Applications of Expander Graphs in Cryptography

Aleksander Maricq

May 16, 2014

Abstract

Cryptographic hash algorithms form an important part of information security. Over time, advances in technology lead to better, more efficient attacks on these algorithms. Therefore, it is beneficial to explore either improvements on existing algorithms or novel methods of hash algorithm construction. In this paper, we explore the creation of cryptographic hash algorithms from families of expander graphs. We begin by introducing the requisite background information in the subjects of graph theory and cryptography. We then define what an expander graph is, and explore properties held by this type of graph. We conclude this paper by discussing the construction of hash functions from walks on expander graphs, giving examples of expander hash functions found in literature, and covering general cryptanalytic attacks on these algorithms.

Contents

List of Figures

1 Introduction

How does one keep a secret? The obvious answer to this question is to only tell the recipient, but what if that secret needs to be transmitted in the presence of a third party? How can one ensure that the intended recipient knows the secret while simultaneously preventing anyone unintended from discovering it? The field of Cryptography attempts to provide answers to the previous question by making interception of a secret message essentially a non-issue. A message is transformed via a process known as encryption, making the result seemingly completely unrelated to the original message, and is then transmitted to be decrypted by the intended recipient. Thus, all any third party, or adversary, would see is the encrypted message, which is worthless to them unless they happened to know the decryption technique.

Consider the following scenario. France decides it is tired of Switzerland's neutrality and goes to war with them. Swiss command wants to be able to transmit messages to the front lines without the French being able to see the content of the messages. The Swiss, therefore, encrypt their messages before transmitting them to the front lines, so that even if those messages were intercepted, they'd have no way of knowing what the messages said without a way of decrypting them. That way, the Swiss military can transmit attack plans, troop movements, and so on, without France knowing about them. This general wartime scenario has been repeated in various forms throughout history. One of the earliest instances of encryption, for instance, was the Caesar cipher, or shift cipher, used by Julius Caesar to send messages to his generals.

With every encryption algorithm, however, there comes somebody looking to break it. If an adversary continuously intercepts messages, the encryption can be broken given enough time. The time it takes is prohibitively long for most generic attacks on encryptions, so cleverer and more specific

attacks are required to bring the time requirement down to acceptable levels. The study of breaking encryption is known as cryptanalysis, and there is a constant struggle between cryptanalysis and cryptography. When a new encryption technique is developed by cryptologists, cryptanalysts will attempt to break it by brainstorming attacks against it. In response to those attacks, the encryption technique is strengthened or a new encryption technique altogether is invented, which prompts the cryptanalysts to figure out new and better attacks, and so on and so forth.

To revisit our wartime example, while Switzerland is encrypting its messages so that France can't read them, France is almost certainly intercepting Switzerland's messages, and is using a team of mathematicians to attempt to decrypt them. Perhaps one of the most famous examples of this in history was during World War II, with the British attempt to decrypt German transmissions that had been encrypted using both Enigma and Lorenz machines. Great Britain employed numerous cryptologists and mathematicians at Bletchley Park to decode German transmissions, and it worked. The efforts of the men and women at Bletchley Park are considered one of the most important contributions to the war effort, and have been credited with an earlier end to the war.

Nowadays, message encryption is still very important for military and wartime purposes, but with the advent and explosive growth of the internet, encryption has become necessary for essentially anybody who uses the internet. For instance, people who check their finances online, use a credit or debit card to purchase things, or submit sensitive personal information over the web benefit from encryption as it allows them to avoid such problems as identity theft and fraud. Cryptography, therefore, has become more important and widespread than ever before. With advances in computers, the development of advanced encryption methods has become a lot easier. However, these same technologic advances can be employed by cryptanaly-

sis to develop more advanced attacks on these methods. Thus, the struggle between cryptography and cryptanalysis continues, and cryptography must always be changing and improving.

This leads to the topic of this paper: We aim to explore the use of a specific class of graph in the field of graph theory to develop new encryption techniques. If these techniques have advantages over more commonly used techniques, then more research and development of that area may lead to better encryption techniques. We will start, then, by introducing requisite topics in graph theory, cryptography, and cryptanalysis. From there, we will introduce expander graphs, discuss expansion and properties of graphs that affect it, and introduce the Ramanujan class of graphs. Finally, we will explore the use of expander graphs to construct cryptographic hash functions, including a general construction method, published examples, and general cryptanalytic attacks on expander hashes.

2 Fundamental Graph Theory

If we want to introduce the concept of an expander graph, we must first talk about the field of graph theory. Graph theory has been around at least since the days of Leonhard Euler, who is considered to have laid the foundation of graph theory with his famous paper concerning the problem of the seven bridges of Königsberg. It has since grown into a mathematical field with many important applications in fields such as computer science, physics, chemistry, and even linguistics.

This section will introduce important concepts in graph theory that will lay the foundation for introducing expander graphs later on in the paper. Section 2.1 provides a general introduction to graph theory, where the definition of a graph, specific graph types, and other basic concepts in graph theory are discussed. Section 2.2 introduces a specific type of graph called a Cayley graph, which combines the fields of graph theory and group theory.

2.1 Basic Concepts

A **graph** in its most abstract form is a set of objects and a set of connections between some pairs of those objects. We give a more formal definition below:

Definition 2.1.1. A **graph** is an ordered pair $G = (V, E)$ where V is a set of vertices and E is a set of subsets $e \subset V$ of cardinality 1 or 2.

There are many different categories of graphs. We will define some of the more common categories below and provide visual examples in figure 1:

A **directed graph** $G = (V, E)$ is a graph in which the elements of E are ordered pairs, and an **undirected graph** is a graph in which the elements

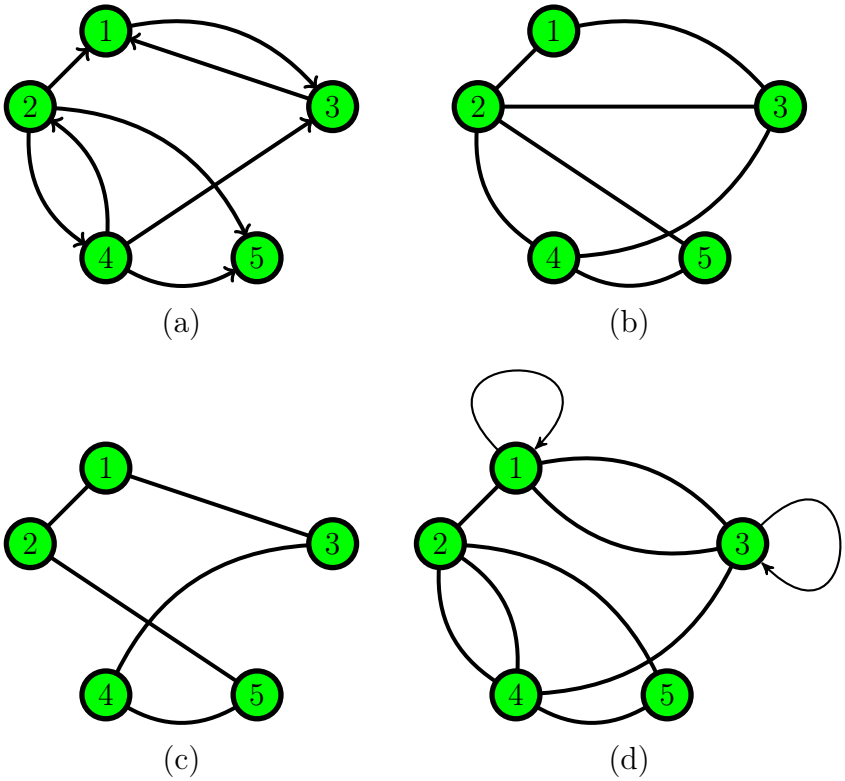

Figure 1: (a) Directed Graph. (b) Simple Graph. (c) k-Regular Graph ($k = 2$). (d) Multigraph.

of E are non-ordered pairs. Among undirected graphs, a graph $G = (V, E)$ is **simple** if there exist one or fewer edges between any two vertex pairs and if there exist no edges connecting a vertex to itself. A **multigraph** allows multiple edges, and while some definitions allow loops, ours will not allow loops in multigraphs.

Another concept that will be important is regularity in graphs:

Definition 2.1.2. A graph $G = (V, E)$ is **k-regular** if every vertex in V is contained in exactly k edges, excluding loops, in E. Furthermore, a directed graph must satisfy the additional property that the number of edges leaving

each vertex must equal the number of edges approaching each vertex.

The number of edges in E that a vertex $v_n \in V$ is contained in is also known as the **degree** of v_n. Additionally, we will say a graph has degree k if it is k-regular. A simple k-regular graph with maximum connectivity has every vertex connected to $n-1$ other vertices, and is called the **Complete Graph** on n vertices, denoted by K_n.

While there are many different types of graphs, some will certainly be more useful to us than others in the coming sections. For instance, the use of k-regular graphs ends up simplifying an important relation in expander graph theory. Unless stated otherwise, we will be working with k-regular, undirected multigraphs in our introduction to expander graphs. Additionally, we will usually be working with **families of graphs** rather than individual graphs, where a family of graphs is simply a set of graphs that share one or more common characteristics.

It is also important to mention terms dealing with the connectivity of graphs. Informally, a graph with many edges is **dense**, and a graph with relatively few edges is **sparse**. A more formal definition follows, which also extends the notion of the density of edges to families of graphs:

Definition 2.1.3. Consider a sequence of graphs $\{G_i\}_{i\in\mathbb{N}} = \{(V_i, E_i)\}_{i\in\mathbb{N}}$ of number of vertices increasing with i. Then

- If $|E_i| = \Theta(|V_i|^2)$ for all i, then $\{G_i\}_{i\in\mathbb{N}}$ is a family of dense graphs.
- If $|E_i| = \Theta(|V_i|)$ for all i, then $\{G_i\}_{i\in\mathbb{N}}$ is a family of sparse graphs.

Where Θ represents the asymptotic behavior of $|E_i|$. [Pre99]

In simpler terms, the number of edges in a sparse graph should be on the order of the number of vertices in the graph, and the number of edges in

a dense graph should be on the order of the number of vertices in the graph squared.

Now that we've introduced graphs and their general structure, we can introduce the idea of moving between vertices on a graph. A **walk** on a graph is an alternating sequence of vertices and edges, beginning and ending with a vertex. Additionally, each edge in the walk must directly connect the preceding and following vertices in the sequence. A walk is closed if the beginning and ending vertices in the sequence are the same, and open if they are not.

The **diameter** of a graph G is defined as the maximum of the set of lengths of the shortest-length walks between every pair of vertices in the graph. A similar property of the graph is its **girth**. The girth of a graph $G(V, E)$ is simply the length of a shortest cycle in the graph. These three concepts will be important in our later discussion of expander hashes.

2.2 Cayley Graphs

A Cayley graph is a graph that contains the structure of a group. For a group $\mathcal{G}$ and a subset S of that group, the **Cayley graph** $C_{\mathcal{G},S}$ is constructed as follows: V contains a vertex v_g associated with each element $g \in \mathcal{G}$, and E contains the directed edge (v_{g_1}, v_{g_2}) if and only if there is an $s \in S$ such that $g_2 = g_1 \circ s$ [Pet09].

Certain aspects of the set S of $\mathcal{G}$ determine certain properties that $C_{\mathcal{G},S}$ will hold:

- The graph is connected if and only if the elements of S generate the whole group.

- The graph is k-regular when $|S| = k$.
- The graph is undirected when S is symmetric, where a set is symmetric if $s \in S \iff s^{-1} \in S$.

Regardless of the properties of S, the elements of S are referred to as the **graph generators**. Cayley graphs are also **vertex transitive** as for any $g_1, g_2 \in \mathcal{G}$ the mapping $v_x \to v_{g_2 g_1^{-1} x}$ is a graph automorphism that sends g_1 to g_2 [Pet09].

Cayley graphs also hold interesting properties with regards to their diameter. We find that for Cayley graphs of finite non-Abelian groups, the diameter is often small. In fact, for finite simple non-Abelian groups, this follows from Babai's conjecture [Pet09]:

Conjecture 2.2.1. (Babai) *There exists a constant c such that for any non-abelian finite simple group $\mathcal{G}$, for any symmetric set $S \subset \mathcal{G}$ that generates G,*

$$D(C_{\mathcal{G},S}) < (\log |G|)^c.$$

3 Fundamental Cryptography

Before we talk about expander graphs, we will shift our focus over to the field of cryptography to introduce hash functions. This is necessary in order to provide some background for our later discussions of hash functions from expander graphs. Hash functions form a very important part of the field of cryptography, and there are many applications in pseudorandom number generators, file verification, password verification, and they form a part of many cryptographic authentication systems.

This section is constructed as follows. Section 3.1 will provide a basic introduction to hash functions, as well as some basic requirements for a hash function to be cryptographically secure. Section 3.2 will give an overview of computational security in the context of cryptographic algorithms. Section 3.3 will introduce the Merkle-Damgård construction for hash algorithms. Section 3.4 will discuss general attacks on cryptographic hash functions, as well as some attacks designed for iterated hash functions such as those utilizing the Merkle-Damgård construction. Sections 3.1-3.4 were derived from Christophe Petit's dissertation on graph-based cryptographic hash functions [Pet09]. Finally, section 3.5 gives examples of the construction of two hash functions, MD5 and SHA1, utilizing the Merkle-Damgård construction.

3.1 Hash Functions

A **hash function** is a function that maps messages of large, arbitrary size to hash values of small, constant size:

$$H : \{0,1\}^* \to \{0,1\}^\lambda.$$

Often times, especially when considering security definitions, we will be interested in not just single hash functions, but in **families** of hash functions $\{H_n\}_{n\in\mathbb{N}}$ parametrized by a **security parameter** n:

$$H_n : \{0,1\}^* \to \{0,1\}^{\lambda(n)}$$

for some function $\lambda(n)$. Additionally, it is useful to define a family of **keyed** hash functions:

$$H_n : \{0,1\}^{\kappa(n)} \times \{0,1\}^* \to \{0,1\}^{\lambda(n)}$$

for some functions $\kappa(n)$ and $\lambda(n)$. If we are dealing with fixed-length input messages, then the hash function is known as a **fixed-length** hash function:

$$H_n : \{0,1\}^{\kappa(n)} \times \{0,1\}^{\mu(n)} \to \{0,1\}^{\lambda(n)}$$

for some functions $\kappa(n)$, $\mu(n)$, and $\lambda(n)$. While any number of hash functions all share the common characteristics outlined above, they may differ greatly both in terms of the function used and their construction. There exist many possible hash functions, not all of them feasible: they may range from easy to construct but not useful, to very useful in theory but hard or practically impossible to build. For the sake of cryptography, we are interested in only hash functions that are easy to construct yet cryptographically useful. That is, we want a hash function where it is easy to compute the hash value from any given message yet is also cryptographically secure. The most common security requirements for hash functions are as follows:

- **Preimage resistance:** For essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any pre image m' such that $H(m') = h$ for which a corresponding input is not known.

- **Second preimage resistance:** It is computationally infeasible to find any second input which has the same output as any specified input, i.e., given m, to find a second preimage $m' \neq m$ such that $H(m) = H(m')$.

- **Collision resistance:** It is computationally infeasible to find any two distinct inputs m, m' which hash to the same output, i.e., such that $H(m) = H(m')$.

Roughly, collision resistance implies preimage and second preimage resistance, but the converse is not true. More rigorous explorations of these concepts can be found in section 2.2 of [Pet09].

3.2 Computational Security

We have now defined our system of security requirements, but what does it mean for something to be "computationally infeasible"? To illustrate what it means for something to be computationally infeasible or hard, consider a collision-resistant hash function based on the definition outlined above. Consider two algorithms that attempt to find collisions:

- **Algorithm** A_1 constructs a database of couples $(m_i, H(m_i))$. Until it finds a collision, A_1 picks a random message m_i, computes $h_i = H(m_i)$, checks the database for a previous occurrence of h_i (in which case it has found a collision). If h_i has not yet appeared it stores the new couple (m_i, h_i) in the database, otherwise it returns the collision found.

- **Algorithm** A_2 picks two random messages m and m'; it returns (m, m') if $H(m) = H(m')$ and returns $\perp$ otherwise, where $\perp$ indicates that the two messages don't hash to the same value.

Algorithm A_1 always finds collisions after $2^\lambda + 1$ hash computations, and about $2^{\lambda/2}$ in mean, and Algorithm A_2 produces collisions with a probability at least $1/2^\lambda$. We can see that once λ becomes sufficiently large, these algorithms no longer become feasible. In particular, Algorithm A_1 would require a long time and prohibitively huge memory to execute, and the probability of Algorithm A_2 succeeding becomes so low as to become negligible in practice.

As we have alluded to above, we tend to consider our cryptographic definitions within the scope of **computational security** since unconditional cryptographic security cannot be practically achieved. The above two algorithmic examples reflect two main approaches for computational security, which we will identify and formalize:

- **Concrete approach:** Some protocol will be (ϵ, t, m)-secure in some sense if any algorithm running in time less than t and using a memory smaller than m succeeds in some task with probability smaller than ϵ.
- **Asymptotic approach:** The hash family $\{H_n\}$ is secure if each H_n is $(\epsilon(n), t(n), m(n))$-secure, the functions $t(n)$ and $m(n)$ do not grow too fast with n, and the function $\epsilon(n)$ decreases fast enough with n.

To further explore concepts in cryptographic hash functions, it is useful to define the following:

- An algorithm A is **efficient** or **probabilistic polynomial time** or **PPT** if it can be solved in polynomial time by a probabilistic Turing machine, or in other words, if there exists a polynomial p such that for every input $x \in \{0,1\}^*$, the computation of $A(x)$ terminates within at most $p(|x|)$ steps, where x denotes the length of the string x.
- A function f is **negligible** if for every polynomial p there exists an N such that for all integers $n \geq N$ it holds that $f(n) < \frac{1}{p(n)}$.

- A function f is **noticeable** if there exists a polynomial p such that for all sufficiently large integers n it holds that $f(n) > \frac{1}{p(n)}$.

It is important to note that a PPT algorithm cannot use more than a polynomial amount of memory, and that the notions of negligible and noticeable functions are **strong negations** to one another. That is, there exist functions that are neither noticeable nor negligible. The prior definitions allow us to introduce the general form of an asymptotic security definition:

> The cryptographic scheme X is secure in the sense Y if for any PPT algorithm given an input of size n, there exists a negligible function $\epsilon(n)$ such that the algorithm succeeds with probability smaller than $\epsilon(n)$ in performing some task Z.

It ends up that many theorems in cryptography have a similar form:

> If the computational assumption X holds, then the cryptographic scheme Y is secure in the computational sense Z.

The consequence of having cryptographic theorems in that form is that it reduces the question of whether Y is secure down to whether or not X is true. The following are widely-used examples of computational assumptions in cryptography:

- **Integer Factorization Problem**: given a large integer $n = p \cdot q$ where p, q are prime, it is computationally hard to find p and q.
- **Discrete Logarithm Problem:** given a prime p, an element g of $\mathbb{F}_p$ with large prime order, and the element $g^k \bmod p$ for some randomly chosen k, it is computationally hard to return the k value.
- **Elliptic Curve Discrete Logarithm Problem:** given an elliptic curve E defined over a prime field $\mathbb{F}_p$, a rational point $P \in E$ with

some large order, and the point $Q = k \cdot P$ for some randomly chosen k, it is computationally hard to return the k value..

Many cryptographic algorithms base their security on the fact that no algorithms exist to solve these problems in any feasible amount of time.

3.3 The Merkle-Damgård transform

Typically, cryptographic hash functions are built upon two main components. The **compression function** hashes messages of a fixed size to hash values of fixed, smaller size. The **domain-extension transform** uses the compression function as a building block to construct hash functions with arbitrary-length inputs.

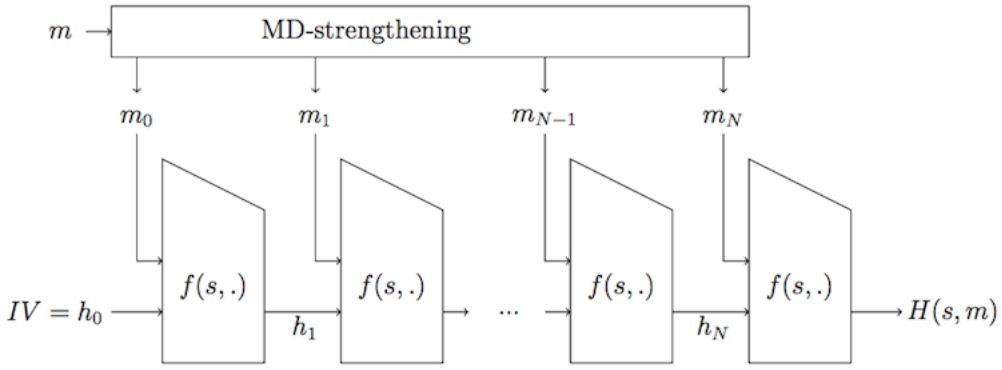

Figure 2: Merkle-Damgård transform [Pet09]

The **Merkle-Damgård transform**, as seen in figure 2, is a common example of a domain-extension transform. It is utilized in a variety of families of hash functions, most notably in the SHA and MD families. Consider a collision-resistant compression function f that takes as input a key s and a message of size $\mu + \lambda$, and returns a bitstring of size λ. The Merkle-Damgård transform of f takes as input a key s and a message m of length L smaller than 2^λ, and returns a bitstring of size λ.

For the sake of collision-resistance, we use the **MD-strengthening** process on our message m. This process produces $N+1$ bitstrings $m_0, \ldots, m_N$ of size μ, where $N = \lceil L/\mu \rceil$. The message m is first decomposed into N blocks of μ consecutive bits. If L is not a multiple of μ, the last block is completed with zeroes. An additional block is constructed that contains a binary representation of L on λ bits.

Let $h_0 = IV$ be some fixed **initial value**. The Merkle-Damgård transform of f is defined as $H_f(s, m) = h_{N+1}$, where $h_i = f(s, h_{i-1} \| m_{i-1})$ and $x \| y$ denotes the concatenation of x and y. The Merkle-Damgård transform satisfies the following property:

Theorem 3.3.1. (Merkle-Damgård) *If (Gen, f) is a fixed-length collision-resistant hash function, then (Gen, H) is a collision-resistant hash function.*

Proof. Suppose an adversary finds $m \neq m'$ such that $H_f(s, m) = H_f(s, m')$. Write m_i and m'_i for the output blocks of the MD-strengthening of m and m', and h_i and h'_i for the intermediate values of the computation of $H_f(s, m)$ and $H_f(s, m')$. Then there exists $i \leq n+1$ such that $h_i = h'_i$ but $h_{i-1} \| m_{i-1} \neq h'_{i-1} \| m'_{i-1}$, so the adversary has found a collision $\left(h_{i-1} \| m_{i-1}, h'_{i-1} \| m'_{i-1}\right)$ on the compression function. □

The Merkle-Damgård transform is a domain-extending transform that preserves collision resistance: it transforms a fixed-length collision-resistant hash function into an arbitrary-length collision-resistant hash function. It is important to note, however, that it does not preserve other properties such as preimage resistance and second preimage resistance. For hash functions utilizing the Merkle-Damgård transform, those properties will need to be obtained from other aspects of their construction.

3.4 Cryptanalysis

An attack on a cryptographic protocol is a proof that this protocol does not satisfy its claimed security properties. Attacks on hash functions tend to target the collision and preimage-resistance properties. In the asymptotic setting, a theoretical attack against the collision resistance of a hash function is a PPT algorithm that finds collisions for asymptotically large values of the security parameter. In practice, however, many hash functions are only defined for a finite small set of values of the security parameter.

The feasibility of an attack is largely dependent on the resources the attacker is willing to invest in an attack. The feasibility is usually estimated in terms of time and memory complexities, to which we add the lengths of hashed messages. By the standards of 2014, attacks running in time greater than approximately 2^{80} are considered infeasible (per DES standards), and attacks with memory requirements greater than approximately 2^{60} (approximately an Exabyte) or 2^{70} (approximately a Zettabyte) are considered infeasible due to storage costs. That is, if a Terabyte hard drive is \$50, then to get an Exabyte worth of storage, $\$50,000,000$ is required.

Since the codomain of any concrete hash function is a finite set, there exist unavoidable **generic** attacks that can only be mitigated by fixing large enough parameters. In practice, a preimage or collision attack is often considered successful if it computes preimage or collisions faster than generic attacks. For iterated hash functions, such as those based around the Merkle-Damgård transform, more efficient attacks exist. We will use this section to identify and define both generic attacks and attacks on iterated hash functions.

3.4.1 Generic Attacks

Brute force attack. Given the value h of a randomly chosen message, an adversary can find a preimage m such that $H(s, m) = h$ (where the key is known by the adversary) by trying successive values $m = 1, 2, \ldots$ until he finds a preimage. If the output is of size λ, then the attack is expected to succeed in time 2^λ.

Birthday attack. This attack is based on the birthday paradox, which is concerned with the probability that two people have the same birthday given a set of n people. We know by the pigeonhole principle that when $n = 366$ (we are excluding February 29^{th}), the probability that two people have the same birthday is 100%. Surprisingly, we find that when $n = 23$ this probability is approximately 50.73%, and when $n = 57$ this probability is approximately 99.01%.

Similarly, if the codomain of a hash function is of size 2^λ, an adversary can find collisions after $2^{\lambda/2}$ hash computations on random messages. Let $N = 2^\lambda$ be the number of output values. The probability of finding collisions after $N' = 2^{\lambda/2}$ random trials is

$$\begin{aligned} P[\text{col}] &= 1 - \prod_{i=0}^{N'-1} \frac{N-i}{N} \\ &= 1 - \prod_{i=0}^{N'-1} 1 - \frac{i}{N}. \end{aligned}$$

From Taylor's first-order approximation $e^x \approx 1 + x$ we obtain

$$P[\text{col}] \approx 1 - e^{-\sum_{i=1}^{N'-1} i/N} \approx 1 - e^{\frac{N'^2}{2N}} = 1 - \frac{1}{\sqrt{e}} \approx 0.39.$$

The birthday attack requires time and memory $2^{\lambda/2}$. However, by translating the collision problem to the problem of detecting cycles in an iterative mapping, the memory requirement becomes negligible while the time requirement remains essentially the same. In this modified birthday attack, instead of choosing the messages randomly, the adversary chooses them deterministically according to the previous hash value. This induces a deterministic mapping on a finite set that will eventually repeat, resulting in cycles. The main advantage of this approach is that there is no need to store all of the hash values.

3.4.2 Attacks on Iterated Hash Functions

Meet-in-the-middle attack. If the compression function is invertible, pre-images can be computed in time approximately $2^{\lambda/2}$ by extending the birthday attack as follows: Apply the compression function to $2^{\lambda/2}$ random messages and apply it backward to $2^{\lambda/2}$ other random messages. By the birthday paradox, there is a large probability that the adversary finds a common value "In the middle". This attack also has a memory-free version, but it is not feasible if the compression function is preimage resistant.

Fixed point attack. This attack looks for an intermediate value h_{i-1} and a message block m_i such that $f(s, h_{i-1}\|m_i) = h_i$. The attack allows inserting any number of blocks m_i without changing the hash value. In a Merkle-Damgård construction, it becomes very practical if the initial value can be selected by the adversary.

Multicollision attack. The birthday attack allows a collision to be found to a compression function in time $2^{\lambda/2}$. Repeating the collision search $\lambda/2$

times, an adversary can find message blocks $m_1, m'_1, m_2, m'_2, \ldots, m_{\frac{\lambda}{2}}, m'_{\frac{\lambda}{2}-1}$ such that

$$\begin{aligned} h_1 &:= f(s, h_0 \| m_1) = f(s, h_0 \| m'_1), \\ h_2 &:= f(s, h_1 \| m_2) = f(s, h_1 \| m'_2), \\ &\ldots \\ h_{\frac{\lambda}{2}-1} &:= f(s, h_{\frac{\lambda}{2}-1} \| m_{\frac{\lambda}{2}}) = f(s, h_{\frac{\lambda}{2}-1} \| m'_{\frac{\lambda}{2}}). \end{aligned}$$

These message blocks can be combined into $2^{\lambda/2}$ messages of $\lambda/2$ blocks that hash to the same value. Finding these multi collisions hence requires time only $\frac{\lambda}{2}2^{\lambda/2}$ while on an ideal function it would require a time $2^{\lambda(2^{\lambda/2}-1)/2^{\lambda/2}} \approx 2^{\lambda}$.

This observation has been used to improve the birthday attack on a class of hash functions. Suppose G and H are two hash functions with ideal collision-resistance (the best attack has expected time $2^{\lambda/2}$). If G and H were ideal, the function $F(s, m) := G(s, m) \| H(s, m)$ would have ideal collision-resistance (the best attack has expected time 2^{λ}). However, if G is an iterated hash function, an adversary can construct $2^{\lambda/2}$ collisions for G in time $\frac{\lambda}{2}2^{\lambda/2}$. By the birthday paradox, these $2^{\lambda/2}$ messages are likely to give one collision for H, hence for F.

3.4.3 Differential Cryptanalysis

The basic premise of differential cryptanalysis is that a small difference in only one of the input variables can be controlled in such a way that the differences occurring in the computations of the two associated hash values are compensated for at the end. The goal is to find good **differential paths** through the whole algorithm computation.

Differential cryptanalysis is mostly applied on Merkle-Damgård based, unkeyed hash functions to find collisions of the form $f(s, h\|m) = f(s, h\|m')$ on the compression function. Since the targeted functions are unkeyed, the key s can be considered to be fixed. The value h is considered as random since in a Merkle-Damgård construction it is the output of the compression function from the previous round.

The compression functions of recent hash algorithms have many rounds that improve the bit interdependencies through non-linear functions. Thus, it is no longer possible to find a full differential path resulting in a collision for the compression function with probability 1. However, the attacker can search in all stages of the algorithm for particular differences in the input bits that with a large probability on the value h will result in small differences a few stages later. Combining these **differentials** to cancel differences may lead to a collision at the end of the hash computation with some probability. When the attack is repeated, it is likely to succeed after a time inversely proportional to this probability.

Near-collisions are also targeted by differential cryptanalysis, in which case a full collision is obtained by iterating the attack, to produce a pair of colliding messages whose lengths are a few blocks long.

3.5 Examples of Hash Functions

This section will describe two hash functions, MD5 and SHA-1, utilizing Merkle-Damgård construction. These hash functions in particular are both extensions of the MD4 algorithm and are, or were, commonly used in such applications as TLS, SSL, PGP, and SSH. While MD5 in particular is no longer secure, we include it in this section as a relatively simple example.

3.5.1 MD5

The following description of MD5 is taken from the Internet Enginering Task Force (IETF) Request For Comments (RFC) 1321 [Riv92].

Begin by assuming a b-bit message as input, where $b < 2^{64}$ is a non-negative arbitrary-size integer. The following five steps are performed to compute the message digest of the message.

Padding. First, the message is padded so that is length in bits is congruent to 448 mod 512: A single 1 bit is appended to the message, and then 0 bits are appended until the length of the padded message is congruent to 448 mod 512. The message is padded in this way so we are 64 bits away from being a multiple of 512.

A 64-bit representation of b is appended to the padded message from above. The resulting message will have a length that is an exact multiple of 512 bits, or equivalently, an exact multiple of sixteen 32-bit words. We can represent the resulting message as a series of 512-bit blocks, which we denote as $M[0 \ldots n]$ for some integer n where $M[i]$ is a 512-bit block.

Functions and Constants. We define the following functions that each take as input three 32-bit words and produce one 32-bit word as output:

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z),$$

$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z),$$

$$H(X, Y, Z) = X \oplus Y \oplus Z,$$

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z),$$

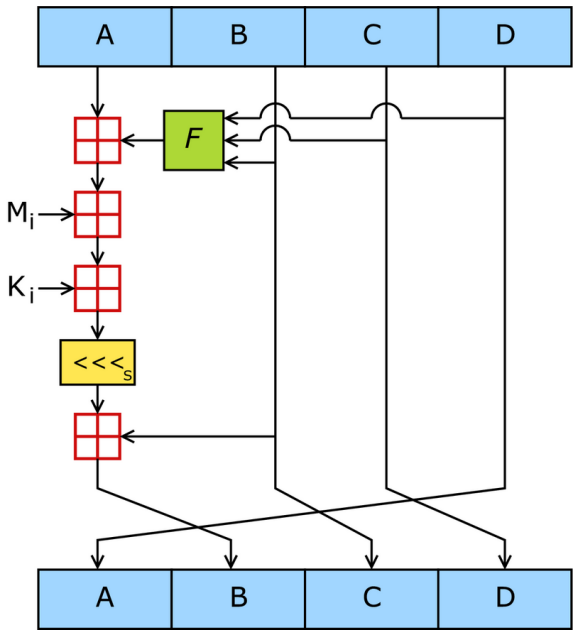

Figure 3: One round of MD5. F is one of the functions (not necessarily $F(X, Y, Z)$ itself) defined in our functions section, M_i is $M[i]$, K_i is $T[i]$, $<<<_n$ denotes the circular left shift by n positions, and $\boxplus$ denotes addition modulo 2^{32}.

Source: `http://en.wikipedia.org/wiki/File:MD5.svg`

where the operations in the above functions are bitwise logical operations on the 32-bit words.

Additionally, we will be using a 64-element table $T[0 \ldots 63]$ which is constructed as follows: The integer part of $2^{32}|\sin(i+1)|$ is the i-th element of the table T, where i is in radians.

We will also use another 64-element table $X[0 \ldots 63]$ which contains elements:

$$
\begin{aligned}
X[t] &= 7, && (t = 0, 4, 8, 12),\\
X[t] &= 12, && (t = 1, 5, 9, 13),\\
X[t] &= 17, && (t = 2, 6, 10, 14),\\
X[t] &= 22, && (t = 3, 7, 11, 15),\\
X[t] &= 5, && (t = 16, 20, 24, 28),\\
X[t] &= 9, && (t = 17, 21, 25, 29),\\
X[t] &= 14, && (t = 18, 22, 26, 30),\\
X[t] &= 20, && (t = 19, 23, 27, 31),\\
X[t] &= 4, && (t = 32, 36, 40, 44),\\
X[t] &= 11, && (t = 33, 37, 41, 45),\\
X[t] &= 16, && (t = 34, 38, 42, 46),\\
X[t] &= 23, && (t = 35, 39, 43, 47),\\
X[t] &= 6, && (t = 48, 52, 56, 60),\\
X[t] &= 10, && (t = 49, 53, 57, 61),\\
X[t] &= 15, && (t = 50, 54, 58, 62),\\
X[t] &= 21, && (t = 51, 55, 59, 63).
\end{aligned}
$$

For our hash computation, we need to define a set of four 32-bit registers (A, B, C, D). Since the initial values (A_0, B_0, C_0, D_0) are based off of the hash calculation of the previous 512-bit block, we are required to set values for these registers for the computation for the first block. These values, represented in little-endian (least significant byte first) hexadecimal, are as follows:

$A_0 = 01\ 23\ 45\ 67,$

$B_0 =$ 89 AB CD EF,

$C_0 =$ FE DC BA 98,

$D_0 =$ 76 54 32 10.

The last constant used, g, is defined as:

$$\begin{aligned} g_t &= t, && (0 \leq t \leq 15), \\ g_t &= (5t+1) \bmod 16, && (16 \leq t \leq 31), \\ g_t &= (3t+5) \bmod 16, && (32 \leq t \leq 47), \\ g_t &= 7t \bmod 16, && (48 \leq t \leq 63). \end{aligned}$$

Futhermore, we will define the circular left shift operation as follows:

$$S^n(X) = (X << n) \vee (X >> 32 - n)$$

where $X << n$ discards the left-most n bits of X and pads the result with n zeroes on the right, and $X >> n$ discards the right-most n bits of X and pads the result with n zeroes on the left.

Finally, consider the addition of two 32-bit words X and Y, $X+Y$. Let words X and Y represent integers $0 \leq x < 2^{32}$ and $0 \leq y < 2^{32}$. Compute $z = (x+y) \bmod 2^{32}$, and convert z to a word Z. Then $X + Y = Z$.

Computation of the Message Digest. For each 512-bit block, $M[i]$, of the padded message, we perform the following steps:

1. Break the block into sixteen 32-bit words, which we denote as $W[j]$, $0 \leq j \leq 15$.

2. Initialize the 32-bit registers A, B, C, and D to the following values:

$A = A_0$,

$B = B_0$,

$C = C_0$,

$D = D_0$.

3. For $t = 0$ to 63 do

   (a) Define a variable dTemp $= D$.

   (b) Let $D = C$, $C = B$, and $A =$ dTemp.

   (c) Set B according to the following:

$$
\begin{aligned}
B \mathrel{+}= S^{X[t]}(A + F(B, C, D) + T[t] + W[g_t]), &\quad (0 \leq t \leq 15),\\
B \mathrel{+}= S^{X[t]}(A + G(B, C, D) + T[t] + W[g_t]), &\quad (16 \leq t \leq 31),\\
B \mathrel{+}= S^{X[t]}(A + H(B, C, D) + T[t] + W[g_t]), &\quad (32 \leq t \leq 47),\\
B \mathrel{+}= S^{X[t]}(A + I(B, C, D) + T[t] + W[g_t]), &\quad (48 \leq t \leq 63).
\end{aligned}
$$

4. Set $A_0 \mathrel{+}= A$. $B_0 \mathrel{+}= B$, $C_0 \mathrel{+}= C$, and $D_0 \mathrel{+}= D$.

The MD5 process can be visualized in figure 3 which pictures one iteration of the MD5 function. The resulting message digest of our original b-length is $A_0 \| B_0 \| C_0 \| D_0$ in little-endian notation.

3.5.2 SHA-1

The following description of SHA-1 is taken from the IETF RFC 3174 [rJ01].

Padding. The padding scheme for SHA-1 is the same as the padding scheme for MD5. A message of length l will be padded so that its length in bits is a multiple of 512, and it will contain $16 \cdot n$ words for some $n > 0$. The padded message can be regarded as a sequence of n blocks $M[1 \ldots n]$.

Functions and Constants. A sequence of logical functions is used in SHA-1, denoted by $f(0), f(1), \ldots, f(79)$, where each $f(t)$, $0 \leq t \leq 79$, operates on three 32-bit words B, C, D and produces a 32-bit word as output. We define $f(t; B, C, D)$ as follows:

$$\begin{aligned}
f(t; B, C, D) &= (B \wedge C) \vee (\neg B \wedge D), && (0 \leq t \leq 19),\\
f(t; B, C, D) &= B \oplus C \oplus D, && (20 \leq t \leq 39),\\
f(t; B, C, D) &= (B \wedge C) \vee (B \wedge D) \vee (C \wedge D), && (40 \leq t \leq 59),\\
f(t; B, C, D) &= B \oplus C \oplus D, && (60 \leq t \leq 79).
\end{aligned}$$

A sequence of constant words $K(0), K(1), \ldots, K(79)$ is used in SHA-1, where $K(t)$ is defined in hexadecimal, big-endian (most significant byte first) notation as:

$$\begin{aligned}
K(t) &= \text{5A 82 79 99}, && (0 \leq t \leq 19),\\
K(t) &= \text{6E D9 EB A1}, && (20 \leq t \leq 39),\\
K(t) &= \text{8F 1B BC DC}, && (40 \leq t \leq 59),\\
K(t) &= \text{CA 62 C1 D6}, && (60 \leq t \leq 79).
\end{aligned}$$

Finally, we will use the circular left shift and addition modulo 2^{32} operations, which are both defined in the same manner as we used for our overview of MD5.

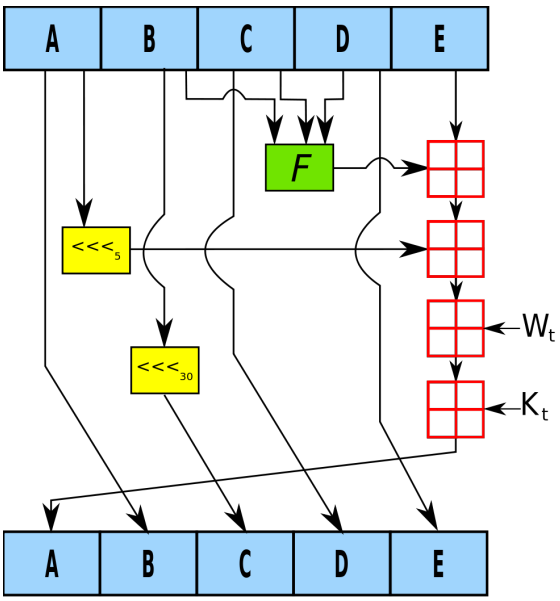

Figure 4: One round of SHA-1. F is one of the functions f defined in our functions section, W_t is the expanded message word of round t, K_t is $K(t)$ as defined in our constants section, $<<<_n$ is S^n as defined in our functions, and $\boxplus$ denotes addition modulo 2^{32}.

Source: `http://en.wikipedia.org/wiki/File:SHA-1.svg`

Computation of the Message Digest. The message digest of the padded message is described using two buffers, each consisting of five 32-words, and a sequence of eighty 32-bit words. The words in the first buffer are labeled A, B, C, D, E and in the second buffer are labeled H_0, H_1, H_2, H_3, H_4. The words of the 80-word sequence are labeled $W(0), W(1), \ldots, W(79)$. Additionally, a single word buffer TEMP is used.

First, initialize the words in the second buffer in big-endian hexadecimal notation as follows:

$H_0 =$ 67 45 23 01,

$H_1 =$ EF CD AB 89,

$H_2 =$ 98 BA DC FE,

$H_3 =$ 10 32 54 76,

$H_4 =$ C3 D2 E1 F0.

Now we process $M[1], M[2], \ldots, M[n]$ by processing $M[i]$ as follows:

1. Divide $M[i]$ into 16 words $W(0), W(1), \ldots, W(15)$ where $W(0)$ is the left-most word.

2. For $t = 16$ to 79 let
   $W(t) = S^1(W(t-3) \oplus W(t-8) \oplus W(t-14) \oplus W(t-16))$.

3. Let $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$.

4. For $t = 0$ to 79 do
   TEMP= $S^5(A) + f(t; B, C, D) + E + W(t) + K(t)$;
   $E = D$; $D = C$; $C = S^{30}(B)$; $B = A$; $A =$TEMP.

5. Let $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$.

The SHA-1 process can be visualized in figure 4 which pictures one iteration of the SHA-1 function. After processing $M[n]$, the message digest is the 160-bit string represented by the 5 words H_0, H_1, H_2, H_3, H_4.

4 Expander Graphs

Now that we have introduced the requisite graph-theoretic concepts, we can discuss expander graphs. Expander graphs are a class of graph that hold the seemingly contradictory properties of being both sparse and well-connected (we will introduce the notion of a graph being well-connected in this section). These graphs turn out to have many applications in computer science, such as the construction of optimized network configurations and, as we will discuss later, the construction of graph-based cryptographic hash functions.

The section is organized as follows. Section 4.1 will introduce expander graphs and their properties, families of expander graphs and examples, and the explicit construction of expander graphs. Section 4.2 will introduce concepts in spectral graph theory, as well as relate the edge expansion ratio of an expander graph to its spectral gap. Section 4.3 will further restrict the relation of the edge expansion ratio of an expander graph to its spectral gap, as well as introduce the Ramanujan class of expander graphs. Section 4.4 will discuss the analogous theory of expansion in graphs for Cayley graphs. Finally, section 4.5 will introduce the notion of random graphs, and relate them to expander graphs.

4.1 Graph Expansion

An **expander graph** is a graph in which every subset S of vertices is connected to many vertices in the complementary set $\bar{S}$ of vertices. Expander graphs are sparse graphs that have many useful properties, such as low diameter, high connectivity, and a high chromatic number [Tao11].

Suppose a graph $G = (V, E)$ has n vertices. Consider a subset S of the vertices V in G, and its complement $\bar{S}$. The **edge boundary** of S, denoted

∂S, is defined as the set of edges $(v, w) \in E$ such that $v \in S$ and $w \in \bar{S}$ [Nie05]. We can use the edge boundary to define the following:

Definition 4.1.1. The **edge expansion ratio** of a graph $G = (V, E)$ on n vertices is given by

$$h(G) = \min_{S \subset V : 1 \leq |S| \leq n/2} \frac{|\partial S|}{|S|}.$$

The edge expansion ratio of a graph gives in part an idea of the connectivity of the graph: A higher value of $h(G)$ means a higher minimum value for the ratio of the edge boundary of a vertex subset S and the size of the subset $|S|$. Since we define our subsets to be of size $|S| \leq \frac{n}{2}$, this implies that subsets of vertices that comprise less than half of the total number of vertices will be well-connected to larger subsets of vertices. This means that G itself will be well-connected.

As mentioned previously, it is important for expander graphs to be sparse in addition to well-connected. A complete graph on n vertices K_n, for instance, is an example of a graph that is well-connected, but not sparse: Each subset of the vertices in K_n is connected to every vertex in the complement, so the edge boundary is $|\partial S| = |S| \cdot |\bar{S}| = |S|(n - |S|)$, which leads to the following edge expansion ratio [Nie05]:

$$h(K_n) = \min_{S \subset V : 1 \leq |S| \leq n/2} (n - |S|) = \left\lceil \frac{n}{2} \right\rceil.$$

Contrary to the prior example, we typically do not consider a single graph when discussing the expansion properties of graphs. Instead, we look at families of expander graphs which are constructed in accordance with the following criterion:

Definition 4.1.2. A sequence of k-regular graphs $\{G_i\}_{i \in \mathbb{N}}$ of size increasing

with i is a **Family of Expander Graphs** if there exists $\epsilon > 0$ such that $h(G_i) \geq \epsilon$ for all i [HLW06].

So when we talk about k-regular expander graphs, we usually mean an infinite collection, or family, of k-regular graphs that satisfy the properties of Definition 4.1.2. The idea is that a family of expanders should allow us to construct arbitrarily large graphs which are both sparse and well-connected [Kow13].

Example 4.1.3. The following are examples of families of expander graphs [HLW06]:

1. A family of 8-regular graphs G_m for every integer m is a family of expander graphs. The vertex set is $V_m = \mathbb{Z}_m \times \mathbb{Z}_m$. The neighbors of the vertex (x, y) are $(x + y, y), (x - y, y), (x, y + x), (x, y - x), (x + y + 1, y), (x - y + 1, y), (x, y + x + 1)$, and $(x, y - x + 1)$ where all operations are modulo m.

2. A family of 3-regular p-vertex graphs for every prime p is a family of expander graphs. Here $V_p = \mathbb{Z}_p$, and a vertex x is connected to $x + 1, x - 1$, and to its inverse x^{-1}, where all operations are modulo p, and the inverse of 0 is 0.

When we apply expander graphs to the field of computer science, we become increasingly interested in the **explicit construction** of expander graphs from families of expander graphs and the **efficiency** of these explicit constructions. The performance of an algorithm that employs expander graphs is at least partially dependent on how efficiently the graphs can be constructed, so we can see why this topic is important.

There are two natural levels of efficiency to be considered in the construction of such graphs. In the first we require that an n-vertex graph should

be generated "from scratch" in time polynomial in n. In the stronger version we demand that the neighborhood of any given vertex should be computable in time that is polynomial in the description length of the vertex, which is usually polynomial in $\log n$. A more rigorous definition follows [HLW06]:

Definition 4.1.4. Let $\{G_i\}_{i\in\mathbb{N}}$ be a family of expander graphs where G_i is a k-regular graph on n_i vertices and the integers $\{n_i\}$ are increasing with i such that $n_i < n_{i+1} \leq n_i^2$.

1. The family is called **Mildly Explicit** if there is an algorithm that generates the j-th graph in the family G_j in time polynomial in j. (That is, G_j is computed in time $< Aj^B$ for some constants $A, B > 0$).

2. The family is called **Very Explicit** if there is an algorithm that on input of an integer i, a vertex $v \in V(G_i)$, and $m \in \{1, \cdots, k\}$ computes the m-th neighbor of the vertex v in the graph G_i. This algorithm's run time should be polynomial in its input length (the number of bits needed to express the triple (i, v, m)).

The explicit construction of expander graphs is important, but we must be certain that our expander graphs are truly good expanders. To that end, being able to compute the edge expansion ratio is important. However, for most graphs, computation of $h(G)$ is not nearly as simple as it was in our K_n example. For a graph $G = (V, E)$ on n vertices, finding the minimum value of $|\partial S|/|S|$ for all $S \subset V, 1 \leq |S| \leq n/2$ can be very difficult and time-consuming, as the total number N of subsets that fit those criteria is:

$$N = \sum_{k=1}^{n/2} \binom{n}{k}.$$

Since this number is equal to the total number of subsets S of the vertices of G of size between 1 and $n/2$, and since binomial coefficients follow the

relation $\binom{n}{k} = \binom{n}{n-k}$, we can say that N is approximately half of the total number of subsets of V.

We know that the total number of subsets of a set of size n is 2^n, so $N \approx 2^{n-1}$. Thus, the number of subsets to look at in our computation of $h(G)$ increases exponentially with n, which makes finding $h(G)$ prohibitively difficult at sufficiently large n. Thankfully, there exists a method for approximating $h(G)$ that is much easier. This method involves the spectrum of graphs, which we will introduce in the following section.

4.2 Spectral Graph Theory

A graph $G = (V, E)$ on n vertices can be expressed in terms of an $n \times n$ matrix with the rows and columns indexed by V and the elements determined by the number of edges between each pair of vertices [Kow13]. A more formal definition follows:

Definition 4.2.1. Let $G = (V, E)$ be an undirected graph on n vertices. Then the **adjacency matrix** A_G is an $n \times n$ matrix with elements determined by:

$$a(i, j) = |\{(v_i, v_j) \in E\}|$$

Since G is undirected, edges in E are unordered pairs, which means $a(i, j) = a(j, i)$ for all $1 \leq i \leq n$ and for all $1 \leq j \leq n$. As a result, A_G is a symmetric matrix, which means that the **eigenvalues** of A_G are all real. That is, all values $\lambda_m(G)$ that satisfy the relation $A_G \mathbf{v}_m = \lambda_m(G)\mathbf{v}_m$, where $\mathbf{v}_m \in \mathbb{R}^V$ is called the **eigenvector** of A_G corresponding to the eigenvalue λ_m, are real. For a symmetric graph, $\mathbf{v}_m \neq \mathbf{0}$, and the eigenvectors form an orthonormal basis for $\mathbb{R}^V$ [Spi11]. Additionally, the set of eigenvalues of A_G is called the **spectrum** of the graph. In the case of k-regular graphs, the

eigenvalues of A_G hold some interesting properties:

Theorem 4.2.2. *Let $G = (V, E)$ be a k-regular undirected multigraph, and let A_G be its corresponding adjacency matrix. Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ be the real eigenvalues of A_G. Then*

1. *$\lambda_1 = k$ and $\lambda_n \geq -k$.*
2. *$\lambda_2 = k$ if and only if G is disconnected.*
3. *$\lambda_n = -k$ if and only if at least one of the connected components of G is bipartite.*

A proof of this theorem can be found in the following resource: [Tre11]. Interestingly enough, we find that the edge expansion ratio of a graph is related closely to the second largest eigenvalue, λ_2, of A_G. Before we get into the exact relation, it is useful to define the following:

Definition 4.2.3. If $G = (V, E)$ is an undirected graph with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, then $\Delta(G) \equiv \lambda_1 - \lambda_2$ is the **Spectral Gap** of G.

This leads to the relationship between $h(G)$ and λ_2 [Nie05]:

Theorem 4.2.4. (Cheeger's Inequality) *The edge expansion ratio $h(G)$ for a k-regular graph is related to the spectral gap $\Delta(G)$ by:*

$$\frac{\Delta(G)}{2} \leq h(G) \leq \sqrt{2k\Delta(G)}.$$

Since we know that $\lambda_1 = k$ for a k-regular, we can also write this inequality as:

$$\frac{k - \lambda_2}{2} \leq h(G) \leq \sqrt{2k(k - \lambda_2)}.$$

The Cheeger constant for graphs is actually analogous to a theory in Riemannian geometry concerning Riemannian manifolds. As such, Cheeger's Inequality for graphs can be considered a discrete version of Cheeger's inequality for Riemannian manifolds. For a more in depth discussion of the Cheeger constant and Cheeger inequality in Riemannian geometry, we direct the reader to Appendix C.

4.3 Eigenvalue Bound and Ramanujan Graphs

Theorem 4.2.4 leads to an interesting question concerning the size of the spectral gap. We know that the spectral gap of a k-regular, n-vertex graph is dependent on both k and n, but exactly how big can the spectral gap be? The answer to that question not only depends on k and n, but also their relationship to each other. In the context of expander graphs, we are mostly interested in graphs with fixed k and large n where $n \gg k$ [HLW06].

First, consider a k-regular, n-vertex graph G. Let $k = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq -k$ be the real eigenvalues of A_G. Define

$$\lambda(G) = \max_{|\lambda_i|<k} |\lambda_i|.$$

Simply put, $\lambda(G)$ is the maximum of the absolute value of all eigenvalues of A_G excluding those equal to $\pm k$ [Lub11]. The bound for $\lambda(G)$ is given by the Alon-Boppana result:

Theorem 4.3.1. (Alon-Boppana) *Let $\{G_i\}_{i\in\mathbb{N}}$ be an infinite family of k-regular connected graphs on n vertices where k is fixed and n increases with i. Then for all i:*

$$\lambda(G_i) \geq 2\sqrt{k-1} - o(1)$$

where $o(1)$ tends to zero for every fixed k as $n \to \infty$ [Lub11].

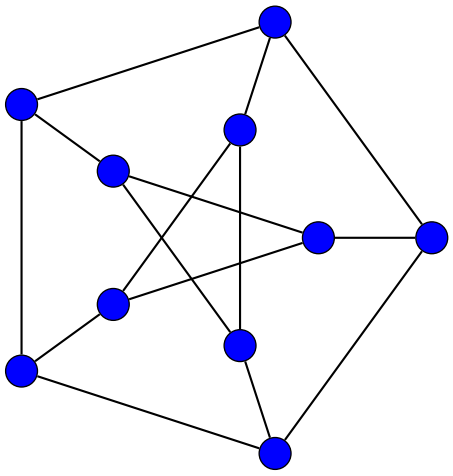

Figure 5: The Petersen graph is an example of a Ramanujan graph.

This definition leads to the following classification of graph, as seen in figure 5, where the Alon-Boppana bound is tight:

Definition 4.3.2. *(Ramanujan Graph)* A k-regular finite graph G is called a **Ramanujan Graph** if

$$\lambda(G) \leq 2\sqrt{k-1}.$$

The work of Lubotzky, Phillips, and Sarnak [LPS88] proved that the explicit construction of infinite families of k-regular Ramanujan graphs is possible when $k-1$ is prime. Their use of the Ramanujan conjecture in the proof, which will not be discussed here, provides the origin for the name. The work of Morgenstern [Mor94] extended the construction of infinite families of k-regular Ramanujan graphs to the case when $k-1$ is a prime power [HLW06]. We can see from both the Alon-Boppana bound and the definition of Ramanujan graphs that Ramanujan graphs define a classification of graph where the spectral gap is almost as large as possible.

4.4 Expansion in Cayley Graphs

The ideas we have previously explored regarding expander graphs in the general case can be applied rather effectively to the specific case of Cayley graphs. An infinite family of groups $\{\mathcal{G}_n\}$ can be made into a family of expanders if there is some constant k and a generating set S_n of size k in each $\mathcal{G}_n$ so that the family $\{C_{\mathcal{G}_n,S_n}\}$ is a family of expanders. That said, not all classes of groups are suitable to be made expanders. Abelian groups, for instance, generally cannot be made expanders with generating sets of bounded size. Most simple groups, however, can be made into families of expanders, as well as the special linear groups $SL_d(\mathbb{F}_{p^m})$ for any $d \geq 2$, $m \geq 1$, and prime p [Pet09].

The study of expanding properties in a Cayley graph constructed from an Abelian group $\mathcal{G}$ is the same as the study of its **characters**, where the character on a group $\mathcal{G}$ is a group homomorphism from $\mathcal{G}$ to the multiplicative group of a field, which in our case is the field of complex numbers $\mathbb{C}$ [Pet09].

Proposition 4.4.1. *Let A be the normalized adjacency matrix of a Cayley graph $C_{\mathcal{G},S}$. Let χ be a character of $\mathcal{G}$. Then the vector $(\chi(g))_{g\in\mathcal{G}}$ is an eigenvector of A with eigenvalue $\frac{1}{|S|}\sum_{s\in S}\chi(s)$.*

This approach is generalized in non-Abelian groups $\mathcal{G}$ by the study of the **representations** of the groups, where a representation of a group is a homomorphism from $\mathcal{G}$ to matrix groups over $\mathbb{C}$ [Pet09].

The notion of expansion in graphs is translated to group theory by the **Kazhdan constant**. Let the **regular representation** r of a group $\mathcal{G}$ be the representation where any $g \in \mathcal{G}$ is associated with a matrix of size $|\mathcal{G}|$ which is 1 at the entries corresponding to (u, ug) for all u and 0 elsewhere.

The Kazhdan constant of $\mathcal{G}$ and S is defined as [Pet09]

$$K(\mathcal{G}, S) = \min_{v \in \mathbb{C}^{|\mathcal{G}|}, v \perp 1} \max_{s \in S} \frac{\|r(s)v - v\|^2}{\|v\|^2}.$$

For a group $\mathcal{G}$ and a symmetric subset S of size k of $\mathcal{G}$, the Kazhdan constant $K(\mathcal{G}, S)$ is related to the spectral gap of $C_{\mathcal{G},S}$ by [Pet09]

$$\frac{K(\mathcal{G}, S)}{2k} \leq k - \lambda_2 \leq \frac{K(\mathcal{G}, S)}{2}.$$

4.5 Expanders and Random Graphs

While expander graphs can be covered from combinatorial and algebraic perspectives, they can also be looked at probabilistically and statistically. For instance, random walks (which will be covered later) are very important with regard to cryptographic applications of expander graphs. It also turns out that expander graphs are rather closely related to random graphs. We will take this section to introduce what random graphs are and a simple example of how they are related to expander graphs.

A **random graph** on n vertices starts as a set of n isolated vertices, and develops by successively acquiring edges at random. The aim of constructing such graphs is to determine at what stage of random graph evolution a particular property of the graph is likely to arise [Bol85].

The most studied model of random graphs is the Erdös-Rényi model, of which two closely related variants exist. The first variant, the $G(n, m)$ variant, chooses a graph uniformly at random from the set of all graphs on n vertices with m edges. The second variant, the $G(n, p)$ variant, constructs a graph on n vertices by choosing each edge independently with probability p. In this variant, a graph on n vertices and r edges is chosen with probability

$p^r(1-p)^{\binom{n}{2}-r}$ [Chu08].

The following lemma provides a connection between expander graphs and random graphs:

Lemma 4.5.1. (Expander Mixing Lemma) *Let $G = (V, E)$ be a k-regular graph with n vertices. Then for all $S, T \subseteq V$:*

$$\left| \; |E(S,T)| - \frac{k|S||T|}{n} \; \right| \leq \lambda(G)\sqrt{|S||T|}.$$

This result shows that a small second eigenvalue in a graph implies that its edges are "spread out", which is a hallmark of random graphs. The left-hand side of the previous equation measures the deviation between the number of edges between S and T, and the number of expected edges between S and T in a random graph of edge density k/n. A small $\lambda(G)$ (or large spectral gap), which is the case for good expanders, implies that the deviation between these two values is small, which makes the graph nearly random [HLW06].

The converse of this result turns out to have useful applications as well. When the spectral gap of a k-regular graph G is much smaller than k, the upper and lower bounds in theorem 4.2.4 differ considerably. A converse to the Expander Mixing Lemma captures the spectral gap more tightly [HLW06]:

Lemma 4.5.2. (Converse of the Expander Mixing Lemma) *Let $G = (V, E)$ be a k-regular graph with n vertices and suppose that*

$$\left| \; |E(S,T)| - \frac{k|S||T|}{n} \; \right| \leq p\sqrt{|S||T|}$$

holds for every two disjoint sets S, T and for some positive p. Then $\lambda(G) \leq \mathcal{O}(p \cdot (1 + log(k/p)))$. The bound is tight.

5 Random Walks and Expander Hashes

Many applications of expander graphs focus around **random walks** on the graphs. On a k-regular expander graph G, a random walk involves starting at some chosen vertex and moving at random to one of the k neighbors. This step is repeated for each new vertex, and is performed independently of any prior choices [Nie05]. An interesting aspect of these random walks is that a length t random walk on an expander graph is similar to a set of t vertices sampled uniformly and independently. The computational significance of this is that fewer random bits are required for a length t random walk than for an independent sampling of t vertices, so applications which would require such a sampling can be performed with greater speed [HLW06].

We find that for hash functions using message input to perform walks on a graph, these walks are very similar to random walks when these graphs are good expanders. As such, we will structure this section as follows. Section 5.1 will discuss random walks on graphs, as well as discuss properties of random walks on expander graphs. Section 5.2 will introduce the concept of an expander hash, or a hash function based on a walk on an expander graph. Section 5.3 will give an overview of expander hash functions from original proposals to more recent constructions. Finally, section 5.4 will take the attacks discussed in 3.4 and apply them to expander hash functions, as well as introduce some attacks designed specifically for expander hashes. This section assumes knowledge of finite fields, a summary of which can be found in Appendix A.

5.1 Introduction to Random Walks

A vector $\mathbf{p} \in \mathbb{R}^V$ is called a **probability distribution vector** if its coordinates are nonnegative and $\sum_{i=1}^{n} p_i = 1$. The probability vector that corresponds to the uniform distribution on $\{1, \ldots, n\}$ is denoted by $\mathbf{u} = (1, \ldots, 1)/n$. A random walk on a graph $G = (V, E)$ is usually initiated by selecting the first vertex v_1 from some **initial probability distribution** $\mathbf{p}_1$ on V. This induces a sequence of probability distributions $\mathbf{p}_i$ on V so that the probability that $v_i = x \in V$ equals $\mathbf{p}_i(x)$ for every i and x. For every finite connected nonbipartite graph G, the distributions converge to a limit, or **stationary**, distribution. Furthermore, if G is a k-regular graph, then this distribution is simply $\mathbf{u}$ [HLW06].

When we apply one step of the random walk procedure on a k-regular graph $G = (V, E)$, moving from the i-th to the $(i+1)$-th vertex in the walk, the updated probability distribution is:

$$\mathbf{p}_{i+1} = \hat{A}_G \mathbf{p}_i$$

where $\hat{A}_G = A_G/k$ is the **normalized adjacency matrix**. More generally, we say that the probability distribution after performing a random walk of length t is:

$$\mathbf{p}_t = \hat{A}_G^t \mathbf{p}_1.$$

A more rigorous definition of the random walk on $G = (V, E)$ is that it is a **Markov Chain** with state set V and transition matrix $\hat{A}_G$. Something we are interested in is the rate at which the Markov Chain converges to its stationary distribution. The following theorem shows that when the uniform distribution is a stationary distribution for the chain, then the Markov

chain converges to the uniform distribution exponentially quickly at a rate determined by $\lambda_2(M)$ [Nie05]:

Theorem 5.1.1. *Suppose M is a normal transition matrix for a Markov chain on n states with the uniform distribution $\boldsymbol{u}$ as a stationary point $M\boldsymbol{u} = \boldsymbol{u}$. Then for any starting distribution $\boldsymbol{p}$:*

$$\|M^t\boldsymbol{p} - \boldsymbol{u}\|_1 \leq \sqrt{n}\lambda_2(M)^t.$$

The expression $\|\cdot\|_1$ denotes the L^1-norm, and the **L^p-norm** gives the length of a vector in a Lebesgue space L^p, and is defined as follows:

$$\|x\|_p := \left(\sum_{i=1}^{n}|x_i|^p\right)^{1/p}.$$

The fact that M is a symmetric matrix allows us to make the connection to expander graphs by considering the case $M = \hat{A}_G$. For a k-regular graph G:

$$\|\hat{A}_G^t\mathbf{p} - \mathbf{u}\|_1 \leq \sqrt{n}\left(\frac{\lambda_2(G)}{k}\right)^t.$$

Using the Cheeger inequality, Theorem 4.2.4, we can obtain the rate of convergence as it relates to the edge expansion ratio:

$$\|\hat{A}_G^t\mathbf{p} - \mathbf{u}\|_1 \leq \sqrt{n}\left(1 - \frac{h(G)^2}{2k^2}\right)^t.$$

Thus we can see that for a family of expander graphs, the rate of convergence of the Markov chain is exponentially fast in the number of steps t [Nie05]. We can also see that the Markov chain converges faster when the ratio $h(G)/k$ is large, which occurs both when k is small and when $h(G)$ is large.

Another useful property of random walks on expander graphs is that

the probability that they stay within a given subset of vertices decreases exponentially with every step unless that subset is very large. The general result for Markov chains states:

Theorem 5.1.2. *Let X_0 be uniformly distributed on n states, and suppose there is a time-homogeneous Markov chain $X_0, \ldots, X_t$ with transition matrix M. Suppose the uniform distribution (u) is a stationary point of M. Let B be a subset of the states, and $B(t)$ be the event that that $X_j \in B$ for all $j \in 0, \ldots, t$. Then*

$$Pr(B(t)) \leq \left(\lambda_2(M) + \frac{|B|}{n}\right)^t$$

where $Pr(B(t))$ is the probability of $B(t)$ [Nie05].

When applied to random walks on expander graphs, X_0 becomes some vertex chosen uniformly at random from the graph. We use X_0 as the starting point for a random walk $X_0, \ldots, X_t$ where X_t is the vertex after the t-th step. Given a subset of vertices B, $B(t)$ becomes the event that the entire random walk is in B after the t-th step. From this, we obtain

$$Pr(B(t)) \leq \left(\frac{\lambda_2(G)}{k} + \frac{|B|}{n}\right)^t.$$

The probability exponentially decreases provided $\frac{\lambda_2(G)}{k} + \frac{|B|}{n} < 1$. For a family of expander graphs, there exists some constant $\epsilon > 0$ such that we get an exponential decrease for any B such that $|B|/n < \epsilon$ [Nie05].

5.2 Expander Hashes

At its simplest, an **Expander Hash** is a hash function where the input to the hash is used as directions for walking around an expander graph without backtracking, and the output of the hash function is the ending vertex of the

walk. For a fixed hash function, the walk starts at a fixed vertex in the given expander graph, and a family of hash functions can be defined by allowing this starting vertex to vary.

5.2.1 General Construction

To execute a walk on a k-regular undirected expander graph, the input to the hash function must be broken into chunks of size c such that $2^c = k - 1$. Starting at the first vertex, each step of the walk chooses an edge emanating from that vertex to follow to get to the next vertex where the choice of the edge to follow is determined by the next c bits of the input. Since we do not allow backtracking, there are only $k - 1$ choices for the next edge at each step [CGL09]. Directed k-regular graphs, on the other hand, cannot have backtracking in them unless the graph has multiple edges, so the input of the hash function must be broken into chunks of size c such that $2^c = k$, as there are k choices for the next edge at each step.

As we have explored previously, we know that random walks on expander graphs converge quickly to the uniform distribution, so the output of an expander hash employing a random walk will be uniform provided the input was uniformly random [CGL09]. We also know that random walks on expander graphs tend not to stay in any small subset of vertices for a long time, which means that our expander hash construction will not take place in a significantly smaller portion of the graph than the entire graph.

5.2.2 Cayley Hashes

As we have applied general expander graphs to the construction of hash functions, we can also construct hash functions based on Cayley graphs. To con-

struct a **Cayley hash** from a directed Cayley graph $C_{\mathcal{G},S}$, let $\{1,\ldots,k\}^*$ be the set of arbitrary-length sequences $(m_1, m_2, \ldots, m_l)$ consisting of elements of $\{1,\ldots,k\}$. Fixing an initial value g_0 in $\mathcal{G}$ and an ordering $\sigma : \{1,\ldots,k\} \to S$, determines a Cayley hash function $H : \{1,\ldots,k\}^* \to G$ defined by $H() = g_0$, $H(m_1) = g_0\sigma(m_1)$ and $H(m_1, m_2, \ldots, m_l) = H(m_1, m_2, \ldots, m_{l-1})\sigma(m_l)$. The successive computations of $g_0\sigma(m_1)$, $g_0\sigma(m_l)\sigma(m_2)$, etc... correspond to a walk from v_{g_0} to $v_{g_0\sigma(m_l)}$, $v_{g_0\sigma(m_l)\sigma(m_2)}$, etc... in the Cayley graph $C_{\mathcal{G},S}$ [PLQ07].

If S is stable under inversion, the corresponding Cayley graph $C_{\mathcal{G},S}$ is undirected, which makes backtracking on our walk an issue. To rectify this, the messages are decomposed in chunks of size $(k-1)$ bits rather than k bits, and H is defined as $H(m_1, m_2, \ldots, m_l) = H(m_1, m_2, \ldots, m_{l-1})\sigma_l$ with $\sigma_1 = \sigma(m_1)$ and $\sigma_i = \sum(\sigma_{i-1}, m_i) := \sigma(\sigma^{-1}(\sigma_{i-1}^{-1}) + m_i \mod k), i \in \{2,\ldots,l\}$ [PLQ07].

A hash function $H' : \{0,1\}^* \to \{0,1\}^{\log_2 |G|}$ sending bitstrings to bitstrings is derived from H as $H' = \pi_2 \circ H \circ \pi_1$, where $\pi_1 : \{0,1\} \to \{1,\ldots,k\}^*$ and $\pi_2 : \{0,1\} \to \{0,1\}^{\log_2 |G|}$ are respectively initial and final mappings. A keyed Cayley hash is constructed similarly, by letting the element g_0 be a function of the key [PLQ07].

5.2.3 Security considerations

Based on our prior exploration of expander graphs, we gather that the following requirements should hold for graphs used in expander hashes [Pet09]:

- **Large expansion:** This requirement guarantees that the hash values of relatively short messages are well-distributed in the output set.

- **Short diameter:** A short diameter implies that all vertices are the output of short messages.

- **Large girth:** A large girth guarantees that no short collision exists and it bounds the distance between any two colliding messages. This requirement may not be necessary if the initial vertex is chosen randomly, but for Cayley hashes a large girth is necessary.

- **Efficiency:** Computing the neighbors of any given vertex must be very efficient.

- **Collision, preimage, and second preimage resistances**: The following problems must be hard:

  - **Constrained two-paths problem:** Given a randomly selected starting point v_0 in a randomly selected graph G, find two paths in G of length at most l that start in v_0 and end at the same vertex.

  - **Constrained cycle problem:** Given a randomly selected starting point v_0 in a randomly selected graph G, find a cycle in G of length at most l that goes through v_0.

  - **Path problem:** Given a randomly selected starting point v_0 and an ending point v in a randomly selected graph G, find a path in G of length at most l that starts in v_0 and ends in v.

  - **Two-path problem:** Given a randomly selected graph G, find two paths in G of length at most l that start and end at the same vertices.

  - **Cycle problem:** Given a randomly selected graph G, find a cycle in G of length at most l.

Interestingly, some of the above problems can be interpreted as problems in group theory. First, we define the **length** of a product of group elements $g_0 g_1 \dots g_{\mu-1}$ as μ. This product is said to be a **reduced product** if $g_i g_{i+1} \neq 1$ for $0 \leq i \leq \mu - 2$. In particular, the cycle problem, two-path problem, and path problem are respectively equivalent to the following three group-theoretical problems for Cayley hashes [Pet09]:

- **Factorization problem:** Given a group $\mathcal{G}$, a subset S of $\mathcal{G}$, and a group element g, find a reduced product of subset elements of length at most l that is equal to g, that is

$$\prod_{0 \leq i < \mu} s_i = g$$

  with $s_i \in S$, $s_i s_{i+1} \neq 1$, and $\mu \leq l$.

- **Balance problem:** Given a group $\mathcal{G}$ and a subset S of $\mathcal{G}$, find two reduced products of subset elements of lengths at most l that are equal, that is

$$\prod_{0 \leq i < \mu} s_i = \prod_{0 \leq i < \mu'} s'_i$$

  with $s_i, s'_i \in S$, $s_i s_{i+1}, s'_i s'_{i+1} \neq 1$, and $\mu, \mu' \leq l$.

- **Representation problem:** Given a group $\mathcal{G}$, a subset S of $\mathcal{G}$, and a group element g, find a reduced product of subset elements of length at most l that is equal to the unit element of the group, that is

$$\prod_{0 \leq i < \mu} s_i = 1$$

  with $s_i \in S$, $s_i s_{i+1} \neq 1$, and $\mu \leq l$.

5.3 Examples of Expander Hashes

5.3.1 Zémor's first proposal

The first exploration of expander hashes was performed by Gilles Zémor in the early 1990s. His scheme was a Cayley hash on a directed Cayley graph built from the group $\mathcal{G} = SL_2(\mathbb{F}_p)$, or the set of 2×2 matrices over the field $\mathbb{F}_p$ with determinant 1 under the group operations of matrix multiplication and matrix inversion. The graph generator set is

$$S_1 = \{A_1, B_1\} = \left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \right\}.$$

Computation of a message of length μ for $C_{\mathcal{G},S_1}$ requires μ multiplications by A or B, and each of these multiplications requires 2 additions modulo p, which makes this scheme reasonably computationally efficient [Pet09].

This scheme, however, is no longer secure due to an attack utilizing a lifting strategy: The representation problem is lifted from $SL_2(\mathbb{F}_p)$ to $SL_2(\mathbb{Z})$ by exploiting the fact that A and B generate the set $SL_2(\mathbb{Z}^+)$ in $SL_2(\mathbb{Z})$. This allows for the representation problem to be solved easily using the Euclidean algorithm. To rectify this, two other generator sets were proposed [Pet09]:

$$S_2 = \{A_2, B_2\} = \{A_1^2, B_1^2\} = \left\{ \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \right\},$$

$$S_3 = \{A_3, B_3\} = \{A_1, A_1B_1\} = \left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \right\}.$$

These changes in A and B affect the efficiency of computation of a message. For $C_{\mathcal{G},S_2}$, each bit of message requires 2 additions and 2 multiplications by 2 modulo p. For $C_{\mathcal{G},S_3}$, each bit of messages requires on average 3 additions

modulo p. No known attacks exist for the schemes utilizing S_2 and S_3 [Pet09].

5.3.2 Zémor-Tillich hash function

Three years after the first proposal of an expander hash, Jean-Pierre Tillich and Gilles Zémor proposed a new family of hash functions based on computations over a finite field of characteristic 2 [TZ94]. Rather than construct the expander hash using the group $SL_2(\mathbb{F}_p)$, the duo opted to use $SL_2(\mathbb{F}_{2^n})$, or the group of 2×2 matrices of determinant 1 in the finite field $K := \mathbb{F}_{2^n}$, which may also be represented by the quotient field $K = \frac{\mathbb{F}_2[X]}{P_n(X)}$ where $P_n(X)$ is an irreducible binary polynomial of degree n [Pet09].

The Zémor-Tillich hash function has as its defining parameter an irreducible polynomial $P_n(X)$ of degree n, has as its starting point the 2×2 identity matrix I_2, and has the following graph generator set [Pet09]:

$$S = \{A_0, A_1\} = \left\{ \begin{pmatrix} X & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix} \right\}.$$

Now define the mapping

$$\begin{aligned} \pi : \{0,1\} &\to \{A_0, A_1\}, \\ A_0 &\mapsto 0, \\ A_1 &\mapsto 1. \end{aligned}$$

Then the hash code of a binary message $m = m_0 m_1 \dots m_{\mu-1}$ is the matrix product $H_{ZT}(P_n(X), m) := \pi(m_0)\pi(m_1)\dots\pi(m_{\mu-1})$, and thus some element of the group $SL_2(\mathbb{F}_{2^n})$ [TZ94]. More compactly, we can say that the Zémor-

Tillich hash value of a bitstring $m = m_0 m_1 \dots m_{\mu-1}$ is given by [Pet09]:

$$H_{ZT}(P_n(X), m) := \prod_{i=0}^{\mu-1} \begin{pmatrix} X & 1 + m_i X \\ 1 & m_i \end{pmatrix}.$$

Since this hash function uses only arithmetic in a field of characteristic 2 in its computation, it is even more efficient than Zémor's previous proposals. Although attacks on this hash function exist, the function remains fundamentally unbroken to this day if the parameters are large enough and well-chosen [Pet09]. A full cryptanalysis of the Zémor-Tillich hash function can be found in [Pet09], Chapter 5.

5.3.3 LPS hash function

Around 15 years after the Zémor-Tillich hash function was proposed at the 14th Annual International Cryptology Conference, the team of Charles, Goren, and Lauter rediscovered expander hashes [CGL09]. Charles et al. proposed two constructions of expander hashes based on undirected graphs, in contrast with the directed graphs of Zémor's first proposal and the Zémor-Tillich hash function, and solved the issue of trivial backtracking collisions by explicitly forbidding backtracking in their construction [Pet09]. This section will focus on the scheme based on the non-bipartite LPS Ramanujan graph construction described in the paper of Lubotsky, Phillips and Sarnak [LPS88].

We define the previously mentioned graph construction as follows. Let l and p be primes, where l is a small prime and p is relatively large, such that $p, l \equiv 1 \pmod 4$ and l is a quadratic residue mod p (there exists an integer x such that $x^2 \equiv l \pmod{\text{p}}$). We will denote the LPS graph, with parameters l and p, by $X_{l,p}$. It is easy to see why LPS graphs were picked for this scheme, as they are Ramanujan, they have a large girth, and they

have a small diameter:

$$g(X_{l,p}) \geq 4\log_l p - \log_l 4,$$
$$D(X_{l,p}) \leq 2\log_l \left(\frac{p(p-1)(p+2)}{2}\right) + 2\log_l 2 + 1.$$

The vertices of $X_{l,p}$ are the matrices in $PSL_2(\mathbb{F}_p)$, which is the quotient group of $SL_2(\mathbb{F}_p)$ by the equivalence relation $M \sim -M$ for any matrix M [CGL09, Pet09]. An equivalent definition follows: $PSL_2(\mathbb{F}_p)$ is the group of 2×2 matrices over $\mathbb{F}_p$ with non-zero square determinant, modulo the equivalence relation $M_1 \sim \lambda M_2$, $\lambda \in \mathbb{F}_p^*$ [Pet09]. A matrix M is connected to the matrices gM, where the g's are the following explicitly defined matrices. Let j be an integer satisfying $j^2 \equiv -1$ (mod p). There are exactly $8(l+1)$ solutions (g_0, g_1, g_2, g_3) to the equation

$$g_0^2 + g_1^2 + g_2^2 + g_3^2 = l.$$

Among these solutions, we will consider those with odd $g_0 > 0$ and even g_1, g_2, g_3. To each such solution, we associate the matrix

$$g = \begin{pmatrix} g_0 + jg_1 & g_2 + jg_3 \\ -g_2 + jg_3 & g_0 - jg_1 \end{pmatrix}.$$

This gives us a set S of of $l+1$ matrices in $PGL_2(\mathbb{F}_p)$, but their determinants are squares modulo p and hence they lie in the index 2 subgroup of $PGL_2(\mathbb{F}_p)$, which is $PSL_2(\mathbb{F}_p)$. Thus we can see that the LPS graph is Cayley, and $X_{l,p} := C_{SL_2(\mathbb{F}_p),S}$. Additionally, this graph is undirected since S is stable under inversion. Since l is small, the set of matrices in S can be found by exhaustive search very quickly [Pet09]. The graph $X_{l,p}$ has $p(p^2-1)/2$ vertices and is $(l+1)$-regular. Recommended parameters are $l = 5$ and p a 1024-bit number [CGL09].

Collisions for the LPS hash functions have been found by Tillich and Zémor [TZ08], and their algorithm has been extended to a preimage attack by Petit [Pet09]. Both attacks can be defeated with a small modification to the graph generator set. The details of these attacks can be found in [TZ08], in [Pet09], Chapter 6, or in [PLQ08].

5.3.4 Morgenstern hash function

Morgenstern's Ramanujan graphs generalize LPS graphs from an odd prime $p \equiv 1 \bmod 4$ to any q which is an even power of 2 or a power of another prime. Arithmetic in fields of characteristic 2 is typically more efficient and easier to implement than arithmetic modulo a large prime integer. This led to the proposal of the Morgenstern hash function, which uses Morgenstern graphs for small even q [Pet09].

Morgensterm graphs for even q are defined as follows. Let q be a power of 2 and let $\epsilon \in \mathbb{F}_q$ such that $f(x) := x^2 + x + \epsilon$ is irreducible in $\mathbb{F}_q[x]$. Let $P_n(X) \in F_q[X]$ be irreducible of even degree n and let $\mathbb{F}_{q^n}$ be represented by $F_q[X]/P_n(X)$. We denote the Morgenstern graph by $\Gamma_{q,n}$. Like LPS graphs, Morgenstern graphs are Ramanujan, have a large girth, and have a small diameter [Pet09]:

$$g(\Gamma_{q,n}) \geq \frac{2}{3}\log_q[q^n(q^{2n}-1)],$$
$$D(\Gamma_{q,n}) \leq 2\log_q[q^n(q^{2n}-1)] + 2.$$

The vertices of the graph are elements of the group $PSL_2(\mathbb{F}_{q^n})$, which is the group of 2×2 matrices over $\mathbb{F}_{q^n}$ with non-zero square determinant, modulo the equivalence relation $M_1 \sim \lambda M_2$, $\lambda \in \mathbb{F}_{q^n}^*$ [Pet09, PLQ08]. Let $j \in \mathbb{F}_{q^n}$ be

a root of $f(x)$. The set S is taken to be $S = \{s_k\}_{k=0,\ldots,q}$, where

$$s_k = \begin{pmatrix} 1 & \gamma_k + \delta_k j \\ (\gamma_k + \delta_k j + \delta_k)X & 1 \end{pmatrix}, \; j = 0, \ldots, q;$$

and $\gamma_k, \delta_k \in \mathbb{F}_q$ are all the $q+1$ solutions in $\mathbb{F}_q$ for $\gamma_k^2 + \gamma_k\delta_k + \delta_k^2\epsilon = 1$. The Cayley graphs $\Gamma_{q,n} := C_{PSL_2(\mathbb{F}_{q^n}),S}$ are undirected as each s_k has order 2 [Pet09, PLQ08].

The collision and preimage attacks against the LPS hash function can be extended to the Morgenstern hash function, and a variation can be made to the Morgenstern hash that makes it immune to these attacks. This can be seen in more detail in [Pet09], Chapter 6, or [PLQ08].

5.3.5 Pizer hash function

The second of two expander hash schemes proposed by Charles et al. [CGL09] is the Pizer hash function. This expander hash uses the Ramanujan graph family of Pizer, which, unlike previous proposals we have explored, is not a family of Cayley graphs. This section assumes knowledge of basic results on elliptic curves, a summary of which can be found in Appendix B.

The graphs. Let l be a small prime and let p be a large prime such that $p \equiv 1 \pmod{12}$. The vertices of the Pizer graph $\Pi_{l,p}$ are the members of the set V of all supersingular elliptic curves over the finite field $\mathbb{F}_{p^2}$ up to isomorphism. This set has $\lfloor p/12 \rfloor + \epsilon$ elements, where $\epsilon \in \{0, 1, 2\}$ depending on the congruence class of p modulo 12. For the case where $p \equiv 1 \pmod{12}$, $\epsilon = 0$. The elements of V can be labeled by their j-invariants, where we write $E(j_i)$ for an elliptic curve with j-invariant j_i. There is an edge from j_1 to j_2 if and only if there is an l-isogeny from $E(j_1)$ to $E(j_2)$. The Pizer

graph $\Pi_{l,p}$ is a Ramanujan $(l+1)$-regular graph, and it is undirected with no multiple edges if $p \equiv 1 \pmod{12}$ [Pet09, CGL09].

Walking around the graph. To take a step in a walk on the graph, we compute isogenies as described in [CL05], algorithm 1, by explicitly writing down generators for the rank 2 l-torsion and listing the $l+1$ subgroups of order l. For a subgroup $\mathcal{H}$ of the group of points on an elliptic curve E, we give the formulas for determining the equation of the isogeny $E \to E/\mathcal{H}$ and the Weierstrass equation of the curve $E/\mathcal{H}$ when l is an odd prime. Let E be given by the equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

We define the following two functions in $\mathbb{F}_q(E)$ for a point $Q = (x, y)$ on $E - \{O\}$

$$\begin{aligned} g^x(Q) &= 3x^2 + 2a_2x + a_4x + a_6, \\ g^y(Q) &= -2y - a_1x - a_3, \end{aligned}$$

and set

$$\begin{aligned} t(Q) &= 2g^x(Q) - a_1g^y(Q), \\ u(Q) &= (g^y(Q))^2, \\ t &= \sum_{Q \in (\mathcal{H} - \{O\})} t(Q), \\ w &= \sum_{Q \in (\mathcal{H} - \{O\})} (u(Q) + x(Q)t(Q)). \end{aligned}$$

Then the curve $E/\mathcal{H}$ is given by the equation

$$Y^2 + A_1XY + A_3Y = X^3 + A_2X^2 + A_4X + A_6$$

where

$$A_1 = a_1,\ A_2 = a_2,\ A_3 = a_3,$$
$$A_4 = a_4 - 5t,\ A_6 = a_6 - (a_1^2 + 4a_2)t - 7w.$$

From the Weierstrass equation of $E/\mathcal{H}$ we can determine the j-invariant of $E/\mathcal{H}$ by applying our formulas for subgroups of order l. This procedure can be done using $\mathcal{O}(l)$ elliptic curve operations for each of the $l+1$ groups of order l [CGL09].

The collision and preimage resistances of this function are implied by, but not equivalent to, the hardness of some isogeny problems for supersingular elliptic curves that were previously studied by Galbraith [CGL09]: the best algorithm today has a time complexity $\mathcal{O}(p \log p)$ [Pet09]. More detailed security considerations for the Pizer hash function can be found in [Pet09], Chapter 7.

5.3.6 ZesT hash function

Christophe Petit, in his 2009 PhD dissertation [Pet09], constructed the ZesT expander hash function as an improvement to the Zémor-Tillich hash function. This section is derived from said dissertation, and contains a brief overview of the ZesT hash.

A binary polynomial $P_n(X)$ and a vector $(a\ b) \in \mathbb{F}_{2^n}^2$ can be represented as bit sequences of sizes n and $2n$ respectively. In this section, we will often identify a polynomial $P_n(X) = X^n + p_{n-1}X^{n-1} + \cdots + p_1 X + p_0$ to its corresponding bit sequence $p_{n-1} \ldots p_1 p_0$. Moreover, the elements of $\mathbb{F}_{2^n}$ can be seen as polynomials of degree less than or equal to $n-1$ once an

irreducible polynomial has been fixed, hence the vector

$$(a\ b) = (a_{n-1}X^{n-1} + \cdots + a_1X + a_0\ b_{n-1}X^{n-1} + \cdots + b_1X + b_0) \in \mathbb{F}_{2^n}^2.$$

will be identified to the bit sequence $a_{n-1} \ldots a_1 a_0 b_{n-1} \ldots b_1 b_0$.

The ZesT algorithm takes as entry a key made of an irreducible binary polynomial $P_n(X)$ and of a starting point $(a_0\ b_0) \in \mathbb{F}_{2^n}^2 \setminus (0\ 0)$, and a bitstring $m = m_0 m_1 \ldots m_{\mu-1}$ of arbitrary length μ. We define the vectorial Zémor-Tillich hash function on parameters $P_n(X)$ and $(a_0\ b_0)$ by

$$H_{ZT}^{vec}(P_n(X) \| (a_0\ b_0), m) := (a_0\ b_0) H_{ZT}(P_n(X), m).$$

where H_{ZT} is the Zémor-Tillich hash function. The ZesT hash function is defined by

$$H_{\text{ZesT}}(P_n(X) \| (a_0\ b_0), m) := H_{ZT}^{vec}(P_n \| (a_0\ b_0), (m \| H_{ZT}^{vec}(P_n(X) \| (a_0\ b_0), m))).$$

The ZesT algorithm is made of two rounds of the vectorial Zémor-Tillich hash function: After the first round, the intermediary result

$$(a\ b) := H_{ZT}^{vec}(P_n(X) \| (a_0\ b_0), m)$$

is seen as a bit sequence of $2n$ bits that are processed as a continuation of the message bits.

5.4 Cryptanalysis of Expander and Cayley Hashes

An entire paper could be written on the cryptanalysis of any one of the previously discussed hash functions, and as such, we do not have room to fully discuss any or all of them. However, we can, and will, discuss general crypt-

analytic attacks on Expander and Cayley Hashes. Several improvements can be made to generic collision and preimage attacks, which leads to the automorphism attack, subgroup attacks, meet-in-the-middle preimage attacks, improved multicollision attacks, and trapdoor attacks specific to small-girth expander hashes. We then discuss differential cryptanalysis in the context of expander hash functions. This section is derived from [Pet09].

Automorphism Attack. Let $G = (V, E)$ be a k-regular graph. The automorphism attack assumes that there exists an efficiently computable automorphism of the graph such that the average distance from a vertex v to its image $f(v)$ is small. The attack works as follows. Take a random walk w of length μ from the initial point v_0, and let v_μ be the last vertex reached by this walk. Take a random walk w' of length μ' from v_μ, and let $v_{\mu+\mu'}$ be the last vertex reached by this walk. By a brute force attack, search for two paths w_μ from v_μ to $f(v_\mu)$ and $w_{\mu+\mu'}$ from $v_{\mu+\mu'}$ to $f(v_{\mu+\mu'})$. The two paths $w\|w_\mu\|f(w')$ and $w\|w'\|w_{\mu+\mu'}$ solve the constrained two-paths problem with large probability.

Subgroup Attacks. The group structure of Cayley hashes allows for subgroup attacks, a very powerful collision attack technique. Suppose there exists a subgroup tower sequence $\mathcal{G} = \mathcal{H}_0 \supset \mathcal{H}_1 \supset \cdots \supset \mathcal{H}_N = \{I\}$ such that $|\mathcal{G}_{i-1}|/|\mathcal{G}_i| \leq B$ for all i and some computational bound B. By successively moving from $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$, it is possible to reach the identity faster than the birthday attack.

The attack first computes two sets of size about $\sqrt{B}$ of random products of length at most μ_1 of the graph generators s_i. The length μ_1 of the products is chosen in such a way that by taking random products of length at most μ_1 in an appropriate way we roughly get $\sqrt{B}$ different random coset representatives.

Choosing for each left coset of $\mathcal{H}_1$ a representative x, each element g of the first set can be written as $g = xh_1$, $h_1 \in \mathcal{H}_1$. This element is stored in a hash table of size $\sqrt{B}$ which is used to store g and its corresponding message at the address $lb_{\log_2(B)/2}(x)$ (the integer given by the $\log_2(B)/2$ least significant bits of x). Choosing for each right coset of $\mathcal{H}_1$ a representative x, each element g' of the second set can be written as $g' = h_1 x$, $h_1 \in \mathcal{H}_1$. If an element g has been stored at the address $lb_{\log_2(B)/2}(x^{-1})$, then the product $s_{0,1} := g'g$ belongs to $\mathcal{H}_1$. This operation is repeated with another choice for the second set to get a second product $s_{1,1}$ of the graph generators which belongs to $\mathcal{H}_1$.

This trick is iterated from $i = 1$ to N: Two elements $s_{0,i}$ and $s_{1,i}$ of $\mathcal{H}_i$ are obtained by random products of $s_{0,i-1}$ and $s_{1,i-1}$ of length at most μ_i. In the last step, the identity is produced from two elements $s_{0,N-1}, s_{1,N-1} \in \mathcal{H}_{N-1}$. The collision size is about $2^N \prod_{i=1}^{N} \mu_i$, the storage cost is of order $\sqrt{B}$, and the computational cost is about $2N\sqrt{B}$.

Meet-in-the-Middle and Multicollision Attacks. The expander hash design may be seen as a Merkle-Damgård transform of a very simple compression function with message blocks made of only one digit. Because of this, collisions can easily be combined into multi-collisions. For Cayley hashes, if $H(m_1) = H(m_1')$ and $H(m_2) = H(m_2')$, then

$$H(m_1\|m_2) = H(m_1\|m_2') = H(m_1'\|m_2) = H(m_1'\|m_2'),$$
$$H(m_2\|m_1) = H(m_2\|m_1') = H(m_2'\|m_1) = H(m_2'\|m_1').$$

Additionally, as the compression function is invertible, meet-in-the-middle attacks compute pre images in a time equal to the square root of the output size.

Trapdoor attack. For graphs of small girth, the collision resistance of the hash function is not necessarily broken, as the existing small cycles may be hard to find from a given, randomly chosen starting point v_0. However, these graphs are more susceptible to the trapdoor attack if it is possible to find the starting points of short collisions. Suppose that message m and m' have the same hash values when starting from v. An attacker who is given the ability to choose the starting point can choose $v_0 = v$. More generally, if the compression function is efficiently invertible, as is typically the case with expander hashes, the attacker can produce collisions of the form $(m_1\|m\|m_2, m_1\|m'\|m_2)$ by computing the hash function backward from v according to the digits of m_1, then choosing the last vertex reached as a starting point for the hash function.

Differential Cryptanalysis. Differential cryptanalysis is unlikely to work on expander hashes, especially when the expander graph girth is large. We recall that these attacks are typically applied to compression functions made of several rounds. The structure of expander hashes, however, is very different, so the attack requires adaptation. In differential cryptanalysis, the attacker searches for combinations of bit flips in the message whose changes a few rounds later are compensated with a high probability. It exploits the fact that after a small number of rounds, the change induced by some bit flips remain local: it does not influence the whole state of the algorithm. However, for expander hashes, the whole state is updated at each bit, and two states may coincide only after considering a number of rounds equal to the girth. Thus we can see that this attack is rendered impractical for graphs with large girth. For Cayley hashes, differential attacks are best replaced by subgroup attacks.

6 Conclusion

Over the course of this paper we have successfully introduced the notion of an expander hash, discussed the general construction of such a hash function, given examples of expander hash functions in published literature, and identified weaknesses that these hash functions have to certain types of cryptanalytic attack. In this review of the use of expander graphs in cryptography, we aimed to identify the benefits of the cryptographic use of expander graphs as well as some shortcomings they might have. While we were unable to delve deep into the cryptanalysis of published expander hash functions due to space considerations, we were able to direct the reader towards resources that provide full cryptanalysis of each example, as well as identify general cryptographic strengths and weaknesses of expander hash functions.

As long as there is information to protect, there will always be a demand for more secure, more efficient hash algorithms. As such, when a new potential avenue for hash function construction is found, it should be explored as much as possible. With current and future cryptanalysis of expander hash functions, it is our hope that the area of expander hash functions will continue to grow and be explored, and will lead to better cryptographic hash functions.

References

[Bol85] Béla Bollobás, *Random graphs*, Cambridge Studies in Advanced Mathematics, Cambridge University Press, 1985.

[CGL09] Denis Charles, Eyal Goren, and Kristin Lauter, *Cryptographic hash functions from expander graphs*, Journal of Cryptology **22** (2009), no. 1, 93–113.

[Chu08] Fan Chung, *A whirlwind tour of random graphs*, Web, April 2008.

[CL05] Denis Charles and Kristin Lauter, *Computing modular polynomials*, LMS Journal of Computational Mathematics **8** (2005), 195–204.

[Gal14] Jean Gallier, *Notes on Differential Geometry and Lie Groups*, Web, January 2014.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson, *Expander graphs and their applications*, Bulletin of the American Mathematical Society **43** (2006), no. 4, 439–561.

[Jud12] Thomas W. Judson, *Abstract algebra: Theory and applications*, Orthogonal Publishing L3C, 2012.

[Kow13] E. Kowalski, *Expander Graphs*, Web, March 2013.

[LPS88] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak, *Ramanujan graphs*, Combinatorica **8** (1988), no. 3, 261–277.

[Lub11] Alexander Lubotzky, *Expander graphs in pure and applied mathematics*, Bulletin of the American Mathematical Society **49** (2011), no. 1, 113–162.

[Mor94] Moshe Morgenstern, *Existence and explicit constructions of $q + 1$ regular ramanujan graphs for every prime power q*, Journal of Combinatorial Theory, Series B **62** (1994), no. 1, 44–62.

[Nie05] Michael A. Nielsen, *Introduction to expander graphs*, Web, June 2005.

[Pet09] Christophe Petit, *On graph-based cryptographic hash functions*, Ph.D. thesis, Catholic University of Louvain, May 2009.

[PLQ07] Christophe Petit, Kristin Lauter, and Jean-Jacques Quisquater, *Cayley Hashes: A Class of Efficient Graph-based Hash Functions*, Web, 2007.

[PLQ08] ———, *Full cryptanalysis of lps and morgenstern hash functions*, Security and Cryptography for Networks **5229** (2008), 263–277.

[Pre99] Bruno R. Preiss, *Data structures and algorithms with object-oriented design patterns in c++*, John Wiley and Sons, 1999.

[Riv92] R. Rivest, *The MD5 Message-Digest Algorithm*, RFC 1321 (Informational), April 1992, Updated by RFC 6151.

[rJ01] D. Eastlake 3rd and P. Jones, *US Secure Hash Algorithm 1 (SHA1)*, RFC 3174 (Informational), September 2001, Updated by RFCs 4634, 6234.

[Sil08] Joseph H. Silverman, *The arithmetic of elliptic curves*, 2nd ed., Graduate Texts in Mathematics, Springer, 2008.

[Spi11] Daniel A. Spielman, *Spectral Graph Theory and its Applications*, Web, March 2011.

[Tao11] Terence Tao, *254B, Notes 1: Basic theory of expander graphs*, Web,

December 2011.

[Tre11] Luca Trevisan, *Graph Partitioning and Expanders Lecture 2*, Web, January 2011.

[TZ94] Jean-Pierre Tillich and Gilles Zémor, *Hashing with SL_2*, Tech. report, Ecole Nationale Superieure des Telecommunications, 1994.

[TZ08] ———, *Collisions for the lps expander graph hash function*, Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings, Lecture Notes in Computer Science, vol. 4965, Springer, 2008, pp. 254–269.

A Finite Fields

A **group** $(\mathcal{G}, \circ)$ is a set $\mathcal{G}$ together with an operation $\circ : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ on the elements of $\mathcal{G}$ such that the following axioms are satisfied [Jud12]:

- **Associativity:** For all $a, b, c \in \mathcal{G}$, $(a \circ b) \circ c = a \circ (b \circ c)$.
- **Identity:** There exists an element $e \in \mathcal{G}$ such that $e \circ a = a = a \circ e$ for all $a \in \mathcal{G}$.
- **Inverse:** For each element $a \in \mathcal{G}$, there exists an element a^{-1} such that $a \circ a^{-1} = e = a^{-1} \circ a$.

Furthermore, a group is **abelian** if for all $a, b \in \mathcal{G}$, $a \circ b = b \circ a$, and **non-abelian** otherwise. The **order** of a group is its number of elements, and the group is **finite** if the number of elements is finite. A **group homomorphism** is a map between two groups that preserves the group structure: If $(\mathcal{G}, \circ)$ and $(\mathcal{H}, \bullet)$, a group homomorphism $\phi : \mathcal{G} \to \mathcal{H}$ satisfies $\phi(g_1 \circ g_2) = \phi(g_1) \bullet \phi(g_2)$ for all $g_1, g_2 \in \mathcal{G}$ [Pet09].

A **field** $(K, +, *)$ is a set K together with two operations $+ : K \times K \to K$ and $* : K \times K \to K$ such that the following axioms are satisfied [Pet09]:

- $(K, +)$ is a group with identity element written 0.
- $(K^\star, *)$ is a group with identity element written 1, where $K^\star := K \setminus \{0\}$.
- **Distributivity:** For any $a, b, c \in K$, $a * (b + c) = (a * b) + (a * c)$.

The operations $+$ and $*$ are called addition and multiplication, and the operations together with the set K are respectively called additive and multiplicative groups of the field K. A **finite field** is any field whose number of elements is finite, and an isomorphism between two fields K_1 and K_2 is

a bijective map that is a group homomorphism for both the additive and multiplicative groups. For each prime p, the set of integers modulo p is a finite field denoted $\mathbb{F}_p$ for the typical addition and multiplication operations [Pet09].

A **monic irreducible polynomial** $P(X)$ over a field K is a polynomial with coefficients in K whose coefficient of highest degree is $e \in K$ and cannot be factored. For each prime p and monic irreducible polynomial $P_n(X)$ of degree n over $\mathbb{F}_p$, the set of polynomials over $\mathbb{F}_p$ modulo $P_n(X)$ is a finite field denoted F_{p^n} for the typical addition and multiplication operations on polynomials. $\mathbb{F}_p$ is a subfield of $\mathbb{F}_{p^n}$ and is thus called an **extension field** of $\mathbb{F}_p$. The **characteristic** of such a field is p [Jud12, Pet09]. For more information on finite fields, refer to chapter 22 of Thomas W. Judson's Abstract Algebra textbook [Jud12].

B Elliptic Curves

Under some conditions, the discrete logarithm problem on elliptic curves is believed to be much harder than the discrete logarithm problem on the multiplicative group of finite fields of equal size. As a result, Elliptic curves have cryptographic applications in hash functions, public key cryptosystems, symmetric key cryptosystems, and random number generators. This section provides a basic overview of elliptic curves as a requisite for our introduction of the Pizer hash function in section 5.3.5, and is mostly taken from [Pet09].

An **elliptic curve** over a field K is a set of points $(x, y) \in K^2$ satisfying an equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

together with a point at infinity O with no singular point; we will write this set $E(K)$ or, more simply, E. When the characteristic of K is neither 2 nor 3, the curve can be given in Weierstrass form

$$E : y^2 = x^3 + a_4x + a_6$$

by changing coordinates. The set of points of an elliptic curve can be given an Abelian group structure with O as the neutral element; additions formulae are given in [Sil08], Section 3.2. The l-torsion of an elliptic curve is the subgroup made of the points of order dividing l in any sufficiently large extension of K.

Two elliptic curves E and E' are **isomorphic** if there exists a change of coordinates mapping the points of E to the points of E'; isomorphic curves are often thought of as a single curve represented by two different equations. Isomorphic elliptic curves also have the same **j-invariant**, which is defined for Weierstrass equations as $j(E) := 1728\frac{4a_4^3}{4a_4^3+27a_6^2}$.

Given two elliptic curves E, E' defined over the same field, a **homomorphism** from E to E' is a rational map preserving the group addition. An **isogeny** from E to E' is a non-zero homomorphism; its degree is the cardinality of its kernel. An isogeny from E to itself is called an **endomorphism**. The set of endomorphisms of an elliptic curve is a ring, and is isomorphic either to $\mathbb{Z}$, to an order in a quadratic number field, or to an order in a quaternion algebra. An isogeny of degree 1 is called an **automorphism**.

When K is a finite field $\mathbb{F}_q$ of characteristic p, an elliptic curve over $\mathbb{F}_q$ is **supersingular** if for every finite extension $\mathbb{F}_{q^r}$, the curve $E(\mathbb{F}_{q^r})$ has no point of order p. The j-invariants of supersingular elliptic curves are called supersingular j-invariants. The endomorphism ring of a supersingular elliptic surge is an order in a quaternion algebra. For more information regarding elliptic curves, see Silverman's book [Sil08].

C Expander Graphs in Riemannian Geometry

The majority of this section is taken from a paper by Alexander Lubotzky [Lub11]. The Cheeger constant and Cheeger inequality both originally appeared in the context of Riemannian manifolds, so we will take this section to introduce these concepts in their original environment. This section contains material that will be unfamiliar to readers who have not studied Riemannian Geometry.

Let M be an n-dimensional connected closed Riemannian manifold (i.e. compact with no boundary). Let $\Delta = -div(grad)$, where div and $grad$ denote the divergence and gradient on the manifold respectively, be the Laplacian operator of $L^2(M)$, which is the intrinsic Lebesgue space of the Manifold. Its eigenvalues $0 = \lambda_1(M) < \lambda_2(M) \leq \lambda_3(M) \leq \dots$ form a discrete subset (with multiplicities) of $\mathbb{R}_+$ called the **spectrum** of M.

The spectrum of M is very much related to the geometry of M and these relations are the subject of **spectral geometry**. A more intuitive description of Δ is given by the following formula:

$$(\Delta f)(p) = \lim_{r \to 0} \frac{2n}{r^2} \left(\frac{\int_{S_r} f}{vol(S_r)} - f(p) \right)$$

where $n = \dim M$, $p \in M$, $f \in L^2(M)$ and S_r is the sphere of radius r around p. This description is similar to the combinatorial Laplacian as an averaging operator.

We will specifically be interested in the second eigenvalue $\lambda_2(M)$ of the manifold, which can be described as follows:

$$\lambda_2(M) = \inf \left\{ \frac{\int_M \|df\|^2}{\int_M |f|^2} | f \in C^\infty(M), \int_M f = 0 \right\}.$$

Another important geometric invariant of M, whose connection with expander graphs is even more evident, is the Cheeger constant:

Definition C.1. The **Cheeger isoperimetric constant** $h(M)$ is

$$h(M) = \inf_E \frac{\mu(E)}{\min(\nu(A), \nu(B))}$$

where E runs over all the compact $(n-1)$-dimensional submanifolds of M which divide M into disjoint submanifolds A and B, $\mu(E)$ is the "area" of E, and ν is the volume form of M.

Just as for graphs, $h(M)$ and $\lambda_2(M)$ are closely related. In fact, the relation between the two was discovered for manifolds before it was discovered for graphs.

Theorem C.2. *The **Cheeger inequality** for a manifold M is as follows:*

$$\lambda_2(M) \geq \frac{h^2(M)}{4}.$$

A converse to this inequality exists that depends on the Ricci curvature $\mathrm{Ric}(M)$ of the manifold, where the Ricci curvature is defined as follows [Gal14]:

Definition C.3. Let M be a Riemannian manifold with the Levi-Civita connection. The **Ricci curvature**, Ric, of M is the $(0,2)$-tensor defined as follows: For every $p \in M$, for all $x, y \in T_pM$, set $\mathrm{Ric}_p(x,y)$ to be the trace of the endomorphism, $v \mapsto R_p(x,v)y$. With respect to any orthonormal basis, $(e_1, \ldots, e_n)$ of T_pM, we have

$$\mathrm{Ric}_p(x,y) = \sum_{j=1}^{n} \langle R_p(x,e_j)y, e_j \rangle_p = \sum_{j=1}^{n} R_p(x, e-j, y, e_j)$$

where $R_p(x, y, z, w)$ is the Riemannian curvature $(0, 4)$-tensor, $R_p(X, Y)Z$ is the Riemannian curvature $(1, 3)$-tensor, and T_pM is the tangent space of M at a point p.

This leads to the following inequality:

Theorem C.4. *If* $Ric(M) \geq -(n-1)a^2$ *for some* $a \geq 0$ *where* $n = dim\ M$, *then*

$$\lambda_2(M) \leq 2a(n-1)h(M) + 10h^2(M).$$

What is important for us is that in the case of bounded Ricci curvature, which will hold in all our considerations, $\lambda_2(M)$ is also bounded above by a function of $h(M)$.