Gilad Asharov, Daniel Demmler, Michael Schapira, Thomas Schneider, Gil Segev, Scott Shenker, and Michael Zohner

Privacy-Preserving Interdomain Routing at Internet Scale

Abstract: The Border Gateway Protocol (BGP) computes routes between the organizational networks that make up today's Internet. Unfortunately, BGP suffers from deficiencies, including slow convergence, security problems, a lack of innovation, and the leakage of sensitive information about domains' routing preferences. To overcome some of these problems, we revisit the idea of centralizing and using secure multi-party computation (MPC) for interdomain routing which was proposed by Gupta et al. (ACM HotNets'12). We implement two algorithms for interdomain routing with state-of-the-art MPC protocols. On an empirically derived dataset that approximates the topology of today's Internet (55 809 nodes), our protocols take as little as 6 s of topology-independent precomputation and only 3 s of online time. We show, moreover, that when our MPC approach is applied at country/region-level scale, runtimes can be as low as 0.17 s online time and 0.20 s pre-computation time. Our results motivate the MPC approach for interdomain routing and furthermore demonstrate that current MPC techniques are capable of efficiently tackling real-world problems at a large scale.

DOI 10.1515/popets-2017-0033
Received 2016-11-30; revised 2017-03-15; accepted 2017-03-16.

1 Introduction

Interdomain routing is the task of computing routes between the administrative domains, called "Autonomous Systems" (ASes), which make up the Internet. While there is a variety of *intra*domain routing designs (e.g., RIP, OSPF, IS-IS) to compute routes *within* an organizational network, there is only one *inter*domain routing algorithm: the Border Gateway Protocol (BGP). BGP stitches together the many (over 55 000) ASes that the Internet is composed of and can thus be regarded as the glue that holds together today's Internet. BGP was specifically designed to meet the particular demands of routing between Internet domains, allowing each AS the freedom to privately and freely implement arbitrary *routing policies*, i.e., the expressiveness to both (i) select a route from the routes learned from its neighboring ASes according to its own local business and operational considerations and (ii) decide whether to advertise or not advertise this route to each of its neighboring ASes. Importantly, ASes' routing policies can leak sensitive information about their business relationships with other ASes and are therefore often kept private.

BGP achieves the dual goals of policy freedom and policy privacy through an iterative, distributed route computation. At each stage of the computation, a domain (AS) chooses which routes to use (among the routes being advertised to it by its neighbors), and then chooses to which neighboring ASes the resulting routes should be advertised. This process is repeated until convergence, thus allowing each domain to make its own policy-induced choices, without needing to explicitly reveal these choices to other domains. However, as pointed out in [1, 2] (and the references within), while BGP computation does not force domains to explicitly reveal policies, much information about routing policies can be inferred by passively observing routing choices.

While BGP has served the Internet admirably, it has many well-known drawbacks ranging from slow convergence to inability to deal with planned outages. Thus, we should explore alternative methods for interdomain routing. As suggested in [3], the use of secure multi-party computation (MPC) offers an intriguing possibility: executing the route computation centrally (among a few mutually distrustful parties) while using MPC to retain policy privacy. However, while the MPC technology provides ASes with provable privacy guarantees, scaling this approach to current Internet infrastructure sizes with over 55 000 ASes is a significant challenge. Specifically, the computation in [3] already requires 0.13 s for a toy example of only 19 ASes and would hence require

Gilad Asharov: Cornell Tech, E-mail: asharov@cornell.edu
Daniel Demmler: TU Darmstadt, E-mail: daniel.demmler@crisp-da.de
Michael Schapira: Hebrew University of Jerusalem, E-mail: schapiram@cs.huji.ac.il
Thomas Schneider: TU Darmstadt, E-mail: thomas.schneider@crisp-da.de
Gil Segev: Hebrew University of Jerusalem, E-mail: segev@cs.huji.ac.il
Scott Shenker: University of California, Berkley, E-mail: shenker@icsi.berkley.edu
Michael Zohner: TU Darmstadt, E-mail: michael.zohner@crisp-da.de

several hundreds of seconds for today's Internet, even when assuming a low number of neighbors per AS. Our paper is devoted to the cryptographic paradigms, protocols, optimizations, and concrete tools necessary to compute interdomain routes at Internet scale in a privacy-preserving way.

1.1 Centralizing BGP

The vision of (logically) centralizing interdomain routing can be regarded as the interdomain-level analogue of the software-defined networking (SDN) approach to routing within an organization (i.e., intradomain routing), which is revolutionizing computer networking. SDN intradomain routing decouples route-computation from the forwarding of data packets. Specifically, route-computation is delegated to a software-implemented, centralized "controller", which installs the resulting forwarding rules in the decentralized switching hardware. Thus, with SDN, altering the routing scheme (e.g., to be more efficient, more resilient to failures, provide quality-of-service assurances, etc.) only involves changing the controller software, as opposed to replacing the hardware and software of multiple proprietary network devices (routers, switches, etc.). However, reaping these benefits in the context of interdomain routing involves overcoming two grand challenges: (1) preserving *privacy* of the business-sensitive routing policies of the many independent organizations that take part in the computation, and (2) computation at very large *scale* (outputting a routing configuration spanning tens of thousands of organizations).

To overcome these challenges we combine knowledge of the two research areas of networking and secure computation. We use a state-of-the-art secure two-party computation framework and outsource the route computation to two computational parties CP_1 and CP_2, who are managed by two different operators, which we assume do not collude. To protect the privacy of their business relations, the ASes secret share their routing preferences with these two computational parties, such that no party gets any information about the routing preferences between the ASes. The computational parties run our secure interdomain routing computation protocols to determine the routes for each AS in the network. The computation results are sent back to each AS, which can then reconstruct the plain text output of the algorithm. More specifically, the CPs only send a small message to the ASes which contains their next hop for a specific destination, while the communication- and round-intensive MPC protocol is run between the CPs. An example setting with 6 ASes is depicted in Fig. 1.

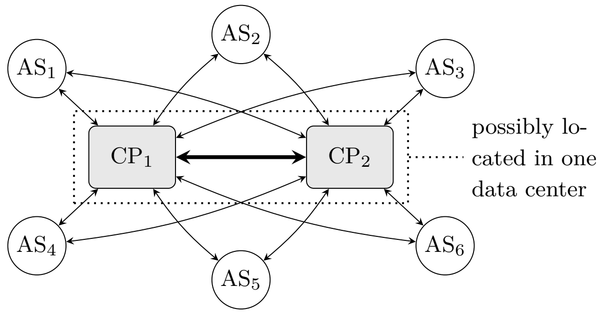

Fig. 1. Example setting with 6 ASes that secret share their inputs with 2 computational parties CP_1 and CP_2. Thin arrows correspond to 1 round of communication with small messages, while the **bold** arrow symbolizes the execution of a secure computation protocol with many rounds and high bandwidth.

1.2 Outline and Our Contributions

After motivating a centralized approach and the use of MPC for interdomain routing in §1.3, we provide an overview of related work for BGP and MPC in §1.4 and §2. Finally, we present our contributions:

Interdomain routing algorithms for MPC (§3). Interdomain routing is a long-standing research topic for which several efficient algorithms exist. When transferred to the MPC domain, however, the complexity of the algorithms changes drastically, since data-dependent optimizations of the algorithms are not possible in MPC. In this paper, we select two interdomain routing algorithms, which provide different capabilities for setting routing policies: one approach is based on neighbor *relations* and the other approach is based on neighbor *preferences*. The neighbor relations-based routing algorithm is due to [4] and uses business relations between ASes to perform routing decisions (§3.1). The neighbor preferences-based routing algorithm was used in the MPC protocol of [3] and allows ASes to rank neighbors based on their preferences and give export policies which specify whether a route to a neighbor i should be disclosed to a neighbor j (§3.2). We implement these algorithms in a centralized setting, which allows for consistency checks of the ASes' inputs and thereby prevents malicious ASes from providing inconsistent input information (cf. §6.2).

Construction and optimization of Boolean circuits for BGP (§4). We convert the neighbor relation BGP algorithm of [4] and the neighbor preference

BGP algorithm of [3] into a distributed secure computation protocol between two parties to provide privacy. We explain challenges facing the implementations of these functionalities as a Boolean circuit, optimized for both low multiplicative depth (the number of AND gates on the critical path of the circuit) and low multiplicative size (the total number of AND gates) for evaluation with the GMW protocol [5] implemented in the ABY framework [6]. We provide details on several building blocks, how we optimize them, and the techniques we use to achieve high performance so as to be able to process real-world data.

Benchmarks and evaluation (§5). We benchmark both implementations on a recent empirically derived BGP dataset with more than 50 000 ASes with maximal degree 5 936 and almost 240 000 connections between them. We propose to exclude stub nodes from the computation as further optimization and evaluate complex Boolean circuits with several million AND gates. The neighbor relation algorithm requires about 6 s of topology-independent precomputation time and an online time of about 3 s on two midrange cloud instances, that are comparable to off-the-shelf desktop computers. The neighbor preference algorithm takes about 13 s of topology-independent precomputation time and 10 s online time. We argue that while the online runtimes alone are not sufficient to provide adequate response to network failures, it allows the precomputation of routes for many failure scenarios, which would enable almost instantaneous failure recovery.

Deployment and future directions (§6). Our aim is to demonstrate the practical feasibility of using MPC for interdomain routing. However, we view our work only as a first stepping stone that should serve as a basis for further research. We first explain our assumptions about the network in §6.1. To spawn interest, we list promising directions for further enhancing robustness and measures against misconfigurations in §6.2 and discuss security against stronger adversaries in §6.3. Of course, transitioning to MPC of interdomain routes is an extremely challenging undertaking that involves cooperation of tens of thousands of independent financial and political entities, alongside significant deployment and operational challenges. We argue, however, that our approach can also yield significant benefits (e.g., in terms of privacy, security, and ability to innovate) when applied at a country/region-level scale. In addition, we show that even when applying our approach to high-density networks in fairly compact areas, namely, the German interdomain network, runtimes decrease to 0.20 s pre-computation time and 0.17 s online time (cf. Fig. 4). We discuss the possibility of deployment and related issues of our approach in §6.4.

1.3 Why MPC for Interdomain Routing?

In the following section, we list some of the benefits that centralizing BGP via MPC can offer.

Better convergence and resilience to disruption. As opposed to BGP's inherently decentralized and distributed computation model, which involves communication between tens of thousands of ASes, in our scheme interdomain routes are computed by only two computational parties. BGP can take seconds to minutes to converge [7–9]. In the interim period, BGP's path exploration can have adverse implications for performance. Indeed, a huge fraction of VoIP (e.g., Skype's) performance issues are the result of bad BGP convergence behavior [10]. Worse yet, BGP's long path exploration can even lead to intermittent connectivity losses. By centralizing computation and thus avoiding the long distributed (and asynchronous) path-exploration process, convergence time is reduced significantly. We demonstrate that our approach, despite harnessing MPC machinery, is much faster to compute global routing configurations than today's decentralized convergence process and, consequently, faster to recover from network failures and to adapt to changes in ASes' routing policies.

Less congestion. BGP permits ASes great expressiveness in specifying local routing policies at the potential cost of persistent global routing instability. However, as shown in [11], under natural economic assumptions (the so called "Gao-Rexford Conditions") BGP convergence to a stable routing configuration is guaranteed. Unfortunately, even under these conditions, convergence might take exponential time (in the number of ASes) due to the exchange of an exponential number of messages between ASes [12]. Our scheme, in contrast, guarantees fast convergence to the desired routing outcomes as our communication overhead is polynomial (linear time in the size of the network).

Enhanced privacy. Many ASes regard their routing policies as private and do not reveal them voluntarily, as routing policies are strongly correlated to business relationships with neighboring ASes. BGP seemingly offers policy freedom and policy privacy, as each AS is free to choose which routes to use and which routes to advertise to others, without having to explicitly reveal its routing policies. However, BGP's privacy guarantees are limited, and are even fictitious. Monitoring selected BGP routes, in particular when done from multiple vantage points, can reveal much information about ASes' routing policies, e.g., their local preferences over BGP routes (see, e.g. [MK06, GZ11] and references within). In addition, AS business relationships can be reconstructed from publicly available datasets [CAI]. We refer the reader to [2] for an illustration of how monitoring the BGP convergence process can yield much information about ASes' routing policies.

Using MPC for interdomain routing can remedy this situation by provably providing strong privacy guarantees that cannot be achieved under today's routing on the Internet. Our scheme guarantees that no information about routing policies and inter-AS business relationships, other than that implied by the routing outcome, is leaked. In fact, each node (AS) learns *only its "next hop" node in the final routing outcome* with respect to a destination, and not even the full route. We also hide the entire convergence process, which potentially leaks information. As a side note, even with multiple vantage points, inferring routing policies is no easy task. While our scheme would not completely remove this kind of leakage, it would definitely decrease it compared to routing using BGP, where the ASes broadcast their full routing table. Furthermore, it is unclear whether all information about the policy preferences can be gained using only the next hop as information.

Enhanced security. BGP's computation model, which is distributed across all ASes, enables ASes to launch devastating attacks against the protocol, which can result in Internet outages [13]. By outsourcing BGP computation to a few parties, attacks on BGP that manipulate its decentralized computation, e.g., propagating bogus AS-level routes to neighboring nodes, are eliminated. We point out that centralizing interdomain routing is also compatible with the ongoing efforts to deploying the Resource Public Key Infrastructure (RPKI) — a centralized certification infrastructure for issuing cryptographic public keys to ASes and for mapping IP addresses to owner ASes, thus preventing ASes from successfully announcing IP prefixes that do not belong to them ("prefix hijacking"). By outsourcing BGP computation, verifying routing information through the RPKI can be executed efficiently in a centralized manner.

Freedom to adapt and innovate. The computational parties can easily update to new and more advanced protocols, thus offering more complex functionality, such as new security solutions, multiple paths per destination prefix, multicast routing, fast failover in response to network failures, etc.

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

1.4 [3] and other Related Works

The innovative idea of using MPC for BGP was first proposed in [3]. The aim of that paper was to illustrate benefits and challenges of this approach, and to explore generic cryptographic schemes towards its realization. We take an important step forward, providing a more concrete cryptographic approach that is tailored to interdomain routing, and thus leads to significant improvements, and show that we achieve reasonable performance in a real topology of today's Internet.

Even though [3] outsources the route computation to few mutually distrustful parties, it does not utilize this approach to the fullest extent possible. In terms of functionality, the internal protocol of [3] that the clusters execute is very close to the BGP protocol. That is, the computational parties get the secret shares of routing preferences of all ASes, and then simulate an execution of the BGP protocol on "virtual" ASes with these shared preferences: The clusters run several iterations until convergence, where in each iteration, each virtual AS (i) selects a route from the routes learned from neighboring ASes and (ii) decides whether or not to advertise this route to each of its neighbors based on a set of export policies. These decisions, however, incur a high overhead in cryptographic computations since, in order to hide the policy that is applied, all policies have to be applied once per iteration. In order to securely evaluate the routing algorithm, [3] uses the MPC protocol of [14], which provides passive security in the case of a honest majority, i.e., if $t < n/2$ of the n parties have been corrupted.

In this paper, we follow up on the work of [3] and suggest the use of a second established interdomain routing algorithm that avoids these computation-heavy policies using a simpler routing strategy based on business relations [4], given in §3.1. We compare the performance of this algorithm to the preference-based algorithm that was used in [3] and which we outline in §3.2. In addition, we use the secure computation protocol of Goldreich-Micali-Wigderson [5] for secure evaluation of the algorithms, since it provides security in case of no honest majority. Also, [15] recently showed that the protocol of [5] scales better to a larger number of parties than the protocol of [14].

Other related works proposed privacy-preserving graph algorithms, but did not consider the more complex BGP algorithm: [16] proposes STRIP, a protocol for vector-based routing that computes the shortest path based on the Bellman-Ford algorithm. In their protocol the routers forward encrypted messages along the possible paths that accumulate the costs along the path using additively homomorphic encryption. This approach requires many messages until it converges and the routers need to implement costly public-key encryption whereas in our solution all cryptographic operations are outsourced to the two mutually distrustful computational parties. [17] provides privacy-preserving graph algorithms with security against passive adversaries for all pairs shortest distance and single source shortest distance. [18] provides data-oblivious graph algorithms for secure computation, such as breadth-first search, single-source single-destination shortest path, minimum spanning tree, and maximum flow, the asymptotic complexities of which are close to optimal for dense graphs. [19] introduces an outsourced secure computation scheme that is secure against active adversaries and uses it to compute Dijkstra's shortest path algorithm. [20] introduces a framework that compiles high-level descriptions into programs that combine secure computation and ORAM and gives speed-ups for Dijkstra's shortest path algorithm. However, the complexities of these algorithms that hide the topology of the graph are too high to scale to the size of the Internet consisting of thousands of nodes.

2 Preliminaries

We provide preliminaries on secure multi-party computation (§2.1), modeling the BGP protocol (§2.2), and detail the input data for our protocols (§2.3) next.

2.1 Secure Multi-Party Computation

Secure multi-party computation was introduced in the 1980s [5, 21]. These works show that multiple computing devices can carry out a joint computation of *any* function on their respective inputs, without revealing any information about the inputs (except for what is logically learned from the output).

More concretely, consider n parties $P_1, \ldots, P_n$ that hold private inputs $x_1, \ldots, x_n$ and wish to compute some arbitrary function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where the output of P_i is y_i. MPC enables the parties to compute the function using an interactive protocol, where each party P_i learns exactly y_i, and nothing else. Natural applications of MPC include voting, digital auctions, survey computations, set operations, and many more.

The security of the protocol is preserved even in the presence of some adversarial entity that corrupts some of the participating parties, combines their transcripts and coordinates their behaviors. Usually, there are two types of adversaries that are considered. A semi-honest adversary (also known as "honest-but-curios" or "passive"), follows the protocol specification but may attempt to learn secret information about the private information of the honest parties from the messages it receives. A malicious adversary (also known as "active") may, in addition, deviate from the protocol specification and follow any arbitrary behavior. In our setting (as well as in [3]), we assume that the computing parties are semi-honest. Our basic variant of the protocols assumes that the ASes are semi-honest as well (and may collude with some of the computing parties). In the more involved variant of our protocols (unlike [3]), we tolerate even malicious behavior of the ASes (cf. §6.2).

In this work, we use MPC in an *outsourcing scenario*, where many ASes secret-share their private inputs to two computational parties, who run the secure computation on these inputs. The outputs are then sent back to the ASes, who reconstruct the plaintext output.

Despite the immense potential of MPC, it is a great challenge to implement it in practice. The aforementioned constructions of MPC are purely theoretical, and protocols for secure computation can require many rounds of interaction and the transformation of massive data between the computing parties. A very productive line of research, e.g., [6, 20, 22–25], has been devoted to positioning MPC as a practical tool and off-the-shelf solution for a wide variety of problems, and to minimize the complexity of the current schemes. Using these recent breakthroughs, the benefits of MPC can be utilized in some real-life applications [26, 27]. Still, its deployment is somewhat insufficient and far beneath its true potential. Our work is a new real-life large-scale application of MPC, which is interesting in its own right.

An important building block for secure computation is oblivious transfer (OT), where a sender inputs two ℓ-bit messages (m_0, m_1) and a receiver inputs a selection bit $s \in \{0, 1\}$ and obliviously receives one message m_s. OT guarantees that the sender does not learn the receiver's choice s, while the receiver only learns m_s and nothing about m_{1-s}. OT can be computed efficiently using OT extension [28, 29].

Boolean sharing. The MPC techniques we use to implement our protocols rely on Boolean sharing, where a value is XOR secret-shared and processed using a circuit that operates on bits. The parties emulate computation of the circuit gate-by-gate, where in each gate the parties compute shares of the gate's output wire using the shares of the inputs. The computation of the outputs of an AND gate requires interaction. Consequently, the round complexity of the protocol depends on the depth of the circuit, resulting in high latency for circuits with high depth. We use the protocol by Goldreich-Micali-Wigderson (GMW) [5], which falls into this category. The main advantage of the GMW protocol is that it allows to precompute *all* (symmetric) cryptographic operations in a setup phase that is independent of both the function that is being evaluated and the inputs to the function. Each AND gate needs as precomputation two Oblivious Transfers (OTs) [28, 29] on random inputs, which are used to precompute a multiplication triple that consists of random bits $a_0, a_1, b_0, b_1, c_0, c_1$ for which $c_0 \oplus c_1 = (a_0 \oplus a_1) \wedge (b_0 \oplus b_1)$ holds, cf. [29]. We heavily utilize this precomputation. In our setting, the setup phase is independent of the topology of ASes and their routing preferences and hence can be easily evaluated by multiple machines in parallel. The remainder of the protocol is an online phase that consists solely of bit operations. Moreover, the GMW protocol allows to efficiently evaluate the same sub-circuit in parallel, similar to Single Instruction Multiple Data (SIMD) instructions in a CPU. Finally, the GMW protocol also allows for highly efficient instantiation of multiplexers using *vector ANDs* [6], which reduce the cost for evaluating a ℓ-bit multiplexer to the cost of evaluating a single AND gate and, in our experiments, reduce the number of required OTs by a factor between 3 and 45. We describe how this optimization is applied to our use case in §4.2.1. More detailed, an ℓ-bit vector AND (or multiplexer) is evaluated using a vector multiplication triple, which consists of two random bits $a_0, a_1 \in \{0, 1\}$ and four random ℓ-bit strings $b_0, b_1, c_0, c_1 \in \{0, 1\}^\ell$ with $c_0[i] \oplus c_1[i] = (a_0 \oplus a_1) \wedge (b_0[i] \oplus b_1[i])$ for $1 \leq i \leq \ell$, where $[i]$ denotes the i-th bit of a string. Similar to a regular multiplication triple, a vector multiplication triple can be generated using two OTs on random inputs [6].

2.2 Modeling BGP

We now give an overview on important aspects of modeling BGP, as discussed in [11, 30, 31] (and references therein). Throughout this work the terms AS, domain, vertex or node are used interchangeably.

2.2.1 The AS-Level Graph

The AS-level topology of the Internet is modeled as a network graph $G = (V, E)$ where vertices represent ASes and edges represent connections between them. Each edge is annotated with one of two business relationships: *customer-provider*, or *peering*. A *customer-provider* edge is directed from customer to provider; the customer pays its provider for transmitting traffic to/from the customer. A *peering* edge represents two ASes that agree to transit traffic between their customers at no cost. We assume that these relationships are symmetric, i.e., if AS a is a peer of AS b, then b is also a peer of a and if AS c is a customer of AS d, then d is a provider of c. ASes with customers are Internet Service Providers (ISPs). We call an AS with no customers a "*stub* AS".

2.2.2 Routing Policies

ASes' routing polices reflect their local business and performance considerations. Consequently, routing policies are considered sensitive information as revelation of an AS's routing policy can potentially leak information about its business relationships with others to its competitors (or other relevant information). We use the standard model of routing policies from [11, 30]. Each AS a computes routes to a given destination AS dest based on a *ranking* of simple (loop-free) routes between itself and the destination, and an *export policy*, which specifies, for any such route, the set of neighbors to which that route should be announced. We next present a specific model of routing policies that is often used to simulate BGP routing (see, e.g., [4, 31, 32]).

Ranking. AS a selects a route to dest from the set of paths it learns from its neighbors ASes according to the following ranking of routes:

- **Local preference.** Prefer outgoing routes where the "next hop" (first) AS is a customer over outgoing routes where the next hop is a peer over routes where the next hop is a provider. This captures the intuition that an AS is incentivized to select revenue-generating routes through customers over free routes through peers over costly routes through providers. Optionally, an AS can have preferences within each group of neighbors, i.e., it can prefer a certain provider over another one.
- **Shortest paths.** Break ties between multiple routes with the highest local preference (if exist) in favor of shorter routes (in terms of number of ASes on them). Intuitively, this implies that an AS

breaks ties between routes that are equally good from a business perspective, in favor of routes that offer better performance.

– **Arbitrary tie breaking.** Break ties between multiple remaining routes (if exist) arbitrarily.

Export policies. The following simple export policy captures the idea that an AS is willing to transit traffic between two other ASes if and only if one of these ASes is a paying customer: AS b announces a path via AS c to AS a iff at least one of a and c is a customer of b.

2.2.3 BGP Convergence

BGP computes routes to each destination independently and so, henceforth, we consider route computation with respect to a single destination AS dest. In BGP, each AS repeatedly uses its ranking function to select a single route from the set of routes it learns from its neighbors, and then announces this route to the set of neighbors dictated by its export policy. This goes on until BGP computation *converges* to a stable routing outcome where no AS wishes to change its route. Observe that as an AS can only select a single route offered to it by a neighbor. The set of selected routes upon convergence must form a tree rooted in the destination dest, referred to as the *routing tree* to AS dest. Under the routing policies specified in §2.2.2, BGP is guaranteed to converge to a unique stable routing tree [11] given the arbitrary tie-breaking strategy.

2.3 BGP Input Data

It is our goal to simulate the algorithms under realistic conditions and show its practicality on real-world data. To this end, we use the network topology and AS business relationships provided in the CAIDA dataset from November 2016 [33]. This dataset is empirically generated and provides us with both a realistic network topology, which we can use as public input, as well as inferred business relationships between domains, which we use to simulate the private inputs of the ASes. We furthermore compare properties of the most recent dataset with the available historic data for every month starting from 1998 and depict this information in Fig. 6 in Appendix A. We evaluate our protocols on datasets from the past 10 years for full topologies and recent subgraphs thereof to show how our implementations scale and provide detailed results in §5.

A possible way of deploying our solutions could be with the help of a Regional Internet Registry (RIR) or on smaller, regional scale. Starting from the original CAIDA topology, we created subgraphs using the GeoLite database [34] for each of the 5 RIRs and Germany as an example of a regional topology (cf. §6.4).

3 Centralized BGP Algorithms

We consider two centralized algorithms for computing interdomain routes: an algorithm based on business relations (§3.1) and an algorithm that ranks neighbors based on preferences (§3.2). We first outline the pseudocode for these algorithms, which can be considered as the “code of the trusted party” in terms of secure computation and then show how to reduce the complexity of the route computation by removing stub-nodes (§3.3).

3.1 Centralized Algorithm with Neighbor Relations

We present the algorithm from [4] for computing the BGP routing tree for the routing policies described in §2.2.2. The algorithm gets as input the AS-topology $G = (V, E)$, where each outgoing edge $(u, v) \in E$ is associated with one of three labels: customer (v is a customer of u), peer (u and v are peers) or provider (v is a provider of u). The algorithm also receives as input the destination AS dest $\in V$. The output of the algorithm is, for each AS, the next hop on the routing tree to destination dest. As shown in [4], the induced routing tree generated by this algorithm agrees with the BGP outcome for the routing policies described in §2.2.2.

The algorithm computes for each AS its next hop on the routing tree using the following three-stage breadth-first search (BFS) on the AS graph:

1. **Customer routes.** A partial routing tree is constructed by performing a BFS “upwards” from root node dest using only customer edges.
2. **Peer paths.** Next, single peering edges connect new ASes to the ASes already added to the partial routing tree from the first stage of the algorithm.
3. **Provider paths.** The computed partial routing tree is traversed with a BFS, and new ASes are iteratively added to the tree using provider edges.

– **Public inputs:** $(V, \mathsf{dest}, d_{\mathsf{depth}})$, where $V = \{v_1, \ldots, v_n\}$ represents the set of vertices (ASes), $\mathsf{dest} \in V$ is the destination node, and d_{depth} is a bound on the depth of the customer-provider hierarchy, i.e., the longest route in the AS-graph in which each edge is from customer to provider. We use 10 as a very conservative upper bound on this depth. The topology of the AS-graph is assumed to be public knowledge. That is, for every $v \in V$ the list of its neighbors $\mathsf{Adj}[v] \subseteq V$ is public. We discuss hiding the topology in §6.5.
– **Private inputs:** Every AS $v \in V$ inputs a private list type_v, where for every $u \in \mathsf{Adj}[v]$, $\mathsf{type}_v[u] \in \{\mathsf{customer}, \mathsf{peer}, \mathsf{provider}\}$.
– **Outputs:** Upon completion of the algorithm, every AS $v \in V$ obtains its next hop in the routing tree $\mathsf{next}[v]$.

```
 1: Initialize a vector next of size $|V|$, that stores the next hop
    in the routing tree to node dest, where for every $v \in V$
    set $\mathsf{next}[v] = \mathtt{DUMMY}$, where $\mathtt{DUMMY} \notin V$ is an unconnected
    node. Set $\mathsf{next}[\mathsf{dest}] = \mathsf{dest}$.
 2: Initialize a Boolean vector fin of size $|V|$, that indicates
    if a route to dest was found, and set all the elements to
    false. Set $\mathsf{fin}[\mathsf{dest}] = \mathsf{true}$.
 3: Initialize a distance vector dist of size $|V|$ that holds the
    number of hops to dest. Set all elements to $\infty$, except for
    $\mathsf{dist}[\mathsf{dest}] = 0$.
 BFS of customers:
 4: for $d_{\mathsf{depth}}$ iterations do:
 5:   for all $v \in V$ do:                          ▷ in parallel
 6:     for all $u \in \mathsf{Adj}[v]$ do:         ▷ for all neighbors of $v$
 7:       if $\mathsf{fin}[u] = \mathsf{true}$ and $\mathsf{fin}[v] = \mathsf{false}$ and
          $\mathsf{type}_v[u] = \mathsf{customer}$ then
 8:         $\mathsf{next}[v] \longleftarrow u$
 9:         $\mathsf{dist}[v] \longleftarrow \mathsf{dist}[u] + 1$
10:         $\mathsf{fin}[v] \longleftarrow \mathsf{true}$
 BFS of peers:
11: for all $v \in V$ do:                            ▷ in parallel
12:   for all $u \in \mathsf{Adj}[v]$ do:           ▷ for all neighbors of $v$
13:     if $\mathsf{fin}[u] = \mathsf{true}$ and $\mathsf{fin}[v] = \mathsf{false}$ and
        $\mathsf{type}_v[u] = \mathsf{peer}$ and $\mathsf{dist}[v] > \mathsf{dist}[u] + 1$ then
14:       $\mathsf{next}[v] \longleftarrow u$
15:       $\mathsf{dist}[v] \longleftarrow \mathsf{dist}[u] + 1$
16:       $\mathsf{fin}[v] \longleftarrow \mathsf{true}$
 BFS of providers:
17: for $d_{\mathsf{depth}}$ iterations do:
18:   for all $v \in V$ do:                          ▷ in parallel
19:     for all $u \in \mathsf{Adj}[v]$ do:         ▷ for all neighbors of $v$
20:       if $\mathsf{fin}[u] = \mathsf{true}$ and $\mathsf{fin}[v] = \mathsf{false}$ and
          $\mathsf{type}_v[u] = \mathsf{provider}$ and $\mathsf{dist}[v] > \mathsf{dist}[u] + 1$ then
21:         $\mathsf{next}[v] \longleftarrow u$
22:         $\mathsf{dist}[v] \longleftarrow \mathsf{dist}[u] + 1$
23:         $\mathsf{fin}[v] \longleftarrow \mathsf{true}$
24: return next
```

Algorithm 1. Neighbor Relation Routing [4]

We proceed with a detailed pseudo-code of the above algorithm. Implementing this algorithm involved several decisions that will mitigate the conversion to a secure protocol, and careful selection of data-structures. E.g., variables with Boolean values are preferred when possible, since these simplify the conversion to the Boolean circuit. Also, sometimes further optimizations of the algorithm (like breaking a loop according to some condition), are avoided as to not reveal information about the internal state. Note that all nodes are processed in parallel, i.e., the state is read once for all nodes and updated at the end of each iteration.

We distinguish between public and private algorithm inputs. We assume public inputs are global knowledge and do not reveal sensitive information. Private inputs describe the privacy-sensitive input of each AS.

A formal description of the algorithm is given in Alg. 1. The state of the algorithm consists of three vectors, each of size $|V|$: next, fin and dist. The vector next stores nodes, where for every node $v \in V$, $\mathsf{next}[v]$ stores the next hop in the routing tree to the node dest. The vector fin is a Boolean vector, where $\mathsf{fin}[v]$ stores whether the route from v to dest is already determined. The vector dist is a vector of integers, where $\mathsf{dist}[v]$ stores the number of hops in the current route between v and dest (this helps us to break ties between multiple routes with the highest local preference, if exist, in favor of shorter routes). It is easy to see that this pseudo-code is a concrete implementation of the algorithm presented in [4], and thus we conclude that the routes computed by this algorithm agree with the outcome of BGP (where the preferences of the ASes are according to §2.2.2).

3.2 Centralized Algorithm with Neighbor Preferences

The algorithm in §3.1 can be extended such that it allows the ASes to specify preferences for each neighbor route and freely choose an individual export policy. In [3], such an algorithm was proposed, that behaves similar to the one in [4], but is computationally more complex due to the added degree of freedom.

We can emulate the behavior of Alg. 1 by grouping each neighbor relation to a certain range of preferences: we ensure that customers have a higher preference than all other nodes, and that providers have lower preference than others. The advantage of this algorithm is, that within each neighbor relation we can have a preferred node, e.g. a favorite provider. In addition, this algorithm allows a node to freely specify his export pol-

```
– Public inputs: Same as in Alg. 1.
– Private inputs: Every AS $v \in V$ inputs a private list of preferences $\mathsf{pref}_v$, where for every $u \in \mathsf{Adj}[v]$, $\mathsf{pref}_v[u]$ corresponds to the preference for $u$, and a private bit-matrix $\mathsf{pub}_v$ of size $|\mathsf{Adj}[v]+1| \times |\mathsf{Adj}[v]|$ that specifies the export policy, i.e., if a route to a neighbor is published to other neighbors.
– Outputs: Same as in Alg. 1.

1: Initialize a vector next of size $|V|$, that stores the next hop in the routing tree to node dest. For every $v \in V$ set $\mathsf{next}[v] = \mathtt{DUMMY}$, where $\mathtt{DUMMY} \notin V$ is an unconnected node. Set $\mathsf{next}[\mathsf{dest}] = \mathsf{dest}$.
2: Initialize a Boolean vector fin of size $|V|$, that indicates if a route to dest was found, and set all the elements to false. Set $\mathsf{fin}[\mathsf{dest}] = \mathsf{true}$.
3: for all $v \in V$ do: Initialize $\mathsf{pub}_v[\mathtt{DUMMY}, u] = \mathsf{true}$ for all $u \in \mathsf{Adj}[v]$ and $\mathsf{pref}_v[\mathtt{DUMMY}] = 0$.
BFS:
4: for $2d_{\mathsf{depth}} + 1$ iterations do:
5:   for all $v \in V$ do:                         ▷ in parallel
6:     for all $u \in \mathsf{Adj}[v]$ do:            ▷ for all neighbors of $v$
7:       if $\mathsf{fin}[u] = \mathsf{true}$ and $\mathsf{pub}_u[\mathsf{next}[u], v] = \mathsf{true}$ and
         $\mathsf{pref}_v[\mathsf{next}[v]] < \mathsf{pref}_v[u]$ then
8:         $\mathsf{next}[v] \longleftarrow u$
9:         $\mathsf{fin}[v] \longleftarrow \mathsf{true}$
10: return next
```

Algorithm 2. Neighbor Preference Routing [3]

icy, i.e., to choose whether he wants to disclose a certain route to a neighbor or not. The pseudo-code of the algorithm is given in Alg. 2. We use a preference bit-length of $\rho = 4$ for good expressiveness in practice.

3.3 Removing Stub ASes

To reduce the complexity of the route-computation, our protocols are only run on the subgraph of the AS-graph that is induced by the non-stub ASes (i.e., by the ISPs). Stubs (by definition) have no customers, and so should never transit traffic between other ASes. Hence, in our scheme MPC is used to compute routes between ISPs (that form the core of the Internet). Then, stubs select an ISP through which to connect to the Internet according to their local routing policies. We point out that:

1. As stubs are roughly 85% of ASes, this means that the MPC protocol needs only be run on a fairly small part of the AS graph. In our experimental evaluation in §5 we show that removing the stubs improves the runtime of the MPC protocol by a factor of ≈ 2.5. For the CAIDA topology from November 2016, the number of ASes is reduced from almost 56 000 to 8 407 ASes, when excluding stubs.
2. Whether an AS is a stub (and not an ISP) is not considered confidential information and so our partition of the AS graph into these two distinct groups of ASes does not leak any sensitive information.
3. Observe that according to the routing policies presented in §2.2.2, to select between ISPs after the MPC step is complete, a stub needs only to know whether or not the ISP has a route to the destination, and the length of the route. This information can be announced directly to the stub by its ISP.

4 Circuit Representation

As a first step towards securely computing Alg. 1 and Alg. 2, we show how to construct Boolean circuits that implement these algorithms. In the following section we detail how to construct a Boolean circuit from the neighbor relation algorithm given in Alg. 1 (§4.1), describe the optimizations that we apply to it (§4.2), give a summary of the circuit for the neighbor preference algorithm in Alg. 2 (§4.3), and show security and privacy of our protocols (§4.4).

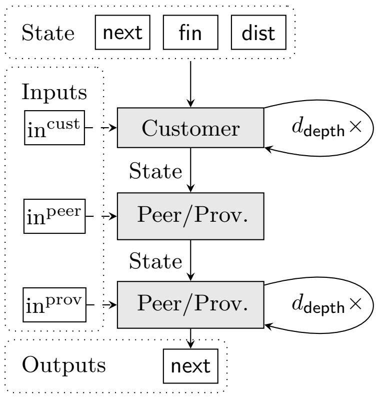

Fig. 2. Circuit Structure Overview.

4.1 'Naive' Implementation of Alg. 1

We first outline the general structure of the circuit (§4.1.1) and then show how to implement subroutines and analyze their complexities (§4.1.2). We provide a circuit structure outline in Fig. 2 and the complete circuit of our centralized BGP algorithm in Circ. 1.

```
 1: Initialization of next, fin and dist identical to Alg. 1.
BFS of customers:
 2: for $d_{\mathsf{depth}}$ times do:
 3:   for all $v \in V$ do:                              ▷ in parallel
 4:     for all $u \in \mathsf{Adj}[v]$ do:              ▷ for all neighbors of $v$
 5:       sel     ⟵ $\mathsf{fin}[u] \wedge !\mathsf{fin}[v] \wedge \mathrm{in}_v^{\mathrm{cust}}[u]$
 6:       sum     ⟵ $\mathrm{ADD}(\mathsf{dist}[u], 1)$
 7:       next[$v$] ⟵ $\mathrm{MUX}(\mathsf{next}[v], u, \mathsf{sel})$
 8:       dist[$v$] ⟵ $\mathrm{MUX}(\mathsf{dist}[v], \mathsf{sum}, \mathsf{sel})$
 9:       fin[$v$]  ⟵ $\mathrm{MUX}(\mathsf{fin}[v], \mathsf{sel}, \mathsf{sel})$
BFS of peers:
10: for all $v \in V$ do:                                ▷ in parallel
11:   for all $u \in \mathsf{Adj}[v]$ do:                ▷ for all neighbors of $v$
12:     sum     ⟵ $\mathrm{ADD}(\mathsf{dist}[u], 1)$
13:     cmp     ⟵ $\mathrm{GT}(\mathsf{dist}[v], \mathsf{sum})$
14:     sel     ⟵ $\mathsf{fin}[u] \wedge !\mathsf{fin}[v] \wedge \mathsf{cmp} \wedge \mathrm{in}_v^{\mathrm{peer}}[u]$
15:     next[$v$] ⟵ $\mathrm{MUX}(\mathsf{next}[v], u, \mathsf{sel})$
16:     dist[$v$] ⟵ $\mathrm{MUX}(\mathsf{dist}[v], \mathsf{sum}, \mathsf{sel})$
17:     fin[$v$]  ⟵ $\mathrm{MUX}(\mathsf{fin}[v], \mathsf{sel}, \mathsf{sel})$
BFS of providers:
18: for $d_{\mathsf{depth}}$ times do:
19:   for all $v \in V$ do:                              ▷ in parallel
20:     for all $u \in \mathsf{Adj}[v]$ do:              ▷ for all neighbors of $v$
21:       sum     ⟵ $\mathrm{ADD}(\mathsf{dist}[u], 1)$
22:       cmp     ⟵ $\mathrm{GT}(\mathsf{dist}[v], \mathsf{sum})$
23:       sel     ⟵ $\mathsf{fin}[u] \wedge !\mathsf{fin}[v] \wedge \mathsf{cmp} \wedge \mathrm{in}_v^{\mathrm{prov}}[u]$
24:       next[$v$] ⟵ $\mathrm{MUX}(\mathsf{next}[v], u, \mathsf{sel})$
25:       dist[$v$] ⟵ $\mathrm{MUX}(\mathsf{dist}[v], \mathsf{sum}, \mathsf{sel})$
26:       fin[$v$]  ⟵ $\mathrm{MUX}(\mathsf{fin}[v], \mathsf{sel}, \mathsf{sel})$
```

Circuit 1. Neighbor Relation Routing Circuit

4.1.1 Structure

Inputs. The circuit gets as input from each AS the secret shared relationship information for its neighbors. For efficiency reasons we have three separate input bit arrays for each AS, one for each type of AS relation: $\mathrm{in}^{\mathrm{cust}}$, $\mathrm{in}^{\mathrm{peer}}$, and $\mathrm{in}^{\mathrm{prov}}$. For a node with n neighbors, we have three inputs of length n bits each, where the i-th bit corresponds to the relation to the i-th neighbor. Note that the ASes only need to secret share their inputs among the computational nodes when updating their relationships since the computational nodes can reuse existing data for multiple executions. (If it should be hidden that a particular AS changed its relationships, then the secret sharing can be run again by all ASes. This might happen on a regular basis.) The routing destination dest and network topology are public inputs and thus not secret-shared.

State. We operate on a secret-shared state where we store for each node: a *finish* bit fin, a δ-bit long *destination id* of the next node on the routing tree next, and a σ-bit long *hop distance* to the target node dist. Initially, fin is set to false and next is set to zero, while dist is set to the maximum value $2^{\sigma} - 1$. This state is then iteratively updated using the methods for each relation. The *peer* and *provider* methods are identical, except for the different iterations and type of AS relation. We have d_{depth} iterations of the customer sub-circuit and $1+d_{\mathsf{depth}}$ iterations of a combined peer/provider sub-circuit, where we use the peer relation as input once, and the provider relation for the remaining iterations.

Outputs. After secure evaluation of the next hop on the route to dest for each AS $v \in V$, the computational parties send their share next[v] to every v, who can then reconstruct the plaintext output.

Parameters. According to the CAIDA dataset, we set the parameters in the protocol of §3 to $d_{\mathsf{depth}} = 10$ and therefore have 10 iterations of the customer routine, a single iteration for the peer routine, and 10 iterations for the provider routine. Furthermore, we set the id-bit length $\delta = \lceil \log_2 |V| \rceil$ and the destination bit-length to $\sigma = \lceil \log_2 21 \rceil = 5$, since we can at most achieve a distance of 21 hops between any AS and the destination (1 hop per each of the 10 customer and provider iterations and one peer iteration).

4.1.2 Operations and Complexities

The operations of the pseudo-code in Circ. 1 can be implemented using standard circuit constructions. Apart from the standard bit-wise operations AND ($\wedge$) and NOT (!), we use the following operations:

Addition (ADD). We use the Ripple-Carry addition circuit [35] with ℓ AND gates and a multiplicative depth of ℓ for addition of ℓ-bit values.

Greater-Than (GT). We use the greater-than circuit of [36] which has ℓ AND gates and a multiplicative depth of ℓ when comparing two ℓ-bit values.

`if`-Condition (MUX). To compute the `if`-condition securely, both branches must be evaluated to hide which branch has been chosen. The results are then assigned to the variables depending on the condition bit sel $\in \{0, 1\}$ using a multiplexer MUX. A multiplexer for ℓ-bit values requires ℓ AND gates [37] and its multiplicative depth is 1. There exist optimizations for the GMW protocol that allow the evaluation of an ℓ-bit multiplexer at the cost comparable to a single AND gate [6], as explained in §4.2.1.

We estimate that naively implemented circuits would have more than one hundred million AND gates, and a multiplicative depth of several hundred thousand. Furthermore, we estimate that the *total* number of gates is around 400 million. Since modern secure computation frameworks are able to evaluate 4-8 million AND gates per second [6, 20], securely evaluating the circuit would require nearly 100 s, which is arguably too long. Finally, the high depth translates to a huge number of communication rounds for the GMW protocol. In the next section, we show how the circuit can be optimized substantially to overcome these limitations.

4.2 Optimized Implementation of Alg. 1

In order to counter the problem of the high number of AND gates, the memory complexity for storing the circuit, and the high number of communication rounds, we propose to use the following three optimizations: reduce the complexity for evaluating AND gates by using *vector ANDs* (§4.2.1), decrease the number of gates in the circuit description by using *SIMD circuits* (§4.2.2), and decrease the circuit depth by building certain parts of the circuits as *tournament-like evaluation* (§4.2.3). We describe these optimizations in more detail next.

4.2.1 Vector ANDs

The naive circuit in Circ. 1 consists of many multiplexer gates operating on ℓ-bit values, needed to realize `if` conditions. As outlined in [6] and summarized in §2.1, these multiplexers can be instantiated using vector ANDs that reduce the precomputation cost from ℓ AND gates to the cost of one AND gate. Overall, the multiplexers constitute to around 75% of the total number of AND gates in the circuit. By using the vector AND optimization, we can therefore reduce the number of AND gates by factor 3 for the neighbor relation algorithm in Alg. 1 and by up to factor 45 for the neighbor preference algorithm in Alg. 2 (as shown in column Total ANDs vs. Vector ANDs in Tab. 2 in the Appendix). Note, however, that this optimization can only be applied when performing the evaluation with the GMW protocol, while an evaluation with Yao's garbled circuits has to process the total number of AND gates.

4.2.2 SIMD circuits

In order to cope with large circuits, there are two common approaches: pipelining the circuit construction and evaluation [24] and building a Single Instruction Multiple Data (SIMD) circuit. While the approach of pipelining the circuit construction and evaluation is especially suited for processing circuits of arbitrary size, we decided to pursue a solution based on SIMD techniques. A SIMD circuit also consists of gates, but instead of operating on single bits, it operates on multiple bits in parallel. Thereby, the time for the load / process / store operations of a gate amortizes, which drastically speeds up the evaluation [38]. In contrast, a pipelined construction and evaluation approach would need to perform a load / process / store operation per bit of evaluation. We will now describe how to build such a SIMD circuit that evaluates our BGP functionality.

Note that for the customer, peer, and provider functionality, we perform the same operation for each node $v \in V$ in parallel. Using SIMD circuits, we can combine the values for each node into vectors instead of single bits and thereby only build a single copy of the functionality. Thereby, we can operate on multiple values in parallel, which allows us to reduce the memory footprint of the circuit as well as to decrease the time for circuit evaluation. However, applying the SIMD programming style is not straight-forward since for each node the circuit depends on its degree, i.e., the number of its neighbor nodes n, which differs drastically between ASes. The obvious solution, that builds the circuit for the node with the highest degree $n_{\max}$ and pads the number of neighbor nodes for all other nodes to $n_{\max}$, introduces a non-tolerable overhead in terms of AND gates. We solve this problem as described next.

All nodes are divided into groups of similar degree. After each iteration the results from all groups are merged into a state, that is used as input to the next iteration. The challenge is to find the right amount and size of groups to partition the nodes. For our experiments, we use the following partitioning: $\{1, 2, \ldots, 6, 8, 12, 20, 32, 64, 128, 256, \ldots, n_{\max}\}$, where $n_{\max}$ is the highest number of neighboring nodes that any AS in the topology has.

4.2.3 Tournament evaluation

The current circuit has a high multiplicative depth, which makes it inefficient for secure computation protocols which require communication rounds linear in the circuit depth, e.g., the GMW protocol. The reasons for the high depth of the circuit are the iterative structure of Alg. 1 and the sequential processing of neighbors, which results in a circuit depth linear in $n_{\max}$, i.e., the highest number of neighbors of any AS in the graph (for the CAIDA dataset, $n_{\max}$=5 936). In order to reduce the depth for processing the neighbors, we adopt a

```
   Input: $v \in V, (\mathsf{next}[u_L], \mathsf{dist}[u_L], \mathsf{fin}[u_L], \mathrm{in}_v^{\mathrm{cust}}[u_L]),$
          $(\mathsf{next}[u_R], \mathsf{dist}[u_R], \mathsf{fin}[u_R], \mathrm{in}_v^{\mathrm{cust}}[u_R])$ with
          $u_L, u_R \in \mathsf{Adj}[v], u_L \neq u_R$
1: $\mathsf{sel} \longleftarrow \mathsf{fin}[u_L] \wedge !\mathsf{fin}[v] \wedge \mathrm{in}_v^{\mathrm{cust}}[u_L]$
2: $\mathsf{next}' \longleftarrow \mathrm{MUX}(\mathsf{next}[u_R], \mathsf{next}[u_L], \mathsf{sel})$
3: $\mathsf{dist}' \longleftarrow \mathrm{MUX}(\mathsf{dist}[u_R], \mathsf{dist}[u_L], \mathsf{sel})$
4: $\mathsf{fin}' \longleftarrow \mathrm{MUX}(\mathsf{fin}[u_R], \mathsf{fin}[u_L], \mathsf{sel})$
5: $\mathrm{in}' \longleftarrow \mathrm{MUX}(\mathrm{in}_v^{\mathrm{cust}}[u_R], \mathrm{in}_v^{\mathrm{cust}}[u_L], \mathsf{sel})$
   Output: $(\mathsf{next}', \mathsf{dist}', \mathsf{fin}', \mathrm{in}')$
```

Function 1. Selection Function customer

```
   Input: $v \in V, (\mathsf{next}[u_L], \mathsf{dist}[u_L], \mathsf{fin}[u_L], \mathrm{in}_v^{\mathrm{cust}}[u_L]),$
          $(\mathsf{next}[u_R], \mathsf{dist}[u_R], \mathsf{fin}[u_R], \mathrm{in}_v^{\mathrm{cust}}[u_R])$ with
          $u_L, u_R \in \mathsf{Adj}[v], u_L \neq u_R$
1: $\mathsf{dist}_{\max} = 2^\sigma - 1$
2: $\mathsf{sum}_L \longleftarrow \mathrm{MUX}(\mathsf{dist}_{\max}, \mathsf{dist}[u_L], \mathsf{fin}[u_L] \wedge \mathrm{in}_v^{\mathrm{peer/prov}}[u_L])$
3: $\mathsf{sum}_R \longleftarrow \mathrm{MUX}(\mathsf{dist}_{\max}, \mathsf{dist}[u_R], \mathsf{fin}[u_R] \wedge \mathrm{in}_v^{\mathrm{peer/prov}}[u_R])$
4: $\mathsf{sel} \longleftarrow \mathrm{GT}(\mathsf{sum}_R, \mathsf{sum}_L)$
5: $\mathsf{next}' \longleftarrow \mathrm{MUX}(\mathsf{next}[u_R], \mathsf{next}[u_L], \mathsf{sel})$
6: $\mathsf{dist}' \longleftarrow \mathrm{MUX}(\mathsf{dist}[u_R], \mathsf{dist}[u_L], \mathsf{sel})$
7: $\mathsf{fin}' \longleftarrow \mathrm{MUX}(\mathsf{fin}[u_R], \mathsf{fin}[u_L], \mathsf{sel})$
8: $\mathrm{in}' \longleftarrow \mathrm{MUX}(\mathrm{in}_v^{\mathrm{peer/prov}}[u_R], \mathrm{in}_v^{\mathrm{peer/prov}}[u_L], \mathsf{sel})$
   Output: $(\mathsf{next}', \mathsf{dist}', \mathsf{fin}', \mathrm{in}')$
```

Function 2. Selection Function peer / provider

tournament evaluation style by arranging operations in form of a tree and thereby achieve a logarithmic depth. We give the selection function for the customer functionality in Func. 1 and for the peer / provider functionality in Func. 2. Note that we can compute sel in Func. 1 as well as sum_L and sum_R in Func. 2 once in the beginning and pass/re-use them during the tournament evaluation to decrease the number of AND gates. Thereby, the overall number of AND gates in the circuit remains the same as for the sequential circuit.

4.3 Implementation of Alg. 2

The structure of the neighbor preference algorithm described in §3.2 is very similar to that of the peer/provider part of Alg. 1 described in §3.1. Thus, we can use the same structure, ideas and optimizations as described before to efficiently realize it as a Boolean circuit optimized for the evaluation with the GMW protocol. The main difference between the neighbor preference and the relation algorithm is the publish matrix pub, held by each AS. This matrix has dimension $|\mathsf{Adj}[v]| \times |\mathsf{Adj}[v]|$ and hence becomes very large for ASes with many neighbors. In fact, for the full CAIDA dataset from November 2016, only the AS with the most neighbors ($n_{\max} = 5\,936$) has a matrix with 35 236 096 bits. Each bit of this matrix has to be accessed once for each of the $2d_{\mathsf{depth}} + 1$ rounds in order to hide the current next hop, which costs one AND gate per bit. Overall, the total number of AND gates in the circuit for the full CAIDA dataset from November 2016 amounts to nearly 8 billion. The vector AND optimization allows us to perform a more efficient access and reduces the cost for processing this matrix in the setup phase to 130 million AND gates. However, during the online phase we have to evaluate the total number of AND gates, regardless of the vector AND optimization, which results in a communication of approximately 2 GiB of data which is an order of magnitude higher than for the relation-based algorithm of §3.1.

Additionally, in the neighbor preference algorithm, the computation parties need to perform lookups by secret-shared values in Step 7 (i.e., the lookups $\mathsf{pub}_u[\mathsf{next}[u], v]$ and $\mathsf{pref}_v[\mathsf{next}[v]]$). We implement these lookups by updating the values pub_u and pref_v for all nodes each time a new next hop is chosen. Note that updating the values can be done using the vector AND optimization, which greatly reduces the costs.

4.4 Security and Privacy

Given the above circuit representations, our final protocols apply the GMW protocol [5, 39] while using these circuits as public input to all computational parties. In a nutshell, all ASes secret share their inputs to the computational parties. Then, the parties use the GMW protocol to evaluate the circuit gate-by-gate, while maintaining the invariant that the value on each wire is secret shared among the computational parties. The parties evaluate the circuit, by securely computing a secret sharing of the output wire of the gate using its secret shared input wires. At the final stage, the computational parties send back to each AS its respective shares of the output, and never learn the output by themselves. Denote by Π_1 the protocol for computing Alg. 1, and by Π_2 the protocol for computing Alg. 2.

It is easy to see that our naive circuits (e.g., Circ. 1) correctly implement our algorithms. These are direct translations of the algorithms into lower level components, such as AND ($\wedge$) and NOT (!) gates, as well as ADD, GT and MUX. We rely on the correctness of the implementations of [35] for ADD-gates, [36] for GT and [6] for MUX, to conclude our final circuits, that use only AND and XOR gates.

The correctness of the protocols is derived from correctness of the GMW protocol, and the correctness of our circuits. The privacy of our protocols is derived from the proof of security [5, 39] of the GMW protocol for privately computing a given circuit while hiding the intermediate values on its internal wires. We specify these properties formally in Theorem 1.

Theorem 1. Security and Privacy
Protocol Π_1 (resp. Π_2) privately computes Alg. 1 (resp. Alg. 2) in the presence of a semi-honest adversary, corrupting at most $n-1$ out of n computational parties in addition to all but one ASes.

5 Benchmarks and Evaluation

In this section, we provide benchmark results of our protocols and evaluate the practicality of our solution.

We show that we are capable of securely evaluating the circuit for the full dataset in a reasonable runtime and further improve it using the algorithmic optimization of excluding stubs from the computation (cf. §3.3).

We implement our protocols using the ABY framework [6] which provides the two-party variant of the secure computation protocol by Goldreich-Micali-Wigderson (GMW) [5] with security against passive adversaries. The main reason for the GMW protocol are the optimizations from §4.2, that are only possible with GMW. Using Yao's garbled circuits, runtimes would become impractical. We provide further arguments for choosing the GMW protocol in Appendix B.

To the best of our knowledge, the optimizations described in §4.2 are only implemented in the ABY framework. We are not aware of automated tools capable of using the same optimizations to the same extent that we do in our hand-built circuits. We would like to point out, however, that our efficient circuits are generic Boolean circuits that could be evaluated with any secure computation framework and could thus be extended to more than 2 parties or even security against malicious adversaries (e.g. using [40] or [41]), with additional cost in communication and runtime.

Benchmarking Environment. Our MPC benchmarks are run on two Amazon EC2 `c4.2xlarge` instances with 8 virtual CPU cores with 2.9 GHz and 15 GiB RAM located in the same region, connected via a Gigabit network connection. The symmetric security parameter in our experiments is set to 128 bits. All runtime results are median values of 10 protocol executions and their standard deviation. The communication numbers provided are the sum of sent and received data for each party.

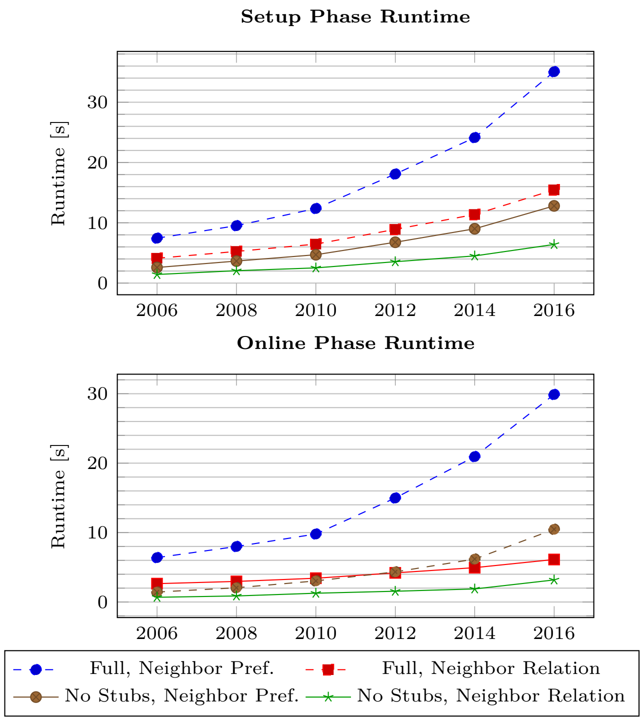

Fig. 3. Median runtimes for setup and online phase, for the CAIDA topology of November of every year, with and without stub nodes, comparing the neighbor relation algorithm (Alg. 1, §3.1) and the neighbor preference algorithm (Alg. 2, §3.2).

CAIDA (Fig. 3). A visualization for the evaluation of both protocols (*neighbor relationship* Alg. 1 and *neighbor preference* Alg. 2) on CAIDA datasets of the past 10 years is provided in Fig. 3. Detailed results are given in Tab. 2 in Appendix C. Our results show that both protocols spend most of the time in the setup phase which takes 15.46 s for the neighbor relation algorithm (35.07 s for the neighbor preference algorithm) for the full CAIDA November 2016 topology and reduces to 6.41 s (12.80 s) if we exclude stub nodes. Note that this part of the computation is less critical than the online phase for two reasons: a) it is independent of the network topology and input of the ASes and can thus be precomputed at any time and b) it can be ideally parallelized by adding more machines and thus is just dependent on the available resources. The online phase is the time required from a secret shared input of the ASes until the resulting next hop on the routing tree can be provided to them. For the full network, the online runtime is 6.12 s for the neighbor relation algorithm (29.89 s for the neighbor preference algorithm) and decreases to 3.18 s (10.47 s) when leaving out stub nodes.

Generally speaking, the algorithmic improvement of removing stub nodes from the network topology speeds up both protocols by a factor between 2 and 3.

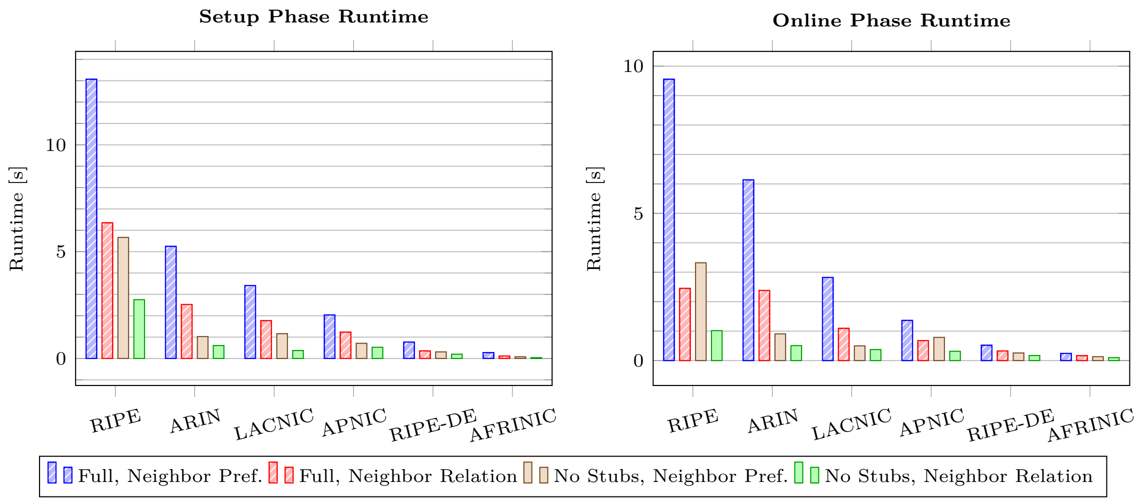

Fig. 4. Median runtimes for setup and online phase, for subgraphs of CAIDA's November 2016 topology, with and without stubs, comparing the neighbor relation algorithm (Alg. 1, §3.1) and the neighbor preference algorithm (Alg. 2, §3.2).

The required bandwidth between the two computational parties in the online phase for the neighbor relationship algorithm is less than 60 MiB, while the more complex neighbor preference algorithm requires around 800 MiB for the most recent topology without stub nodes. Our results also show that the online runtime of the preference algorithm scales worse with a growing topology size. In general, communication between the two computing parties takes approximately 1/3 of the runtime of the online phase and is also the part that induces the biggest runtime variations, even on a local network. The remaining 2/3 of the runtime in the online phase are spent on local computations consisting of simple bit operations and memory accesses.

RIRs (Fig. 4). We show similar measurements for subgraphs of CAIDA's November 2016 topology for 5 RIRs and a regional topology for German ASes (RIPE-DE) in Fig. 4 and provide detailed numbers in Tab. 3 in Appendix C. When considering the smaller RIR topologies, the online time decreases below 10 s for both algorithms, even for big sub-topologies such as RIPE or ARIN. For smaller sub-topologies, the online runtime decreases even further. For instance, on the full RIPE-DE sub-topology, the online runtime for the neighbor relation algorithm is 0.33 s (0.52 s for the neighbor preference algorithm) and decreases to 0.17 s (0.26 s), when leaving out stubs. In a similar fashion, the required bandwidth decreases to 1.0 MiB for the neighbor relation algorithm and to 3.6 MiB for the neighbor preference algorithm withour stub nodes.

6 Deployment and Future Work

In this section, we explain our network assumptions and propose further enhancing security and privacy by enforcing input consistency. We discuss how to handle failures and byzantine behavior, and possible deployment.

6.1 Network Considerations

The connection between the CPs is a critical point in our system and has to be low-latency and high-throughput. We argue that this is realistic, since reputed entities that run the CPs are often co-located in the same data centers, yet managed by different authorities. The communication between ASes and CPs can be an arbitrary Internet connection with no special requirements. Error correction can be applied on the usual network layers (e.g. within TCP/IP or on application level). Packet loss between ASes and CPs does not cause severe problems, as old inputs from ASes can be re-used in multiple protocol iterations. Lost outputs have the same consequences as lost BGP messages nowadays. However, packet loss is covered by the use of TCP/IP that re-sends lost packets. The ASes have to do one round of communication with the computational parties that run our protocols. Thus, we have measured the round trip time (RTT) between our computational parties and other Amazon EC2 regions and observed average RTTs between 90 ms and 311 ms, as depicted in Tab. 1. This time has to be added to the MPC runtime to get the time an individual AS has to wait for a computation result. The standard deviations of the RTTs and packet loss were less than 1%.

Table 1. Average round trip times between Amazon EC2 regions measured from EU (Frankfurt) to the listed regions.

Location	∅ RTT
US East (Northern Virginia)	90.2 ms
US West (Northern California)	162.5 ms
South America (São Paulo)	193.4 ms
Asia Pacific (Mumbai)	112.5 ms
Asia Pacific (Tokyo)	228.5 ms
Asia Pacific (Sydney)	311.3 ms

6.2 Input Consistency

The centralized evaluation gives us the powerful ability to check the ASes' inputs for consistency. Since our solutions aim at protecting AS relations and local preferences, the CPs could have an overview of the announced prefixes and can detect malicious behavior such as prefix hijacking or misconfigurations. One more specific attack that can be prevented by this, is the following: AS a claims that AS b is its peer, while b claims that a is its provider. Clearly, one of them is lying. To verify the symmetry of input relations (cf. §2.2), we require only a single layer of AND gates that processes the inputs, adding negligible complexity to the overall algorithm. This check has to be done only once whenever an AS changes its inputs. Further, more complex sanity checks of encrypted data can be added on top of that at the expense of longer runtimes, while validity checks of plaintext inputs can be done rather easily.

When inconsistencies are detected, the CPs can discard these new and inconsistent inputs and fall back to previous inputs. Involved ASes can be queried to resend their inputs if we suspect that the inconsistency happened erroneously or due to faulty transmission. If an AS is detected as malicious or permanently faulty, the computational parties, can virtually remove this AS from the public topology and ignore it until recovery. This has the effect that no route will be sent via a faulty or malicious AS.

6.3 Handling Failures and Byzantine Behavior

Our approach preserves the privacy of interdomain route-computation against honest-but-curious attackers. However, the MPC itself is a single point of failure, as the routing depends on the availability and the honesty of two computational parties. We propose to add robustness by running multiple, independent 2-party MPC sessions in parallel. Alternatively, one could also use secure multi-party computation protocols based on t-out-of-n secret sharing that work even if all but t out of the n computing parties fail. Identifying more efficient schemes is an interesting direction for future research.

In §6.2 we showed that our approach can be adapted efficiently to the case of malicious ASes, but where the computational parties are still semi-honest. Another future direction is to protect against malicious computational parties, while keeping runtimes practical, at least at country- or region-level scales. Another possible approach is to obtain security in a slightly weaker adversarial model, which is the covert security model [42]. By small adaptations of our protocol we can obtain a semi-honest version of GMW for the multi-party case (instead of two-party as we considered). Assuming honest majority, such a protocol can be transformed easily (and in a black-box way) to efficient protocols in the more robust setting of covert adversaries [42], at the expense of just running the protocol several times (in parallel) [43]. In this setting, the corrupted party might not follow the protocol, and by doing so it can also sometimes break the security of the protocol. Nevertheless, the security guarantee is that any cheating attempt can be recognized by the honest party with some high probability (say, 50%). Furthermore, by some additional (cheap) adaptations of the protocol, any cheating attempt can also be publicly verified [44, 45], which enables the honest party to persuade other third parties (e.g., a "judge") about the cheating attempt. Since the computational parties in our settings are reputed authorities such as IANA/NANOG/RIPE, etc., we believe that the fear of being caught, the public humiliation or even the legal consequences is enough of a deterrent to prevent any cheating attempt.

6.4 Deployment

Our approach is primarily intended as a broad vision for the future of interdomain routing. Of course, transitioning to MPC of interdomain routes is an extremely challenging undertaking that involves cooperation of tens of thousands of independent financial and political entities, alongside significant deployment and operational challenges. We believe, however, that our approach can also yield significant benefits (e.g., in terms of privacy, security, and ability to innovate) when applied at a smaller scale, while alleviating many of the challenges a global transition entails. Often, for performance and security reasons, traffic within a geographic/political region is expected to not leave the boundaries of that region. One example for this is that many end-users retrieve content from servers in their geographic region due to the popularity of content delivery networks (à la Akamai). Thus, a natural deployment scenario for MPC of interdomain routes is focusing on a specific region and executing MPC only for routes between the ASes in that region, while all other routes will be computed via traditional (decentralized) BGP routing.

This would also offer very natural instantiations of the MPC parties: The computation could be done by the RIR as well as a local IXP, such as DE-CIX. While being independent entities, they typically share a fast and low latency network connection, which is required for our protocols. We give the runtimes for such subgraphs for the RIRs in Fig. 4 and observe that such a regional execution also drastically decreases the runtime of our algorithms (e.g., 0.20 s setup time and 0.17 s online time for the German RIR RIPE-DE). We point out that such a scheme, beyond MPC's inherent privacy guarantees and the other benefits listed in §1.3, can provide the guarantees that all routes between ASes in the region will indeed only traverse other ASes in that region. This should be contrasted with today's insecure routing with BGP, which allows a remote AS to manipulate the routing protocol so as to attract traffic to its network. Our evaluations show that, beyond the above guarantees, it also yields better running times due to the smaller size of the "input". We believe that a region/country-level implementation of such MPC of routes is a tangible and beneficial first step en route to larger-scale deployment scenarios.

6.5 Hiding the Network Topology

Currently, we exploit the fact that the network topology is public for many implementation optimizations in the circuit. However, we could also keep the topology private, which comes at an overhead of $O(n^3)$, where n is the number of ASes [46]. For country-level ASes, especially when excluding stub-nodes, this overhead seems tolerable. E.g., the RIPE-DE country-level AS has 250 non-stub ASes, which would result in a circuit with 30 million AND gates for the neighbor relations algorithm of §3.1 and around 200 million gates for the neighbor preference algorithm of §3.2.

Acknowledgments. We thank the anonymous reviewers for their valuable feedback on our paper. This work has been co-funded by the German Federal Ministry of Education and Research (BMBF) and by the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP, and by the DFG as part of projects S5 and E3 within the CRC 1119 CROSSING. The third author is supported by an ERC Starting Grant and the Israel National Cyber Bureau. This work was supported by a Junior Fellow award from the Simons Foundation, by the European Union's 7th Framework Program (FP7) via a Marie Curie Career Integration Grant, by the European Union's Horizon 2020 Framework Program (H2020) via an ERC Grant (Grant No. 714253), by the Israel Science Foundation (Grant No. 483/13), by the Israeli Centers of Research Excellence (I-CORE) Program (Center No. 4/11), by the US-Israel Binational Science Foundation (Grant No. 2014632), by NSF Grant No. 1420064, and by a Google Faculty Research Award. Part of the work was done while the first author was a post-doc at the IBM TJ Watson Research Center, supported by NSF Grant No. 1017660.

References

[1] S. Machiraju and R. H. Katz. Leveraging BGP dynamics to reverse-engineer routing policies. Technical Report UCB/EECS-2006-61, EECS Department, University of California, Berkeley, May 2006.

[2] V. Giotsas and S. Zhou. Inferring AS relationships from BGP attributes. *CoRR*, abs/1106.2417, 2011.

[3] 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 *Workshop on Hot Topics in Networks (HotNets'12)*, pages 37–42. ACM, 2012.

[4] P. Gill, M. Schapira, and S. Goldberg. Let the market drive deployment: a strategy for transitioning to BGP security. In *SIGCOMM'11*, pages 14–25. ACM, 2011.

[5] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC'87*, pages 218–229. ACM, 1987.

[6] D. Demmler, T. Schneider, and M. Zohner. ABY – a framework for efficient mixed-protocol secure two-party computation. In *NDSS'15*. The Internet Society, 2015. Code: https://github.com/encryptogroup/ABY.

[7] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. *SIGCOMM'00*, 30(4):175–187, 2000.

[8] B. Zhang, D. Massey, and L. Zhang. Destination reachability and BGP convergence time [border gateway routing protocol]. In *GLOBECOM'04*, volume 3, pages 1383–1389. IEEE, 2004.

[9] R. Oliveira, B. Zhang, D. Pei, and L. Zhang. Quantifying path exploration in the internet. *IEEE/ACM Transactions on Networking*, 17(2):445–458, 2009.

[10] N. Kushman, S. Kandula, and D. Katabi. Can you hear me now?!: It must be BGP. *SIGCOMM'07*, 37(2):75–84, 2007.

[11] L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, 2001.

[12] A. Fabrikant, U. Syed, and J. Rexford. There's something about MRAI: timing diversity can exponentially worsen BGP convergence. In *INFOCOM'11*, pages 2975–2983. IEEE, 2011.

[13] K. R. B. Butler, T. R. Farley, P. McDaniel, and J. Rexford. A survey of BGP security issues and solutions. *Proceedings of the IEEE*, 98(1):100–122, 2010.

[14] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC'88*, pages 1–10. ACM, 1988.

[15] A. Ben-Efraim, Y. Lindell, and E. Omri. Optimizing semi-honest secure multiparty computation for the internet. In *CCS'16*, pages 578–590. ACM, 2016.

[16] W. Henecka and M. Roughan. STRIP: privacy-preserving vector-based routing. In *International Conference on Network Protocols (ICNP'13)*, pages 1–10. IEEE, 2013.

[17] J. Brickell and V. Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT'05*, volume 3788 of *LNCS*, pages 236–252. Springer, 2005.

[18] M. Blanton, A. Steele, and M. Alisagari. Data-oblivious graph algorithms for secure computation and outsourcing. In *ASIACCS'13*, pages 207–218. ACM, 2013.

[19] H. Carter, B. Mood, P. Traynor, and K. Butler. Secure outsourced garbled circuit evaluation for mobile phones. In *USENIX Security'13*, pages 289–304. USENIX, 2013.

[20] C. Liu, X. Shaun Wang, K. Nayak, Y. Huang, and E. Shi. ObliVM: A programming framework for secure computation. In *S&P'15*, pages 359–376. IEEE, 2015.

[21] A. C. Yao. How to generate and exchange secrets. In *FOCS'86*, pages 162–167. IEEE, 1986.

[22] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *USENIX Security'04*, pages 287–302. USENIX, 2004.

[23] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS'08*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.

[24] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security'11*, pages 539–554. USENIX, 2011.

[25] 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 *CT-RSA'12*, volume 7178 of *LNCS*, pages 416–432. Springer, 2012.

[26] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *FC'09*, volume 5628 of *LNCS*, pages 325–343. Springer, 2009.

[27] D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht. How the Estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In *FC'15*, volume 8975 of *LNCS*, pages 227–234. Springer, 2015.

[28] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[29] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS'13*, pages 535–548. ACM, 2013.

[30] T. Griffin, F. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Transactions on Networking*, 10(2):232–243, 2002.

[31] P. Gill, M. Schapira, and S. Goldberg. Modeling on quicksand: dealing with the scarcity of ground truth in interdomain routing data. *Computer Communication Review*, 42(1):40–46, 2012.

[32] S. Goldberg, M. Schapira, P. Hummon, and J. Rexford. How secure are secure interdomain routing protocols. In *SIGCOMM'10*, pages 87–98. ACM, 2010.

[33] The CAIDA AS relationships datasets. http://www.caida.org/data/as-relationships/.

[34] GeoLite data created by MaxMind. http://dev.maxmind.com/geoip/.

[35] J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$. *Theoretical Computer Science*, 235(1):43–57, 2000.

[36] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS'09*, volume 5888 of *LNCS*, pages 1–20. Springer, 2009.

[37] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP'08*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[38] T. Schneider and M. Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *FC'13*, volume 7859 of *LNCS*, pages 275–292. Springer, 2013.

[39] O. Goldreich. *The Foundations of Cryptography - volume 2, Basic Applications*. Cambridge University Press, 2004.

[40] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *CRYPTO'12*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[41] J. B. Nielsen, T. Schneider, and R. Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In *NDSS'17*. The Internet Society, 2017.

[42] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC'07*, volume 4392 of *LNCS*, pages 137–156. Springer, 2007.

[43] I. Damgård, M. Geisler, and J. B. Nielsen. From passive to covert security at low cost. In *TCC'10*, volume 5978 of *LNCS*, pages 128–145. Springer, 2010.

[44] G. Asharov and C. Orlandi. Calling out cheaters: Covert security with public verifiability. In *ASIACRYPT'12*, volume 7658 of *LNCS*, pages 681–698. Springer, 2012.

[45] Vladimir Kolesnikov and Alex J Malozemoff. Public verifiability in the covert model (almost) for free. In *ASIACRYPT'14*, volume 9453 of *LNCS*, pages 210–235. Springer, 2014.

[46] A. Aly, E. Cuvelier, S. Mawet, O. Pereira, and M. Van Vyve. Securely solving simple combinatorial graph problems. In *FC'13*, volume 7859 of *LNCS*, pages 239–257. Springer, 2013.

[47] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT'15*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.
[48] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *S&P'13*, pages 478–492. IEEE, 2013.

Appendix

A BGP Network

In the following, we provide insight into the development and growth of the BGP network in the past years. With the historic data collected by CAIDA [33], we can assess how the number of ASes, the number of connections between them, and the maximum degree of ASes develops over time. In Fig. 6, we show these values for both the full CAIDA dataset, as well as the set with the stub nodes removed. In Fig. 5, we depict the count of ASes with a given number of neighbors. We show that ASes are only sparsely connected and most nodes only have a small number of neighbors. Note that both axes are in a logarithmic scale. 91% of the nodes have at most 8 neighbors in the full topology, while in the no-stub topology the average degree is higher and the number of nodes with degree 8 and less is 69%.

B GMW vs. Yao

We decided to rely on the MPC framework ABY [6] for implementing our protocols, since it is publically available and implements many recent optimizations. ABY provides two different approaches for secure two-party computation on Boolean Circuits: the GMW protocol [5] or Yao's garbled circuits [21]. In the following, we justify our decision to choose GMW for our implementation.

B.1 Generic Protocol Differences

The features of each protocol make it advantageous for use in different scenarios. Typically, the main factors to decide which protocol to use are the network latency and the multiplicative depth of the function that is evaluated, since GMW requires a number of communication rounds that is linear in the depth while Yao's garbled circuits has a constant number of rounds.

For our evaluation, however, we use GMW, even though our circuit has a high multiplicative depth. Our decision is due to the following reasons:

Precomputation. GMW allows precomputation of all symmetric cryptographic operations and communication independently of the circuit (in our case independent of the AS topology) and its inputs. Additionally, this setup phase can be heavily parallelized and easily computed by multiple machines.

Multi-Party. GMW allows for easy extension to multiple computing parties, which is good for our setting where we might want to use more parties for better trust assumptions.

Balanced Workload. Unlike Yao's protocol, GMW balances the workload equally between all computational parties.

Lower Memory Consumption. The memory consumption for circuit evaluation is much lower for GMW, since GMW only needs to process single bits while Yao's garbled circuits needs to process symmetric keys of length 128 bits.

SIMD Evaluation. Only GMW allows more efficient parallel evaluation of gates by processing multiple bits per register, which is especially important for large circuits (cf. §4.2.2).

Vector ANDs. Only GMW supports vector ANDs, that reduce the number of OTs that have to be evaluated and allows the construction of very efficient MUX gates that allow for highly efficient instantiation of multiplexers which occur frequently in the circuit (cf. §4.2.1). For our circuit, this reduces the required number of OTs by a factor between 3 and 45.

Note, however, that the Boolean circuit we designed for computing the routing tree of BGP is independent of the underlying secure computation scheme. Hence, for networks with high latency, one could simply use an implementation of Yao's protocol [21] with pipelining [24] for evaluation instead of GMW.

B.2 Why Yao's Protocol Won't Work

In the following, we argue why we cannot achieve reasonable runtimes when evaluating the BGP circuits with Yao's protocol.

Communication. When evaluating the BGP circuit with Yao's Garbled Circuits protocol, we will have higher bandwidth requirements, since we cannot use the optimized vector MUX gates and thus have to evaluate between 3 to 45 times more AND gates. Today's

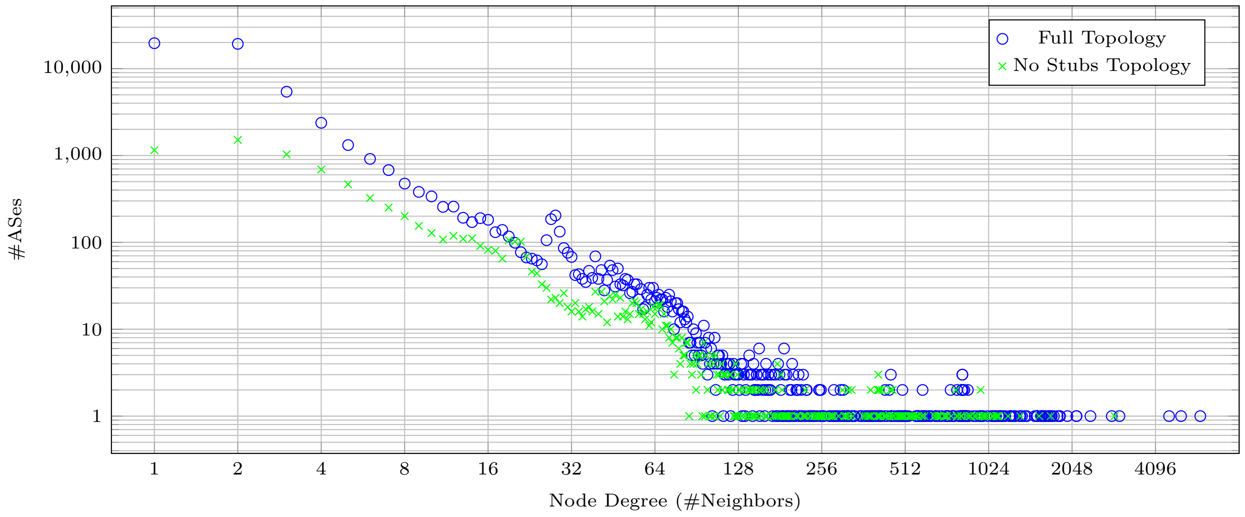

Fig. 5. Distribution of node degree for the full 20161101 CAIDA dataset compared to the set without stub nodes.

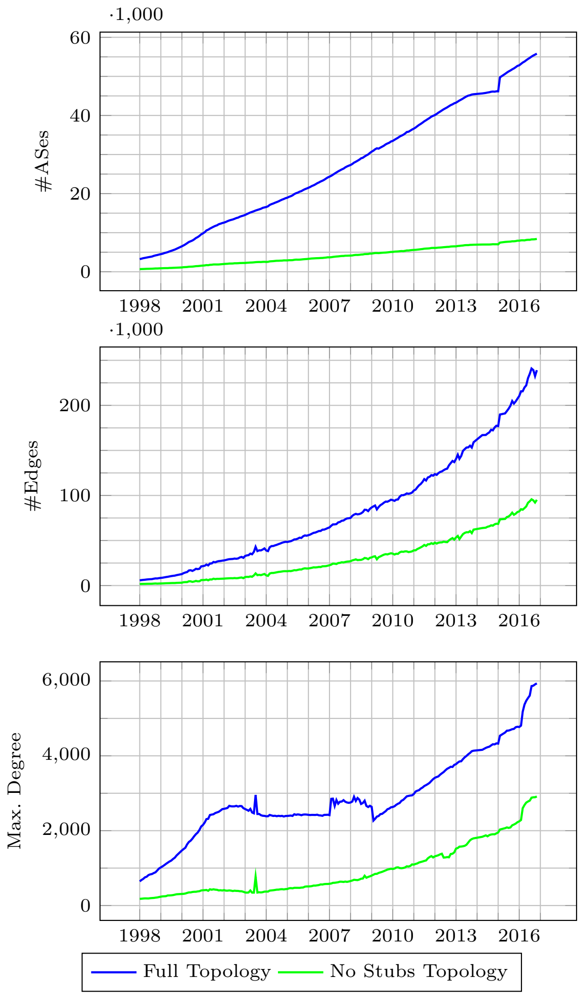

Fig. 6. Statistics for the number of ASes, edges between them and the maximum degree for all available CAIDA ASrel datasets from January 1998 until November 2016 for the full topology compared to the topology without stub nodes.

most communication-efficient method for garbled circuits [47] requires $2 \cdot 128 = 256$ bit of communication per AND gate. We also have to consider the problem, that we cannot precompute the setup phase independently of the AS topology and hence have to use circuit pipelining [24], i.e., generate and transmit the garbled circuit in the online phase. Assuming an ideal Gigabit network connection the resulting runtime for communication alone will be $380 \cdot 10^6$ AND gates $\cdot$ 256 bit per AND gate / 1 Gbps $\approx$ 100 s for the full topology and ≈ 35 s without stubs. We emphasize that these ideal runtimes are already higher than our combined setup and online time.

Computation. The fastest available Yao's garbled circuits implementation is JustGarble [48] that requires 48.4 cycles for evaluating and 101.3 cycles for garbling gates in "larger" circuits [48, Fig. 10]. Using a 3.5 GHz CPU we need $(380 \cdot 10^6$ AND gates $+$ $797 \cdot 10^6$ XOR gates$) \cdot 101.3$ cycles per gate / $(3.5 \cdot 10^9$ cycles per second$) \approx$ 34 s for the full topology and ≈ 12 s without stubs for garbling the circuit (evaluation can be done in parallel when using pipelining).

Summary. Overall, the runtime for Yao's garbled circuits, even with the fastest available implementation, most recent optimizations, and an ideal network, would be significantly slower than the runtimes we achieve with the GMW protocol.

C Benchmark Results

Here, we provide detailed numbers for our performance evaluation, which we describe in §5.

In Tab. 2, we give detailed evaluation results for both of our protocols on the CAIDA datasets of the past 10 years, where the most recent results for the topology of November 2016 are marked in bold. We list the number of ASes, the connections between them, and the maximum degree for each benchmarked topology. Furthermore, we list the circuit sizes as total number of AND gates (that one would have to evaluate *without* using the vector gate optimization of the GMW protocol respectively when using Yao's protocol), the number of gates when using the vector gate optimization for GMW, and the depth of the circuit, i.e. the number of communication rounds between the two computational parties.

In Tab. 3, we list the same values for subgraphs of the CAIDA dataset from November 2016 that correspond to the RIR networks and a local topology as described in §6.4.

Table 2. Comparison of topology, circuit and MPC runtimes of both algorithms from §3, using CAIDA datasets from past years, comparing the full dataset with the topology without stubs. The depicted communication happens solely between the CPs. We used the November dataset from every respective year. Most recent values are marked in **bold**.

			Topology			Circuit			Benchmarks			
									Setup Phase		Online Phase	
	CAIDA Dataset		ASes	Edges	max. Degree	Total ANDs [·10^6]	Vector ANDs [·10^6]	AND Depth	Runtime [s]	Comm. [MiB]	Runtime [s]	Comm. [MiB]
Full Topology	2008	Alg. 1 §3.1	30 018	82 630	2 632	123	37	1 042	5.25 (± 1 %)	1 136	2.96 (± 1 %)	47
	2012		42 847	138 306	3 703	217	63	1 042	8.91 (± 1 %)	1 921	4.20 (± 0 %)	82
	2016		55 809	239 064	5 936	380	110	1 118	**15.46** (± 0 %)	**3 344**	**6.12** (± 1 %)	**143**
	2008	Alg. 2 §3.2	30 018	82 630	2 632	2 074	58	1 430	9.52 (± 1 %)	1 767	7.99 (± 2 %)	520
	2012		42 847	138 306	3 703	4 428	99	1 430	18.09 (± 0 %)	3 028	14.98 (± 2 %)	1 105
	2016		55 809	239 064	5 936	6 603	184	1 535	**35.07** (± 0 %)	**4 577**	**29.89** (± 0 %)	**2 130**
No Stubs	2008	Alg. 1 §3.1	4 550	29 275	764	43	14	890	2.05 (± 3 %)	418	0.87 (± 1 %)	16
	2012		6 483	52 661	1 384	78	25	966	3.55 (± 2 %)	760	1.55 (± 0 %)	30
	2016		8 407	95 157	2 913	147	45	1 042	**6.41** (± 1 %)	**1 374**	**3.18** (± 1 %)	**55**
	2008	Alg. 2 §3.2	4 550	29 275	764	434	22	1 220	3.65 (± 2 %)	687	2.05 (± 1 %)	111
	2012		6 483	52 661	1 384	1 173	41	1 325	6.77 (± 1 %)	1 255	4.34 (± 1 %)	296
	2016		8 407	95 157	2 913	3 142	75	1 430	**12.80** (± 0 %)	**2 283**	**10.47** (± 1 %)	**785**

Table 3. Comparison of topology, circuit and MPC runtimes of both algorithms from §3, using subgraphs of the CAIDA datasets from November 2016, comparing the full dataset with the sub-topology, with and without stub nodes. The depicted communication happens solely between the CPs.

			Topology			Circuit			Benchmarks			
									Setup Phase		Online Phase	
	Dataset		ASes	Edges	max. Degree	Total ANDs [·10^6]	Vector ANDs [·10^6]	AND Depth	Runtime [s]	Comm. [MiB]	Runtime [s]	Comm. [MiB]
Full Topology	CAIDA 2016	Algo. 1 §3.1	55 809	239 064	5 936	380	110	1 118	**15.46** (± 0 %)	**3 344**	**6.12** (± 1 %)	**143**
	RIPE		21 723	95 909	1 769	149	44	966	6.35 (± 1 %)	1 358	2.45 (± 1 %)	56
	ARIN		16 942	39 563	3 047	56	17	1 042	2.53 (± 3 %)	518	2.38 (± 1 %)	21
	APNIC		7 505	18 802	727	25	8.1	890	1.23 (± 4 %)	249	0.68 (± 1 %)	9.5
	LACNIC		5 283	28 374	1 066	39	12.5	966	1.77 (± 3 %)	381	1.09 (± 1 %)	15
	RIPE-DE		1 328	5 375	372	6.8	2.4	814	0.35 (±11 %)	72	0.33 (± 1 %)	2.6
	AFRINIC		916	1 644	199	1.8	0.7	738	0.12 (±16 %)	21	0.17 (± 2 %)	0.7
	CAIDA 2016	Algo. 2 §3.2	55 809	239 064	5 936	6 603	184	1 535	**35.07** (± 0 %)	**4 577**	**29.89** (± 0 %)	**2 130**
	RIPE		21 723	95 909	1 769	2 844	71	1 325	13.07 (± 1 %)	2 185	9.55 (± 2 %)	711
	ARIN		16 942	39 563	3 047	1 325	26	1 430	5.25 (± 1 %)	789	6.14 (± 1 %)	330
	APNIC		7 505	18 802	727	200	12.5	1 220	2.04 (± 4 %)	386	1.36 (± 1 %)	52
	LACNIC		5 283	28 374	1 066	693	20	1 325	3.42 (± 3 %)	622	2.82 (± 2 %)	174
	RIPE-DE		1 328	5 375	372	44	3.8	1 115	0.77 (± 8 %)	118	0.52 (± 2 %)	12
	AFRINIC		916	1 644	199	6.3	1.0	1 010	0.27 (±14 %)	33	0.24 (± 3 %)	1.8
No Stubs	CAIDA 2016	Algo. 1 §3.1	8 407	95 157	2 913	147	45	1 042	**6.41** (± 1 %)	**1 374**	**3.18** (± 1 %)	**55**
	RIPE		3 646	41 274	918	58	19	890	2.75 (± 2 %)	583	1.02 (± 1 %)	22
	ARIN		1 849	8 501	665	11	3.7	890	0.61 (± 7 %)	112	0.51 (± 1 %)	4.0
	APNIC		1 140	5 398	338	6.7	2.3	814	0.37 (±11 %)	71	0.31 (± 1 %)	2.5
	LACNIC		1 012	8 367	484	9.9	3.6	814	0.52 (± 9 %)	109	0.37 (± 1 %)	3.8
	RIPE-DE		250	2 219	167	2.5	1.0	738	0.20 (±16 %)	31	0.17 (± 2 %)	1.0
	AFRINIC		178	371	77	0.4	0.2	662	0.04 (±12 %)	5.1	0.10 (± 3 %)	0.2
	CAIDA 2016	Algo. 2 §3.2	8 407	95 157	2 913	3 142	75	1 430	**12.80** (± 0 %)	**2 283**	**10.47** (± 1 %)	**785**
	RIPE		3 646	41 274	918	884	32	1 220	5.67 (± 2 %)	971	3.32 (± 3 %)	223
	ARIN		1 849	8 501	665	86	6	1 220	1.02 (± 6 %)	182	0.91 (± 1 %)	23
	APNIC		1 140	5 398	338	36	3.7	1 115	0.71 (± 9 %)	117	0.50 (± 2 %)	9.7
	LACNIC		1 012	8 367	484	109	5.8	1 115	1.16 (± 6 %)	182	0.79 (± 2 %)	28
	RIPE-DE		250	2 219	167	13	1.6	1 010	0.31 (±14 %)	53	0.26 (± 2 %)	3.6
	AFRINIC		178	371	77	1.0	0.2	905	0.08 (±10 %)	9.2	0.13 (± 4 %)	0.3