Seagull: Privacy Preserving Network Verification System

1st Jaber Daneshamooz
Department of Computer Science
University of California, Santa Barbara
Snata Barabara, USA
jaber@ucsb.edu

2nd Melody Yu
Department of Computer Science
University of California, Santa Barbara
Snata Barabara, USA
melodyyu@ucsb.edu

2nd Sucheer Maddury
Department of Computer Science
University of California, Santa Barbara
Snata Barabara, USA
smaddury@ucsb.edu

2nd
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, USA
@cs.ucsb.edu

2nd
ECE Department

Santa Barbara, USA
@ecs.umass.edu

***Abstract*—Todo list:**

- **Put the initial version on Arxive**

***Index Terms*—**

I. Introduction

A. notes for doing in future

Function secret sharing rather than data secret sharing

To search: VC3, Reusable garble circuit, homomorphic message authentication, AES better of shifting the key

TO read: Distributed ZN-prrof article by author of Opaque, Vuvuzela, https://github.com/vschiavoni/sgx-papers Facebook massive outage in Oct 4, 2021

Network configuration refers to the proper assignment of settings and policies in a network. Appropriate network configuration keeps the internet up and running and yields to efficient use of network resources. Configuration of routers is the most important and challenging part of network configuration.

Routers use routing protocols to determine the best path to a destination based on the given configuration. BGP is the de facto routing protocol in backbone network. All the routers in backbone network use BGP protocol to communicate with their neighbours and exchange reachability information. Each of these backbone routers belong to an autonomous system(AS) and the AS admins manually apply BGP configurations on the backbone routers.

The BGP configuration is the process of defining and applying BGP polices. BGP policies are a list of clauses defined by AS admins and determines the criteria for selecting the best path among all of the paths learned by the router. They also determine whether to advertise a known path to a neighbour or not. Since these BGP configurations are performed manually and separately in each AS, there may be some inconsistencies among them. These inconsistencies would result in internet outage which is the nightmare of internet clients and e-commerce companies. Network verification is a process which is used to find the possible problems in the network before applying the configurations and prevent further catastrophes.

Verification of network raises privacy concerns. The input data of the verification systems may contain sensitive private information like BGP polices and the network admins are not willing to share their private data with the verifier party. The privacy concerns vary in different levels of the network hierarchy.

The privacy concerns in inter AS level verification are more crucial than intra AS level one. In intra AS level, all of the networking devices are managed by a single entity. Therefore, the AS admin who configures the network has access to all of the data required for network verification. Thus, the AS admin can locally run the verification system without worrying about privacy issues. But in inter AS level, network verification is more challenging. The verification system needs to gather some data from all of the ASes and the AS admins are not willing to share this data due to economical and security aspects of network management. Hence, the inter AS level network verification system must provide a guaranteed level of privacy in order to encourage AS admins to share the required data.

Based on the report of network incidents by Alibaba in 2016 and 2017, 66% of networking incidents are caused by misconfigurations,including the mistakes in configuration updates (56%) and configuration bugs triggered by hardware failures (10%) [10].

In addition to that, Verizon caused a great network outage in 2019 by wrongly accepting a network misconfiguration from a small ISP in Pennsylvania, USA. For nearly three hours, web traffic that was supposed to go to some of the biggest names online including Amazon, Cloudflare and Facebook was instead accidentally rerouted through a steel giant based in Pittsburgh. This issue caused huge financial damage and customer dissatisfaction [2].

To the best of our knowledge, all of the works in network verification focus on the solubility and utility of the system and ignore privacy aspects. In this paper, we present Seagull, a system which privately verifies the network.

II. Problem Statement

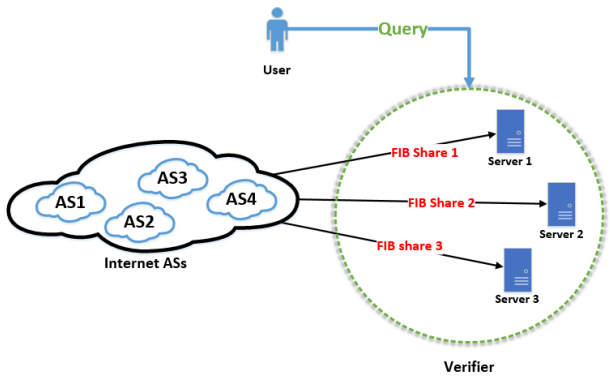

Fig. 1: Scenario

A. Scenario

https://arxiv.org/abs/2202.02729

1) Internet ASes: The backbone of the internet is a collection of ASes connected together and governed by BGP. Figure 1 shows a group of ASes in the network we want to verify. Since we want to verify the global network in this scenario, this group represents all of the ASes in the internet. Each of these ASes have their own BGP policies and pefrom route computation algorithm to generate the FIB.

2) FIB: Generally, FIB is the output of route computation process. In the AS level, FIB is a table which maps the destination prefixes to the next-hop AS. The AS would route the traffics for destination d to the corresponding next-hop AS in the FIB table. Table I shows an example of AS level FIB.

TABLE I: Network layer FIB at AS level

Prefix	Next Hop(ASN)
25.8.0.9/16	30125
12.244.0.0/24	870
184.0.0.0/8	43127
128.69.16.0/24	870

3) User and query: The user is an AS which wants to update its policies. As mentioned earlier, the network verification process is performed before applying the new polices in order to make sure that policy updates do not cause any problem in the network. Therefore, the user sends a query to the system which contains the changes it want to make in its BGP policies. If the system determines that the policy modifications are consistent with the configuration of the rest of the network, it would sent the corresponding response and the user would apply the desired changes. Otherwise, the system would notify the user about the possible problems and the user would not apply the updates. Worthy to mention that in the last step, we assume that the user follows the protocol; If its updates cause problem it the network, it would not apply them and if they updates are consistent with the rest of network, it would apply them. Therefore, we can assume that the system always has the most updated version of policies/FIB.(Policies would be reflected in FIB)

4) Threat model: An AS does not trust other entities(verifier servers, other ASes) in this scenario. We consider anytrust threat model in the verifier; it means that when at least one the servers in the verifier is not colluding, the data would be kept private. For the entire system, we assume that all the entities are honest but curious(HBC).

B. Goals

The goal of this paper is designing a network verification systems which checks the following properties:

- Loopfree: The network has loop or not
- Waypoint: Whether node x is in the path from s to d
- Reachability:Every node has found a next hop for reaching destination d
- Prefix hijacking
- BGP route leak

The easy solution for privacy preserving verification is just applying the MPC on the existing solutions. The state of the art network verification systems mostly use SMT solver or formal methods. These methods have scalablity problem and can not be applied to the network with several thousand ASes, let alone we add the overhead of private computation(supposing that we can apply private computation over SMT or formal method). jd: Saying that why I use forwarding graph representation?

1) Loopfree: In order to find loops in the forwarding graph, For example, in order to check the network is loop free or not, we can use DFS and MPC. Our experiment showed that simply running DFS in MPC framework does not scale

2) BGP leak: BGP Route Leak is "the propagation of routing announcement(s) beyond their intended scope. That is, an announcement from an Autonomous System (AS) of a learned BGP route to another AS is in violation of the intended policies of the receiver, the sender, and/or one of the ASes along the preceding AS path."

3) Waypoint: The waypoint property determines whether a specific node w is in the path from s to d or not. In order to verify this property, we reconstruct the path from s to d. In this process, whenever we discover the next node in the path form s to d, we compare it with w and if the values are the same, we determine that w is a waypoint node.

4) Prefix hijacking: Many of routing incidents are caused by prefix hijacking which is due to the inherent vulnerability of BGP protocol; BGP is not able to verify whether the ASes are sending announcements are permitted to do that or not. In order to hijack a prefix, AS x would announce a prefix which does not belong to it and try to manipulated the routing table of other ASes. In this way, the traffic destined to that prefix would be routed to x instead of the legit owner. AS x can use these traffic for malicious purposes. Therefore, identifying prefix hijackings as soon as possible is so important for AS admins due the security issues related to that. Based on [3], there were 14 prefix hijacking incident every day from January to July 2020(Figure 2). jd: Should I provide a toy example for prefix hijacking here?

Some incident descriptions here

Some notes for myself:

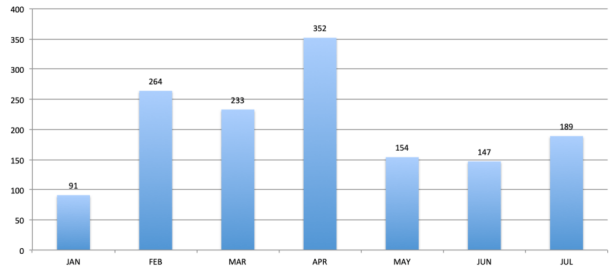

Fig. 2: Number of prefix hijacks from Jan to Jul 2020

1) Is it more important than Loop detection? (for two hours most of youtube traffice were going to Pakistan and youtube could find it in one hour and solve it in one hour)
2) Mentioning the Regional Internet Registry for dedicating the IP addresses
3) Owning a prefix appears as a loop back in the FIB (make sure about it)
4) Can check the ownership by that entry in the FIB and also the RIR database –¿ cause the data is secretly shared, when we are doing the loop check etc, if there are more than one nodes where source and next hop are the same, we can deduce that there is a hijacking
5) We can check the ownership when they upload their FIB (maybe previous solution is better in terms of performance and reducing redundancy)
6) some RIRs host other databases known as Internet Routing Registries (IRRs), where network operators can store information about their routing policies and routed prefixes – which they can also use to generate filters to prevent prefix hijacking.(say it's not accurate cause not updated frequently enough and AS admins must use specific IRR format etc)
7) One of the state of the art solution is: A more robust and solid mechanism to help prevent prefix hijacking is Resource Public Key Infrastructure (RPKI). This is based on digital certification, which allows anyone consulting its repository to validate that an association between a prefix and autonomous system is correct.

We want to run the aforementioned algorithms in a privacy preserving way. Our final privacy goal is to prevent the leakage of AS policies. We will show that one can infer the policies from FIB data. Hence, in order to keep the policies private, we need to keep the FIB private while running our algorithms. For this purpose, we need to make sure that the data in FIB is kept confidential during computation and the access patters and number of accesses to the FIB does not reveal any information that may lead to inferring the policies. Also, we need to show that the final result of computation does not reveal any information that may result in identifying policies. The only thing that can be learned from the system is the result of algorithm(True or False).

The entire system should be able to answer the queries in order of several minutes. For the initial step, it can tolerate more overhead(loading all of the policies and verifying the entire network) but later on, we need an interactive system. The reason behind this limitation is that the AS admins can not wait indefinitely for applying their desired policies. The need to update the policies at reasonable time frame to properly react to the security incidents, load balancing etc.

C. Challenges

The scalibility is the main challenge for this scenario. As mentioned before, running the state of the art loop detectoin algorithms in a private way is not scalable. Therfore, we need to devise some heuristics and customize the loop detection algorithm for our specific problem.

jd: Can I mention the definition of problem, scenario and threat model as challenge due to lack of prior work???

III. Background and Motivation

A. Background: BGP

Border gateway protocol (BGP) is the de facto routing protocol in the backbone network. BGP is designed to exchange routing and reachability information among ASes. The route computation and path selection in BGP is performed based on the following criteria:

- path properties like shortest AS_PATH
- Routing policies like business relations following Gao-Rexford rule
- rule-sets like LOCAL_PREFERENCE to route the traffic through a specific AS

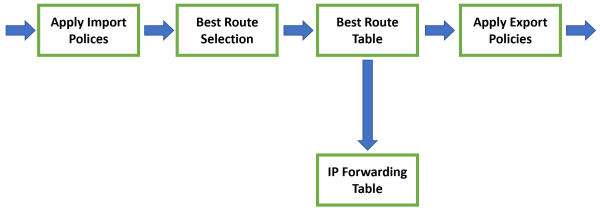

Fig. 3: BGP route computation process

```
RouterA# show ip bgp
BGP table version is 14, local router ID is 172.31.11.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal, r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network          Next Hop            Metric LocPrf Weight Path
*> 10.1.0.0/24      0.0.0.0                  0         32768 i
* i                 10.1.0.2                 0    100      0 i
*> 10.1.1.0/24      0.0.0.0                  0         32768 i
*>i10.1.2.0/24      10.1.0.2                 0    100      0 i
*> 10.97.97.0/24    172.31.1.3                             0 64998 64997 i
*                   172.31.11.4                            0 64999 64997 i
* i                 172.31.11.4              0    100      0 64999 64997 i
*> 10.254.0.0/24    172.31.1.3               0             0 64998 i
*                   172.31.11.4                            0 64999 64998 i
* i                 172.31.1.3               0    100      0 64998 i
r> 172.31.1.0/24    172.31.1.3               0             0 64998 i
r                   172.31.11.4                            0 64999 64998 i
r i                 172.31.1.3               0    100      0 64998 i
*> 172.31.2.0/24    172.31.1.3               0             0 64998 i
<output omitted>
```

Fig. 4: An example of BGP table content

1) BGP route computation: The ASes running BPG protocol use three types of routing table: BGP neighbour table, BGP table and BGP routing table. The BGP neighbour table contains information about BGP neighbours. BGP Table also known as BGP RIB contains the network layer reachablility information(NLRI). The BGP table contains all the routes from all the neighbours. In this table, for every destination, maybe there are several routes with different attributes. The last table is BGP routing table which contains only the best routes from BGP table. After selecting the best path to a network based on the BGP policies, this path is added to FIB.

2) BGP policy: The BGP policies determine which path should be chosen among all of the identified paths to a destination. They also determine whether to advertise a path to the other neighbouring ASes or not. These decisions are highly impacted by the relation of AS with it's neighbouring ASes.

There are two types of relation between neighbouring ASes:

- Customer-Provider: The customer purchases the connectivity through the provider
- Peering relation: Two ASes agree to carry the traffic of each other without any charge. It's usually used as a shortcut in the network to increase the performance.

Based on Gao-Rexford rule, ASes have the following preference for routing the traffic regarding the commercial deals:

1) A route through the customer has the highest preference(They get paid)
2) A route through the peer is preferred to a route through provider
3) A route through the provider has the lowest preference(They need to pay)

Based on the aforementioned rules, the ASes are not willing to advertise new routes to their provider. For example, in Figure 5, node X is customer of B and C. Node X knows a path to Y through C, but it would not advertise it to B. Since B does not know that X has a path to Y, it would not route the traffic destined to Y through X. If X advertises the XCY path to B, then B with forward the traffic destined to C and Y through X and would charge X for that.

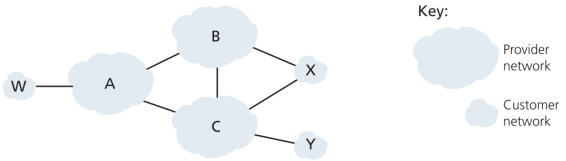

Fig. 5: BGP customer-provider relation

3) BGP policy conflicts: AS admins change BGP policies for security issues, traffic management and commercial reasons. These network configurations and updates are set manually; Therefore, they are prone to errors. For example, the BGP polices of one AS may have conflict with the policies of other ASes.

BGP

1. Highest local preference
2. Shortest AS-path length
3. Lowest origin type
4. Lowest MED (with same next-hop AS)
5. eBGP-learned routes over iBGP-learned
6. Lowest IGP cost to egress router
7. Lowest router ID of the BGP speaker

Fig. 6: List of BGP policies

Policy updates may cause route oscillation, network loops or unreliability to a portion of the network. For example, imagine that node 4 in figure 7 withdraws it's prefix from 2 and 3 but keeps advertising the same prefix to node 6(for load balancing, changes in AS relations etc). Node 2 knows that 3 and 6 have route to 4 through previous route advertisement. Also, node 3 knows that 2 and 6 have route to 4. If both 2 and 3 prefer each other over 6 as the new route to 4, it would cause a routing cycle.

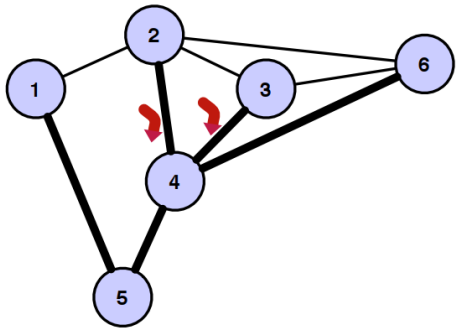

Fig. 7: Policy change causing BGP loops at 2 and 3 when 4 withdraws a prefix from 2 and 3 but not 6

jd: Paper: Consensus Routing: The Internet as a Distributed System paper for loop cases. Add the other example from this paper. Also check if I can tie these issues to Pittsburg issues

B. Network verification

Due to aforementioned reasons, we need to be cautious before applying new policies. The ASes needs to check for probable problems before running the configuration updates. Network verification systems are designed for this purpose.

C. Importance of privacy in network verification

Business owners specially in network area are not willing to share their commercial information because it can negatively

affect their business. These commercial information can be leaked by observing their BGP policies. Hence, it is important for AS admins to keep the BGP routing policies secret.

D. FIB

Routers in the network has a forwarding table for routing the packets. The forwarding table is filled by running the routing algorithms with respect to network configuration and finding the best path for each packet The AS level FIB(Table II) shows the mapping of IP prefixes and the corresponding next hop(Next AS) for reaching to that destination. Based on this table, the traffic destined for 184.130.10.20 would be forwarded the AS with AS number = 43127.

TABLE II: Network layer FIB at AS level

Prefix	Next Hop(ASN)
25.8.0.9/16	30125
12.244.0.0/24	870
184.0.0.0/8	43127
128.69.16.0/24	870

The network verification system checks the probable policy conflicts in the network. Since the AS level FIBs are populated by considering BGP policies, any changes in the routing policies would be reflected in the FIB. Therefore, any conflict between BPG policies would appear as a form of conflict in the forwarding tables. BGP policies are like some conceptual information about preference of one route over the other, while the FIB contains concrete information about each route, like the next-hop ID etc. For this reason, finding conflicts and network bugs like loops is easier when utilizing FIBs rather than policies. Hence, Seagull utilizes FIB of each node in order to verify the network. We use the FIB information in all of the network nodes in order to verify the network. jd: Adding example that how FIB can reveal policies

E. Challenges

jd: Talk about the existing solutions:SMT and Formal method solutions While most of the current works focus of scalablity of network verification in large scale networks, the privacy issues are undermined. A large scale network verification system which ignores the privacy issues is useless. Although it can verify a large network so fast, the ASes would avoid to share the required information with the versifier. In other words, preserving the privacy is as important as the scalability of these network versifiers. To the best of our knowledge, there is no prior work on privacy issues in network verification.

Network verification is an expensive operation. Therefore, if the privacy preserving operation is not designed efficiently, the system would be so costly and not applicable. Therefore, we can not blindly apply privacy preserving computation. We need to customize the algorithms for solving this specific problem.

Most of the works in privacy preserving systems try to remove the link between the data owner and the data. However, in network verification, we can not ignore or eliminate this bound. In network verification, the data records are related to each other and release some information about the network design.

Unlike many applications of privacy preserving systems where the program tolerate some level of inaccuracy, network verification systems are so sensitive and require a fully accurate response. Hence, we can not apply differential privacy and sacrifice accuracy to get privacy. Therefore, the common approaches like node differential privacy or edge differential privacy are not applicable here.

Here, we present a novel way to meet both privacy and efficiency. Seagull is able to perform required operations with respect to the bound between data owner and data record and is designed in a way that does not add intolerable cost.

IV. Security and Threat Model

The private data can be leaked from any part of the system. The attacker may gain access to the data when the data provider is sharing the private date, or a malicious insider can access the stored data in the system. The querier also may obtain some private data from the query results. In order to design a privacy preserving system, the leakage of private data should be prevented at every single steps. These steps include:

1) Data collection
2) Data storage
3) Computation
4) Sharing the result

Choosing the proper approach in each step highly depends on the utilized approach in the other steps.

1) Briefly explaining private data collections and storage part. The result is a one bit answer(considering the privacy budget). The main part is private computation:

A. Attack model

Seagull utilizes the FIB in order to verify the network. By accessing the FIB, the attacker can only discover the best route of each node for a specific destination; But the active adversary can play another role here. By removing the links corresponding the the best path of a node, the BGP protocol would select the second best path and so on. By this method, the attacker can find all the routes of a node to a specific destination with the ranking of each route. By having the knowledge of all possible routes with their ranking, the attacker can find the BGP policies of a node.

Suppose graph G is the graph created based on the BGP table for destination D. Graph G is a directed graph where edge $E(u, v)$ means that there is a route from u to D where the next-hop is node v. The forwarding graph created based on FIB is a sub graph of G that the edges only represent the routes with the highest rank. Therefore, the FIB contains some information about the BGP table.

If the attacker gain access to the forwarding graph, he knows the best route of the node x to destination D. The next-hop of best route from x to D exists in the FIB. The attacker can remove the link between x and its next-hop to D and run the query again for this new topology. This time, the FIB will contain the best route from x to D which is different from the

previous best route. Hence, the attacker would know the route with second highest ranking from x to D. By continuing this process, the attacker can find all the routes from x to D with the ranking of each route. The attacker should run the query n times where n is equal to maximum number of neighbours of the nodes in graph G. ???(ASK question)

Some may claim that the BGP tables are public and attacker can easily access them, but this is not true. Only some of these tables are public and the others are private or partially revealed. Since these information are private, and as explained earlier, they are critical for AS admins, accessing these information is an interesting issue for attackers. The focus of seagull is hiding the FIB so that an attacker can not gain any information about the BGP table from observation of FIB.

1) Private Computation: In this paper, we focus on providing privacy in computation. Privacy in other parts of the system is trivial and out of the scope of this paper. In data collection and storage, state of the art cryptography and secret sharing method are used to keep the data private. In addition, the query result does not have any specific information about the private data; It is a boolean value indicating that a specific network property is satisfied or not. Since the verification system requires the exact result, differential privacy is not used in any part of seagull.

2) Threat model: Seagull considers honest but curious(HBC) threat model. In HBC threat model, parties may try to gain as much information as possible; but they would not manipulate the output of perform faulty operations. An HBC is a t_private protocol if any t parties who collude at the end of the protocol can not learn the private data. Seagull is $(t-1)$_private where t is the number of servers in the verifier. In this way, if only one server avoids to collude with other servers, the data would be kept private.

V. Seagull

A. Sharing FIB

In this setup, the ASes would share their FIB with the verifier. This step should be designed in a way that transferring the FIB to the verification section does not reveal any secret information. We use secret sharing method to send the FIB entries to the verifier [**?**]. In this way, each server in the verifier would have a share of each entry. Running the seagull for the first time requires that all the ASes to send their entire FIB to the verifier. All of this data would be saved by the verifier. After this step, the ASes only send the changes in their FIB(as a result of update in BGP configurations and policies)..

To keep the FIB private, seagull uses multiparty computation(MPC) in verifier. The verifier consists three servers(parties). These three servers cooperate with each other to secretly perform the verification. Seagull utilizes additive secret sharing method for MPC. In this way, every data in the FIB would be broken into three shares and each share would be given to one of the servers in the verifier. The summation of these shares would create the original data. The summation would reveal the secret values but based on the properties of additive secret sharing, these servers do not get any information about the original data till at least one of them avoids to compromise.

B. Loop detection(initial phase)

jd: Things for loop free:

1) Mentioning general loop detection stuff?
2) Mentioning the other paper(just cite, little overview or full analysis and showing the deficiency in that solution?)
3) Showing the several modification in one place or explain each one in separate subsection?

1) Overview: The forwarding graph is the graph of network which is built upon FIB. As shown in Figure **??**, it is a digraph where each edge connects a node to its next hop for a specific destination in the network. Since every node has one and exactly one next hop for a specific destination in the FIB, the out degree of every node would be exactly one; except that the destination node which does not have any outgoing edge.

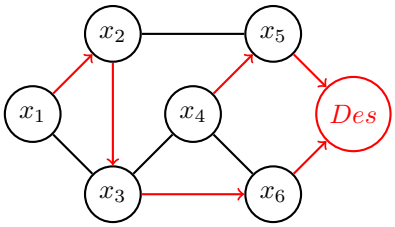

Fig. 8: Network topology and its corresponding forwarding graph of destination Des(annotated in red)

2) Finding loops in the network by traversing forwarding graph: If a set of entries in the FIB create a loop, that loop would appear in the forwarding graph. A loopfree forwarding graph would be weakly connected; It means that if we replace all of the directed edges in the forwarding graph with undirected edges, it produces a connected undirected graph. In other words, when a set of FIB entries create a loop, the equivalent undirected forwarding graph would be a disconnected graph(figure **??**). So, in order to check the loop free property of the network, we only need to check whether the forwarding graph is connected or not.

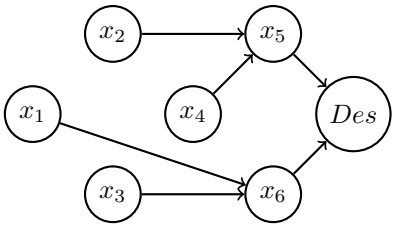

Fig. 9: Forwarding graph with loop

3) Algorithm 1: In order to check the graph connectivity, seagull uses BFS. Starting from destination node, BFS iteratively visit the nodes. If the graph is connected, the BFS would visit all of the nodes.

Since the starting point of BFS is the destination node, it should travers the graph in direction which is the opposite of edges' direction. For this purpose, we reverse the forwarding graph. Reversing the forwarding graph is performed by reversing the FIB(Table III and IV).

TABLE III: FIB

Source	next-hop
x1	x2
x3	x6
x2	x5
x4	x5

TABLE IV: Reversed FIB

Source	next-hop
x2	x1
x6	x3
x5	x2
x5	x4

The reversed FIB table is secretly shared with MPC parties. Each entry of this table contains the ID of two nodes which represent an edge in the reversed forwarding graph. Also, a list of visited nodes V is secretly shared among MPC parties. We can consider this list as the third column in FIB. The entries in this list are boolean values indicating that the edge(somehow the next hop node) is already visited or not.

After visiting the destination node d, seagull iterates thorough the reversed FIB to find the neighbours of d. To do this, Seagull secretly compares the source ID of entries in the reversed FIB with the ID of d. Whenever the source ID matches the ID of d, the BFS visits the next-hop node and updates the corresponding entry in V respectively. For a graph with n nodes, finding the neighbours of a node requires n MPC comparisons.

When all the neighbours of destination is visited, seagull chooses the next node in the V and tries to find the neighbours of this node. Seagull repeats these steps until it meets one of the following criteria: 1) All the nodes are visited. 2) There are some uvisited nodes but seagull is unable to discover new nodes.

In the first scenario implies that the notwork is loop free. In this scenario, all values of V would be true. If it is not the case, it implies that the reversed forwarding graph is not connected and hence, the graph is not loop free. (we can use 1 in case of true value and secretly take some over V)

4) Algorithm 2:

5) Less comparison due to special property of graph: In the BFS algorithm, before adding a node x the the list of visited nodes V, we check that if is already visited or not.

$$if(x \in V) \tag{1}$$

This ensures that back edges do not cause revisiting a node and having an infinite loop in the graph. In FIB, the out-degree of each node is equal to 1. The BFS algorithm here works with reverse FIB, therefore the in-degree of each node in the input of BFS algorithm would be 1. When the in-degree of every vertex is equal to 1, it means that there would be no back edge to the same node. Since we do not have a back edge to a node in our graph, we can ignore running the (1). Considering that there is no loop in the graph and the graph is connected, the (1) would take $O(n^2)$ MPC comparison.

6) Removing the leaves: Can I do it while I don't have the mirror? Can I do it in offline(not query mode) without revealing information? If yes, the increamental approach needs to be updated.

C. Loop detection(Query phase)

Verifying LoopFree property over the entire forwarding graph is expensive. As mentioned earlier, we need to discover all of the nodes in the forwarding graph. The internet is consists of more than 35,000 ASes and running the aforementioned algorithms for every policy update is not a desirable. First of all, there would be too much computation for the verifier servers. Moreover, the system should be interactive and response in order of several minutes; but the previous Loop detection method is not that fast. Finally, we should mention that the system responses to one query at a time. So, if AS X has already initiated a query and Y need and update on it's policies, then why should wait till the fulfillment of X's query. jd: It may be able to make it support concurrent queries initiated at the same time(I need to prove that) but new queries can not preempt the running queries. So, we need an approach which can respond to the queries in the acceptable time frame.

If applying policy updates of AS X does not create a loop in the path of X to the other nodes(All nodes are reachable by X), we can conclude that the entire network would be loop free. In this way, when AS X sends the query, we take the following steps to verify loop free properties for destination d

1) We get the FIB after applying those policies.
2) We start from X and go through the graph to reach d
3) Previous step is like waypoint algorithm(when the waypoint node is not discovered in the path)
4) If we can reach to d:
   - The path is loop free
   - The new policies would not create a loop in the network
   - AS X can apply the policy changes in the real network
5) If we don't reach to destination after several iterations(MAX is 15 which is maximum acceptable number of hops in the network) or we reach to the node X in the path(detecting the loop):
   - The path has a loop
   - The network would have loop under new condition
   - AS X must avoid applying those policies

1) Cost: The cost would be equal to the worst case in waypoint(Although there is not much difference between the worst case and average case of the waypoint).

2) Proof of correctness: A tree is a connected graph without any loop. Our forwarding graph G is a tree with n vertices. A connected graph with n vertices and n-1 edges is a tree; Therefore, our forwarding graph has $n-1$ edges. Suppose that we want to connect another AS called x to the forwarding graph. We need to add a vertex and an edge(connects x to it's next hop in the forwarding graph). We can prove that adding

x to G does not create a loop. Adding x increases the number of nodes to $n+1$ and the number of edges to n. Based on the theorems explained earlier, if the new graph is connected, it is a tree and does not have a loop. Since one point of the added edge is x, the graph is connected and we can deduce that the resulting graph is loop free.

Suppose that G and G' are two trees with m and n nodes respectively; So, they would have m-1 and n-1 edges. We can prove that by adding edge $E(u,v)$ where $u \in G$ and $v \in G'$(or vice versa), the resulting graph would still hold the tree properties.jd: I can cite Kruskal algorithm or explain myself. Which one is better?

We suppose that node x updates its policies. This update in policy only affects the next-hop value of x. So, for every node which is between x and destination d, this policy change does not have any effect and destination is reachable by those nodes. Suppose that we divide the graph into two subgraphs A and B. Subgraph A contains the node which reach the the destination throuhg x(in the other word x is a waypoint of those nodes) and the other subgraph contains the rest of the nodes. We consider that x is a member of A and the corresponding edge of x is not added yet. Lets call this edge $E(x,u)$. u is the next hope of x. If $u \in B$, based on the theorem in "Union of two subgraphs", the resulting graph would be a tree and loopfree. Otherwise if $u \in A$, it is obvious that the forwarding graph would be disconnected and we would have a loop.

Scale mamaba does not allow separation of offline and online phase but it provides a mean to save the output of offline data(like triples etc) and restart the program. I should find a way to update those stored data for new policies or retrieve those data and update the corresponding rows in online phase).

D. Obliviousness

Intuitively, the security definition requires that the server learns nothing about the access pattern. In other words, no information should be leaked about:

1) Which data is being accessed
2) The last time the data was accessed
3) Tthe same data is being accessed (linkability),
4) Access pattern (sequential, random, etc)
5) Whether the access is a read or a write

Data-oblivious, or just oblivious, execution is defined as having the same sequence of operations regardless of the input data and data-independent memory accesses, which makes it suitable for use in outsourced tasks. The ORAM protects many things like which index is accessed etc but does not hide the number of read/writes to the data structure. Here, we prove that our algorithm provides complete obliviousness and even the number of accesses to the oblivious data structure does not reveal the private information.

Definition 1: Let d denote input to a graph algorithm. Also, let $A(d)$ denote the sequence of memory accesses that the algorithm makes. The algorithm is considered data-oblivious if for two inputs d and d' of equal length, the algorithm executes the same sequence of instructions and access patterns $A(d)$ and $A(d')$ are indistinguishable to each party carrying out the computation.

Definition 2: Let the forwarding graph d denote input to a graph algorithm. Also, let $A(d)$ denote the sequence of memory accesses that the algorithm makes. The algorithm is considered data-oblivious if for two inputs d and d' of equal length, the sequence of instructions, access patterns and the number of accesses are irrelevant to the structure of graph d and d'.

Algorithm 1 IsLoopFreeRoute(node d, array FIB(three columns), array V, int n)

Require: Destination node d, secretly shared array of visited nodes, FIB and number of nodes n
Ensure: All paths to d are loop free

```
1:  V ← empty array
2:  MPC.append( V , d.ID)
3:  flag = True
4:  while V.size()! = n && flag do
5:      flag = False
6:      for all row in FIB do
7:          if row.src in V && row.visited = false then
8:              MPC.insert(V, row.src)
9:              row.visited = true
10:             flag = True
11:         end if
12:     end for
13: end while
14: if V.size() == n then
15:     return True
16: else
17:     return False
18: end if
```

E. Complexity

We examine the complexity of loop free algorithm based on the number of MPC rounds. Comparison(Line **??**) is the only MPC operation in this algorithm which takes 3 MPC rounds. The number of times that the algorithm calls the MPC comparison depends on the nested for loops.

1) Worst case: The worst case scenario happens when the reversed forwarding graph is a skewed binary tree. In this scenario, the inner loop traverse the entire FIB and only find one next-hop in the FIB. Therefore, the outer loop runs n times where n is the number of nodes in the network. The eq 2 and 3 imply that the loop free algorithm performs $O(n^2)$ MPC operations in the worst case.

$$T(n) = n + T(n-1) \quad \text{And} \quad T(1) = 1 \tag{2}$$

$$T(n) = n+(n-1)+...+1 = \frac{n^2}{2}+\frac{n}{2} \rightarrow T(n) = O(n^2) \tag{3}$$

2) Average case: Suppose that the average number of neighbours for every node in the reversed forwarding graph is c. Every time that the algorithm runs the inner loop, it finds c next-hops(visits c nodes) and shrinks the FIB by removing c entries. The eq 4 shows the number of MPC comparsions in average case.

$$T(n) = \sum_{i=1}^{n/c} n - (i-1) \times c = \frac{n^2}{c} + n - \sum_{i=1}^{n/c} i \times c$$
$$= \frac{n^2}{c} + n - \frac{\frac{n^2}{c} + n}{2} = \frac{n^2}{2c} + \frac{n}{2} \rightarrow T(n) = O(n^2) \quad (4)$$

The average case and worst case are both in $O(n^2)$. Considering the constant factor $\frac{1}{c}$ in the average case, it outperforms the worst case by the factor of c. The experiments show that by March 2021, the number of allocated AS numbers exceeded 100,000. Also, the average AS path length in the network is between 4 and 5; it means that we can expect the value of c is between 10 and 18.

F. MPC operations

In this section, we explain the MPC operations which are the building blocks of seagull. Comparison and table look up are the main MPC operations used by seagull to verify loop free and waypoint property. Seagull also requires to secretly share node IDs and remove a node from a list of nodes.

1) MPC comparison: Seagull performs MPC comparison using multiplicative secret sharing(MSS) [**?**]. The MSS takes an input x in devides it into n shares $[x_1, x_2, ..., x_n]$ in a way that:

$$x_1 \times x_2 \times ... \times x_n = x \quad (5)$$

Secure comparsion presented in [**?**] uses Beaver triples [5]. Beaver triples are the building block of secrue MSS comparison; therefore, we describe them here and then use them for construction of comparison operation.

Beaver triples were originally proposed to efficiently implement the secure multiplication over MSS. The MSS multiplication gets the secretly shared values as input and retruns the secretly shared product of those values(6). Worthy to mention that the MSS multiplication can not be constructed $[\![x.y]\!]$ by simply multiplying the inputs(7).

$$Secure\_Product([\![x]\!],[\![y]\!]) \rightarrow [\![x.y]\!] \quad (6)$$

$$[\![x]\!].[\![y]\!] \neq [\![x.y]\!] \quad (7)$$

VI. Evaluation

A. Benchmarking environment

In order to evaluate the performance of seagull, we implemented several different versions in two different environments. First, we implemented seagull with a MPC framework called $Scale-Mamba$ and run the algorithm on a Surface book 2 core i7 with 16 GB RAM. All the MPC parties were on the same computer and therefore, the communication cost is mitigated. (explain distributed implementation after doing that part)

B. Network topology

In this experiment, we used real world network topologies. Since seagull is designed for inter-AS network, we use the AS-level network topology dataset. In this dataset, every AS in the network is considered as a node in the network topology which is identified by a unique AS number(ASN). The datasets represent the network topology by the set of edges in the network where each eage is defined by a pair of ASN.

We used seagull to privately verfiy the network in different scales. Different networks with 12,280 nodes. I will test for 1421vand 22972 and 34000 nodes after resolving the issue for reading files by MPC parties.

C. FIB generation

Every node in the network has a FIB which contains neccessary information for reaching to all of the IP prefixes. However, seagull verifies the network properties by considering the forwarding graph for each destination. Hense, we need to generate destination based FIBs. In contrast to the source based FIB, the destination FIB contains information for reaching to an IP prefix by every node.

1) Random FIB generation: In order to generate a random destination based FIB, we used BFS as a graph search algorithm. Since the AS network topology is a connected unweighted graph, the BFS search would generate a minimum spanning tree(forwarding graph). In addition, in an unweighted graph, the shortest path is the path with least number of edges. Therefore, by running BFS on AS network graph, we always reach the other node from the given source using the minimum number of edges.

2) Deriving FIB based on policies(not necessary):

VII. Related works

A. Related works on loop detection

Using DFS and DAG are common approaches for finding loops in a graph. Running these recursive algorithms for network verification ruquire complete information about the graph connectivity and node policies. This issue leads to the leak of information even using privacy preserving systems. Current approaches are trying to run DFS in MPC without revealing infromation. Adding fake nodes and fake links to the graph is one of the approaches to hide private information in the process of running DFS in MPC. These approaches fail to keep the information private enough. Also, the process is so expensive in terms of comutation and communication. Here, we presenet another approach to find loops in the netwrok graph.

This section is not the part of paper. Instead of writing report about the papers I read, I included the related information here.

B. Privacy-Preserving Interdomain Routing at Internet Scale

Instead of MPC, it utilizes 2PC, with the assumption that parties are semi-honest and non-colluding. Removing the stub nodes would be a very good optimization. Stub nodes are the nodes which are only customer and constitude 85 percent of the ASes. But the assumption of the system about two

non colluding parties does not seem a satisfactory priavcay guarantee. Also, the fact that both parties are located in the same datacenter makes the benchmark of the system questionable; That's because a great portion of MPC cost is related to communication over netwokr. Also, having both parties in the same data center increase the risk of successful cyber attack or intruder attacks.

The system is prone to false input data. Focuses on optimizing the boolean garbled circuit for this problem. Also, using PKI for router advertizements which can eliminate prefix hijacking is a good point here(how implemented?). The approach which is like SDN does not seem to work correctly in the current environment.

Due to our low runtimes, we can precompute paths for cases of failure, i.e., simulate the removal of nodes and therefore significantly reduce the recovery times for these cases. This is possible since the network topology is known publicly and therefore topology changes can be simulated."

The configuration of current network and updating the routing table in current network is more complex than just finding a new path or a stable configuration and announcing it to the ASes.

In fact, each node (AS) learns only its "next hop" node in the final routing outcome with respect to a destination, and not even the full route. We also hide the entire convergence process, which potentially leaks information. As a side note, even with multiple vantage points, inferring routing policies is no easy task. While our scheme would not completely remove this kind of leakage, it would definitely decrease it compared to routing using BGP, where the ASes broadcast their full routing table. Furthermore, it is unclear whether all information about the policy preferences can be gained using only the next hop as information. The setup phase in this system is independent of topology of ASes and increase the parallelism in the offline phase.

REFERENCES

[1] zenodo.org
[2] theregister.com
[3] bgpstream.com
[4] Xiong, Lizhi, et al. "Efficient privacy-preserving computation based on additive secret sharing." arXiv preprint arXiv:2009.05356 (2020).
[5] D. Beaver, "Efficient multiparty protocols using circuit randomization," in Annual International Cryptology Conference. Springer, 1991, pp. 420–432.
[6] Aly, Abdelrahaman, et al. "Scale–mamba v1. 12: Documentation." (2021).
[7] Blanton, Marina, Aaron Steele, and Mehrdad Alisagari. "Data-oblivious graph algorithms for secure computation and outsourcing." Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. 2013.
[8] Cao, Ning, et al. "Privacy-preserving query over encrypted graph-structured data in cloud computing." 2011 31st International Conference on Distributed Computing Systems. IEEE, 2011.
[9] Hastings, Marcella, et al. "Sok: General purpose compilers for secure multi-party computation." 2019 IEEE symposium on security and privacy (SP). IEEE, 2019.
[10] Liu, Hongqiang Harry, et al. "Automatic life cycle management of network configurations." Proceedings of the Afternoon Workshop on Self-Driving Networks. 2018.