol00] Oded Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, New York, NY, USA, 2000.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 1 edition, April 2008.

[Har83] Juris Hartmanis. Generalized kolmogorov complexity and the structure of feasible computations. In *SFCS '83: Proceedings of the 24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 439–445, Washington, DC, USA, 1983. IEEE Computer Society.

[HILL99] Johan Hastad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.

[HO02] Lane A. Hemaspaandra and Mitsunori Ogihara. *The Complexity Theory Companion.* Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[HP08] John M. Hitchcock and A. Pavan. Hardness hypotheses, derandomization, and circuit complexity. *Computational Complexity*, 17(1):119–146, April 2008.

[Joz01] Richard Jozsa. Quantum factoring, discrete logarithms, and the hidden subgroup problem. *Computing in Science and Engg.*, 3(2):34–43, 2001.

[KL94] Erich Kaltofen and Austin Lobo. Factoring high-degree polynomials by the black box berlekamp algorithm. In *ISSAC '94: Proceedings of the international symposium on Symbolic and algebraic computation*, pages 90–98, New York, NY, USA, 1994. ACM.

[Kob87] Neil Koblitz. Elliptic curve cryptosystems. *Math. Computation*, 48:203–209, 1987.

[Kol68] Andrey N. Kolmogorov. Logical basis for information theory and probability theory. *Information Theory, IEEE Transactions on*, 14(5):662–664, 1968.

[Lev84] Leonid A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Inf. Control*, 61(1):15–37, 1984.

[LL93] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The development of the number field sieve.* Number 1554 in Lecture Notes in Mathematics. Springer-Verlag, 1993.

[Lut97] Jack H. Lutz. The quantitative structure of exponential time. pages 225–260, 1997.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In *CRYPTO '85: Advances in Cryptology*, pages 417–426, London, UK, 1986. Springer-Verlag.

[MV93] Alfred J. Menezes and Scott A. Vanstone. Elliptic curve cryptosystems and their implementation. *Journal of Cryptology*, 6(4):209–224, September 1993.

[PH78] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.

[Pol74] John M. Pollard. Theorems on factorization and primality testing. *Proceedings of the Cambridge Philosophical Society*, 76:521–528, 1974.

[Rab79] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Cambridge, MA, USA, 1979.

[Reg02] Oded Regev. Quantum computation and lattice problems. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 520–529, Washington, DC, USA, 2002. IEEE Computer Society.

[Rie85] Hans Riesel. *Prime numbers and computer methods for factorization.* Birkhauser Boston Inc., Cambridge, MA, USA, 1985.

[Ron04] Detlef Ronneburger. *Kolmogorov complexity and derandomization.* PhD thesis, New Brunswick, NJ, USA, 2004. Director-Allender,, Eric W.

[RR94] Alexander A. Razborov and Steven Rudich. Natural proofs. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 204–213, New York, NY, USA, 1994. ACM.

[RSA83] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math. 20*, pages 415–440, 1971.

[Sho94] Peter W. Shor. Polynominal time algorithms for discrete logarithms and factoring on a quantum computer. In *ANTS-I: Proceedings of the First International Symposium on Algorithmic Number Theory*, page 289, London, UK, 1994. Springer-Verlag.

[Sho98] Victor Shoup. Subquadratic-time factoring of polynomials over finite fields. *Math. Comp*, 67:398–406, 1998.

[Sol64] Ray J. Solomonoff. A formal theory of inductive inference, parts i and ii. *Information and Control*, 7:1–22, 1964.

[Wil82] H. C. Williams. A $p + 1$ Method of Factoring. *Math. Comput.*, 39:225–234, 1982.

[Yao82] Andrew C. Yao. Theory and application of trapdoor functions. In *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Washington, DC, USA, 1982. IEEE Computer Society.