Preserving Privacy at IXPs

Xiaohe Hu[†] Arpit Gupta[◇] Nick Feamster [◇] Aurojit Panda[‡] Scott Shenker[*]
[†] Tsinghua university [◇] Princeton [‡] NYU/Nefeli [*] UC Berkeley/ICSI

ABSTRACT

Autonomous systems (ASes) on the Internet increasingly rely on Internet Exchange Points (IXPs) for peering. IXPs provide the physical infrastructure including a fabric required to interconnect ASes. A single IXP may interconnect several 100s or 1000s of participants (ASes) all of which might peer with each other. This requires each participant to maintain a BGP session with every other participant in an IXP and poses a scaling challenge. IXPs have addressed this challenge through the use of *route servers*. When route servers are used IXP participants outsource parts of their policy to the route server and maintain a single BGP session with it. The route server is responsible for implementing parts of each participant policies – often export and import policies. While route servers allow IXPs to scale, they require participants to trust the IXP and reveal their policies, a drastic change from the accepted norm where all policies are kept private. In this paper we look at techniques to build route servers which provide the same functionality as existing route servers (thus providing scalability) without requiring participants to reveal their policies thus preserving the status quo and enabling wider adoption of IXPs. Prior work has looked at secure multi-party computation (SMPC) as a means of implementing such route servers however this affects performance and reduces policy flexibility. In this paper we take a different tack and build on trusted execution environments (TEEs) such as Intel SGX to keep policies private. We present results from an intial route server implementation that runs under Intel SGX and show that our approach has $20\times$ better performance than SMPC based approaches. Furthermore, we demonstrate that the additional privacy provided by our approach comes at minimal cost and our implementation is at worse $2.1\times$ slower than a current route server implementation (and in some situations up to $2\times$ faster).

1 Introduction

Since its commercialization in the early 1990s, the Internet has comprised of a set of independent autonomous systems (ASes), each of which owns a portion of the Internet infrastructure and is responsible for building and maintaining this infrastructure. Two ASes can connect to each other whenever they are physically connected (*e.g.,* through a link or a L2 fabric), and policies on what traffic they are willing to carry on each other's behalf. Routing decisions between ASes are commonly implemented using the Border Gateway Protocol (BGP). BGP policies have generally afforded a great deal of flexibility to ASes, allowing them to specify policies for how paths are used to route traffic, what paths are advertised to other connected ASes, etc.

Increasingly ASes interconnect at *Internet eXchange Points* (IXPs) such as the ones run by AMS-IX [1], DE-CIX [12], etc. IXPs are large physical locations (*e.g.,* a datacenter) where multiple ASes are interconnected through an IXP provided fabric. We refer to each AS connected to the IXP fabric as a participant[1]. Large IXPs interconnect several 100 participants, *e.g.,* 780 ASes participate in DE-CIX's Frankfurt location [11] while AMS-IX in Amsterdam interconnects 870 participants [2]. IXPs centralize physical infrastructure required for ASes to interconnect, thus reducing the cost of peering between ASes.

While IXPs reduce the cost of physical interconnectivity between ASes, they pose a scalability challenge for BGP implementations. BGP requires the use of long-lived BGP sessions between any pair of connected ASes. Traditionally, ASes participated in several exchange points, each of which contained at most of 10s of participants. As a result BGP implementations are designed to support a small number of BGP sessions and cannot scale to the 100s or 1000s of sessions required when an AS participates in an IXP.

Many IXPs rely on *route servers* [21] to address this scalability challenge. Intuitively a route server acts as a single AS which peers with each participant at an IXP. The route server is responsible for aggregating and distributing route updates from all participants in an IXP. IXP participants are also required to outsource a part of their policy computation to reduce the number of messages sent by the route server to any participant. Currently deployed route servers can filter announcements based on a participant import policies (which determine the set of paths considered by the participant's routing algorithm) and export policies (which limit the paths a participant makes available to other participants), thereby reducing the number of BGP announcements that need to be processed by each participant. Beyond enabling scalability, router servers also provide a convenient location for adding new interdomain functionality. For example, recent work on software defined internet exchanges [17, 16] has looked at using route servers to reconfigure the IXP fabric (in addition to making BGP announcements) as a way of providing additional functionality such as load balancing (between interdomain paths), etc. As a result route servers have both enabled IXPs to scale to large number of customers, and to provide additional services that differentiate them from competitors.

However, route servers also change the existing Internet trust model by requiring that participants trust the IXP with their policies. A recent survey of network operators [9] found that concerns about policy privacy are an impediment to the widespread adoption of route servers. As a result enabling policy privacy for route servers is essential to enabling IXP adoption.

Can one construct route servers which are scalable (allowing increasing numbers of participants at an IXP) and efficient (adding little or no overhead to BGP route computation), while keeping participant policy private? Prior approaches, including SIXPACK [9],

[1]Note, a participant can either physically place equipment at an IXP facility or connect remotely through the use of remote peering [8].

have proposed using secure multi-party computation (SMPC) [14] to construct such route servers. SMPC is a cryptographic primitive where computation is securely split across multiple servers. The servers are assumed to be honest but curious, and are required to cooperate to execute the program. SMPC mechanisms guarantee that in the absence of collusion servers learn neither the inputs nor outputs of a computation. While SMPC based mechanisms are sufficient for implementing route servers which ensure that policies remain private, as we show later the use of SMPC leads to a significant increase in a route server's runtime (§5). Furthermore, existing SMPC mechanisms require programs to be specified either as arithmetic circuits (*e.g.,* BGW schemes [4]) or as boolean circuits (*e.g.,* GMW schemes [10]), and some of the circuit elements (*e.g.,* products or logical conjunction) require several rounds of communication. As a result the use of SMPC also makes it harder to extend route server functionality, and hence restricts the IXPs ability to add new features.

In this paper we propose an alternate approach based on the use of trusted execution environments (TEEs) such as Intel SGX or ARM TrustZone to implement route servers that preserve policy privacy. Our approach leverages new hardware features to allow participants to verify that the IXP is running a trusted route server and to ensure that the IXP cannot use operating system or hypervisor exploits to read router server state. This allows us to ensure that the route server provides policy privacy, while placing little or no restriction on route server functionality[2]. This allows IXPs to easily extend route server functionality and allows greater generality when compared to SMPC based approaches. We more completely describe our approach in §3. We implemented our approach as an extension to SDX, and evaluated its performance using realistic traces. Our evaluation shows that the use of TEEs allows us to achieve up to a $20\times$ improvement in performance when compared to SIXPACK's SMPC based approach. Furthermore, our approach induces minimum performance penalty over a router server that does not provide policy privacy: our implementation is at most $2\times$ slower than the original SDX implementation, and in some cases is up to $2\times$ faster. We start by describing our assumptions and threat model.

2 IXPs and Privacy Model

We begin by providing some background on IXP route servers, and then provide a brief overview of the threat model considered in this work.

2.1 IXP Route Server

Internet exchange points provide a physical facility where ASes can peer. They do so by both providing a physical facility (similar to a server colocation facility) where participants can place equipment and a L2 fabric providing connectivity between participants. Participants can then use this fabric to exchange data or routing information through BGP. As is standard, ASes participating in an IXP exchange route information using BGP, and must establish a BGP sessions with every other AS with which they exchange routing information. In the absence of a route server an IXP would require each participant to establish a session with every other participant, which requires that each participant store state for $O(n)$ sessions (where n is the number of participants) and the IXP fabric carry data for all $O(n^2)$ active sessions in the IXP, thus limiting scalability. IXPs rely on route servers to address this scalability challenge: each participant opens a single BGP session with the route server, which is then responsible for aggregating and propagating routing announcements from all participants. This reduces the number of active BGP sessions by a factor of n, improving scalability.

Route servers, *e.g.,* BIRD [7], implement mechanisms for filtering and ranking BGP route announcements [21]. Similar to BGP routers, route servers store any announcements that they receive (from participants) in a set of tables referred to as the routing information base (RIB). As opposed to BGP routers which use a single RIB, most existing route server deployments rely on a global RIB (containing routes that the route server can propagate to participants) and a set of per-participant RIBs (containing routes that are known to the participant). Participants configure a route server by providing it with an *export policy*, an *import policy* and a *ranking policy*. They then connect to the route server by establishing a BGP session. When a participant X advertises a route it is added to the participant's RIB, and the route server uses IXP specific import polices[3] to determine whether the announcement should be added to the global RIB. When an announcement from participant X is added to the global RIB, the route server applies X's export policy to check whether the route should be advertised to another participant Y, if so the route server uses Y's import policy to determine whether the announcement should be copied to Y's RIB. Finally, if the announcement is copied to Y's RIB the route server uses Y's ranking policy to order all available paths to each destination and it then announces any path changes caused by X's announcement. The same process is repeated for all other participants in an IXP. See [21] for a more detailed description of how route server's function. Finally, note that current route server deployments do not support the use of complex ranking policies, and instead rank routes based on path length and other standard BGP policies. Furthermore, many IXPs including AMS-IX [1], DE-CIX [12], etc. allow for hybrid deployments where a participant can both directly establish BGP sessions with some peers, while using the route server for others.

In this work we focus on general route servers that can be easily extended to implement more complex inter-domain policies, including those that make use of additional fields in BGP announcements. Recent work from Google [31] and Facebook [23] have demonstrated the value of richer interdomain policies, but are not widely available outside of these companies. In the absence of IXPs, widespread adoption of these mechanisms would have required router vendors to implement these changes and ASes to install new routers. However, recent work [17, 16] has shown that IXP route servers provide a convenient insertion point for implementing new interdomain mechanisms. As a result we believe that it is important to ensure that route server implementations can be easily extended to implement additional features where required.

2.2 Threat Model and Privacy

Threat Model. In this work we focus on ensuring that each IXP participant policies (*i.e.,* AS policies) are kept private from both the IXP and other participants. As is standard we assume that standard isolation mechanisms are used to ensure that each participant is limited to modifying its own policy and receiving BGP updates. We assume IXPs are *honest but curious*, *i.e.,* they may attempt to eavesdrop or read data stored in devices they control, but will not engage active attacks against participants. Finally, we assume that all participants and the IXP trust the hardware vendor who supplies the trusted execution environment, *i.e.,* Intel, or any of the manufacturers licensing ARM's TrustZone technology, and that the TEE

[2]Similar to restrictions imposed by Haven [5] and SCONE [3] we require some limits on how syscalls are used.

[3]These are commonly derived from the Internet Routing Registry (IRR) [19] and prevent participants from advertising malicious or erroneous routes.

Information	Publicly Visible	RS Visible
Route announcements	yes	yes
Possible routes (RIB)	no	configuration dependent
Best route	yes	yes
Filtering policy	no	yes
Ranking policy	no	configuration dependent
Auxiliary state (*e.g.* intradomain link property)	no	configuration dependent
Data plane behavior	yes	yes

Table 1: Summary of information in a BGP implementation: State is publicly visible when it can be deduced on the Internet, and RS visible when it can be deduced by the route server.

implementation is correct[4] Through the rest of this paper we report on a prototype implementation that makes use of Intel SGX as a TEE, and therefore in the rest of this paper we explain our system in terms of features provided by SGX.

Policy Privacy. Table 1 lists policy and intermediate state in current BGP implementations. Observe that some information including the route chosen by an AS and its dataplane behavior are always observable by observing how packets are routed, the set of announcements made by the AS, and through the use of tools such as looking glass servers [25]. However, when route servers are used ASes must reveal additional information including filtering and routing policies, and intermediate RIBs to the route server (and by extension the IXP). In this work we aim to provide ASes using route servers with the same level of privacy as is availabe when route servers are not used. Specifically we aim to ensure that IXPs do not have access to an AS's policies or intermediate state (*e.g.,* RIBs). Ensuring IXP scalability is the main challenge in providing such a guarantee since route server's rely on this information to both reduce the number of active BGP sessions and the number of messages sent as a part of each session.

3 Preserving Privacy

Next we provide an overview of two techniques that can be used to implement route servers that do not leak policy information: (a) SMPC [14] which has been used by prior efforts for both centralizing BGP computation [18] and for building privacy preserving route servers; and (b) TEE based approaches which we describe in this paper. We delay a detailed comparison between these techniques to §5.

3.1 Secure Multi-Party Computation

SMPC is a cryptographic technique [14, 29, 30, 15] that allows a user's computation to be split across multiple players (*e.g.,* different cloud providers) while ensuring that (a) the players can cooperatively perform the computation and (b) in the absence of collusion players discover neither inputs to the computation, nor its output. A variety of SMPC protocols have been described in literature, and the two protocols that have been applied to ensuring policy privacy in BGP deployments are BGW [6] and GMW [15]. Both protocols function similarly: first, the computation is converted into an arithmetic (BGW) or boolean (GMW) circuit that is provided to all players; next the user calls on a secret sharing scheme (Shamir's secret sharing scheme for BGW, XOR-based secret sharing for GMW) to produce a share for each player; next players use protocol specific algorithms to compute the result of applying the circuit to their share and send the results back to the user; finally the user combines the outputs received from each player to determine the result of the computation. Secret sharing mechanisms used in these techniques are reversible (by collecting all or a majority of shares) and as a result these techniques must assume no (or limited) collusion between players for privacy.

SIXPACK [9] is a route server that uses SMPC (specifically GMW) to provide policy privacy without requiring special hardware. The use of SMPC however imposes a few limitations: first, it increases the cost of running a route server by requiring that a portion of the computation be outsourced to one or more third-party providers to ensure there is no collusion; second, current SMPC implementations impose significant computational overheads and as a result systems that rely on them (including SIXPACK) need to minimize computation performed using SMPC protocols, making it harder to add new functionality thus hampering flexibility; finally, as we show in §5, despite efforts to minimize the use of SMPC protocols, SIXPACK is over an order-of-magnitude slower at processing BGP anouncements when compared to an insecure alternative, thus hampering IXP scalability.

3.2 TEE-Based Route Server

Trusted execution environments [27] (TEEs) such as Intel SGX and ARM TrustZone provide alternative mechanisms that can be used to implement route servers which preserve policy privacy. In current implementations applications making use of TEEs run within a trusted *enclave*. Each enclave implements two mechanisms (a) remote attestation which can be used by remote clients to verify the identity of the code executed within an enclave; and (b) memory protection mechanisms which allow applications to protect parts of their memory region so that external applications – including those running on another processor or device – cannot access the contents of this memory region. Both of these mechanisms are implemented using standard cryptographic primitives: remote attestation relies on signature schemes, while memory protection is implemented by encrypting data written to memory and decrypting data when it is fetched into cache. A variety of recent work including Haven [5], SCONE [3], etc. have designed systems that can be used to execute largely unmodified applications in trusted execution environments. Applications running within TEEs must be modified to address two challenges: first, system calls from an enclave are more expensive than from applications (requiring OCalls); second, existing trusted execution environments limit the amount of protected memory that is accessible (*i.e.,* resident in the enclave's page cache) by an enclave , and requires page swapping when this limit is exceeded. As a result applications running within TEEs must be designed so that they avoid syscalls when possible, and have small working sets that can fit within limited enclave memory so as to minimize swapping overheads.

In this paper we design a route server that uses Intel SGX to provide policy privacy. Our route server builds on the remote attestation, memory protection, and standard mechanisms (which similar to TLS rely on Diffie-Hellman key exchange) to establish secure channels between the route server and participants. Our implementation assumes that IXP participants trust Intel (or the hardware vendor), that SGX is correctly implemented, and that IXPs do not use side-channel attacks [28] to violate privacy guarantees. Side-channel attacks are a known concern for current TEE implementations, and we can adopt any current or future approaches to ad-

[4] Recent work [26] has shown that some current TEE implementations (specifically ARM TrustZone) are susceptible to leaking information, however this is due to a bug in the implementation and not a fundamental limitation of such environments. We do not address such bugs in this work, and assume that the TEE implementation is used is correct and does not leak information.

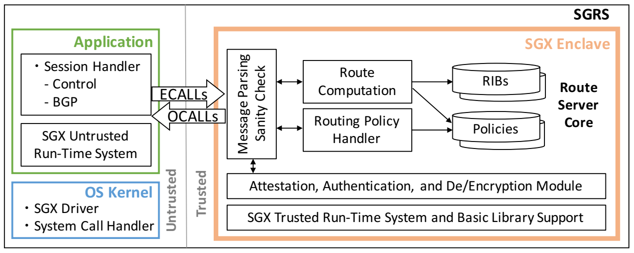

Figure 1: The architecture of SGRS

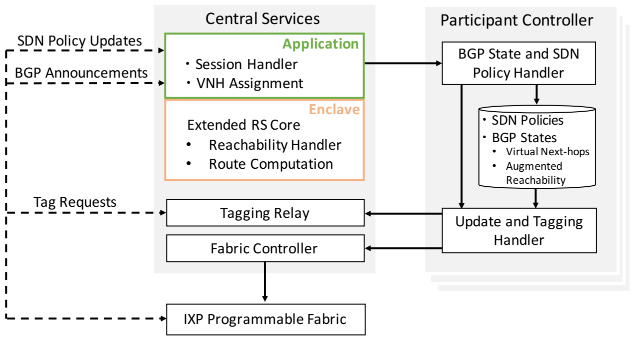

Figure 2: The architecture of SGDX

dressing these problems, including Oblivious RAM [20]. Finally, we also assume that the route server code is available to IXP participants who can examine it (or have a third party examine it) to ensure that the route server does not expose policies to the IXP.

Given these assumptions, our basic approach is to execute the route server within an enclave and using memory protection to protect participant policies and all RIBs. Next each participant is provided with the route server's address and uses remote attestation to ensure that the route server is executing an expected version of the code, thus establishing that it trusts the route server implementation. Next the participant establishes a secure channel with the route server. The client can then use this route server as before, by sending messages over the secure channel. In this model participants ensure policy privacy by first analyzing code (using formal methods, or through manual examination) to find a trusted route server implementation[5], then using remote attestation to ensure that the route server instance they use is executing this trusted implementation and running within an enclave, and finally relying on SGX's memory protection mechanism to ensure that the IXP cannot access policy or intermediate state used by the route server. Next, we describe our implementation in greater detail.

4 Implementation

We implemented the design described in §3.2 in a route server named SGRS. We also extended iSDX [16] to use SGRS, producing a system named SGDX that both implements a privacy preserving route server and a reconfigurable fabric. The main challenge in implementing SGRS and SGDX lay in designing the code so as to minimize the number of system calls (thereby avoiding OCalls and ECalls). Our approach for doing this is to partition the route server into trusted code that accesses private data and hence needs to run inside the enclave, and into untrusted code that we run outside of the enclave. While this is similar to the mechanism employed by SCONE [3], we place a larger fraction of our functionality outside the enclave.

In SGX protected memory regions are refered to as Enclave Pages and cached in the Enclave Page Cache (EPC). These pages are decrypted (or encrypted) by the CPU's memory encryption engine (MEE) when they are fetched (or evicted) from memory. The CPU cache is assumed to be inaccessible and hence is trusted by SGX. SGX relies on standard process isolation mechanisms to ensure pages cannot be accessed by external entities. In our implementation we place the entire route server in a single enclave and store the RIBs and policy state in Enclave Pages as described below.

Figure 1 shows the system overview of SGRS. SGRS is composed of untrusted application code, in the unprotected memory, to initialize enclave and handle incoming sessions and trusted enclave code, in the SGX-shielded memory, to perform cryptography actions and compute routes. The application and enclave code communicate with each other by ECalls and OCalls checked on parameter address and length. Enclave functionality is developed using SGX Linux SDK (version 1.9) [24], and we use ExaBGP [13] as the BGP session handler. At participant edge, each participant runs an SGRS proxy to accomplish the remote attestation procedure and set up a secure control session. BGP session processing is unmodified, still through participant border routers.

To extend SGRS to SGDX, we first identify the private information which should be protected. iSDX control plane integrates a route server and an SDN controller. The newly introduced information is SDN policies. SDN policies which are sent to iSDX will be enforced to the data plane and take effect immediately after control plane computation. Hence, SDN policies are publicly visible. However, SDN outbound policies should be augmented with BGP reachability (RIBs) to forward traffic along BGP-advertised paths. Therefore, except for private information and functionality in SGRS, the augmentation function which accesses the private RIBs should also be protected in the enclave.

Figure 2 shows the SGDX system architecture. iSDX partitions control plane across participants and remains the fabric controller and message relay as central services. We consolidate the route and reachability computation functions and private states from individual participant controllers into the central route server enclave the same way as SGRS implementation. We also move the virtual next-hop (VHN) assignment function to the route server untrusted application to put all routing related logic in central services, reducing communication between central services and participant controllers.

In general privacy-preserving mechanisms including SMPC and TEE require making a trade-off between performance and privacy. SGX performance overhead comes from three aspects: (i) enclave transition cost when the enclave interacts with untrusted software, (ii) enclave access cost when cache missing, $2\times-3\times$ slower than untrusted memory access (iii) page swapping cost when enclave access missing, with a significant performance penalty, around two orders of magnitude times slower than enclave access [3]. Route computation is a memory intensive task, while current version SGX has limited EPC size (128MB). We evaluate the effect of these limitations and analyze SGRS and SGDX performance overhead with large-scale datasets in next section.

5 Evaluation

We analyze and evaluate our implementation of SGRS and SGDX to illustrate (i) the flexibility on developing routing functionality using SGX, (ii) the performance overhead that SGX introduces, (iii) the performance comparison of SGRS and SMPC-based approach SIXPACK, and (iv) the scalability of SGDX compared to original

[5]In practice we expect this will be done by third-party vendors who audit code and guarantee its ability to preserve policy privacy.

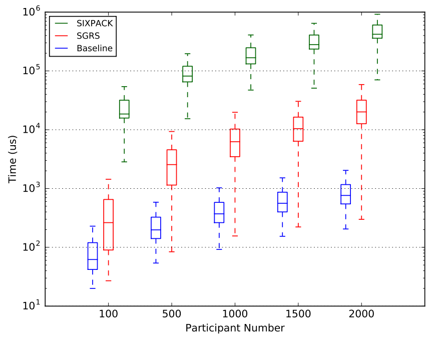

Figure 3: BGP update compute time of SIXPACK, SGRS, and Baseline on increasing participant number

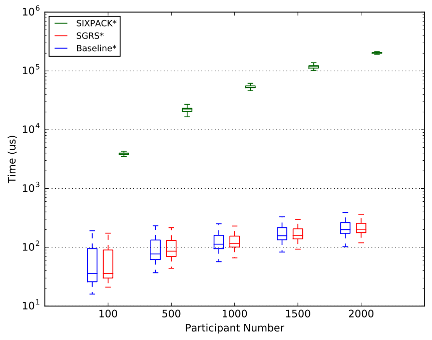

Figure 4: BGP update compute time of SIXPACK*, SGRS*, and Baseline* (each only does route exchange) on increasing participant number

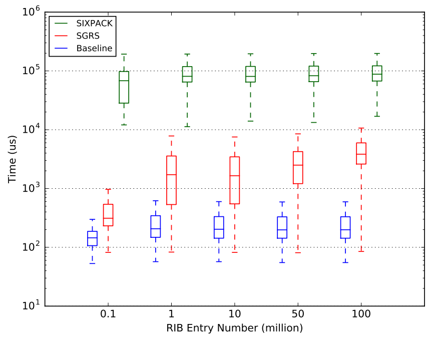

Figure 5: BGP update compute time of SIXPACK, SGRS, and Baseline on increasing route-server-maintained RIB size

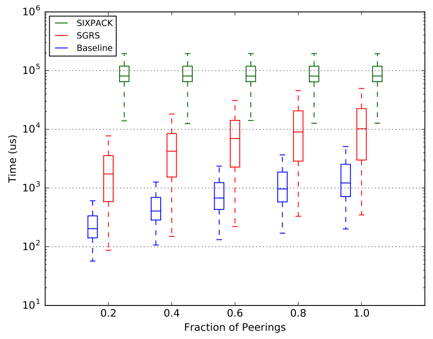

Figure 6: BGP update compute time of SIXPACK, SGRS, and Baseline on increasing peering density

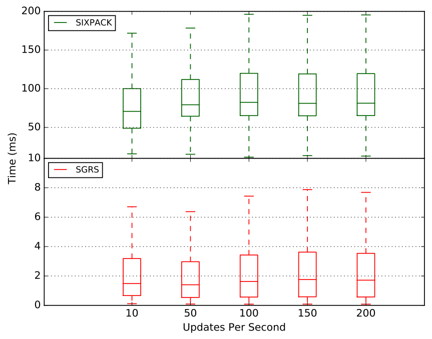

Figure 7: BGP update compute time of SIXPACK and SGRS on increasing sustained update rate

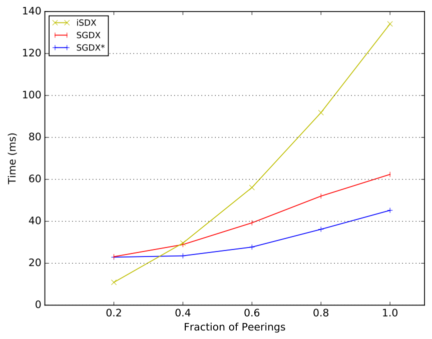

Figure 8: BGP update compute time of iSDX, SGDX, and SGDX* on increasing peering density

iSDX.

Experiment setup. We run our performance tests on a server with a 4-core SGX-enabled processor Intel Xeon CPU E3-1280 v5 3.7GHz and 64 GB of DRAM. We derive the data sets from RIPE RIS data [22] which is collected by route collectors, consists of public BGP updates and RIB dumps (historical BGP updates) but no routing policies, and has a maximum participant number 90. To generate large-scale data sets, we extend the participant number and assign each participant a uniform fraction of peerings. The participant's export/import policy for another participant is set to true (send/accept) if they are peering, and we use random local preferences as participant ranking policies. Then, we propagate the collector RIB to each participant according to the generated policies. Finally, we have a large number of participants with synthetic routing policies and RIBs, and the corresponding real world BGP update traces.

We load the routing policies to the system under test and replay the BGP update traces to evaluate the runtime BGP update compute time. To compare SGRS and SGDX with previous works in the same condition, we modify SIXPACK to use IRRdb based filtering policies instead of BGP Community based filtering policies and replace iSDX RIB backend MongoDB which has limited QPS (around 500-1000) with in-memory hash tables.

Flexibility analysis. We implement SGRS and SGDX trusted routing part in C using SGX SDK. During the implementation, after finishing the private state identification and code path redesign, most functions are written in identical way as general C programs. SGX related logic includes the common functions which can be reused in SGX applications such as $enclave\_init()$ and $remote\_attestation()$, the enclave definition language described ECall and OCall interfaces, and the application-specific ECall and OCall functions. The application-specific transition functions of SGRS consist of 207 lines of C code, while SGRS functions except for BGP session handler implemented by ExaBGP have 2241 lines. For SGDX, the transition functions consist of 277 lines of C code, while the central service functions extended from SGRS have 2807 lines. Both SGRS and SGDX have less than 10% code which belongs to the SGX-introduced development overhead.

SGRS performance. We evaluate the BGP update processing time of SGRS, SIXPACK, and Baseline (without protection) with four experiments. First, we test the scalability with the increase of participant number. We set the maximum participant number up to 2000 which exceeds the size of all current IXPs, generate 80 million peering routes, assign each participant a uniform 20% of peerings, and choose random subsets of participants with corresponding routes and peerings for smaller IXPs. Figure 3 shows the result. With the same settings as above, we truncate each route server to only do route exchange (no RIB storage and operations) and test the computation time. Figure 4 shows the result.

To understand how the RIB size affects the update computation, we then evaluate the processing time on varying the route number ranging from 0.1 million to 100 million. We use 500 participants (the same setting as iSDX) and each participant with a uniform 20% of peerings for all tests. Figure 5 shows the result. We also vary the uniform fraction of peerings of each participant from 20% to 100% with 500 participants and 10 million routes. Figure 6 shows the result. Besides, we vary the number of BGP updates per second and send the updates at constant rates to quantitatively analyze how the systems react to bursts. We use the same setting as the varying-RIB experiment and set the RIB size to 10 million. Figure 7 shows the result.

SGDX performance. We evaluate the BGP update processing time of SGDX, iSDX, and SGDX* (disabling SGX) using the same setting in iSDX[16]. We vary the fraction of peerings of each participant. Participants enforce SDN policies and filtering policies for their peerings. Figure 8 shows the result.

6 Conclusion

IXPs have become a crucial part of the internet infrastructure and are increasingly used by ASes for peering. Route servers have been essential to enabling IXP scalability, however they allow IXPs to learn AS-specific policies and intermediate data, and thus require ASes to trust IXPs. This additional trust requirement is an impediment to wider IXP adoption, and as a result it is important to develop privacy preserving route servers. Prior work [9] has addressed this need through the use of SMPC, however this comes at a large performance penalty – resulting in a performance degradation of up to $20\times$ – and also makes it harder to add new features to a route server. In this paper we proposed an alternate design based on the use of hardware trusted execution environment such as Intel SGX, and implemented a route server based on our design. Our privacy preserving route server imposes modest overheads of at most $2\times$, is easily extended to add support for additional policies, and is readily deployable on existing hardware. We have made our code available at https://github.com/huxh10/SGDX so as to allow further evaluation and extensions to our model.

7 References

[1] Amsterdam Internet Exchange (AMS-IX). https://ams-ix.net.

[2] Amsterdam Internet Exchange: Members. https://ams-ix.net/connected_parties retrieved 11/21/2017.

[3] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. M. Eyers, R. Kapitza, P. R. Pietzuch, and C. Fetzer. SCONE: Secure Linux Containers with Intel SGX. In *OSDI*, 2016.

[4] G. Asharov and Y. Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30:58–151, 2011.

[5] A. Baumann, M. Peinado, and G. C. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *OSDI*, 2014.

[6] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, 1988.

[7] The BIRD Internet Routing Daemon. http://bird.network.cz.

[8] I. Castro, J. C. Cardona, S. Gorinsky, and P. FranÃğois. Remote peering: More peering without internet flattening. In *CoNEXT*, 2014.

[9] M. Chiesa, D. Demmler, M. Canini, M. Schapira, and T. Schneider. SIXPACK: Securing Internet eXchange Points Against Curious onlooKers. In *CoNEXT*, 2017.

[10] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *IACR Cryptology ePrint Archive*, 2011.

[11] Connected Networks: DE-CIX. https://www.de-cix.net/en/locations/germany/frankfurt/connected-networks retrieved 11/21/2017.

[12] Deutscher Commercial Internet Exchange (DE-CIX). https://www.de-cix.net.

[13] ExaBGP overview. https://github.com/Exa-Networks/exabgp/wiki.

[14] O. Goldreich, B. Chor, S. Goldwasser, and L. A. Levin. Secure multi-party computation. 1998.

[15] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, 1987.

[16] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever. An Industrial-Scale Software Defined Internet Exchange Point. In *NSDI*, 2016.

[17] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. J. Clark, and E. Katz-Bassett. SDX: a software defined internet exchange. In *SIGCOMM*, 2014.

[18] 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 *HotNets*, 2012.

[19] International Routing Registry. http://www.irr.net/ retrieved 11/27/2017.

[20] C. S. Liu, A. Harris, M. Maas, M. W. Hicks, M. Tiwari, and E. Shi. GhostRider: A Hardware-Software System for Memory Trace Oblivious Computation. In *ASPLOS*, 2015.

[21] P. Richter, G. Smaragdakis, A. Feldmann, N. Chatzis, J. Böttger, and W. Willinger. Peering at Peerings: On the Role of IXP Route Servers. In *IMC*, 2014.

[22] RIS Raw Data. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data.

[23] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, Í. S. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *SIGCOMM*, 2017.

[24] Intel(R) Software Guard Extensions for Linux* OS. https://github.com/01org/linux-sgx.

[25] G. Siganos and M. Faloutsos. Analyzing bgp policies: Methodology and tool. In *INFOCOM*, 2004.

[26] A. Tang, S. Sethumadhavan, and S. J. Stolfo. Clkscrew: Exposing the perils of security-oblivious energy management. In *USENIX Security Symposium*, 2017.

[27] GlobalPlatform made simple guide: Trusted Execution Environment (TEE) Guide. https://www.globalplatform.org/mediaguidetee.asp.

[28] Y. Xu, W. Cui, and M. Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. *2015 IEEE Symposium on Security and Privacy*, pages 640–656, 2015.

[29] A. C.-C. Yao. Protocols for secure computations. *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

[30] A. C.-C. Yao. How to generate and exchange secrets. *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[31] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. J. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. M. B. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *SIGCOMM*, 2017.