The Graphen Protocol: A Decentralized Graph Database

Matthieu Gavaudan[1], Peiqi Jin, Yang Yang[2] and Ching-Yung Lin[3]

Abstract— **Scalable computing power is starting shift from centralized data centers to decentralized networks of personal devices. In order to keep up with this paradigm shift, new software technologies must adapt. This paper describes a decentralized graph database protocol that leverages cryptographic and economic techniques in order to guarantee the security, privacy and accuracy of its data and queries.**

The system is completely peer to peer and allows for anyone to rent out computational and storage power in order to participate in the network. The owners of these machines can be rewarded with monetary compensation such as US Dollars or cryptocurrencies. This enables users interacting with the graphs to access their data in a trustless, permissionless way, opening the door to new opportunities for true data ownership.

I. INTRODUCTION

A. Problems

The technical problems the Graphen Protocol attempts to solve are twofold: the ones presented by current cloud database solutions and the ones presented by existing distributed ledger technologies (DLTs) such as blockchains[1] and directed acyclic graphs[2].

Starting with the cloud database solutions, we identified three fundamental flaws that would be hard or impossible to fix without the use of decentralized networks.

The first and most important one is security: while your data may be replicated across two, three or ten data-centers across the world it still leaves a single point of failure: the company hosting the data. The advent of cloud computing has led companies to store data in an increasingly centralized manner, thereby telling hackers exactly where all the important information is being kept. Destroying or getting access to all of your data is as easy as hacking the one company that hosts said data on its servers. There have been 2,216 data breaches in 2018 alone according to a Verizon data breach report [3], with the biggest ones affecting companies like Facebook, Google and Amazon.[4]

The second flaw will affect cloud database's latency in the coming 3-4 years. Centralized data centers at large technology companies are starting to hit a bandwidth ”wall” [5] that cannot be fixed without a drastic change. Incremental progress is no longer going to cut it. On the other hand, every year, our personal computing devices get ever more powerful. Moore's law predicted this phenomenon by claiming that transistors on chips would double every year while costs would get halved. What few foresaw was the fact that these devices are now wasting more compute cycles than ever, sitting idle most of the time. ”Fog Computing” and ”Edge Computing” have been proposed as potential solutions, but have so far failed to reach mainstream use.

Finally, our last flaw involves trust and collaboration. Within this particular problem, there actually exist two subproblems: censorship resistance and sharing of data. Cloud databases are stored in highly centralized data centers, spread throughout the world, and are subject to the laws specific to the country they are located in. This means that if your data were to ever be transferred to a country who's jurisdiction allows for censorship you could entirely lose access to it or have your privacy be invaded.

Companies looking to grant their users true data ownership cannot with the current cloud computing paradigm. Data centers are centralized, answer to a single authority and are therefore not permissionless or trustless, two crucial pillars for data ownership.

Similarly, existing DLTs are not perfect either and, in fact, present a significant amount of problems. We will go over the relevant ones and provide concrete examples as to why they have a very narrow use case that cannot be applied to most problems in a post cloud computing era.

The first and most obvious one is accessibility. Blockchains and directed acyclic graphs are, fundamentally, just decentralized immutable databases. Yet, manipulating data within these so called databases has never been harder: there exist no innate querying languages like SQL. The only real existing solutions are apps that offer very limited GUIs for querying the contents of a wallet or a contract. This problem also led to the rise of third party blockchain analysis companies like Chainalysis[6] whose job entirely consists of querying the blockchain in ways that should be trivial for normal graphs or relational databases.

The second problem is the current distributed ledger technologies' transactional based architecture. This is a much deeper flaw that would require the complete revisal of the structure of directed acyclic graphs and blockchains.

Blockchains and DAGs were built with transaction storage in mind, and not anything heavier. As a result of that, they are only able to contain small amounts of data without scaling node resources and becoming centralized. In fact, it costs millions of dollars to store just 1 GB worth of data on Ethereum or Bitcoin [7].

Furthermore, blocks are created irregularly with transactions issued by the users that paid the most, and not according to their timestamp. This is a fundamental flaw because querying data formatted in that way is far from

[1] Matthieu Gavaudan is a Product Lead at Graphen, 500 5th Avenue, New York, NY `mgavaudan at graphen.ai`

[2] Peiqi Jin and Yang Yang are research scientists Graphen, they contributed equally and are listed alphabetically. `[peiqijin, yanyang] at graphen.ai`

[3] Ching-Yung Lin is the CEO at Graphen

optimal. This also means that while data is secure on the blockchain, it is not accessible in an efficient manner.

Imagine an enterprise wanted to manipulate medical data on the blockchain but couldn't do it fast enough to efficiently attend to their patients, they would likely have to resort to a third party service that sorts the data in a time series database and that then provides an API to it. While this would solve the speed conundrum, it destroys the entire concept of decentralization since we once again have a single point of failure: the company sorting the data.

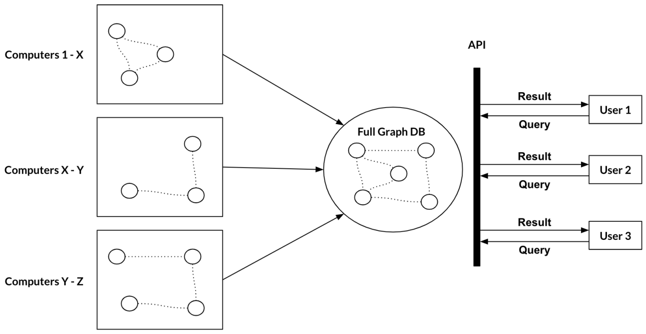
In conclusion, we can definitively say that current DLTs were not built for the storage or manipulation of heavy data in mind and have not since made much progress towards that end.

B. *The Graphen Protocol*

The Graphen Protocol is our solution to the aforementioned problems. It combines the best of the distributed ledger technology and cloud database industries. This section will serve as a brief overview of Graphen Protocol's architecture; each paragraph is further clarified in subsequent sections of the whitepaper.

From a user experience perspective, the Graphen Protocol functions exactly like current cloud graph database solutions, with the same capabilities for queries, storage and complex data manipulation. Users are able to create their own graph databases and have them hosted on a network of masternodes. Masternodes in our system are users that rent out their storage and computational power in return for monetary rewards (similar to Bitcoin miners).

Each graph database isn't entirely replicated across each masternode but rather is sharded in separate subgraphs' so that each masternode isn't overloaded with data. Queries to the graph are broadcasted to the network of masternodes and executed for a fee. The queries are distributed across multiple masternodes storing the relevant data in order to maximize efficiency and throughput.



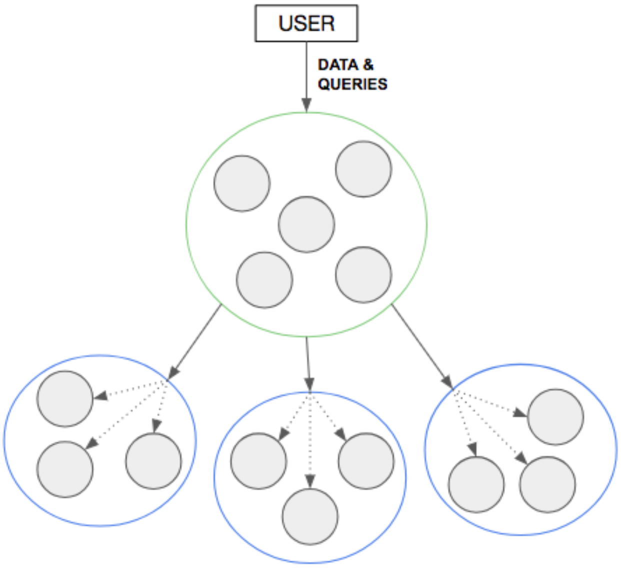
The user-created graphs can be either public or private. If a graph is created as private by its user, the data and queries sent to the masternodes hosting the graph are encrypted so that perfect privacy can be maintained. Furthermore, the result of queries/computations sent to masternodes are guaranteed to be accurate via a verifiable computing algorithm and a reputation system.

Users are also free to choose which method of payment they would like to offer masternodes. This can range from digital assets such as Bitcoin, Ethereum or stablecoins to US Dollars. Depending on the desired security for a given graph, a user can also decide how much a masternode should stake in order to participate in the network. However, a higher stake will correlate with higher storage and query fees.

II. Protocol Architecture Overview

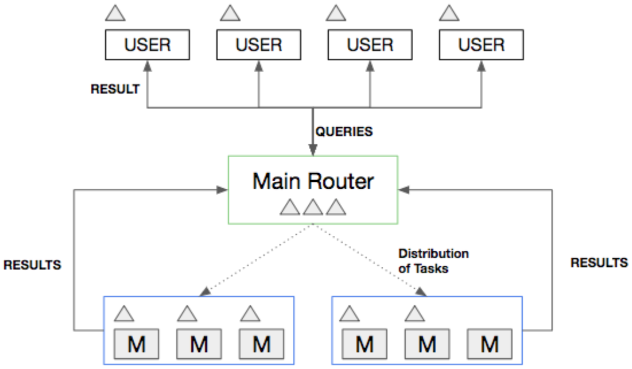
A. *Physical Architecture*

The SmartGraph[8] technology is the inspiration for the high level architecture of the Graphen Protocol. Instead of being vertex centric, the SmartGraph adopts the "Think Like A Router" mentality whereby several vertices are grouped together to form a router. Routers in the Graphen Protocol are simply a collection masternodes working together to maximize security, throughput and fault tolerance.



The figure above is a physical representation of the machines that store a single graph. The small gray circles each represent a single masternode. The blue circles are routers, and the masternodes they contain store the same subgraph. Finally, the green circle containing the five (arbitrary number) masternodes is the "main router" who's main responsibility is to redirect data flow and queries to the appropriate subgraph routers.

The Graphen Protocol has the aforementioned architecture for several reasons. First of which is the random replication of data across machines around the world. The second reason is for the distribution of tasks. The SmartGraph uses artificial intelligence to manage what parts of the graph routers should store in order to maximize efficiency. It does so by checking what queries are often being run and on what parts of the graph and then adopts a distributed task queue system within its own routers and masternodes.



In this figure, the triangles represent tasks (such as queries or data updates) and are handed down from users to the main router and then to the masternodes within the appropriate routers. After completing the assigned tasks, the masternodes return the information or confirmation required to the main router which, in turn, sends it back to the user if needed.

B. Routers

1) The Main Router: Each graph has its own dedicated main router. The main router's purpose is to keep track of subgraphs and their assigned routers and masternodes. It is the main point of contact for users and masternodes in a graph.

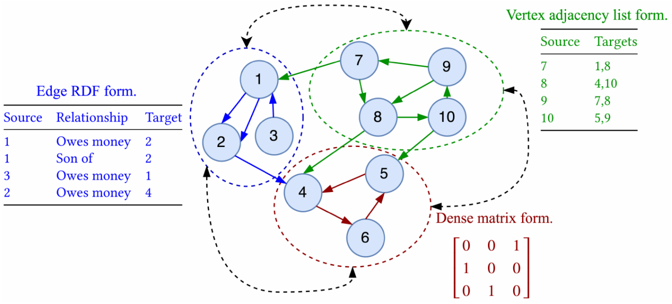
Its selection is done via voting once the graph is created and the routers chosen. We use verifiable delay functions [9] in order to create a fair and secure process to choose the main router.

For example, a malicious adversary might try to manipulate the outcome of the vote by refusing to participate after seeing the other masternodes' picks or by trying to sell their swing vote, thereby corrupting the voting process.

Verifiable delay functions are separated into three steps:

- Step 1 : A Setup function that creates a puzzle with a given difficulty and outputs a verification and an evaluation key. In our scenario, every masternode participating in the vote starts by running setup with a given minimum difficulty.
- Step 2: The Eval function takes the evaluation key and an input and outputs a number and a proof. Here the input is the masternode's vote.
- Step 3: The Verify function takes an input, an output (the number from Eval) and proof and outputs a boolean. Once all masternodes have voted, they can run the verify function on the other masternode's eval output number and proofs to verify their vote.

2) The Regular Router: Every graph stored on the Graphen Protocol is sharded into several subgraphs. Each subgraph contains a set of vertices with their corresponding edges. Each router stores a single subgraph and is connected to the other routers via the edges that originate in one subgraph and end in another. The subgraphs are stored under the RDF convention. The figure below illustrates this system, where the dotted circles represent the information contained within a single router and the blue circles are graph vertices.



Edge RDF form.

Source	Relationship	Target
1	Owes money	2
1	Son of	2
3	Owes money	1
2	Owes money	4

Vertex adjacency list form.

Source	Targets
7	1,8
8	4,10
9	7,8
10	5,9

Dense matrix form.

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

III. Security & Privacy

A. Merkle-Patricia Trees

In order to increase the security of the Graphen Protocol, we decided to have each masternode in the system store three Merkle-Patricia trees (abbreviated to MP tree):

- Router MP Tree: Every masternode in a router stores the same subgraph. This Merkle-Patricia tree's leaf nodes will contain the vertices of the subgraph and an adjacency list of their edges.
- Graph MP Tree: Every masternode in a given graph will also store a Merkle-Patricia tree of the entire graph. However, here, the leaf nodes will be the hash roots of the router trees.
- Graphen Protocol MP Tree: Every masternode in the Graphen Protocol will also store the entire protocol's Merkle-Patricia tree, whose leaf nodes will be the hash roots of the graph MP trees.

These three Merkle Patricia Trees are small enough that from a storage standpoint they are insignificant, and yet, they allow for masternodes to considerably improve the overall Graphen Protocol security.

Consider the following scenario:

- A group of masternodes try to delete or corrupt data on a graph.
- They alter the subgraph data they possess.
- If even a single masternode in the same router detects this error, a conflict resolution vote is initiated.
- Every masternode in the graph (and even possibly any masternode in the system) can vote on which subgraph they believe is correct. They can do so by checking their Merkle-Patricia tree of the graph (which contains the hash root of all router subgraphs in the system) and verify what the current state of the subgraph should be.

B. Query On Encrypted Data

Since we want our users to be able to retain their data privacy we decided to add an encryption feature so that even the masternodes hosting user data will not be able to read it. The problem presented now is how do the masternodes compute queries on data they cannot decrypt? This section gives an high level overview of our solution.

1) Adjustable Query-based Encryption: There exist multiple levels of encryption suited to different sorts of queries. This commonly referred to as onion encryption for adjustable query-based encryption[10][11]. Onion encryption can change the data's encryption levels to suit a given query type. For example, if a user wishes to just store data without being able to manipulate it, he/she will select RND, the strongest encryption algorithm for the data. As the complexity of the data manipulation increases, the encryption algorithm becomes less and less secure. However, even one of the lowest levels of encryption (homomorphic encryption) has been shown to be extremely secure.

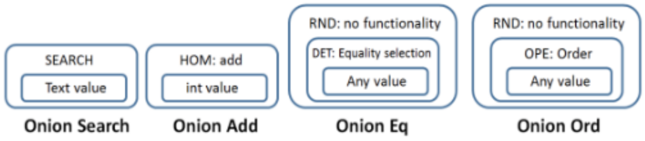


Figure 1: Onion layers of encryption.

As we briefly mentioned above, there are different layers of encryption a user can request. More specifically there are five:

RND (Random): RND provides the most security. Even two equal values will be mapped to different ciphertexts (a ciphertext is the result of applying an encryption algorithm to plaintext). Computations cannot be performed efficiently on the ciphertext with RND.

DET (Deterministic): DET generates same the ciphertext for the same plaintext. This layer allows the masternodes to compare two vertices for equality.

OPE (Order-preserving encryption): OPE establishes order relations between data based on the encrypted values without leaking the data itself. If x¡y, then OPEK(x)¡OPEK(y).

HOM (Homomorphic encryption): HOM allows for masternodes to perform computations on encrypted data and for users to decrypt the final result. It has also been proved that HOM is efficient and secure.

SEARCH (Word Search): SEARCH allows for keyword search on encrypted text. Moreover, SEARCH allows masternode to detect repeating plaintext in their stored data.

2) *Querying Process:*

- Step 1: The user issues a query.
- Step 2 : The user checks if the masternode should be given keys to adjust encryption layers before executing the query, and if so, the user strips off the onion layers by sending the corresponding keys to the masternodes.
- Step 3: The user rewrites his or her query to an encrypted format and adjusts the encryption layer, anonymizing vertices, edges, properties and any constants.
- Step 4: The user sends the encrypted query to the masternode who executes it and returns the encrypted result.
- Step 5: The user decrypts the received piece of information and gets the plaintext result.

IV. Network Functions

A. *Create New Graph*

When a user wants to create a new graph on the Graphen Protocol network, a bid order will be sent to the Graphen Protocol market. The bid contains metadata such as:

- How many masternodes should be in a router (to adjust security and replication)
- What the minimum storage capacity needed is
- What the minimum computational power needed is
- Whether the graph is public or private
- Payment currency
- The required stake
- Pay per hour per megabyte of storage
- The base fee for queries

Similarly, a masternode who wants to provide storage and computational power can create an ask order to price its hardware and effort. The masternode needs to specify the following information in its ask order:

- Desired pay per hour per megabyte
- Hardware specs
- Maximum storage capacity
- Preference for public or private graphs
- Maximum stake allowed
- Desired payment currencies

If a bid's requirements are matched by a group of masternodes, the graph is generated and the masternodes arranged in routers by computational power and storage.

B. *Update*

When users need to modify the data stored in a given graph, (e.g. INSERT, UPDATE and DELETE) the masternodes will change all replicates of the data within a router in an identical fashion, create a new Patricia-Merkle tree hash root and broadcast it to the other graph routers. If a subgraph exceeds a certain predetermined size, a new bid will automatically be placed on the market to form a new router.

C. *Query*

Users with the right permissions can make queries to the Graphen Protocol by including the following information:

- Graph ID
- Query
- Fee Amount

The query is sent to the graph's main router and redirected to the appropriate routers from there.

D. *Repair*

At different time intervals, masternodes within the same router will provide to each other a proof of Spacetime as defined in the Filecoin whitepaper [12] and its given three Merkle-Patricia tree roots. This will also serve as proof to the network that a masternode is storing the correct subgraph on their machine. If a conflict arises and a masternode claims one of the others is storing faulty data, a conflict resolution vote is initiated and resolved in the manner described in the Security & Privacy section.

V. Query Verification

A. Verifiable Computing

In the Graphen Protocol, masternodes within the same router will verify each other's computation results in order to confirm their accuracy. Towards that purpose we present a three steps verification algorithm [13][14] for a masternode.

- Step 1: Given a graph G and security parameter 1^k, the masternode will output an evaluation key eK_G and a verification key vK_G : $\{eK_G, vK_G\} \leftarrow genkey\left(1^k, G\right)$.
- Step 2: When a query q is inputted, the masternode will output a proof π_q and an answer α : $\{\pi_q, \alpha\} \leftarrow compute\left(q, eK_G\right)$
- Step 3: On input π_p, α, q, the masternode outputs 0 or 1: $\{0,1\} \leftarrow verify\left(\pi_q, q, \alpha, vK_G\right)$.

Therefore the computation done by the non-verifying masternode is correct if, for graph G, for all $k \in N$, for eK_G, vK_G outputted by $genkey(1^k, G)$, for a query q on G and for all π_q, α outputted by $compute(q, eK_G)$, it returns $1 \leftarrow verify\left(\pi_p, q, \alpha, vK_G\right)$.

Furthermore, the verification computation determines the query computation to be valid if, for graph G, for all $k \in N$, for eK_G, vK_G outputted by $genkey(1^k, G)$, for any adversary Adv (inverse of verification algorithm) and for negative binomial distribution neg, the probability $P[\{q, \pi_q, \alpha\} \leftarrow Adv\left(eK_G, vK_G\right);\ 1 \leftarrow verify\left(\pi_q, q, \alpha, vK_G\right);\ \alpha\ is\ incorrect.] \leq neg\left(k\right)$.

In the scenario where masternode A has to verify masternode B's computations:

- A user sends a query q to masternode B.
- Masternode A executes $genkey$ and generates eK_G and vK_G. The verification key vK_G is published for anyone to see.
- Masternode B computes the answer α and proof π_q using $compute$.
- Masternode A verifies the validity of response α by executing $verify$. The size of proof of a query answer is proportional to the size of answer of the query.
- If masternode A's verification result is 1, it proves the correctness of masternode B's computation results. Otherwise, a third masternode will intervene and compute the entire query again. A masternode that returns the wrong result, or in an untimely fashion will get a decreased reputation score.

B. Reputation System

We decided to use a Beta Reputation System[15] for the Graphen Protocol, which is one of the most popular reputation systems at the moment. It consists of having masternodes verify each other's computation results and give a score based on result accuracy and speed.

1) Reputation of Beta Distribution: To get the reputation score for a given masternode we first define a Beta density function with parameters α, β:

$$f(p|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)} p^{(\alpha-1)}(1-p)^{(\alpha-1)} \tag{1}$$

The expected probability value of the beta distribution is given by:

$$E(p) = \frac{\alpha}{\alpha+\beta} \tag{2}$$

Assuming a masternode A gives a score of v to masternode B's computation results, the system transforms the computation results with normalization weight w, and returns the positive and negative scores:

$$r_B^A = w(1+v)/2\ and\ s_B^A = w(1-v)/2 \tag{3}$$

$$\alpha = r_B^A + 1\ and\ \beta = s_B^A + 1 \tag{4}$$

Combining equation (2) and (4), we get:

$$E(p|r_B^A, s_B^A) = \frac{r_B^A + 1}{r_B^A + s_B^A + 2} \tag{5}$$

So that the reputation score of B given by A is:

$$Rep(r_B^A, s_b^A) = (E(p|r_B^A, s_B^A) - 0.5) * 2 = \frac{r_B^A - s_B^A}{r_B^A + s_B^A + 2} \tag{6}$$

2) Combined Rating: If n masternodes A_1, A_2,..., A_n review masternode B, the reputation score function of B can be represented as $Rep(R, S)$ where:

$$R = r_B^{A_1,A_2,\ldots,A_n} = \sum_{i=1}^{n} r_B^{A_i} \tag{7}$$

$$S = s_B^{A_1,A_2,\ldots,A_n} = \sum_{i=1}^{n} s_B^{A_i} \tag{8}$$

3) Discounting: Reputation discounting is our mechanism for giving reviewers with a good reputation a larger weight in the reputation system.

If masternode A has positive score R and negative score S and wishes to score the masternode B with r_B^A and s_B^A, the reputation function of B becomes $Rep(R_B^A, S_B^A)$ where:

$$R_B^A = \frac{2R * r_B^A}{(S+2)*(r_B^A + s_B^A) + 2R} \tag{9}$$

$$S_B^A = \frac{2R * s_B^A}{(S+2)*(r_B^A + s_B^A) + 2R} \tag{10}$$

4) Forgetting: Past behavior doesn't always represent current behavior. This also applies to masternodes. Therefore, we introduce a forgetting mechanism.

Assume a group of masternodes each give a sequence Q containing n scores $(r^Q_{B,i}, s^Q_{B,i})$. The overall scores can be expressed as :

$$r^Q_B = \sum_{i=1}^{n} r^Q_{B,i} \text{ and } s^Q_B = \sum_{i=1}^{n} r^S_{B,i} \tag{11}$$

By introducing the forget factor λ $(0 < \lambda < 1)$ into the expression(11), we can get :

$$r^Q_B = \sum_{i=1}^{n} r^Q_{B,i}\lambda^{n-i} \text{ and } s^Q_B = \sum_{i=1}^{n} r^S_{B,i}\lambda^{n-i} \tag{12}$$

We can see that when $\lambda = 1$, this expression is identical to the one without the forget factor; when $\lambda = 0$, only the most recent score counts; whereas when $(0 < \lambda < 1)$, recent reviews gain accrue a bigger weight in the reputation score calculation.

VI. Protocol Economics

A. Query Fees

In the Graphen Protocol, we decided to charge small query fees to avoid DoS and DDoS attacks (Denial of Service). These attacks aims to flood the network with a large amounts of useless queries. If the attacker has to pay a sufficient fee for all queries, this type of attack will become economically unviable.

The base fee for each query is:

$$BaseFee = Stake * Complexity * C$$

where,

Stake: Masternode stake held in escrow (determined by graph creator).
Complexity: Estimated magnitude of computational steps required by the query.
C: Constant base fee in given graph currency.

We also take into account a scenario where the network could be overloaded with query requests. Users will, in that case, be able to pay a premium fee to get priority. The total fee for the query:

$$TotalFee = BaseFee + PremiumFee$$

where,

Premium Fee: Arbitrary amount paid to gain priority in the processing sequence.

B. Storage Fees

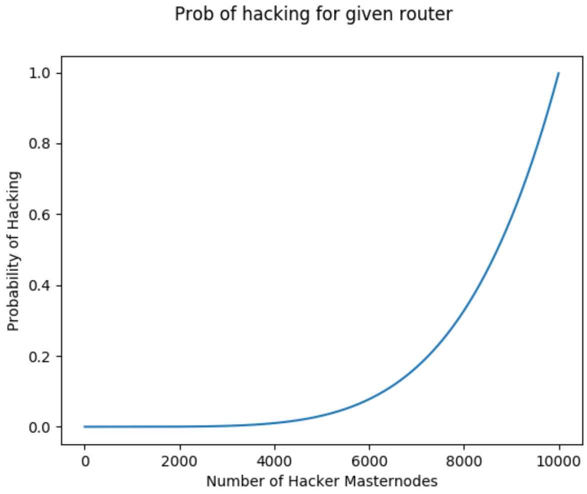
Storage fees, as mentioned in the Network Functions section of this whitepaper, are determined upon the creation of the graph and are distributed frequently. Graphen plans to take a 10% cut from storage fees in order to continue the development of the graph algorithms and other cryptographic research.

VII. Attacks

A. Masternode Collusion - Given Router

This attack is the one we deem to be the biggest threat to our system. Yet, we will also see why it is statistically impossible for hackers to be successful with this method.

In this scenario the hacker(s) desire to alter or destroy a piece of information stored by a given router. The hacker(s) therefore have to control all the masternodes within a given router. This plot shows how many masternodes the hacker(s) would have to possess in order to have a chance to hack that router. This plot assumes the total number of masternodes in the system is 10,000. As we can see the probability of this attack happening with control of 50% of all masternodes in the Graphen Protocol is 2-3%.

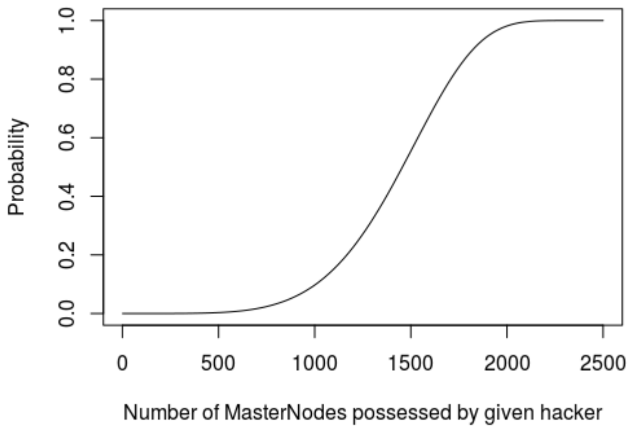


Furthermore, if the graph is private, the hackers would only be able to destroy information without being able to alter it since the data is encrypted. They would also be immediately caught since they would no longer able to produce correct the Merkle Patricia trees.

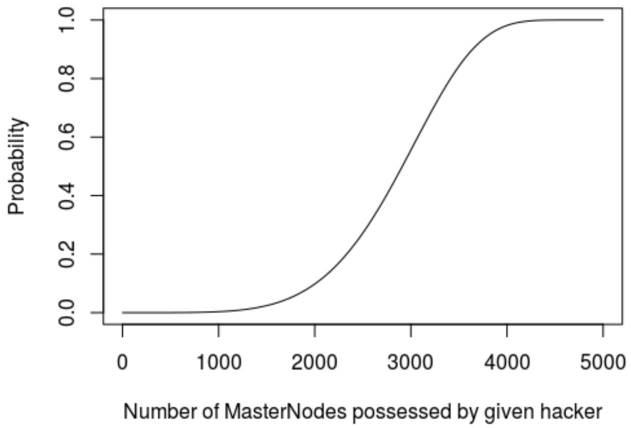
B. Masternode Collusion - Any Router

To gain control over any router in a given graph, which makes a lot less sense for a group of hackers, one would have to stake enough money to have a significant probability of gaining control over a router. If the attack fails the hacker would then have to be a very active participant in the network in order not to lose the money staked to spin up his or her own masternodes.

The figure below plots the probability of gaining control over any router in a given graph versus the number of masternodes controlled by the hacker(s). We assume the total number of graph on the Graphen Protocol is 50 and the total number of masternodes is 2500. Throughout this paper the default number of masternodes per router will also be 5, although it can be changed based on the security level desired by the graph's creator.



Below is the same graph but for 100 graphs and 5000 masternodes.



As you can see, controlling even 50% of all the masternodes in the Graphen Protocol would just give you a 25% chance of gaining controlling over a single router in the graph.

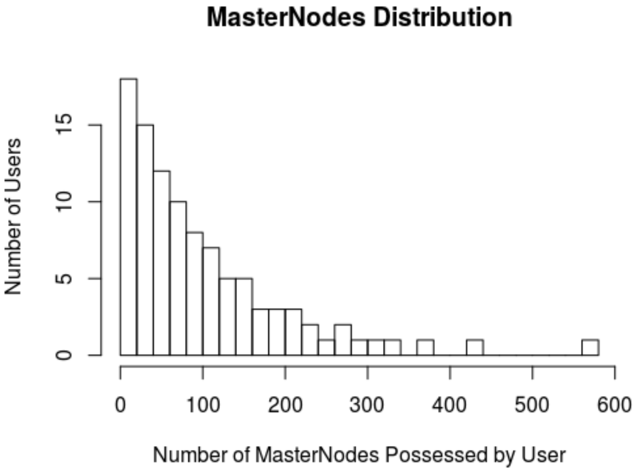
C. *Honest Actor Turned Malicious*

This scenario calculates the probability that through sheer luck an honest actor with at least 5 masternodes gains control over any router in a given graph.

The probabilities of that happening with 5 masternodes per router are: - For 50 graphs in the Graphen Protocol: 0.000475% - For 100 graphs in the Graphen Protocol: 0.000185%

With 6 masternodes per router, we simulated the scenario 10,000,000 times and there was a grand total of 0 hacks.

The assumed distribution of masternodes per user is plotted below and is considered almost a worst case scenario with only 100 users controlling all the masternodes:



The following are the worst actions an honest actor turned malicious could take in the following case:

- Stop responding to queries.
- Stop storing or deleting data.
- Assign him or herself a good reputation.

If the user stops responding to queries, the user will detect this fault and alert the main router. If the user stops storing or deleting data, he/she will be unable to answer queries and once again the main router will be alerted. If the user only assigns him or herself a great reputation, this will be harder to detect. If done excessively, it will be detected by other masternodes as someone who has an abnormal amount of power in the system and eventually the disparity between the queries answered and the reputation assigned will be found out.

Most of these actions result in the masternodes all losing their stake and therefore having a negative economical impact on the ”honest” actor.

VIII. Conclusion

A. *Advantages*

The Graphen Protocol’s features make it an ostensibly compelling alternative to legacy data centers and current distributed ledger technologies for its intended purposes. As we saw, the Graphen Protocol, through cryptographic and economic methods is highly secure and fault tolerant. However, being a decentralized technology has ramifications that give it further real world value propositions.

One is the option for users to have trustless, permissionless access to their data. This means being able to modify read and write permissions and not having to trust a third party in order to do so.

One of the other significant advantages the Graphen Protocol would possess over current cloud database solutions is lower latencies as it scales. Being decentralized it can use devices that reside in close proximity to the users issuing the queries. This would allow for the results of each query to travel a much smaller distance and therefore reduce latency.

Finally, by applying homomorphic encryption to a graph, multiple entities can pool sensitive pieces of data together and run encrypted queries on the whole. This would allow companies and/or users to collaborate in completely new ways without infringing on each other's privacy.

Graphen Protocol's advantages over current DLTs are very straightforward. By taking on the structure and capabilities of a graph database it will drastically improve from a storage, speed and data manipulation perspective. This should also allow users that still have difficulties understanding the merits of decentralization create apps with real world value and mainstream applications.

B. Use Cases

The Graphen Protocol will be able to satisfy all the major use cases presented by current cloud graph database solutions such as: social networking, recommendation engines, fraud detection, knowledge graphs, life sciences data processing and network/it operations.

It is also not meant to replace blockchains or smart contract platforms. In fact, smart contract platforms such as Ethereum [16] or EOS [17] could collaborate with the Graphen protocol to create truly decentralized applications. In a dApp, a smart contract would handle the complex logic of the backend and then query a Graphen Protocol graph for user data and relationships just like any current database.

A more immediate use case for the technology would be cryptocurrency exchange and stablecoin anti money laundering services. Each trader could be represented as a vertex and their transactions as edges to other vertices with information such as: who is the maker and taker, the amounts exchanged, the prices... The exchange could continuously query the graph for suspicious trading patterns and have real-time money laundering and fraud detection.

There have been 2.5 billion US Dollars laundered in Bitcoin alone and exchanges are looking for new solutions to this problem everyday. The market is also so fragmented that many exchanges are still small and often do not have sufficient resources to do proper AML checks, thereby risking immediate shutdown. Our systems would allow for initially small graphs to scale very quickly with our masternode market system, thereby reducing the barrier to entry for these new exchanges.

Furthermore, since the fees can be paid with digital assets such as Bitcoin, there is no need to convert the cryptocurrency exchanges' earnings to FIAT to pay for the hosting and querying services. Finally, a network effect for cryptocurrency exchanges could be achieved: independently, it is hard to track money laundering, but if all exchanges contribute some data, criminals would be caught even when switching platforms. Exchanges can encrypt the data they contribute and the masternodes running the AML algorithms on it would be able to spot suspicious patterns without having to read the underlying data and invading other exchanges' user privacy.

While this is a very tangible first step to using the capabilities of the Graphen Protocol, its use cases extend way beyond that and its potential is limitless. One extremely ambitious use of the Graph could be, for example, the creation of the famous semantic decentralized Internet that Tim Berners Lee (inventor of the World Wide Web) pitched in his TED Talk[18]. In fact, it's very likely that the Graphen Protocol is one of the only technologies that could support that grand vision.

ACKNOWLEDGMENTS

We would like to acknowledge Graphen.ai for giving us the opportunity to write this paper and the resources necessary to implement it. Thanks to Ketan Mehta for his help plotting the attack graphs.

REFERENCES

[1] Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.
[2] $https : //iota.readme.io/docs/what - is - iota$
[3] $https : //www.dataprise.com/resources/blog/2018 - dbir - takeaways$
[4] $https : //techcrunch.com/2018/10/08/google - plus - hack/$
[5] $https : //www.youtube.com/watch?v = n9zEiGyvJ - A$
[6] $https : //www.chainanalysis.com$
[7] $https : //medium.com/ipdb - blog/forever - isnt - free - the - cost - of - storage - on - a - blockchain - database - 59003f63e01$
[8] Hal Cooper, Garud Iyengar, Ching-Yun Lin, SmartGraph Database: Thinking like a Router, Graphen Inc., 2018
[9] Dan Boneh, Joseph Bonneau, Benedikt Bunz, Ben Fisch, Verifiable Delay Functions, Stanford University, New York University, 2018
[10] Raluca Ada Popa, Building Practical Systems that Compute On Encrypted Data, PhD thesis, Massachusetts Institute of Technology, 2014.
[11] Nahla Aburawi, Alexei Lisitsa and Frans Coenen, Querying Encrypted Gragh Database, University of Liverpool, 2018.
[12] Protocol Labs, Filecoin: A Decentralized Storage Network, 2017.
[13] Kalikinkar Mandal, Brasel Alomair,Radha Poovendran, Outsourcing Graph Database with Label-Constrain Query Verification, University of Washington, 2015
[14] Yupeng Zhang, Charalampos Papamanthou, Jonathan Katz, ALITHEIA:Towards Practical Verifiable Graph Processing, University of Maryland, 2014.
[15] Audun Josang, Roslan Ismail, The Beta Reputation System, 15th Bled Electronic Commerce Conference, 2002.
[16] Vitalik Buterin, Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform, 2013.
[17] $https : //github.com/EOSIO/Documentation/blob/master/TechnicalWhi$
[18] $https : //www.ted.com/talks/timbernersleeonthenextweb$