STRIP: Privacy-Preserving Vector-Based Routing

Wilko Henecka
School of Mathematical Sciences
University of Adelaide, Australia
Email: wilko.henecka@adelaide.edu.au

Matthew Roughan
School of Mathematical Sciences
University of Adelaide, Australia
Email: matthew.roughan@adelaide.edu.au

***Abstract*—Security of routing protocols is a critical issue, as shown by the increasing number of attacks on the Internet's routing infrastructure. One often overlooked aspect of security is privacy. In the context of a routing protocol we mean the ability of a router to keep information such as its routing policies private. BGP does this to some extent through design. An Autonomous System's policies are not explicitly revealed to other participants in the routing protocol. Nevertheless, BGP still reveals a great deal of information about the Internet and its participants. We propose a privacy-preserving routing protocol called STRIP that reveals very little information to participants in the protocol. For instance, participants can find shortest-paths to destinations in the network without ever learning the path lengths. Such privacy could be useful for a range of reasons: preserving the proprietary information captured in a routing policy, or preventing an attacker from gaining valuable information about the network. We show the feasibility, performance, and costs of STRIP with simulations and implementations of the protocol.**

I. Introduction

There are a long list of desirable features for a routing protocol. For instance, it should be robust, distributed, scalable, and easy to configure. There are now many protocols with different sets of these properties, but more recently security of routing protocols has become a major issue. The reason for this lies in the spread of routing protocols between untrusted parties. The canonical example is inter-domain routing. The defacto standard inter-domain routing protocol is the Border Gateway Protocol (BGP). The rapid expansion of the Internet has lead to proliferation of BGP speaking Autonomous Systems (ASes), more than 40,000 as we speak. In the Internet of yesteryear the BGP speaking networks were almost like a club. Membership assured an element of civility. However in recent years this trust has led to problems. There have been many documented problems with BGP: spammers have exploited security vulnerabilities of BGP to send unwanted, or illegal emails [1], accidental and/or deliberate hijacking of address space has caused large scale disruptions (for just a few examples see [2]–[4]). Most of the resulting work on adding security to BGP has focussed on authentication.

One aspect of securing routing protocols that has not received wide-spread study is privacy. Of course, one could encrypt the individual transactions of a routing protocol to prevent eavesdropping by external assailants. However, privacy with respect to the other participants in a routing protocol is a much more interesting problem. Routing protocols are generally designed to spread information. This is seen as a necessary step as distributed control is one of the philosophical grounding points for Internet design, but there are good reasons to wish to preserve privacy of routing information. The routing information of each party can contain proprietary information that could be commercially, or politically sensitive, or could contain information that would make an attack against that party easier.

BGP implicitly acknowledges the dichotomy of information hiding/spreading. The protocol allows for each AS to have flexible, heterogeneous and dynamic policies, and BGP attempts to jointly determine a solution to these routing policies. However, BGP hides internal policies by performing a "best route" computation locally and passing on the result. It doesn't pass on the policies used to make the decision. BGP's route computation passes only simple data such as AS-paths and locally defined attributes such as MEDs and communities. This has resulted in a protocol that is not transparent. Its behaviour is unpredictable, and exhibits slow convergence [5], persistent oscillation [6], [7], and other negative features [8], [9], but ISPs have clearly been willing to trade off negatives against the desire to preserve their privacy (BGPv4 has been a defacto standard for nearly two decades [10]).

But does BGP hide enough? BGP routing data has been used to build AS-level topologies of the Internet, and to infer relationships between ASes [11]–[14]. The same ideas can and have been used for tasks such as inferring customers of an ISP. This type of business intelligence can provide one competitor with an unfair advantage. An even more sinister use for such intelligence arises when one considers an antagonist planning an attack. The more information about a network they can gather, the more likely it will be successful. It is precisely this sort of information that would be needed to mount a malicious hijack of address space [15]–[17].

Despite its many security issues, the Internet (or at least the component in which BGP operates) is a relatively benign. The majority of participants in BGP are well intentioned, and more problems (in the routing infrastructure) appear to have been caused by mistakes than deliberate malfeasance. There are other networks where security is the paramount concern, rather than something added on after the fact. There are network operators who may wish to co-operate from time to time (i.e., by sharing traffic), but who are very concerned that private details do not leak from one network into another. A simple example occurs in the military setting where national armed forces often co-operate in joint missions where lives depend on being able to share accurate, up to date information. However,

978-1-4799-1270-4/13/$31.00

today's allies may be tomorrows combatants, and so they may also wish to maintain secrecy regarding their networks design and capabilities. Furthermore, in the operations of their own networks, they may wish to limit the damage that can be done if a router (or group of routers) is compromised. It is therefore interesting to explore the limits to which a routing protocol can preserve the privacy of its participants.

We will present here a routing protocol aimed at preserving exactly that. We show that a great deal of information can be hidden. In particular, we show that a common routing algorithm — the distributed Bellman-Ford shortest-paths algorithm — can be performed without participants learning distances!

We call our new protocol STRIP (Secure-Transitive RIP), after the iconic distance-vector protocol RIP (the Routing Information Protocol). STRIP conceals nearly every aspect of the network from its participants. Participants in the protocol learn their neighbours, and the next-hop router on the route to a destination, but they learn very little else. In particular, participants do *not* learn any routes or distances (other than to their immediate neighbours).

The algorithm has other desirable properties. It uses a strong method to authenticate potential paths, step by step, much as stronger versions of BGP security improvements aim to do. Thus we get authentication almost for free.

It is also easy to see how the algorithm can be extended. BGP uses a path-vector algorithm that resembles a distance-vector protocol in some respects, and this paper also presents a number of ways in which our distance-vector protocol can start to be adapted to the path-vector setting.

As always there are costs to security. We have implemented the proposed protocol in both simulation software, and as a real distributed routing protocol, and analysed the performance overhead, and it is not trivial. We do not suggest that the costs are warranted for the Internet in general, but we believe that a thorough understanding of what is possible should inform future routing protocol designers, and we leave it to them to choose the tradeoff between privacy and overhead.

Privacy-preserving ideas have been applied to interdomain routing before. Zhao *et al.* [18] showed a verification mechanism for routing decisions, and Gupta *et al.* [19] use a *secure outsourcing* approach to move the routing decision process from the routers to a centralised computation cluster. Our work, which is motivated by an earlier presentation [20], maintains the distributed nature of a routing protocol, and we do not rely on any additional players except a key distribution mechanism (PKI). Another difference is the type of information that is kept private. Gupta *et al.* do not consider network information private whereas our solution not only protects policy but also a great deal of the network information.

II. Secure Multiparty Computation

Secure multiparty computation protocols are a set of techniques, often cryptographic in nature, which enable parties to carry out distributed computation tasks without having to reveal their private data.

Perhaps the most famous problem in the area is called the Millionaire's Problem, where two millionaires meet on the street, and wish to determine who is wealthier, but without revealing their own weath. It was shown early on [21] that any polynomial time function could be computed in a secure distributed manner, and this provided a simple solution to the Millionaire's Problem.

Yao's approach [21] is not always practical, but there is now a substantial literature on secure multiparty computation and the closely related area of privacy-preserving data mining, and many available techniques. In our protocol we use homomorphic encryption to securely compute a distributed sum, and we briefly describe this approach below.

A. Homomorphic Encryption

A homomorphic encryption scheme is a public-key encryption scheme with a special homomorphic property.

In a public-key encryption scheme each party gets a pair of keys, one called the public key and the other called the private key. The public key is published and used for encrypting, while the private key is kept private and used for decrypting. This approach eliminates the need for the parties to establish a shared secret before exchanging encrypted messages.

An encryption scheme is called *homomorphic* if there exists an operation on two ciphertexts that is equivalent to another operation on the corresponding plaintexts: i.e.,

$$\mathrm{Enc}(x) \odot \mathrm{Enc}(y) = \mathrm{Enc}(x \oplus y),$$

for some operations $\odot$ and $\oplus$. In our protocol we use an *additive* homomorphic encryption scheme, where the operation $\oplus$ corresponds to standard arithmetic addition.

The elegant feature of this approach is that we can create a sum of a series of values held by different parties, but which is encrypted. Only the holder of the private key can decrypt it and determine the sum.

In detail, the secure distributed sum is computed as follows.

1) Assume we have n parties $P_1, P_2, \ldots, P_n$ with corresponding inputs $x_1, x_2, \ldots, x_n$, and we wish (at completion) for party P_n to know the sum
$$X = \sum_{i=1}^{n} x_i.$$
2) Each party encrypts their data with the public key of P_n to obtain $y_i = \mathrm{Enc}(x_i; P_n)$.
3) WLOG we assume that they transmit cumulative sums to the next party in sequence, i.e., party P_i sends the following to P_{i+1},
$$y_1 \odot y_2 \odot \cdots \odot y_i.$$
4) Finally, P_n receives $y_1 \odot y_2 \odot \cdots \odot y_{n-1}$, and decrypts it with his private key and adds his own value x_n.

Note that for more than two parties P_n learns nothing about $x_1, \ldots, x_{n-1}$ apart from the sum $\sum_{i=1}^{n-1} x_i$.

There are a number of possible homomorphic encryption schemes. Here we use Paillier encryption [22], which is homomorphic with respect to summation (as required), and comparatively simple to implement.

B. Key distribution problem

The one critical requirement for the public-key encryption system is a Public-Key Infrastructure (PKI). For a router to determine a route to a destination it has to know its public key. That is, it has to know the key before it has a valid route to the destination. So it can not ask the destination for the key beforehand, and even if this was possible, how can the router know that this public key belongs indeed to the destination and not to some attacker pretending to be the destination router.

This problem is not limited to our scenario, indeed it is well know in public-key cryptography applications. The most prominent solution to this problem is the introduction of a trusted third party who provides the PKI. They act as a public-key broker: verifying and linking identities to public keys, and distributing those keys in a secure manner.

In general, creation of PKIs is non-trivial. However the pressing need for improved BGP security means the problem has been tackled for Internet routing. A PKI system specially designed for routing infrastructure called RPKI (Resource PKI) [23] has already undergone some testing [24].

III. STRIP

We call our protocol STRIP (Secure-Transitive Routing Information Protocol). Its aim is to find shortest paths through a network, while revealing minimal information about that network.

The purpose of a routing protocol is to provide an automated and distributed means to create *routing tables* at each router. These are essentially tables of destinations accompanied by *next hops* (the first step from the current router along the path to the destination), along with some information about the paths (such as distances).

The "destinations" in our protocol might be the routers themselves, but could equally be some aggregate such as the Autonomous Systems of BGP, or some set of subnets attached to routers. However, for the sake of simplicity, we shall equate destinations and routers here.

There are several approaches for calculating shortest-paths. *Path-Vector Protocols* (PVP) use an interesting approach that combines the information passing and decision processes, and it is this approach that we shall generalise in STRIP.

PVPs are sometimes called "routing by rumour". In a PVP each router shares its routing table entries by announcing them to its immediate neighbours. Thus the neighbours learn of potential destinations, and potential paths towards these destinations. The advertised information in the table includes the path (and in the case of BGP it contains other metrics), and these can be used to discriminate between potential choices when a router learns of more than one path.

In BGP, the best-path decision is made based on multiple metrics, but here we will concentrate on shortest-paths, thereby avoiding some of the intractable problems of BGP convergence [6], [7]. However, it should be clear that the approach we propose generalises to allow for multiple metrics composed as lexicographic products [25].

In our protocols we allow each link in the network to be assigned a weight and the weight of a path is the sum of its edges. The selection criteria then is to select the path with the lowest weight (the shortest path).

The announcements of a typical PVP contain the destination, the weight of the announced path to the destination, and a list of routers contained in the path. Thus selecting a path is just a matter of comparing the new path weight with the existing one, and loops can be detected (and eliminated) if the receiving router is already present in the list of routers.

Clearly such announcements leak information about the configuration of the network. As many of these are sent, and received, they can be used to create a combined picture of the network and its policies [14]. In contrast, the proposed STRIP protocol keeps this information secret.

However, there is still a minimal amount of information that must be public. We assume that routers know their neighbours (this is reasonable as establishing a link requires co-operation). We also assume that a router knows (or determines) the link weight of the directed edges from it, to its neighbours (the weights don't need to be symmetric, and the routers don't need to know the distances of the links towards them).

A. Route propagation

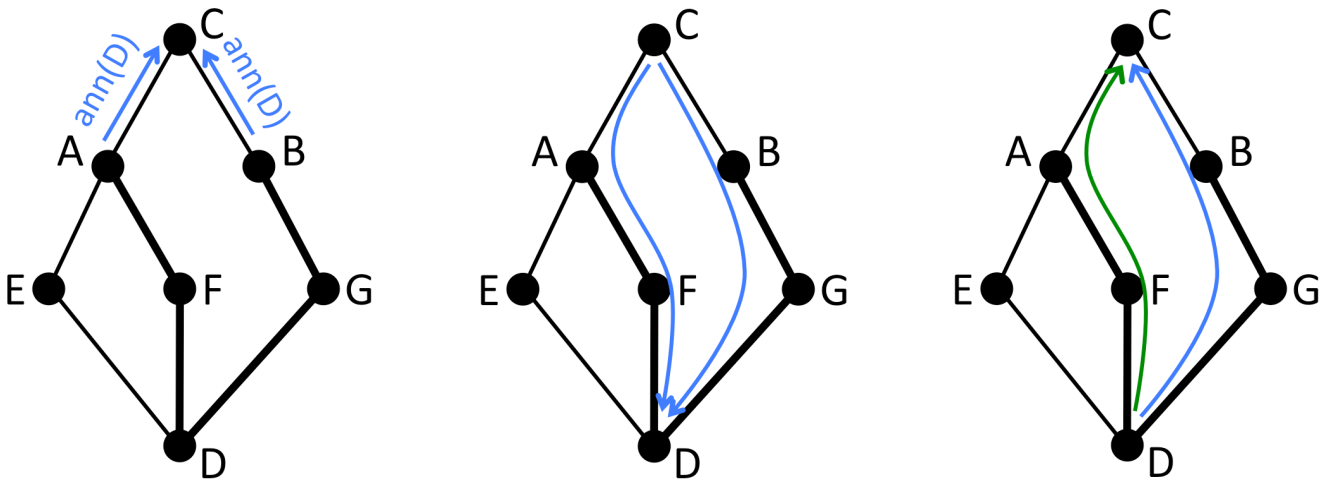
We make two major modifications to a typical PVP. The first is that we alter the information routers transmit from their routing tables. They still transmit potential destinations, but these announcements are now simply a list of destinations without any information about the path. A router will then learn of alternative next-hop routers towards a destination.

The second major difference between STRIP and PVPs is that the router does not itself decide between these alternatives. In order to decide between them, it starts a *shortest path computation*, in which the destination router is asked to make the decision (described in more detail in Section III-B).

Although we modified the PVP significantly, we want to stress that we did not change the underlying mechanism for route propagation: all known routes to a destination are assessed, and the shortest is selected and announced to the neighbours. Hence, its convergence properties are almost identical to those of conventional PVPs.

A schematic overview of the operation of STRIP is given in Figure 1. It shows the process: router C receives announcements of the destination D from A and B; C sends a shortest-path computation request to D via the two alternative paths; D makes the computation, and responds to C with the best path; and then (as in all PVPs) C would announce its new destination to its neighbours, who might then commence their own computation.

It may seem, on the face of it that D needs to know about C before it can return the message, but we will explain in Section III-B how to avoid this problem. So, although the process involves more message passing, it is logically identical to the standard PVPs, which are known to converge correctly for shortest paths (given non-negative weights).



(a) Each node that knows of a route to a destination advertises it to its neighbours. In this example A and B announce the existence of a path to D to router C

(b) C starts a shortest-path computation by sending requests to the announcing routers A and B, who forward the request on their known routes to the destination D. Along the path each involved router adds the distance to the next hop to the path's distance stored in the request.

(c) After receiving all requests for this SPC, D determines the shortest path, and then sends replies to C along the reverse paths. C then updates its forwarding table and send an announcement of the newly learned route (in green) to D to its neighbours.

Fig. 1: Operation of STRIP. The dark lines show the pre-existing paths (though note that each router only knows its next hop).

B. Shortest Path Computation (SPC):

The main change to the PVP is the way shortest paths are selected. In STRIP a router starts a SPC: a distributed computation involving the routers on the known paths to the destination. The idea is that the originating router send a "probe" message to the destination along all paths known to him. The intermediate routers add the path weight for the respective links to the messages and the destination router, after receiving all probes, decides on the shortest path and sends a reply back to the originating router.

We could easily do this with all messages in the clear, resulting in a modified PVP. This would have advantages in itself. For instance, the response proves that the path is valid: this is not at all guaranteed by the current version of BGP (hence the need for BGPsec).

However, we make the additional change that the weight-sum is computed using homomorphic encryption, as described in Section II-A, and we anonymise so that D learns little from its role in the computation.

In detail: the originator of the SPC sends an SPC-request to all neighbouring routers that have announced a route to the destination. An SPC-request contains:

- The address of the destination.
- A random computation ID, unique to each SPC.
- A random path ID, unique to each path in the computation.
- A distance field, holding the encrypted sum of the weights of the links of the path.
- A random "originator" encryption key for a symmetric encryption scheme, encrypted with the destination routers public key.
- A timestamp, holding the creation time of the request.
- The distance of the current shortest path to the destination known to the originator, encrypted with the destination routers public key.

Note that only the destination router can decrypt the message details. No one else can learn anything about the distance, and although D learns a set of distances, it does not know the origin, since the random path and computation IDs and the random key are anonymised.

If an intermediate router receives a SPC request, it looks up the next hop to the destination in his routing table (it must have one for this to be a valid path), and adds the distance for that link to the encrypted distance field. It then stores the last hop, path id and computation id in a temporary routing table, and sends the request to the next hop. The temporary routing tables serves the purpose of enabling the network to route the responses back on the same path as the corresponding request without revealing the identity of the originator.

If the destination router receives a SPC request it waits a certain period of time (the `waitForRequests` time) for other requests of the same computation to arrive. Any requests with the same computation ID after the timeout are ignored. It then decrypts all distances, chooses the smallest one and prepares the response messages. A response is generated for every request it has received. The response contains the path ID, computation ID, and the ID of the path with the shortest weight encrypted with the random key sent by the originator.

C. Avoiding redundant announcements

The above protocol would work, but for the protocol to converge to the optimal solution, a router has to announce

every route change. Every announcement triggers a potentially expensive computation, so we want to omit unnecessary announcements. In particular, we want to avoid re-announcing a route if it hasn't really changed.

How does a router know if a route it is using has really changed? Presuming the router has received new announcements itself, and undertaken a new SPC, it could make one of two decisions:

1) change the next hop – in which case it is perfectly obvious that a route change has occurred and that it should re-advertise; or
2) keep the next hop the same.

In the latter case, we must remember that the router has only local information, so it is not clear whether:

1) the new route is the same; or
2) the next hop is the same, but the route is different at some downstream point.

In the second case, we must re-advertise, because distances may have changed, and this may affect the decisions of other routers (even if the local next hop is the same). In the first case, we can omit re-announcing, and thus avoid overhead.

To enable a router to detect a route change in the second case we have to extend the protocol: the destination router not only returns the shortest path ID but also the encrypted distance of this path. Now if the originator starts a new SPC for that destination, he adds the old distance to the request. The destination router can then compare the new distance to the old distance and adds the result of the comparison to the response. The full response contains:

- The computation ID.
- The path ID.
- The path ID of the shortest path, encrypted with the encryption key sent by the originator.
- The distance of the shortest path, encrypted with the public key of the destination router.
- The "same distance" flag: a Boolean flag showing if the distance of the old and the new shortest route are identical, encrypted with the encryption key sent by the originator.

If the originator of the SPC receives a reply to his computation request it decrypts the shortest path ID and the same distance flag. He will only update his routing table if he received a reply for this computation through the path of the new shortest path and then announce the update to his neighbours if either the new next hop is different to the old one or if they are the same but the flag is not set.

Note that, we only need to pass a same distance flag. We don't need a "same path" flag because the change is only important if it affects downstream decisions, and this will only be the case if the distance changed.

D. Timeouts

For every SPC there has to be temporary routing information stored at routers. In order to reduce memory overhead, we need a mechanism to decide when it is safe to delete this information.

Also, since announcements for the same destination often come in close succession, we introduced a waiting period for the originator to start the SPC after receiving an announcement. This reduces the number of SPCs (at the cost of potentially delaying convergence).

In addition there is the time the destination waits for queries before computing shortest paths. In all there are three timers required:

1) `waitForAnnouncementsTime`: the time a router delays between receiving an announcement for a destination, and commencing the SPC.
2) `waitForRequestsTime`: the time that a destination waits from receiving a SPC request, before beginning a SPC response.
3) `waitForRepliesTime`: the time that information is kept in temporary storage for reverse-path lookups, in particular, the time the originating router waits for replies after creating an SPC request.

Figure 2 shows the timeline of these different timeouts. We will discuss the choice of these parameters in Section IV.

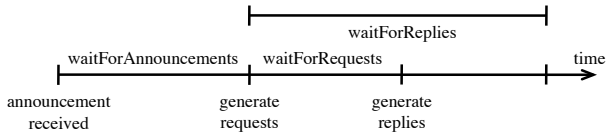


Fig. 2: timeline of `waitTime` timers.

Without timeouts STRIP behaves essentially like a distributed asynchronous Bellman-Ford algorithm, so the same argument for convergence applies (see [26, Chapter 5.2.4]). With the introduction of timeouts convergence behaviour is not that clear. For instance, if `waitForRequestsTime` or `waitForRepliesTime` are chosen too low, then no path with minimum transmission delay greater than those timeouts will ever be discovered. Transmission times for messages depends on processing, queuing and transmission delays at and in-between routers which are all variable in nature, and we might envisage situations where this leads to long-term oscillation. However, most realised protocols have such timers, and standard practices such as including jitter in timers have generally been accepted as approaches to mitigate such problems, and we use these here. In our experiments (detailed later) we saw no problems with long term oscillation, and we examine the correct choice of timers to avoid incorrect convergence problems.

E. Implicit Loop Detection

Since a SPC request only leads to a new routing table entry if the request traveled to the destination and back along the new shortest route, this entry must be loop free.

F. Privacy

The protocol has privacy-preserving properties in the *honest-but-curious* security model, i.e., if the parties correctly follow the protocol, there is no efficient, single adversary

that can extract more information from the transcript of the protocol execution than is revealed by that party's private input and the results.

a) Topology information: From any shortest path computation an originating router will only learn the next hop to the destination. But that is information it already knows, i.e., its neighbouring routers.

The destination router involved in the shortest path computation can also not learn any new information about the topology of the network, since the path information (the path, and the computation ID) is distributively stored in the memory of the routers of the path. Without collusion this information is not obtainable.

b) Distance information: No intermediate router in a shortest path computation can gain distance information since this information is encrypted with the destination routers public key. Only the destination router is able to decrypt. Therefore the security is based on the security of the homomorphic encryption system. And although the destination router learns the distances of all paths it can not link this information to any router in the paths since the only information about the paths it learns is just the last hop. Note that all information about the originator of the request is anonymised.

G. Authentication

Our protocol provides destination authentication, i.e., the originator of a request can be assured that the response was created by the destination router and no one else.

For every shortest path computation the originator creates a random symmetric key K_s. It then encrypts K_s with the public key pk_D of the destination D and adds it to the request. Only D, the holder of the private key corresponding to pk_D, can learn K_s. But since K_s is necessary to create a valid reply it could have only be the destination creating the reply. And the originator can be assured that the reply is fresh and not a replay, because every SPC has a new key K_s.

H. Possible attacks outside the security model

The privacy properties of STRIP only hold in the non-collusion honest-but-curious security model. It assumes that participants follow the protocol, though they may seek to learn additional information.

This is a reasonable assumption for a routing protocol — routing requires protocols to be followed correctly at some level, or it can't reasonably be expected to work at all. However, we do not want to conceal weaknesses of the protocol against more powerful attackers.

If we allow parties to deviate from the protocol or to collude with other parties they might mount the following attacks.

Sabotage: Routers can sabotage the protocol in a number of ways. They can drop random packets (either control packets to sabotage computations, or data packets after the fact). They can use invalid input weights to manipulate the results. They can also perform a Denial of Service (DoS) attack. Since every announcement triggers a rather expensive shortest path computation flooding the network with announcements potentially leads to overload.

The emphasis in our protocol is privacy, not protection from such attacks, which are in any case possible at present with most current protocols.

Attacks to gain information: There are several ways in which a participant in the protocol might actively attempt elicit additional information. Firstly, an originating router O can request multiple route computations with different subsets of its peers. The result is the next hop for each subset of peers, and from this O can deduce the order of the routes. A partial version of this may happen during route convergence, and so partial orderings are one form of leakage in this protocol, when performed multiple times. A more serious form of this attack involves O performing many such calculations and deliberately corrupting its component of the calculation by adding values to partial metrics it learns. This could potentially allow O to learn the actual distance metrics if performed enough times. However, for a metric with a reasonable range this attack requires many computations and is unlikely to be accomplished unnoticed.

Secondly, multiple routers could collude to obtain more information. For instance, the destination router and some other routers along a path sharing a common neighbour can compute its distance value by decrypting the partial sum of the distances and taking the difference.

Finally, once routes are established, traffic will follow. An observer of traffic may be able to learn a great deal about the routes in a network (e.g., see [27], [28]). Furthermore, by changing its own weights, and observing traffic flows, a node may be able to learn a substantial amount. These type of attacks are unavoidable as long as the traffic paths are not hidden from the nodes, but there do exist anonymous forwarding schemes (e.g. [29]) that allow one to disguise sources/destinations and routes through a network. If such a scheme were designed for use on top of privacy-preserving routing, then we may be able to avoid this last type of information gathering attack, though typically at the expense of some loss of efficiency.

IV. Evaluation

Cryptographic protocols usually create overheads, and in the case of STRIP there are messages passed above and beyond those of a typical PVP. Our first goal, therefore, is to determine the overheads of STRIP.

We can see two types of overhead: the additional messages passed (and the additional length of these messages in comparison to those of a PVP protocol), and the cost in terms of extra time to converge. There is an additional computational cost to the protocol, but we will subsume that through the calculation of additional delays.

We assessed the protocol through two means: simulation and implementation. The simulation is necessary because we don't have the resources to assess the performance of the real implementation in distributed hardware, so the simulation is used to show scaling of the protocol, and used for setting features such as timers that required many experiments. The final proof of the pudding lies in the implementation.

We will first discuss the simulation results. We wrote a discrete-event simulation of STRIP focusing on the processing capabilities of the routers in the network. We implemented the simulation using Python and in particular the SimPy [30] package, a framework for implementing a process-based discrete-event simulation.

The routers are modelled to have a processing queue where the SPC-packets are processed sequentially, in order of their arrival. Each router can have one or several of these processing queues.

The STRIP protocol routers were configured with the following parameters:

- `requestTime`: the time a router needs to process a SPC request. This is dominated by the encryption. We set this value on all simulations to 4ms, the time for our Paillier encryption on a fairly standard CPU.
- `replyTime`: the time a router needs to process a SPC reply. That's just a lookup in the temporary routing table. We set replyTime to 0.1 ms.

We implemented the `waitForAnnouncementsTime` (wfat) as a random variable X with a uniform distribution and $1/2$ wfat $\leq X \leq 3/2$ wfat. If the waiting time is fixed it can lead to bursts of almost simultaneous shortest path computations which create load bursts on the routers. With the randomisation we achieve a more equal load on the routers, and avoid potential synchronisation affects.

We also implemented a simulation of the path vector protocol described in III. For a fair comparison the protocol includes an `announcementDelay` parameter similar the `waitForAnnouncementsTime` in STRIP (routing protocols often have such a parameter, for instance the `minRouteAdvertTimer` in BGP). When a router learn a new route it waits for `announcementDelay` before announcing the changes to its neighbours. The purpose is to prevent load spurts and it is randomised the same way as the equivalent parameter of the STRIP protocol.

We test the protocols on several different networks whose topologies are described below. Every link between routers has a transmission delay of $5 + x$ ms, with x being sampled from an exponential distribution with mean 1. In each case, the routers are started at the same time, and start by announcing their immediate neighbours. This is the most stressful test of the performance of the algorithm.

For every configuration we ran the simulation 50 times with different seeds for the random number generator – the reported figures are the averages over these simulations.

A. Comparison

The added privacy measures of STRIP introduce overhead compared to the PVP. Figure 3 shows the convergence time of STRIP vs. the path vector protocol in an Erdös-Rényi graph with $p = 10\%$ and respective number of nodes and a Barabási-Albert graph with 2 new edges for each node.

We observe that the convergence times increase due to the additional message passing delays, but that the increase is perhaps 20% on average over all the cases. If the computational times of the encryptions were reduced, then even these overheads could be substantially reduced.

We don't suggest either of types of network are realistic, but they test opposite extremes. The former is a relatively regular network, while the latter has power-law degree, and this creates a small number of high-degree hub nodes. We can see that in the results where both protocols converge more quickly on the Barabási-Albert graphs, due to the smaller search space (routes concentrate on the few hub nodes).

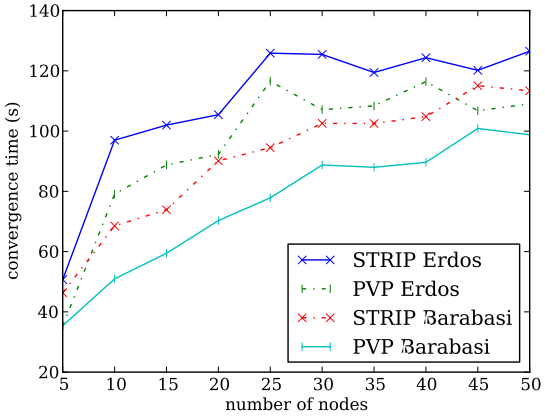


Fig. 3: comparison of the convergence time between STRIP and PVP for Erdös-Rényi and Barabási-Albert graphs.

There is a more significant difference in the communication costs of the two protocols. Figure 4 shows the number of messages sent until convergence for the same graphs as in Figure 3. The number of announcements made by both protocols are similar, but the SPC in STRIP introduces additional messages. The sizes of these messages are dominated by the cypher-texts, a request contains two, and a reply contains one cypher-text. With Paillier's encryption scheme with a key size of 1024 bits, the cypher-text are 2048 bits long. Thus requests are around 512 bytes and replies are around 256 bytes. The extra, larger messages introduce a communications overhead, but it is still manageable simply because todays networks have an exponentially larger available bandwidth than those for which a typical PVP was designed (20 years or more ago).

However, the number of requests and replies grow significantly faster than the number of announcements for bigger graphs, since there are more paths to be evaluated and the average path length grows.

B. Parameter Choice

STRIP has three parameters that need to be configured: `waitForAnnouncementsTime`, `waitForRequestsTime` and `waitForRepliesTime`. Their chronological sequence is shown in Figure 2.

The value for `waitForRequestsTime` depends on how long a request travels from the originator to the destination through the network. Too small, and requests are dropped;

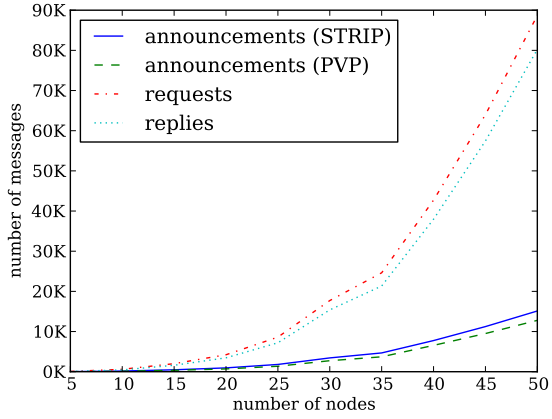


Fig. 4: number of messages sent until convergence in the graphs from Figure 3.

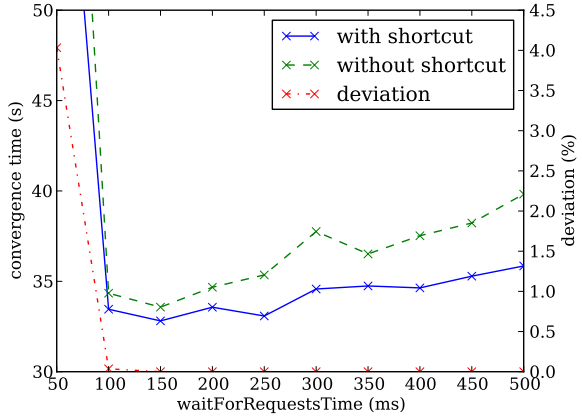


Fig. 5: convergence time and deviation from optimal routing solution for different values for `waitForRequestsTime`.

but large values slow down convergence. Figure 5 shows convergence time and deviation from the optimal routing solution for different values for `waitForRequestsTime` in an Erdös-Rényi graph with 30 nodes and $p = 15\%$.

Figure 5 shows that the convergence times (left axis) generally increase with an increase in the `waitForRequestsTime` parameter. This effect can be dampened by an implementation trick we call *shortcut*. The originator knows how many requests it has created and therefore, after receiving a reply for every request it sent, it does not have to wait any longer because there cannot be any more replies. Sending the number of requests in each request to the destination router means it can stop waiting after it receives all the requests. Figure 5 shows convergence times with and without *shortcut*.

Figure 5 also shows the "deviation" (right axis) from the correct routing solution, which can be non-zero if too many routing messages are dropped. The figure shows that there is a minimum value for this parameter, above which the routes converge correctly. In all our simulations we found that choosing `waitForRequestsTime` to be $n(c + td)$, where n is the number of nodes in the graph, c the time it takes to compute an encryption and td the average transmission delay on a link, results in an optimal solution being found. This was a reasonable compromise between additional convergence time, and finding the optimal solution.

The value for `waitForRepliesTime` depends on a composition of the value for `waitForRequestsTime` and the times it takes for the replies to travel back to the originator. We chose `waitForRepliesTime` to be twice the `waitForRequestsTime`.

The `waitForAnnouncementsTime` parameter also has a dramatic influence on the convergence time. Choosing this to be too small leads to STRIP not finding the optimal solution. If it is too small, the routers receive too many requests to process, so queues build up and at some point the number of requests may exceed the processing capabilities of the router leading to

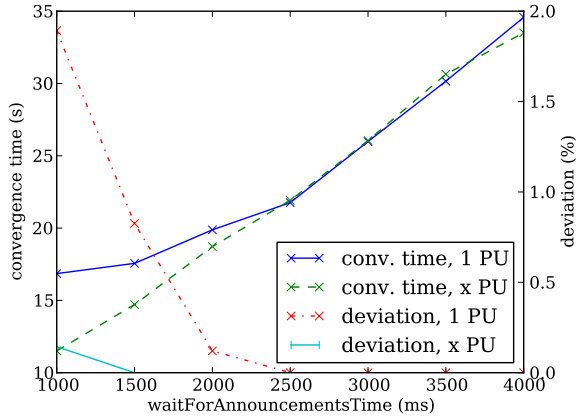


Fig. 6: comparing convergence time and deviation from optimal routing solution for different values for waitForAnnouncementsTime for single and multi processing unit (PU) routers.

dropped requests, but larger values slow convergence. Figure 6 shows the convergence time of STRIP in an Erdös-Rényi graph with 30 nodes and $p = 15\%$. The effect of the deviation from the optimal solution can be dampened by allowing routers to process requests in parallel (in this measurement series we assigned a router one processing unit for every 4 ports). The resulting convergence times are almost linear in the delay time.

C. Performance

The performance of the protocol with regards to the smallest time to converge to the optimal routing solution depends on the number of messages, because processing the cryptography in the messages is the bottleneck. If the processing queue of a router fills faster than it can process the messages it increases the overall round-trip time for a shortest path computation, or in extreme cases messages may be dropped.

Obviously, the more routers in the network, the more messages have to be processed, but the number of links in

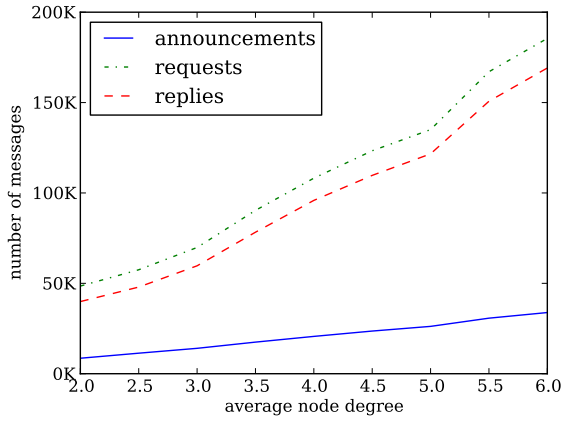


Fig. 7: number of messages for graphs with $n = 30$ but different average node degree.

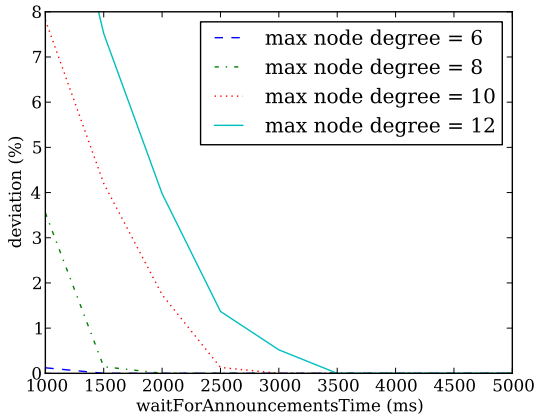


Fig. 8: deviation from optimal routing solution for graphs with $n = 30$ and same average node degree but different maximum node degree.

a network also has an influence on the number of messages sent. Figure 7 shows the number of messages in an Erdös-Rényi graph with 30 nodes with varying average node degree. The increase of edges in the graph leads to more possible paths between nodes, which consequently requires that more paths has to be explored in each computation.

The overall amount of load for the routers is not the only determining factor for the protocol performance. It also depends on how this load is distributed over all routers. Figure 8 show the deviation from the optimal routing solution for an Erdös-Rényi graph with 30 nodes with an average node degree of 3 but different maximal node degrees. It shows, that the nodes with high node degree are the bottle neck for the performance in the protocol, because a higher node degree means being part of more paths and therefore having to participate in more shortest-path computations.

V. Implementation

We implemented the protocol using the Python programming language. We chose Python because of its suitability for rapid prototyping and its vast collection of libraries. The core of STRIP is *twisted* [31], an event-driven networking engine. It abstracts the underlying complexity of networking by providing a suitable set of primitives. It enabled us to implement STRIP in just 680 lines of code.

For example, setting up a TCP server listening for connections from other routers is achieved using just one line of code:

```
stripservice = internet.TCPServer(
    config.getint('server','port'),
    STRIPServerFactory(router))
```

The STRIPServerFactory thus created in turn creates a STRIPServer object for each connection. Processing messages simply requires overwriting of the stringReceived method.

```
class STRIPServer(NetstringReceiver):
    def stringReceived(self, data):
        ...
```

For the homomorphic encryption we chose Paillier's encryption scheme [22]. We based our implementation on the work of [32], but we used a wrapper module to be able to use the very efficient GNU multiple precision arithmetic library to speed up computations dramatically. We also improved the performance of decryption by applying the Chinese Remainder Theorem (CRT). The computationally expensive part of the decryption is a modular exponentiation with a large exponent. The CRT enables us to divide the exponentiation into two exponentiations with much smaller exponents. The new exponents have just half the bit size of the original one. This trick leads to an improvement of approximately a factor of 4.

The source code is available for download at https://github.com/wilko77/STRIP.

A. Emulation

We emulated routers using AutoNetkit [33], which creates a *netkit* [34] lab. Netkit is an environment for performing network experiments with several virtual network devices that can be interconnected to form a network on a single PC. Given the network topology description in graphML format, it automatically generates the netkit configuration files to set up the virtual routers and the connections. We modified AutoNetkit to also generate the STRIP configuration files.

It is important to realise, though, that this is a real protocol stack, running on real router software (on virtualised router hardware). So our emulation experiments can demonstrate success of the protocol, and overhead in terms of messages, but convergence times for such a network are not accurate, hence the need for the earlier simulations.

We used the following networks for the emulation runs. All networks consist of 11 nodes.

- Random tree. It has the smallest possible amount of edges for a connected graph.
- Clique. Fully connected, thus maximum amount of edges for a graph. Random weights on the edges.

- The Abilene network. Weights on the edges are the distance of the edge in km.

Netkit starts the virtual routers in a sequential order, only after the first one is fully running it starts the next one. We run an emulation until the routing converges and then tested the result for correctness. Table I shows the number of sent messages for the different networks.

TABLE I: Comparison of the costs

network	#encryptions	#announcements	#other msgs
tree	512	91	584
fully connected	14 477	2 069	26 570
Abilene	1 286	192	1 954

As seen with the simulations the density of a network is a determining factor for the communication and computation costs. The difference between the best case (tree) and worst case (clique) for the same size network is significant. However, real world networks are rather sparse as every link introduces more costs and does not necessarily result in better performance.

VI. Conclusion

This paper presented STRIP, a shortest-path routing protocol, which doesn't reveal the length of paths to its participants. It opens up a set of operations that could enhance privacy (and hence security) in future protocols.

The protocol has limitations: it introduces overheads, and doesn't implement all of the features that one might like to see in a modern, BGP-like protocol. However, the basic components of the protocol are easily extensible: the homomorphic encryption can implement other types of path metrics, or combinations of them. And we aim to work on improving its efficiency to reduce the overheads of the protocol.

Acknowledgement

We would like to acknowledge contributions to the ideas in this paper from Yin Zhang, who was involved in an earlier proposal [20]. We would also like to acknowledge the support of an Adelaide Scholarship International, a supplementary Scholarship of the Defence Systems Innovation Centre, and Australian Research Council grant DP0985063.

References

[1] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *ACM SIGCOMM*, 2006, pp. 291–302.

[2] R. Hiran, N. Carlsson, and P. Gill, "Characterizing large-scale routing anomalies: A case study of the china telecom incident," in *Passive and Active Measurement Conference*, 2013.

[3] R. McMillan, "Youtube outage underscores big Internet problem," *Infoworld*, 2008, http://www.infoworld.com/t/applications/youtube-outage-underscores-big-internet-problem-702.

[4] J. Cowie, "China's 18-minute mystery," *Renesys blog*, November 2010, http://www.renesys.com/blog/2010/11/chinas-18-minute-mystery.shtml.

[5] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," in *ACM SIGCOMM*, 2000, pp. 175–187.

[6] K. Varadhan, R. Govindan, and D. Estrin, "Persistent route oscillations in inter-domain routing," *Computer networks*, vol. 32, pp. 1–16, 2000.

[7] T. Griffin and G. Wilfong, "Analysis of the MED oscillation problem in BGP," in *ICNP*, 2002, pp. 90–99.

[8] R. Bush, O. Maennel, M. Roughan, and S. Uhlig, "Internet optometry: assessing the broken glasses in Internet reachability," in *ACM SIGCOMM*, 2009, pp. 242–253.

[9] T. Griffin and G. Huston, "BGP wedgies," RFC 4264, 2005.

[10] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 1654, March 1994.

[11] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the internet hierarchy from multiple vantage points," in *INFOCOM*, 2002, pp. 618–627.

[12] F. Wang and L. Gao, "On inferring and characterizing internet routing policies," in *ACM SIGCOMM*, 2003, pp. 15–26.

[13] J. Xia and L. Gao, "On the evaluation of AS relationship inferences," in *IEEE GLOBECOM*, 2004, pp. 1373–1377.

[14] W. Mühlbauer, A. Feldmann, O. Maennel, M. Roughan, and S. Uhlig, "Building an AS-topology model that captures route diversity," in *ACM SIGCOMM*, 2006, pp. 195–206.

[15] P. Boothe, J. Hiebert, and R. Bush, "How Prevalent is Prefix Hijacking on the Internet?" *NANOG 36*, February 2006.

[16] A. Pilosov and T. Kapela, "Stealing the Internet," in *Defcon 16*, 2008, www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-pilosov-kapela.pdf.

[17] A. Barbir, S. Murphy, and Y. Yang, "Generic threats to routing protocols," RFC 4593, 2006.

[18] M. Zhao, W. Zhou, A. J. Gurney, A. Haeberlen, M. Sherr, and B. T. Loo, "Private and verifiable interdomain routing decisions," in *ACM SIGCOMM*, 2012, pp. 383–394.

[19] D. Gupta, A. Segal, A. Panda, G. Segev, M. Schapira, J. Feigenbaum, J. Rexford, and S. Shenker, "A new approach to interdomain routing based on secure multi-party computation," in *ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI, 2012, pp. 37–42.

[20] M. Roughan and Y. Zhang, "Privacy-preserving routing," September 2006, Clean Slate Networks Workshop, Cambridge UK.

[21] A. Yao, "Protocols for secure computations," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1982, pp. 160–164.

[22] P. Paillier, "Public-Key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999, pp. 223–238.

[23] M. Lepinski and S. Kent, "An Infrastructure to Support Secure Internet Routing," RFC 6480, IETF, Feb. 2012.

[24] I. Phillips, O. Maennel, D. Perouli, R. Austein, C. Pelsser, K. Shima, and R. Bush, "RPKI propagation emulation measurement: an early report," IETF Talk, July 2012.

[25] E. Parsonage, H. X. Nguyen, and M. Roughan, "Absorbing lexicographic products in metarouting," in *The 1st International Workshop on Rigorous Protocol Engineering (WRiPE)*, 2011.

[26] D. Bertsekas and R. Gallager, *Data networks*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[27] S. Ratnasamy and S. McCanne, "Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements," in *IEEE INFOCOM*, vol. 1, 1999, pp. 353–360.

[28] M. Coates, M. Rabbat, and R. Nowak, "Merging logical topologies using end-to-end measurements," in *ACM Internet Measurement Converence*, 2003.

[29] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *USENIX Security Symposium*, 2004, pp. 21–21.

[30] "Simpy simulation package." [Online]. Available: http://simpy.sourceforge.net/

[31] "Twisted networking library." [Online]. Available: http://twistedmatrix.com/trac/

[32] D. Evans, Y. Huang, J. Katz, and L. Malka, "Efficient privacy-preserving biometric identification," in *Proc. of Network and Distributed System Security Symposium, NDSS*, 2011.

[33] S. Knight, A. Jaboldinov, O. Maennel, I. Phillips, and M. Roughan, "Autonetkit: simplifying large scale, open-source network experimentation," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 97–98, 2012.

[34] M. Pizzonia and M. Rimondini, "Netkit: easy emulation of complex networks on inexpensive hardware," in *TridentCom*, 2008, pp. 7:1–7:10.