Implementations of graph theory in Cryptographic technique for secure communication

Ahmed Metwally *[1]

[1]Berkit Elsabaa west, Monofia, Egypt

Abstract: *Graph theory is the study of relationships and connections between different vertices(nodes) that can express anything from tiny bits to the largest systems including networking systems and security systems. The purpose of this study is to explore the available practices of cryptographic algorithms and the implementations of graph theory in cryptography to increase security and confidentiality. For the first section, this paper focuses on mathematical theorems and notions in graph theory that progress in complexity from trees, DAGs, Hamiltonian and Eulerian graphs, in which we will explore the different theorem that led to revolutionize the cryptographic techniques in the last century. The second part of this article discusses how graph theory is used in constructing secure one-way, hash functions using trees and two-way communication techniques using Hamiltonian and Eulerian graphs. Modern encryption to store important data relies on hash functions, which are a form of one-way function. We will also analyze some ways to increase the Merkle hash method using more complicated graphs than trees. We will also investigate communication using adjacency matrices to encrypt graphs in a form of a matrix.*

***Keywords*:** *Hamiltonian and Eulerian graphs, cryptography, hash functions, adjacency matrix, trees.*

Introduction

Graphs are growing more and more popular everyday due to their use in real life. They made the lengthy and tedious data easier to convey and comprehend. For instance, several types of graphs, such as directed graphs, trees, and cycles, are used to describe flow networks, web pages on the internet, and electrical networks. They are a great resource for the study of sequential machines and are used in the abstract representations of computer programs. There are many theories regarding graphs that come from our daily experience and investigation owing to our curiosity. For example, regarded in isolation, there would be nothing intriguing about a person. It is only when you start contemplating his or her ties to the world around, individual becomes intriguing.

Financial databases, geographical and location data, and biological data are just a few of the numerous contexts where graph-structured data models are often used. Finding out whether a graph has been updated or if two graphs are the isomorphic is often important. A key building block of authentication techniques is hashing. Hashing is used to assess these operations. Graph-structured data authentication is

*Email: t.a.metwally35@gmail.com

often required as a database security feature. Since a graph contains many nodes and edges, and each node has several incoming and outgoing edges, hashing graphs presents a variety of challenges.

A key foundational element of modern cryptography is hash functions. Hash functions, message authentication codes, and digital signature methods are all used to check the integrity of data. A hash function is any function that can be used to convert data of arbitrary size to fixed-size values. The words "hash values," "hash codes," "digests," and "simple hash" are all used to refer to the results of a hash function. The values are often used to index a fixed-size table called a hash table. Data integrity protection encompasses not just bit strings but also graphs as data objects. Hash functions must have two key characteristics. 1) They must have a collision-resistant property, which indicates that finding two inputs with the identical output is difficult. 2) They should be preimage resistant, which implies they should be difficult to invert[4]. We will also discuss the encryption-decryption of messages using Eulerian and Hamiltonian graphs. The next section of paper will explore the parts of graph theory that are mainly used to update and increase the security of cryptographic systems.

1 Graph theory topics

Leonhard Euler, a Swiss mathematician and one of the most important mathematicians of the 18th century, was the first to define the basic notion of graphs. His work on the famous "Seven Bridges of Königsberg" issue is often credited with creating graph theory. This abstraction of a concrete issue involving a city and bridges to a graph makes the problem mathematically tractable, since this abstract representation contains just the information required to solve the problem[5]. Euler established that this specific problem has no solution. However, he had difficulty in devising an acceptable technique of analysis and subsequent testing that mathematically confirmed this premise[5]. This subject of mathematics known as graph theory was inert for decades. However, its applications are finally flourishing in modern times.

Graph theory is ultimately a study of connections. Given a collection of nodes and connections that might represent anything from city plans to computer data, graph theory is a valuable tool for measuring and simplifying the numerous moving components of dynamic systems. By employing a framework to analyze graphs, many layout, networking, optimization, matching, and operational issues may be resolved[5].

Graphs may be used to mimic a wide number of linkages and processes in different systems, and thus have a vast array of applications. For instance, graphs are employed to create a safe and robust cryptographic system that cannot be readily compromised by unauthorized users. When encoding a problem with graphs, we must verify that we comprehend the kind of graph involved[5]. The two types of graphs that we will be considering are:

- Digraphs (Graphs with directions)

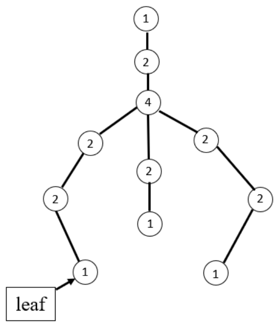
  Orientation or direction between nodes exists in directed graphs. You can only traversal from node A to node B if you have an edge between them.

- Undirected graphs (Graphs that are not directed)

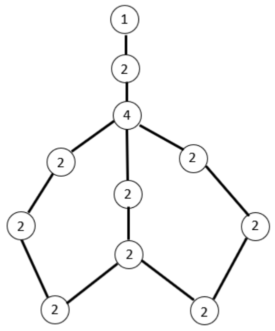
  As the name suggests, there won't be any predetermined paths between nodes. An edge from node A to node B is thus equivalent to an edge from B to A.

1.1 Trees

A graph without cycles is called a tree. The trees outside our windows may be seen as graph-theoretical trees. Cycles are absent from graph-theoretical trees, just as they are absent from the real world's division and reunification of tree branches.



(a) Tree with order 11



(b) A graph with cycles

Figure 1: The difference between trees and regular graphs

The graph in Figure 1(a) depicts a forest that consists only of one tree. A forest is just a grouping of trees. In a tree, a leaf is a vertex of degree one. Graph-theoretic trees exist in a variety of forms and sizes, much as in nature. They can be thin, thick, tall, or short such as $P_{10}, K_{1,1000}, P_{1000}$, or K_1 and K_2, respectively. Even the K_1 and K_2 graphs are classified as trees.

Clearly, the only trees with orders 1 and 2 are K_1 and K_2, respectively. The cardinality of a graph's vertex set determines its order. Lone P_3 is the only tree in the order of three[5]. Figure 1(a) depicts a tree with an order of 11(the number of vertices). Because of this, trees have become popular as role models in a broad range of fields. I will provide a few instances to illustrate my point.

Tree topologies are often used by programmers to facilitate sorting and searching as well as to express the logic of algorithms[5]. In Figure 8, we see an example of a logic tree that can be used to illustrate the logic of a program that tries to discover the largest possible set of four integers (which we call the "maximum" in this case). A binary decision tree is one of them.

1.1.1 Properties of trees

Trees possess many interesting properties that made them a fundamental graph theory topic. All of its properties were implemented in a lot of applications that without it representing data would be much harder than it seems.

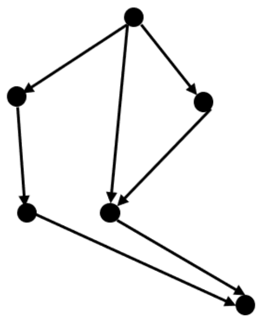
Theorem 1 *If T is a tree of order n, if and only if T is connected and has $n-1$ edges.*
Proof: Let's look at the base case of induction. For a tree of order 1 it will have 0 edges. Our hypothesis holds!! Let's assume the result holds for all tree graphs of order less than $n, n > 1$. Let T be a tree with order k, and v is a leaf of T. $T - v$ will have $k - 1$ edges. From our hypothesis, $T - v$ will have a size of $n - 2$. By adding the vertex v again, the tree T will have a size of $n-1$. So, our hypothesis works for all trees. For the reverse direction, suppose T is connected graph with n vertices and $n - 1$ edges. Let T be not a tree. Thus, it must at least contain one cycle. By removing an edge from this cycle, it will leave us with a tree of $n - 2$ edges which is impossible from the first part of the proof. Thus, our statement holds true[7].

This theorem is used to identify the sparsity of a tree because it leads to many major conclusions such as every tree is $(1, 1)$-tight graph (explained in Section 1.4.).

1.2 Directed acyclic graph (DAGs)

A directed graph with no cycles is known as a *directed acyclic graph (DAG).* It may serve as a model that shows the progression of events. A set of vertices, each representing an activity, are graphically represented by a graph showing the order of the activities[5]. Some of the vertices are connected by lines, reflecting the progression from one action to the next. The term "*directed*" suggests that each edge has a specific direction and that each edge represents a uni-directional flow from one vertex to the next. As seen in Figure 2, a "*acyclic*" network is one that lacks loops (also known as "*cycles*"), meaning that if an edge is followed from one vertex to another, there is no way to go back to the first vertex[5].

What Benefits Do Directed Acyclic Graphs Have? Data processing flows are one sort of flow that may be described using DAGs. Numerous stages and the accompanying sequence for these activities become more clearly structured when considering large-scale processes in terms of DAGs[5]. In a variety of data processing situations, a series of computations are performed on the data to prepare it for one or more ultimate destinations. Data from sales transactions, for instance, may be examined straight immediately to be ready for real-time consumer recommendations. A key characteristic of DAGs and the data processing operations they represent is the existence of several pathways in the data flow. This characteristic is important because it recognizes the need to process data in diverse ways to accommodate different outputs and needs or a single output via several pathways, as illustrated in Figure 2(a)[5].


(a) A DAG graph.

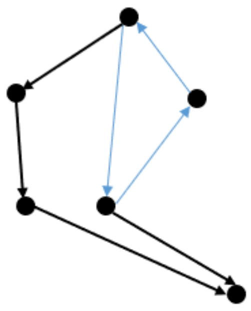

(b) A non-DAG graph

Figure 2: The difference between DAG and non-DAG graphs.

1.3 Sparse graphs

A sparse graph has a total number of edges that is much less than the entire number of possible edges. A. Lee described a sparse graph like this. The graph is (k, L)-*sparse* if each non-empty sub-graph with n vertices has at most $kn - L$ edges. Moreover, if it contains exactly $kn - L$ edges, it is called $(k, L)-$*tight* graph. The degree to which a big network graph deviates from a completely linked graph is reflected by its sparsity. The greater the divergence, the sparser the data[6].

Planar graphs are $(3, 6)-$sparse because every subgraph of a planar graph is also planar. This is due to the fact that every subgraph of a planar graph is likewise planar and that any planar graph with n vertices has at most $3n - 6$ edges, as illustrated in Figure 3. However, planarity is not a property shared by all $(3, 6)-$sparse graphs, which means that not all $(3, 6)-$sparse graphs are planar graphs[6].

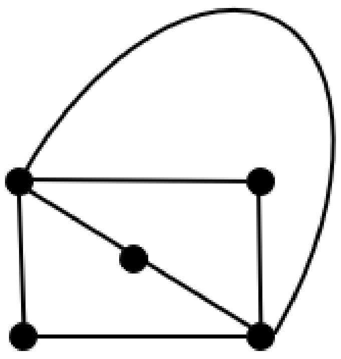


Figure 3: (3,6)-sparse tight planar graph.

1.3.1 Edge density

To measure the sparsity of a graph we usually refer to the edge density. Edge density is the ratio of the edges of a graph to its total possible edges. It is provided by $\frac{|E|}{\binom{v}{2}}$, which represents the edge density of a simple graph $G = (V, E)$. This ratio indicates how near the graph is to becoming complete. A complete graph have an edge density of 1[6].

1.4 Eulerian and Hamiltonian graphs

A path known is a Eulerian trail if and only if it crosses each graph's edges once. It might finish on a different vertex than where it started, which is called a traversable graph. A Eulerian circuit is a circularized Eulerian trail. Alternatively, it starts and finishes at the same vertex. If a graph has a Eulerian circuit, it is referred to as a Eulerian graph[5].

A Hamiltonian circuit is a closed route that precisely traverses each vertex of a graph. The starting point of a Hamiltonian circuit is where it ends. After the Irish mathematician Sir William Rowan Hamilton, who flourished in the nineteenth century, Hamiltonian graphs were given their name. The traveling salesperson or postman paradox is the name for this kind of problem[5].

A graph is referred to be a Hamiltonian graph if it contains a Hamiltonian circuit. While a Hamiltonian circuit traverses each vertex in a graph precisely once, a Eulerian circuit visits each edge inside a graph exactly once.

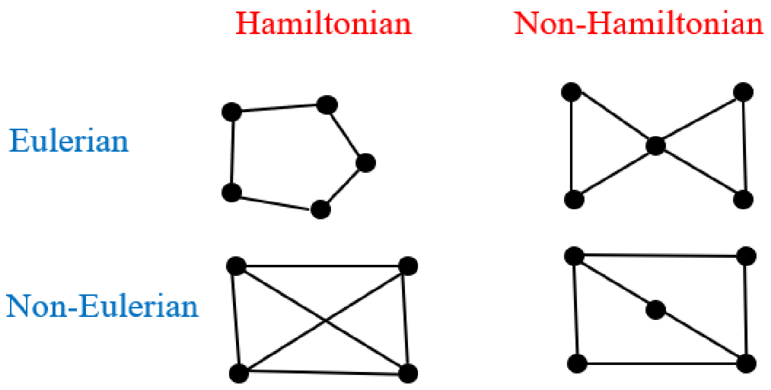


Figure 4: Comparison between Eulerian and Hamiltonian graphs

2 Cryptography

Cryptography is the study of secret writing with the purpose of hiding the meaning of a communication. Historically, spies and combatants have used encryption. Every day, billions of people use it to safeguard their electronic communications and financial transactions. Numerous disciplines are affected by cryptography, including mathematics and CS(computer science)[6]. Cryptology is an area of mathematics that utilizes graph theory, number theory, statistics, and probability. Computer science theories, namely complexity theory, are used to examine cryptography security. The challenging work of performing security studies for specific cryptosystems is within the purview of engineering, and practical computer science. In this part, we will discuss and demonstrate one-way hashing techniques, as well as how they are constructed using trees. We'll also discuss a strategy for encryption and decryption based on Eulerian graphs, as well as how the computer reads the graphs using the adjacency matrices.

2.1 Definitions

In this section we define some important definitions that we use in this paper.

- **Encryption:**

  It is the conversion of information into a code, particularly for the purpose of restricting access to unauthorized people.

- **Decryption:**

  It means to restore data in its original form after it has been encrypted.

- **Plain text:**

  It is an ordinary text that can be read before being converted to cipher text or after being decoded.

- **Cipher text:**

  A cryptographic algorithm transforms plain text into encrypted text, which is referred to as cipher text. Plain text can only be deciphered from cipher text.

- **Hash function:**

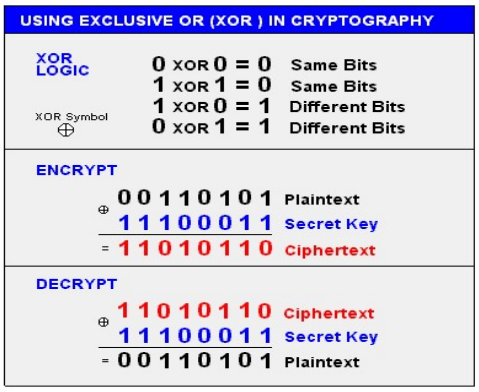
  A hash function is any function that may be used to convert data of arbitrary size to fixed-size values. The values generated by a hash function are referred to as hash values, digests, or simply hash. The values are often used for indexing a fixed-size table known as a hash table.

- **XOR:**

  In cryptography, the simple XOR cipher is a kind of additive cipher and an encryption method based on the following principles:

  - $A = A \oplus 0$,
  - $0 = A \oplus A$
  - $A \oplus (B \oplus C) = (A \oplus B) \oplus C$

  where $\oplus$ denotes the XOR (exclusive-or) operation. Addition modulus 2 is another name for this process. An encrypted string of text is created this way by using the bit-wise XOR operation to each character with a supplied key. To remove the cipher from the output, just perform the XOR function with the key. Since XOR can only return true if both inputs are distinct, it is used to test for equality between two expressions[1].



USING EXCLUSIVE OR (XOR) IN CRYPTOGRAPHY

XOR LOGIC

0 XOR 0 = 0 Same Bits
1 XOR 1 = 0 Same Bits
1 XOR 0 = 1 Different Bits
0 XOR 1 = 1 Different Bits

XOR Symbol ⊕

ENCRYPT

```
  0 0 1 1 0 1 0 1  Plaintext
⊕ 1 1 1 0 0 0 1 1  Secret Key
= 1 1 0 1 0 1 1 0  Ciphertext
```

DECRYPT

```
  1 1 0 1 0 1 1 0  Ciphertext
⊕ 1 1 1 0 0 0 1 1  Secret Key
= 0 0 1 1 0 1 0 1  Plaintext
```

Figure 5: XOR technique [2]

- **Adjacency Matrix:**

  Let $G = (V, E)$ be a graph with $V = \{v_1, v_2, \ldots, v_n\}, E = \{e_1, e_2, \ldots, e_m\}$ and without parallel edges. The adjacency matrix of G is a symmetric binary matrix $X = [x_{ij}]$.

  It is used to represent whether a pair of vertices in a given graph. Two vertices' connection in the adjacency matrix is represented by x_{ij} where i is the number of the row, and j is the number of the column for the the element x_{ij}. The values of x_{ij} are either 1(connected) or 0(disconnected)[1]. The next example shows the correspondence between adjacency matrix X and graph in Figure 6. Notice that $[x_{2,2}]$ is 1 because a_2 is connected to itself. Whenever there is a 1 on the diagonal, there must be a vertex that is connected to itself.

$$[x_{i,j}] = \begin{cases} 1 & \text{if } v_i \text{ and } v_j \text{ are connected with an edge,} \\ 0 & \text{otherwise.} \end{cases}$$

$$X = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

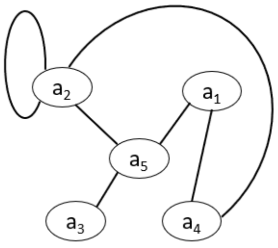


Figure 6: Graph of matrix X

Theorem 2 *Let $G = (V, E)$ be a graph with $V = v_1, v_2, ..., v_n$ and let X be its corresponding adjacency matrix. We can argue that $\forall k$, the (i, j) entry of X^k is equal to the number of walks from v_i to v_j that use exactly k edges.*
Proof:
This can be proved by strong induction over the size of k.
Base case:
For $k = 1$, $[X]_{i,j} = 1$ because there exists only one edge between v_i and v_j. Suppose that the result holds true $\forall i$ and $\forall j$. So that, the number of walks from v_i to v_j is X^{k-1}, which uses exactly $k-1$ edges. There exists an h such that the walk can be thought of as a $(k-1)-$edge walk from v_i to v_h combined with an edge from v_h to v_j[5]. The total number of these $k-$edge walks is the sum of number of $(k-1)-$edge walks from v_i to v_h which can be written as:

$$\sum_{v_h \in \mathbb{N}(v_i)} (\text{the number of}(k-1)\text{-edge walks from } v_i \text{ to } v_h)$$

Using induction, we can rewrite as

$$\sum_{v_h \in \mathbb{N}(v_i)} [X^{k-1}]_{i,h} = \sum_{h=1}^{n} [X^{k-1}]_{i,h}[X]_{h,j} = [X^k]_{i,j}$$

So, this result holds true ∀k ∈ ℕ.

To check Eulerian graphs with adjacency matrix, we should count the number of non-zero elements in each row. If the count is even, then the graph is Eulerian. Otherwise, it is not.

2.2 Hamiltonian and Eulerian graphs

Cryptography is a kind of algorithm that allows for secure communication. We will outline a process in this section for encrypting each character of data into a Eulerian graph. A key known as a Hamiltonian Circuit is used to safeguard the data, so unauthorized people won't be able to decrypt. Decryption is therefore very difficult without knowledge of the Hamiltonian circuit and the encoding scheme. With this encryption method, the complexity and unpredictable nature of decryption and interpretation of the true message are fairly significant and difficult since each graph represents a character in the message. The data security is safeguarded by this algorithm.

2.2.1 Encrypting and decrypting Eulerian graphs

The following encryption method will convert the user's intended message into a Eulerian Graph. We shall be able to extract a Hamiltonian circuit from the encrypted graph. This will act as the decryption key.[1] Encrypting the plain text using Eulerian graph:

1. Converting all letters to alphabet
2. Convert all values to its equivalent ASCII(American Standard Code for Information Interchange).
3. Convert ASCII to binary
4. It should be XOR^{ed} with its binary equivalent of 32.
5. Store the results in an array $X[i]$.

6. Determine K(the number of 1's in the array $X[i]$).
7. Make an adjacency matrix $A = (a_{ij})$ with the count k. It should be symmetric.
8. Count the numbers of $0's$ following 1's, add one to the count, and store it in a_{ij}. For an instant, if 10 is in the array $a_{ij} = 2$. If 1000 is in the array, $a_{ij} = 4$.
9. Repeat till the Hamiltonian circuit reaches the end vertex.
10. This adjacency matrix is sent to the receiver.

Decrypting cipher text:

1. The matrix is stored in an array $Z_{[p]}$.
2. We take into consideration only either the upper or lower triangular because the matrix is supposed to be symmetric.
3. The elements of $Z_{[p]}$ are expanded to build the binary stream,
4. Trace the encrypting procedures backward[1] .

Here is an example:

Let GRAPH be our plain text. Create binary strings from each letter in the plain text. The 32 bit binary string and each alphabet are then XORed[1]. This table represents all the conversions:

Alphabet	ASCII Code	Binary Number	XORed
G	71	1000111	1100111
R	82	1010010	1110010
A	65	1000001	1100001
P	80	1010000	1110000
H	72	1001000	1101000

The adjacency matrix of each alphabet is created using the encryption technique and the Eulerian graph, and is shown below:

Since the alphabet G includes five 1s, its graph has five vertices, and as a result, its adjacency matrix is of the 5x5 order shown below:

$$G = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 3 & 0 & 0 \\ 0 & 3 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}, R = \begin{bmatrix} 0 & 1 & 0 & 2 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 3 \\ 2 & 0 & 3 & 0 \end{bmatrix}, A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 5 \\ 1 & 5 & 0 \end{bmatrix}, P = \begin{bmatrix} 0 & 1 & 5 \\ 1 & 0 & 1 \\ 5 & 1 & 0 \end{bmatrix},$$

$$\text{H}= \begin{bmatrix} 0 & 1 & 4 \\ 1 & 0 & 2 \\ 4 & 2 & 0 \end{bmatrix}$$

For matrix H, there are 3 1's, so it is 3×3 matrix. All these matrices are sent to the receiver one by one. From step 8, we count the number of 0's after the 1's and add one to each count. This will yield 1,2,4 which are assigned in the matrix. Note that the last element in this array, which is 4, was put on the opposite corners of the matrix. By repeating this same algorithm for all letters, we can generate all the matrices.

For G, the recipient receives [1 3 1 1 1], which is the count of zero's after the 1's in the XORed code + 1.

For R, the recipient receives [1 1 3 2].

For A, the recipient receives [1 5 1].
For P, the recipient receives [1 1 5].
For H, the recipient receives [1 2 4].
By expanding these matrices, the receiver gets

Array	XORed
13111	1100111
1132	1110010
151	1100001
115	1110000
124	1101000

Using encoding technique in reverse order

Letter	XORed
G	1100111
R	1110010
A	1100001
P	1110000
H	1101000

As a result, the recipient gets the plain text that the sender initially sent. [1]

2.3 Hashing functions

When utilizing a random oracle as a graph hash function, all random oracle versions must generate the same hash result. In hashing algorithms like SHA-1 and SHA-2, a message is either wholly shared with a user or it is entirely not shared. However, when using graphs, a user may obtain one or more subgraphs of a graph. The challenge is determining how the hash for a graph is created for an updated graph and how much it costs. Furthermore, certain parts of the graph may be updated while others stay intact. Graph models are also occasionally used to represent sensitive data (such as passwords or signature verification). Traditional hashing techniques allow information to be leaked (SHA-1 and SHA-2 do not express the length of the plain text). When dealing with probabilistic polynomial adversaries, the goal is to hash graphs in such a manner that no information is revealed. Aside from dealing with trees, the Merkle hash method (MHT) has been expanded to produce a hash for directed acyclic networks. The Merkle hash method and its extensions, on the other hand, do not address all of the aforementioned concerns. [3]

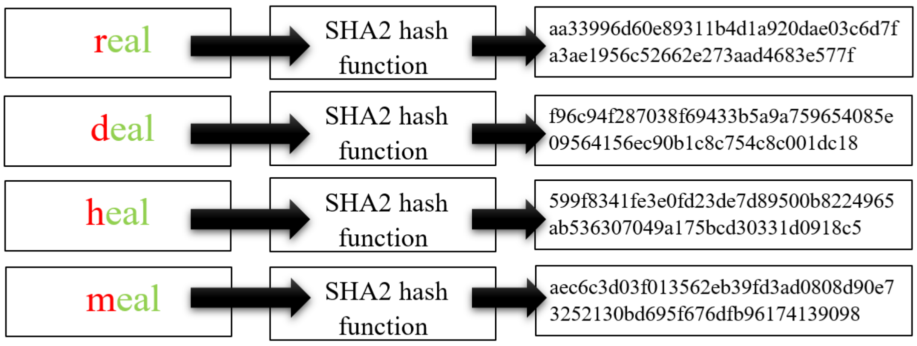

real
SHA2 hash function
aa33996d60e89311b4d1a920dae03c6d7f a3ae1956c52662e273aad4683e577f
deal
SHA2 hash function
f96c94f287038f69433b5a9a759654085e 09564156ec90b1c8c754c8c001dc18
heal
SHA2 hash function
599f8341fe3e0fd23de7d89500b8224965 ab536307049a175bcd30331d0918c5
meal
SHA2 hash function
aec6c3d03f013562eb39fd3ad0808d90e7 3252130bd695f676dfb96174139098

Figure 7: The digest after changing one letter

Figure 7 displays the change in the hashing code in SHA-2 just by changing one letter in the original message. It is very difficult for unauthorized authors to observe any similarities between the hashing digest. This digest has fixed number of bits that it can represent any length of input plain text. So, the no one can have such information about the length of the plain text itself.

2.3.1 Merkle hash techniques with trees

The Merkle hash technique worked the leaf to the root. For a node v in tree $T(V, E)$, it computes a Merkle hash(MH) $mh(v)$ as follows: if v is a leaf node, then $mh(v) = H(c_v)$; else $mh(v) = H(mh(n_1)||\ mh(n_2)||mh(n_3)||....||mh(n_m)))$, where $n_1, n_2, n_3, \ldots, n_m$ are the children of v in T in that order from left to right[4].

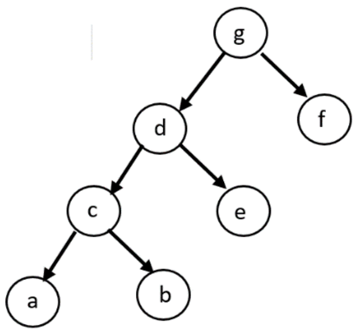


Figure 8: Merkle tree

For example, consider the tree in Figure 8. The Markel hash for this tree is computed as follows. The MH of a and b are computed as $H(c_a)$ and $H(c_b)$, respectively, which are then used to compute the MH of c as $mh(c) = H(mh(a)||mh(b))$. The MH of d is computed as $H(mh(c)||mh(e))$. Similarly, MH of f and g are computed as $H(c_f)$ and $H(mh(d)||mh(f))$, respectively. $||$ means putting the two mh functions together and working on both of them as one input[4].

To account for non-leaf's node contenet, two straightforward Markle hash variations may be used to calculate the MH of a non-leaf node using the MH of its children and the hash of the non-leaf node's content. Suppose V to be a non-leaf node in T. the MH of V is defined as follows: $mh(V) = H(H(c_V)||mh(n_1)$ $||mh(n_2)||mh(n_3)||....||mh(n_m))),$

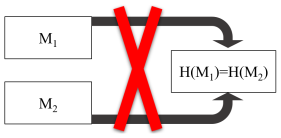
Consider again the tree in fig(8). MH of d and g are computed respectively as $H(H(c_d)|(|mh(c)|)|mh(e)), H(H(c_g)||mh(d)||mh(f))$. There are many ways to construct hashing trees and the more complicated the tree is the more secure the hashing is[4].

2.3.2 Inefficiency of updates

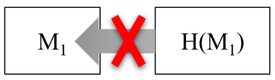
Consider the situation when node a in Figure 8 has been altered. This update has an impact on the hash values of c, d, and g because the values of the hash functions depend on each other. A single adjustment resulted in three different outcomes. A change in one node in MHT may result in changes in $O(n)$, such that n represents the number of nodes in the tree. This has a lot of performance implications, especially in terms of storage stability, data integrity, and index integrity.

2.4 Collision-resistant hashing of graphs

The highest degree of security is achieved by hash functions that are resistant to collision, second pre-image, and pre-image attacks. The most potent idea is collision resistance, which includes second pre-image resistance and pre-image resistance in turn. Every hash function that satisfies (1) also satisfies (2) and (3), but not the other way around. The greatest characteristic, collision resistance, is referred to as the security of hash functions throughout the whole research[4].



(a) Collision resistance



(b) Preimage resistance

Figure 9: Difference between collision resistance and preimage resistance

The traditional definition of hashing algorithms cannot be directly applied to graphs since they function on messages $v \in \{0,1\}^*$ and each message is either entirely shared or not shared at all with a user. Each node in a graph $G(V,E)$ is represented by the symbol v, and a user may retrieve more than one subgraphs rather than the whole graph[4].

As already mentioned, a hashing method for graphs requires both a key creation process and a hash function algorithm. A method for verifying a graph's hash and that of its subgraphs would also be necessary. We call this strategy the "*hash-verification*" approach. This is so that users may validate the integrity of any complete or subgraphs they may get.[4] The user will need further information in addition to the appropriate subgraphs if the hash function used to create a graph's hash is a collision-resistant hash function for that reason. Otherwise, the hash value determined by the user for the received subgraphs would not match the hash value of the graph, unless there is a contradiction to the assumption that the hash function is collision resistant. The term "*verification objects*" (VO) refers to this data. The computation of the VO for a subgraph with relation to a graph is another operation that is included in the explanation of the hashing technique for graphs. This method is known as "*graph hash-redaction.*"[4].

2.5 Sparsity and security

Cryptographically resistant hash functions are built using sparse graphs. A sparse network with high connection features, characterized by vertex, edge, or spectral expansion, is known as an expander graph. Expander graphs are formally described as graphs in which each "not too enormous" subset of vertices has a neighbor set that contains a sufficient number of extra vertices. The hash function's output is the ending vertex, and its input is used as walking directions around a graph[6].

3 Conclusion

Graph theory is a crucial area in many disciplines. When it comes to cryptography, graphs are a very effective tool. Due to its many features and easy representation in computers as a matrix, graph theory is particularly often utilized as a tool for encryption. Graphs may be created from ciphers to facilitate secure communication. Numerous methods that provide one-way (hashing) and two-way communication have been created by scientists, but graphs were the most successful techniques. Of all graphs in this discipline, trees and Eulerian graphs are the most significant. The use of tree-structured hashing algorithms can avoid collisions and adds a level of complexity does not present in earlier hashing processes.

References

[1] P Amudha, AC Charles Sagayaraj, and AC Shantha Sheela. "An application of graph theory in cryptography". In: *International Journal of Pure and Applied Mathematics* 119.13 (2018), pp. 375–383.

[2] Ziff Davis. *pcmag*. 2020. URL: https://www.pcmag.com/encyclopedia/term/xor.

[3] Oded Goldreich. *Modern cryptography, probabilistic proofs and pseudorandomness*. Vol. 17. Springer Science & Business Media, 1998.

[4] Ashish Kundu and Elisa Bertino. *On Hashing Graphs*. Cryptology ePrint Archive, Paper 2012/352. 2012. URL: https://eprint.iacr.org/2012/352.

[5] Gerry Leversha. "Combinatorics and graph theory, by John M. Harris, Jeffry L. ISBN 0 387 98736 3 (Springer-Verlag)." In: *The Mathematical Gazette* 86.505 (2002), pp. 177–178.

[6] Natalia Tokareva. "Connections between graph theory and cryptography". In: *G2C2: Graphs and Groupts, Cycles and Coverings, September* (2014), pp. 24–26.

[7] Carl Yerger. "Course notes". In: (2022), pp. 1–6.