Private and Verifiable Interdomain Routing Decisions

Mingchen Zhao, Wenchao Zhou, Alexander J. T. Gurney, Andreas Haeberlen, Micah Sherr, and Boon Thau Loo

***Abstract*—Existing secure interdomain routing protocols can verify validity properties about individual routes, such as whether they correspond to a real network path. It is often useful to verify more complex properties relating to the route decision procedure – for example, whether the chosen route was the best one available, or whether it was consistent with the network's peering agreements. However, this is difficult to do without knowing a network's routing policy and full routing state, which are not normally disclosed. In this paper, we show how a network can allow its peers to verify a number of nontrivial properties of its interdomain routing decisions *without* revealing any additional information. If all the properties hold, the peers learn nothing beyond what the interdomain routing protocol already reveals; if a property does not hold, at least one peer can detect this and prove the violation. We present SPIDeR, a practical system that applies this approach to the Border Gateway Protocol, and we report results from an experimental evaluation to demonstrate that SPIDeR has a reasonable overhead.**

***Index Terms*—Collaborative verification, interdomain routing, privacy, security.**

I. Introduction

IN INTERDOMAIN routing, there is an inherent tension between verifiability and privacy: both properties are desirable, but they seem contradictory. Communicating networks have expectations about one another's routing decisions, but they are stymied from verifying these expectations because routing configurations are usually kept confidential.

Routing promises. Interdomain routing policies are routinely governed by formal agreements, such as peering and transit contracts, and the correct implementation of these policies is vital for allowing networks to achieve other contractual goals, such as maintaining traffic ratios [8]. In some cases, such as 'partial transit' relationships, the desired policy can be complex, placing additional cost on the implementors [35]. Less formally, networks often publish information on how customers and others can tweak the route selection process, using *BGP communities* [3], [5], [38]. Such capabilities represent an understanding between the networks about how certain routes should be treated. Negotiation about the provision of routing services, including undertakings about routing policy, is an important part of the Internet interconnection marketplace [8].

The value of verifiability. Unfortunately, these promises are not always kept, and violations are hard to detect. Promise-breaking may be deliberate, since networks may have economic incentives to lie about their routes [13]. Other examples of malicious behavior abound [19], [33]; one study found that 18 of 28 peering agreements contained clauses against abuse of the peering relationship through BGP configuration [31]. More innocently, misconfigurations [24], compromised routers [30], or equipment failures [44] can cause routing decisions to deviate from expectations.

Secure variants of BGP (e.g., S-BGP [20]) have been proposed as mechanisms for ISPs to check whether routing announcements correspond to the claimed path and destination, but these mechanisms do not address the important question of whether the *route decision process* matches expectations. Complete verification could be enabled by revealing all routing tables [17], but verifiability is not the only concern.

The value of privacy. For operational security or commercial reasons, ISPs have traditionally been reluctant to disclose details of their routing policy. Some aspects may be revealed to neighbors, included in a route registry, or exposed indirectly via looking glass services, but we cannot expect network operators to agree to use any system that reveals even more of their private information. Existing work has shown that it is possible to make deductions about which autonomous systems are connected, and even about some aspects of policy [6], [10], [22], [36], [41], but these inferences have limited accuracy [35] and require considerable effort to carry out, making them unsuitable for verifying routing decisions.

Can we have both? Intuitively, it seems that verifiability and privacy are conflicting goals—by revealing more information, we can improve verifiability, but we reduce privacy. In this paper, we show that this intuition is incorrect. We present a first step towards an interdomain routing system in which networks can verify each other's promises *without revealing any additional information*. Our approach is to enable the networks to verify promises *collaboratively*: Each promise is broken into small pieces such that a) each piece can be verified by some network using only information it already knows, and b) a successful verification of all pieces implies that the promise has been kept.

We show that collaborative verification is possible for an entire category of nontrivial promises, covering the relative preference assigned to different classes of route, and including the possibility of route filtering. We present a practical verification algorithm called *VPref*, as well as a formal proof that VPref both guarantees detection of broken promises and preserves privacy. We also present a data structure, called a *modified ternary tree*

Manuscript received March 26, 2014; revised December 24, 2014; accepted January 23, 2015; approved by IEEE/ACM Transactions on Networking Editor Y.-C. Hu. Date of publication March 25, 2015; date of current version April 14, 2016. This work was supported by the National Science Foundation (NSF) grants CNS-1054229, CNS-1065130, CNS-1117052, CNS-1149832, CNS-1218066, and CNS-1445967, and by DARPA contract FA8650-11-C-7189.

M. Zhao, A. J. T. Gurney, A. Haeberlen, and B. T. Loo are with the University of Pennsylvania, Philadelphia, PA 19104 USA.

W. Zhou and M. Sherr are with Georgetown University, Washington, DC 20057 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2409233

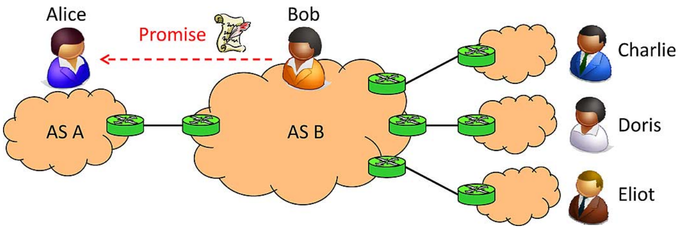

Fig. 1. Motivating scenario. Bob has made a promise to Alice about his routing decisions, but Alice cannot verify whether Bob is keeping his promise.

(MTT), that can be used to run our algorithm efficiently for large numbers of prefixes.

To show that our approach is practical, we present *Secure and Private Inter-Domain Routing (SPIDeR)*, a collaborative verification system that can be deployed as a companion protocol to BGP, possibly on separate hardware, and that makes its decisions based on observing the BGP message flow. We report experimental results to show that SPIDeR's overhead is reasonable. SPIDeR is meant to be a proof of concept: it can verify a nontrivial set of control-plane actions, but it cannot verify the network's performance on the data plane, and it does not cover promises about certain aspects of BGP functionality, such as proxy aggregation.

Although this paper motivates and evaluates collaborative verification in the context of BGP, the VPref algorithm is not specific to BGP, or even to interdomain routing: it could be applied to any scenario where one entity privately chooses between multiple options presented by other entities. Thus, we speculate that VPref could be useful beyond network routing.

In summary, this paper makes the following contributions:

- a novel algorithm, VPref, for collaborative verification of promises about private choices (Section IV);
- a formal proof that VPref guarantees privacy and verifiability (Section IV-F; details in [45]);
- a data structure (MTT) that can be used to efficiently scale VPref (Section V);
- SPIDeR, a companion protocol to BGP that applies VPref to interdomain routing (Section VI); and
- an experimental evaluation of SPIDeR (Section VII).

To provide some context, we begin with a general overview and roadmap (Section II), followed by a definition of 'promises' in the context of BGP (Section III).

II. Overview and Roadmap

Fig. 1 illustrates the problem we are concerned about in this paper. Alice and Bob are operators of two autonomous systems (ASes) A and B, which are connected by a direct link. Alice receives interdomain routes from Bob via BGP [34], and these routes typically do not terminate in Bob's AS but rather traverse one of Bob's neighbors: Charlie, Doris, or Eliot. At any given time, Bob may have several routes to a prefix p; he chooses one of these routes according to his local policy and then decides whether or not to make that route available to Alice. However, Alice is unable to observe which routes (if any) Bob had available at that time.

We are specifically interested in situations where Bob has made some kind of *promise* to Alice regarding his routing policy. A promise is a statement that Bob will prefer certain routes over certain other routes; for instance, Bob could promise that he will always choose the shortest route to p that is available to him, or that he will prefer routes through Charlie's AS over routes through Eliot's. Since Bob's incoming routes are not visible to Alice, Bob can easily break his promise without Alice noticing—for instance, a configuration error might result in Eliot's route being chosen, instead of the preferred route that Charlie had offered. Our goal is to enable Alice to verify whether Bob is keeping his promise, *without* forcing Bob to reveal his incoming routes to Alice. We will refer to the first part as our *verifiability* goal and to the second part as our *privacy* goal.

A. How Much Privacy Do We Need?

Before we can formalize these goals, we need to provide a more specific definition of privacy. A very strong definition could demand that the downstream neighbors of Bob's AS learn *nothing at all* about the routes available to Bob. However, this property seems too strong—certainly much stronger than what BGP offers today. For instance, if Bob announces a route to Alice via BGP, Alice is able to see (from the `AS_PATH` attribute) which upstream neighbor the route traverses, and she can infer that this neighbor had previously exported it to Bob. Since this seems acceptable to the ASes that exist today, we adopt a slightly weaker definition: assuming that BGP is already running and that all participants are correct, operation of the additional protocol should not enable any AS participant to deduce *additional* information about the routing state or policy of any other AS, *beyond what it can already learn via BGP*.

In particular, Alice should not be able to determine which routes were available to Bob at any given time, except for those routes she has already learned from him through BGP; and she should not be able to deduce the relative preference of any two routes, other than as already specified by Bob in his promise. However, if Alice colluded with Charlie, then she could find out which routes he sent to Bob – but she could do this even if our system were not deployed. When some participants break their promises, it is acceptable to reveal new information, e.g., evidence that a promise has been broken.

B. What is a Promise?

We also need to define what it means for an AS to make a promise about its routing policy. Section III contains more detail on this topic, and how our formalism can model existing policy for local preference groups, selective export, etc. The promises of interest to us relate to the routing decision procedure, in which several routes enter and at most one leaves; routing policy determines which route is the winner. A promise does not specify every possible nuance of route selection, but may give partial information about which routes will be considered 'better' than others.

Assume that an AS A, for a prefix p, has a total order over the set $R(A, p)$ of all possible routes from A to p, yielding its preference. During protocol operation, A will choose the best route, according to this order, from all those that are currently available. We envision a promise as dividing $R(A, p)$ into several *indifference classes*, where preferences exist between them, but not within each class. In this way, A could divide $R(A, p)$ into 'routes through customer networks' and 'all other routes', and promise that the customer routes constitute the more preferred class. This is an undertaking that if A ever has both a customer and a non-customer route, then it will choose the customer route; but it promises nothing about what will happen when two customer routes are both possible, or when only non-customer routes are available. In these last two cases, the candidate routes are all within the same class, and so no public preference is specified among them.

In the presentation, we assume that, for each pair of neighboring ASes, only a single promise is in effect for a given prefix. However, real routing policy may legitimately be different at each interconnection point, and the visible routes will differ as well: ASes are not atomic [27]. In Section VIII, we discuss the proper handling of this fact in our system.

C. Goals

We are now ready to define (based on [16]) the four properties we want to provide in SPIDeR:

1) **Verifiability:** If an AS A breaks a promise and all of A's neighbors are correct, then at least one of the neighbors detects this.
2) **Privacy:** No AS can learn information by running SPIDeR that it could not already learn from just running the BGP protocol.
3) **Evidence:** If an AS A detects that another AS B has broken a promise, A can obtain evidence against B that will convince a third party.
4) **Accuracy:** If an AS A is keeping all of its promises and is running SPIDeR correctly, no other AS can obtain valid evidence against A.

The first two properties correspond to the informal goals from the beginning of this section. Properties 3 and 4 are useful when holding ASes accountable for broken promises: evidence ensures that ASes cannot deny genuine violations, and accuracy protects correct ASes from spurious accusations.

Non-Goals. To keep the problem manageable, we set two explicit non-goals for this paper. First, we focus strictly on verifying control-plane behavior; checking data-plane forwarding against control-plane announcements is a separate problem that is discussed, e.g., in [43]. We also do not attempt to handle *all* aspects of BGP operation (such as aggregation [37]), nor of peering agreements (such as cold potato manipulation). The latter can be handled by systems like BorderGuard [9], which is complementary to SPIDeR.

III. Policy and Promises

BGP best-route selection is carried out on the basis of routes' *attributes*, according to the local configuration of each router. The decision procedure is lexicographic, beginning with the *local preference* attribute and proceeding down a chain of tie-breakers as necessary. Pattern-match rules allow attribute values to be modified as routes are propagated. In the case of local preference, the numeric value might be chosen based on the AS neighbor from whom the route was received. Choosing shorter routes (those going through fewer ASes) is the next step, if local preference values were tied.

Route preference can also be influenced by BGP *communities* [3], [5], [38], which are numeric tags attached to route advertisements. For example, routes might get local preference 100 by default, but those with a special community tag might get a preference of 80, making them seem less attractive to the recipient. Other communities can be used to control the export of routes, or to cause AS-number prepending. They can also be attached on export to give additional information (e.g., a route's geographic origin [7]). These capabilities are determined by the pattern matching rules in the router configurations of the recipient AS.

Many lists of supported communities are publicly available, indicating that ASes are often willing to disclose certain aspects of their routing policy, at least at a high level. If an AS reveals its local preference tiers in this way (e.g., the fact that transit routes are the least preferred) then it has committed to only part of the information in its actual route preference order. That is, although the AS will make its routing decisions based on many different route attributes, it has only made a public promise about the first attribute. Outside observers cannot say for sure, concerning two routes in the same tier, which one would actually be preferred. This observation motivates our order-theoretic model of an AS's promises.

A. Promises Formalized

Definition 1: Suppose that $R(A,p)$ is the given total order on all routes that might ever be received by AS A for prefix p. A *promise* on $R(A,p)$ consists of a partial order $(C,\leq)$ and a partition $\biguplus_{c\in C} R_c$ of $R(A,p)$.

The set C is used to label various classes of routes. It is endowed with a partial order, representing the definite preference of one class over another. Every route falls into exactly one of these classes. The semantics of this promise are that each R_c is an *indifference class*, meaning that no preference is stated among the routes within that class. However, if $c < d$ for some c and d in C, then a preference exists between routes in R_c and R_d: if r is in R_c and s is in R_d, then r is stated to be less preferred than s. Although this model is very general, we expect that typical use would not involve many classes, and that the mapping from routes to classes would match contemporary BGP usage. One consequence is that the representation of a promise can be expected to be compact.

We include the *null route* $\perp$ among $R(A,p)$, and it is also represented somewhere in the promise, perhaps but not necessarily in a class of its own. Because $\perp$ is always available, ranking a route lower than $\perp$ means that the route should *never* be propagated by A. This capability allows the handling of promises about routes that should not be exported.

Promises to different neighbors. An AS may make different promises to different neighbors, each consistent with what it is actually doing. For example, suppose that the AS prefers customer routes over peer routes, and in addition prefers one customer over all others. The favored customer could be told that its routes will get the highest preference; other customers might only be told that customer routes are better than peer routes. Both promises can be kept at the same time, although one is more specific than the other.

Promise privacy. Not all neighbors need have access to the full definition of the promise. This is useful for cases where it is important to conceal which promises have been made, on the grounds of policy privacy, A producer of routes will only need to know the definitions of classes into which their routes

Method	ASes
Set local preference	57
Selective export by neighbor group	48
Selective export by specific AS	45
Information about route origin	45

Fig. 2. Summary of BGP community actions supported by 88 ASes [32].

might fall. It does not need to know the partial ordering of those classes, or the definitions of other classes. Therefore, the most they can tell is that the promise makes *some* preference distinction about the known classes. Further obfuscation is possible by splitting classes into mutually indifferent subclasses, to hide the true basis on which decisions are being made.

B. Examples

Next, we present a few examples of AS promises based on current interdomain routing practices. Our examples involve neither many route classes nor complex rules about how routes are assigned to their classes. However, all of the examples represent behavior that neighbors would expect to rely on, but where the AS involved might do something different. Fig. 2 summarizes some supporting evidence that policies and promises of this kind are currently being used in practice, based on documentation provided by AS networks themselves. Note that this does not cover every possible use of communities (for example, to control prepending or blackholing). We briefly describe some example promises, ordered by how frequently they appear in [32]:

Set local preference: Many ASes offer neighboring networks a choice of community values to adjust the local preference of their routes. Of the ASes from Fig. 2, 64% (57 ASes) supported this option by setting their local preferences, with a mode of three tiers and a maximum of twelve.

Selective export: Communities may be used to exclude certain routes from being given to specific neighbors or categories of neighbor [7]. In our model, this can be accomplished by placing the exportable routes in a separate, preferred class to the other routes. If the semantics are that these excluded routes should never be exported (as opposed to merely being de-preferenced) then the null route should be placed, in a class of its own, between the two main classes. In verifying this promise, the original sender of a route can confirm that it was indeed not exported, and the ultimate recipient can be sure that none of the routes that it was entitled to see were falsely excluded. As shown in Fig. 2, selective export is popular; it may be based on naming the specific ASes to be excluded (51%), or may deal with them in groups, such as 'peers in Poland' or 'all transit providers' (54%).

Partial customer or transit relationship: A dual version of the above policy has the recipient of the routes, rather than the sender, asking for a particular export policy. In this 'partial customer' model, certain neighbors are only interested in a subset of the routing table; for example, they might want to receive only routes to destinations in Japan [35]. Correct implementation of these requirements is critical for enabling ASes to balance traffic across a diverse collection of peers [8]. The use of exported communities for geographic information [7] is additional evidence that connected networks are interested in making decisions on a geographic basis. As shown in Fig. 2, such opportunities were provided by 51% (45) of the ASes.

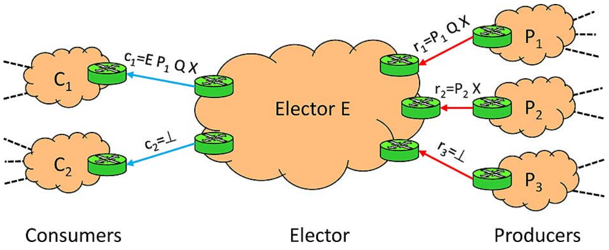

Fig. 3. Simplified system model.

Prefer customer: Following the Gao-Rexford guideline [11], and in accordance with commonly-declared AS practice, a network could promise that all customer routes would be preferred over all non-customer routes. This would yield just two indifference classes. An AS could also differentiate between peer and provider routes, the latter being least preferred of all, for three classes in total.

Path length: For any of these schemes, an AS might additionally want to make a promise about preferring shorter AS-level paths. In this case, each original class would be split: what was the 'peer route' class now becomes 'peer routes of length 2', 'peer routes of length 3', and so on, up to some small maximum. Note that making a promise about path length requires more complete disclosure of the true local preference values being used. In the 'favored customer' scenario, a longer route through that customer will be preferred over a shorter route through a different customer: if the AS has previously promised to deliver the shortest customer route regardless of that customer's identity, then this is a violation of the promise.

IV. The VPref Algorithm

Next, we introduce the VPref algorithm for collaborative verification of promises. The basic algorithm works for routes to a single prefix at a single time point; in Section V, we show how to extend VPref to multiple prefixes and handle route announcements and withdrawals.

A. System Model and Definitions

In the following, we focus on the routing decision of a specific AS, which we will call the *elector* E (Fig. 3). The elector is connected to several upstream ASes P_i, called *producers*, as well as to several downstream ASes C_i, called *consumers*. (These terms, unlike 'provider' or 'customer', are not intended to suggest any particular business relationship among the ASes, but only indicate the flow of routing data.) As in BGP, each producer P_i advertises to the elector a single route r_i, which can be the null route $\perp$. The elector then picks a single route e, which must either be $\perp$ or one of the advertised routes r_i. Finally, the elector advertises to each consumer C_j a route c_j, which must be either $c_j = e$ or $c_j = \perp$. Note that this model can represent import filtering (the elector is free to choose $\perp$ instead of an advertised route) as well as export filtering (the elector can offer $\perp$ to consumers instead of e).

As discussed in Section III-A, we assume that the set of possible routes is divided into k *indifference classes* $R_1, \ldots, R_k$, which are known to all ASes. We further assume that the elector has made a promise to each consumer C_j, which we model as a partial order $\leq_j$ over the classes $R_1, \ldots, R_k$. We say that the elector has *broken his promise* to a consumer C_j iff one of the inputs r_i is in a class R_i that is strictly more preferred than the class R of c_j (that is, $R <_j R_i$): so the elector had a route available that was 'better' than the one he exported to C_j. If the elector has not broken any promise, we say that the elector is *correct.*

B. Assumptions

We make the following seven assumptions:

1) Each AS has access to the same cryptographic hash function H, which is modeled as a random oracle;
2) Each AS i has a private key σ_i and a public key π_i;
3) No AS can forge a digital signature of a correct AS;
4) Signed messages contain timestamps and logical counters to prevent replay attacks;
5) The topology and the public keys are known to all ASes;
6) Each C_i has some representation of the promise $\leq_i$ that is signed by E; and
7) Correct ASes can eventually communicate.

The random oracle model [2] is one of the most popular heuristics in cryptography; the security of virtually all practically deployed public-key encryption and signature schemes rely on it. The next three assumptions can be satisfied, e.g., by using RSA as the signature scheme, and by applying standard security practices. The fifth assumption seems acceptable because the AS-level topology of the Internet can already be inferred [6]; for key distribution, the deployment of RPKI [18] would certainly suffice. The sixth assumption could be satisfied by exchanging a representation of $\leq_i$ out of band, e.g., as part of a peering agreement. The seventh assumption simply states that link failures and other disruptions are eventually repaired.

C. Intuition

Intuitively, VPref works by breaking the complex property that is to be verified into a number of simpler lemmata, such that 1) each lemma can be verified by *some* network, using information it already knows, and 2) in combination, the lemmata imply the original property. A commitment mechanism ensures that all lemmata are verified using the same state. Finally, we add cryptographic signatures to enable networks to prove violations to each other, or to a third party.

VPref has one lemma for each indifference class, which says that the elector has at least one route from that class *or* a more preferred route. When a producer gives the elector a route from some class R_i, it can ask the elector to prove that the corresponding lemma holds for its routing state, but the producer learns nothing from such a proof because it already follows from its own input. When a consumer receives a route from class R_i, it can ask for a proof that the lemmata for all more preferred classes do *not* hold; again, such a proof reveals no additional information because the elector is supposed to output a route from the most preferred class.

D. Commitment Phase

VPref proceeds in two rounds, a mandatory *commitment phase* and an optional *verification phase*. The commitment phase involves the following six steps:

1) Each producer P_i picks his route r_i, signs it with his key σ_{P_i}, and then sends $\sigma_{P_i}(r_i)$ to the elector.
2) The elector responds to each producer with an acknowledgment $\sigma_E(\sigma_{P_i}(r_i))$.
3) The elector chooses e to be either $\perp$ or some r_i. It also chooses k *input bits* $b_1, \ldots, b_k$, one for each R_j, and sets $b_j = 1$ if either $r_i \in R_j$ for some i (i.e., at least one input is from class R_j) or $R_j \leq_i e$ for some i (i.e., R_j is worse than e according to at least one promise). Otherwise it sets $b_j = 0$.
4) The elector chooses k random bitstrings $x_1, \ldots, x_k$ and computes $h := H(H(b_1 \| x_1) \| \ldots \| H(b_k \| x_k))$. The x_i act as salt values to prevent dictionary attacks on h.
5) The elector sends a *commitment* $\sigma_E(h)$ to each producer and each consumer.
6) To each consumer C_j, the elector sends either
   - $\sigma_E(C_j, \perp)$, if $c_j = \perp$, or
   - $\sigma_E(C_j, \sigma_{P_i}(r_i), \sigma_E(r_i))$, if $c_j = r_i$.

If an expected message is not received or the message is not properly signed, the corresponding AS raises an alarm. If no alarm is raised at the end of the protocol, each producer holds a signed proof that the elector has acknowledged his input, each consumer holds a route announced by the elector, and each producer and elector holds an (opaque) commitment $\sigma_E(h)$ to the elector's bits. The acknowledgments and commitments are needed for the optional verification phase.

E. Verification Phase

Any producer or consumer may trigger a verification of the elector's choice by broadcasting a $\text{VERIFY}(\sigma_E(h))$ message to the other producers and consumers. Upon receiving this message, each AS compares $\sigma_E(h)$ to the commitment it has received from the elector. If an AS has received a different commitment $\sigma_E(h')$, the elector has clearly misbehaved, and the AS broadcasts a $\text{INVALID COMMIT}(\sigma_E(h),\ \sigma_E(h'))$ message as proof of the elector's misbehavior. Otherwise the ASes are satisfied that they all have received the same commitment.

Next, each producer or consumer forwards the VERIFY message to the elector, which responds with a number of signed *bit proofs*. A bit proof for the ith bit consists of b_i, x_i, and, for all $j \neq i$, $H(b_j \| x_j)$. The recipient of such a proof can (by recomputing h) verify that the ith bit really did have the value b_i when the commitment was produced. Specifically,

- A consumer C_j that was offered a route c_j earlier will now receive a bit proof for any b_x such that $c_j \leq_j R_x$ and $c_j \notin R_x$ (i.e., the indifference class R_x is preferred over the class containing c_j);
- A producer P_j that sent a route $r_j \neq \perp$ earlier will now receive a bit proof for b_x such that $r_j \in R_x$;
- A producer P_j that sent the empty route $\perp$ earlier will not receive any bit proofs.

When a producer P_i previously provided an input r_i from indifference class R_x but does not receive a valid bit proof for $b_x = 1$, it broadcasts a PROOFCHALLENGE message to the other ASes that contains the acknowledgment $\sigma_E(\sigma_{P_i}(r_i))$ and, if it exists, the invalid bit proof it received from the elector. The other ASes can then use the signed acknowledgment from the

PROOFCHALLENGE message to challenge the elector to produce a valid bit proof for $b_x = 1$. A correct elector would always be able to answer such a challenge for any acknowledgment it has signed, so, if the elector refuses, it effectively admits its own guilt.

Finally, each consumer C_j checks whether it has (a) received all the bit proofs it is due, and whether (b) all the bits are being proven to be 0. If a proof is missing or the bit is being proven to be 1, C_j broadcasts a PROOFCHALLENGE message that contains (i) the message it previously received from the elector in step six, (ii) a signed representation of $\leq_j$, and (iii) any bit proofs received from the elector. Again, any other AS can use this message to challenge the elector to produce the corresponding proofs. Note that a signed representation of $\leq_j$ must exist (Assumption 6) and is included so that the other ASes know which bit proofs to request.

F. Correctness

Next, we briefly sketch our proofs that VPref satisfies all the goals we described in Section II-C. The full proofs are available in [45].

Theorem 1: **Verifiability:** If (1) an AS is faulty during the commitment phase or breaks a promise, (2) both phases of the protocol are executed, and (3) all the neighbors of the faulty AS are correct, then at least one correct neighbor will detect the fault.

Proof sketch: The proof is a case distinction that identifies, for each possible misbehavior, the step that detects it.

A producer is faulty if it provides an invalid signature in its route announcement; such behavior is trivially detected. A correct consumer does not send any messages during the commitment phase, and hence any message received from a consumer immediately reveals that it is faulty. The elector must send identical commitments to all correct ASes, or else an AS will detect it when a VERIFY message is received.

The elector breaks a promise to C_j if there is a route r_q from producer P_i such that $c_j \leq_j r_q$ (where c_j is the elector's choice of route) and c_j and r_q are in different indifference classes. This case is always detected by a producer or consumer: if R_q is the indifference class that contains r_q, then either $b_q = 0$ (P_i discovers that the elector is faulty) or $b_q = 1$ (C_j discovers that the elector broke a promise). This follows from the collision-resistant property of the hash function; i.e., the elector is unable to provide a valid bit proof that inverts any bit used to compute in its previously sent commitment. □

Theorem 2: **Evidence:** If a correct AS X detects that another AS Y was faulty during the commitment phase, then either (1) Y did not send an expected message or did not properly sign a message in which case X raises alarm, or (2) X can convince any correct AS that Y is faulty.

Proof sketch: The proof is a case distinction that lists the possible faults that may be detected in AS Y, and shows that the AS that detects the fault can convince another correct AS that Y is faulty.

Evidence against producers and consumers consist of malformed messages that are signed. These messages serve as *proof of misbehaviors* (PoMs) against the sender; i.e., they can convince any correct AS that the sender is faulty. A correct AS that detects a faulty elector who sends inconsistent commitments can construct a PoM by transmitting an INVALIDCOMMIT message.

A correct producer that detects an invalid commitment (i.e., that $b_z = 0$ where R_z contains its advertised route) can cause other correct ASes to also detect the fault by broadcasting a PROOFCHALLENGE message that contains the elector's signed acknowledgment of the omitted route. Similarly, a correct consumer C_j that detects a broken promise can cause other correct ASes to also detect the violation by sending a PROOFCHALLENGE message that contains the elector's choice from step six along with the encoding of $\leq_j$. In either case, the elector cannot offer a valid bit proof, indicating to the correct AS that it is faulty. □

Theorem 3: **Accuracy:** If an AS X is correct during the commitment phase, then no correct AS will detect a fault in X, and no valid evidence can exist against X.

Proof sketch: The proof is a case distinction of all possible detectable faults and evidence, showing in each case that the fault cannot be detected, and the evidence cannot exist, if X is correct.

A correct AS detects faults (resp. collects evidence) against X if X sends an invalid (resp. and properly signed) message. However, since X is correct, it will send only valid and signed messages. Additionally, a correct AS detects faults against an elector if the elector does not send proper acknowledgments, commitments, or routing choices, or does not provide a valid bit proof. However, a correct elector will always properly acknowledge messages and send a single well-formed commitment to all ASes. From the protocol definition a correct elector maintains sufficient information to construct a bit proof, and therefore can always provide valid bit proofs. Finally, an AS cannot have valid evidence against a correct elector because the elector will not send inconsistent commitments (hence no valid INVALIDCOMMIT messages can exist) or be unable to provide a valid bit proof (hence no PROOFCHALLENGE message will uncover faulty behavior). □

Theorem 4: **Privacy:** If all ASes are correct, then no AS can learn anything from running VPref that it could not already have learned from running BGP.

Proof sketch: When a consumer receives c_j, it cannot use the information revealed in VPref to tell whether E had any other routes at all. Therefore, it cannot deduce unrevealed preferences of E, since it cannot tell whether E even had a choice to make. A producer P can only tell that its route was received: since this happens even when E has a trivial policy to discard every route, P cannot deduce anything about E's policy or state. Finally, E does not receive any data from VPref that it does not already know. □

Theorem 5: **Inconsistent promises:** If an elector makes inconsistent promises, i.e., there exist indifference classes R_i and R_j and consumers C_a and C_b such that $R_i \leq_a R_j$ and $R_j \leq_b R_i$, then there exists a set of inputs such that the elector either must choose $e = \perp$ or break at least one promise.

Proof sketch: The proof considers inputs r_i, r_j that are respectively in classes R_i and R_j. The elector must choose $e \in \{\perp, r_i, r_j\}$. If $e = \perp$, the theorem trivially holds. If $e = r_i$, then the elector breaks its promise to C_a. Otherwise, if $e = r_j$, then the elector breaks its promise to C_b. □

The full proofs are available in an extended technical report [45]. There, we also prove minor variations of the above theorems for cases in which faulty parties collude (i.e., coordinate their efforts). Briefly, if the elector colludes with some of the producers, detection is only guaranteed for violations that would exist for *any* combination of inputs from the colluding producers—if there is any combination that would make the elector's output conform to the promise, the elector can simply ask his confederates to pretend that this is what they provided. The evidence and accuracy theorems remain unchanged.

Our privacy result says that VPref participants learn nothing that they could not have learned from BGP, so long as the elector's promise is not broken, i.e., the algorithm does not leak new information. This holds even if the participants share data: they cannot share any more than they have already obtained from BGP. Colluding parties could manipulate route advertisements in an attempt to discover the elector's hidden preferences, but they do not need VPref to help them.

V. Multi-Prefix VPref

The basic VPref algorithm has two important limitations: it only considers routes to a single prefix, and it only works in a static setting. In this section, we extend VPref to support multiple prefixes as well as route announcements and withdrawals at runtime.

A. Additional Challenges

Adding support for routing changes and multiple prefixes results in two additional challenges that are both related to confidentiality. One way to handle multiple prefixes would be to run a separate instance of VPref for each prefix the elector can reach. However, this would leak some information: If an AS A is invited by its neighbor B to participate in a VPref instance for some prefix p, A can conclude that B has a route to p, even if B never exports such a route to A via BGP. An alternative would be to always run a VPref instance for *every possible* prefix, regardless of whether B ever had a corresponding route, but this would be very expensive: there are $2^{33}-1$ possible prefixes in IPv4! Thus, our first additional challenge is to run multiple parallel VPref instances efficiently, without leaking information. To this end, we introduce a special data structure called a *modified ternary tree (MTT)* that is based on a ternary Merkle hash tree (Section V-B).

Another challenge results from the need to handle route announcements and withdrawals. To guarantee detection of all faults, we must re-verify the elector's routing decision whenever the underlying set of routes changes. However, if the elector simply re-runs VPref whenever a route is announced or withdrawn, it again leaks information: If a neighbor observes an execution of VPref, it can conclude that the elector has received an update from *some* producer, even if the elector has not changed the route it exports via BGP to that particular neighbor. In SPIDeR, we avoid this problem by running VPref *periodically* at short, fixed intervals (Section VI-A). This means that we cannot reliably detect violations whose duration is less than one interval, but in return, the fact that VPref runs at some particular time does not leak any information about the underlying routes.

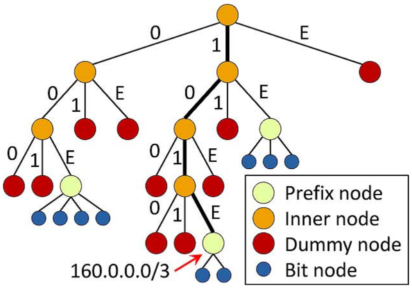

Fig. 4. MTT with three prefixes: `0/2`, `160/3`, and `128/1`.

B. The Modified Ternary Tree

We now describe the modified ternary tree (MTT), which SPIDeR uses for efficient commitments. Intuitively, the leaves of a MTT contain the input bits for separate instances of VPref that each belong to some specific IP prefix; the edges of the tree are labeled in such a way that the labels on each path from the root to a leaf encode the prefix of that particular leaf. Electors can use MTTs to efficiently run multiple instances of VPref in parallel: rather than separately committing to each set of input bits, they can simply commit to the entire tree. Privacy is preserved because, due to the particular structure of the tree, it is possible to construct the bit proofs for each prefix *without* revealing the presence or absence of any other prefix in the tree.

More formally, an MTT is a tree with four types of nodes: *inner nodes*, *prefix nodes*, *bit nodes*, and *dummy nodes*. Each inner node has three children, and we will imagine the edges leading to these children as being labeled 0, 1, and E (for 'end of prefix'). The children at the end of the edges labeled 0 or 1 can be inner nodes, prefix nodes, or dummy nodes; the child at the end of the edge labeled E can be a prefix node or a dummy node. Each prefix node has at least one bit node as its child (but can have more). Bit nodes and dummy nodes do not have any children of their own.

Fig. 4 shows an example MTT in which the highlighted node would be mapped to the prefix `160.0.0.0/3` (`101` in base 2, followed by an E for 'end of prefix'). A full MTT of depth 32 (without dummy nodes) would contain $2^{33}-1$ prefix nodes, one for each possible IPv4 prefix, but, as we will show in Section VII-C, MTTs for realistic routing tables are much smaller. Also, for any set P of prefixes and any function ϵ that maps each prefix to a set of indifference classes, there is a unique minimal MTT $M(P,\epsilon)$ that contains (a) a prefix node $\pi(p)$ for each prefix $p \in P$, and (b) for each element of $\epsilon(p)$, a bit node that is a child of $\pi(p)$.

C. Commitments and Bit Proofs With MTTs

To commit to a given MTT $M(P,\epsilon)$, the elector essentially uses the MTT as a Merkle Hash tree [25]. First, it assigns a label to each leaf node: each dummy node is labeled with a random bitstring, and each bit node is labeled with $H(b_i \,\|\, x_i)$, where b_i is the value of the bit and x_i is a random bitstring. (Random bitstrings must be the same length as a hash value). Then it recursively labels each interior node with the hash $H(l_1 \,\|\, \ldots \,\|\, l_k)$ of the labels l_i of its children. Finally, the commitment is $\sigma_E(h)$, where h is the label of the root node.

A bit proof for a bit b_i then consists of a) the values of b_i and x_i, as well as b) the labels of all direct children of each node on the path from b_i's bit node and the root. As with basic VPref, the recipient of such a proof can verify it by recomputing h from the values in the proof. Crucially, the recipient cannot distinguish between hash values and random bitstrings, and thus has no way of knowing whether a given value in the proof is the label of an inner node or a dummy node. Therefore, a bit proof does not leak any information about the presence or absence of any prefixes in the MTT, other than the specific prefix to which the bit belongs.

An important detail is that the random bitstrings must be replaced for each new commitment. If the bitstrings were reused, neighbors could compare the labels of subtrees (which they can extract from the bit proofs) in consecutive commitments. If a subtree had the same label in two MTTs, this would mean that all the bits in that subtree were almost certainly identical in both commitments, i.e., all of the corresponding routes would have remained the same.

VI. Application to BGP

The MTT-based approach we have described so far is generic and could be applied to different routing protocols. Next, we present SPIDeR, a system that applies our approach to one specific protocol: BGP.

A. System Overview

SPIDeR is designed as a companion protocol to BGP and could run alongside existing BGP equipment. It consists of three components: a *recorder*, a *proof generator*, and a *checker*. The recorder implements the commitment part of VPref: it opens BGP connections to the border routers in its local AS, mirrors their routing state, and then announces the same routes to the recorders in adjacent ASes, but with the appropriate signatures and acknowledgments added. The recorder also maintains a record of all recent messages it has sent or received, and it periodically computes a commitment, which it distributes to recorders in neighboring ASes.

The two other components implement the verification part of VPref. When a verification is triggered for some AS, the proof generator in that AS reconstructs the corresponding MTT from the recorded message trace and then generates a set of bit proofs, which can be distributed to the neighboring ASes. There, they are verified by the checker. If verification fails, the checker in the detecting AS produces evidence that can be distributed to other neighbors, which can validate the evidence with their own checker.

The recorder, proof generator, and checker implement all the SPIDeR-specific functionality, and they need not be run on the routers—they could run on a separate workstation in each AS. The only change that needs to be made to existing routers is a new pair of iBGP/eBGP sessions to the recorder.

B. Signatures and Acknowledgments

When verification is triggered for a past commitment at time T, the proof generator needs to reconstruct its AS's precise routing state at time T; moreover, if some faulty AS reconstructs its state incorrectly, the other ASes need to be able to prove that they (or the faulty AS) were in fact im- or exporting a particular route at time T. For this purpose, the recorders timestamp, sign, and acknowledge each BGP UPDATE, and they keep a record of past updates and acknowledgments they have sent or received.

Specifically, a route announcement sent from an elector E to a consumer C has the form $m := \sigma_E(\text{ ANNOUNCE}, \ t, C, p, \sigma_P(r'), \ \sigma_E(r))$, where t is a timestamp, C is the AS number of the recipient, p is the prefix, r' the underlying route that E itself has imported (if any), and r is the route itself, including all the attributes that are currently in a BGP UPDATE (AS_PATH, communities, etc.). σ_P denotes a signature by the producer who has exported the underlying route r' to E, and σ_E denotes a signature by E. Note that t serves as a nonce, and that both inner signatures are necessary: σ_P proves to C that the route actually exists (and was not fabricated by E); σ_E is necessary so that C can use it as the fifth element when it propagates the route further to its own consumers. A withdrawal has the form $\sigma_E(\text{ WITHDRAW}, t, C, p)$.

When a recorder receives a message m that contains an announcement or a withdrawal, it checks whether m's timestamp is reasonably close to its own clock (within, say, a few seconds); if so, it returns a message $\sigma_r(\text{ ACK}, t, \ C, H(m))$, where $H(m)$ is the hash of message m. If a router fails to acknowledge m after some time $T_{\max}$, even after several retransmissions, the sender raises an alarm, which must be handled outside of the system, e.g., by calling the faulty router's system administrator on the phone. The same process is used when a recorder detects that the signed messages it is receiving from some neighbor's recorder contain different routes than the BGP messages the local routers are receiving via BGP from that neighbor. Thus, when there have been no recent alarms, each non-faulty router must have a valid ACK for any message it has sent more than $T_{\max}$ ago. To limit the number of signatures that are required during message bursts, routers can sign messages in batches, e.g., using a variant of Nagle's algorithm [29].

C. Evidence

Recall from Section IV-E that there are two situations in which an AS must provide something about the elector's past routing state during verification:

- if a producer did not receive a bit proof for a route r it was exporting to the elector, it must prove that the elector had acknowledged r (evidence of import); and
- if a consumer did not receive bit proofs for a route r it had received from the elector, it must prove that the elector had exported r to it (evidence of export).

With periodic commitments, the situation is more complicated than in Section IV-E: announcements and acknowledgments are no longer sufficient proof because they can be withdrawn at a later time. Therefore, we adopt an iterative approach: announcements and acknowledgments are considered initial evidence but can be disproven by other evidence, such as a WITHDRAW message with a higher timestamp. Since clocks are only loosely synchronized, we always use the timestamp of the elector that is being verified, i.e., outgoing announcements and withdrawals are considered effective when they are sent, and incoming ones are effective when they are acknowledged. (A malicious elector cannot change its timestamps by re-signing messages because it cannot forge matching ACKs from other ASes, and because signing multiple ACKs for the same message would constitute a proof of misbehavior.)

Evidence of import: If Alice wants to prove that she was exporting a route r to Bob at time t, Alice presents her ANNOUNCE for r, timestamped $t' < t$, and the matching ACK from Bob. If Alice has made a mistake and is presenting an announcement she has withdrawn again before t, Bob can refute her evidence by presenting Alice's WITHDRAW for r with a timestamp t'' such that $t' < t'' < t$.

Evidence of export: If Alice wants to prove that Bob was exporting a route r *to her* at time t, she presents Bob's ANNOUNCE for r, timestamped $t' < t$. If Alice has made a mistake and Bob has withdrawn r at some time $t'' < t$, Bob can refute her evidence by presenting his own WITHDRAW for r, timestamped $t' < t'' < t$, with Alice's matching ACK.

D. Handling Loose Synchronization

In an ideal world, an AS would update its routing decisions instantly whenever it receives an announcement or a withdrawal. In practice, it can take some time for changes to take effect, e.g., due to propagation delays or due to BGP features such as MRAI or route flap damping [40]. If a commitment is requested shortly after an incoming message, some of the outgoing routes may not yet be consistent, and this could appear to neighbors as if the AS had broken a promise.

To avoid this problem, we allow electors, for the purposes of a commitment at time T, to choose their inputs from a time window $[T - \delta, T]$. For instance, suppose Alice had received route r_1 from Bob at time t_1, but r_1 had been withdrawn at time t_2 and then replaced with r_2 at time t_3 ($T - \delta < t_1 < t_2 < t_3 < T$), then Alice would be free to choose whether she wants her input from Bob to be r_1, $\perp$, or r_2. Alice informs Bob of her choice during verification, so that Bob can know how to check the corresponding bit proofs. Alice may choose different times for different neighbors and different prefixes, as long as they all fall into $[T - \delta, T]$.

Electors could use this freedom to hide promise breaches shorter than δ, as long as *some* combination of inputs from $[T - \delta, T]$ would have satisfied the promise. However, this seems acceptable because it is restricted to periods of instability: when the routes for a given prefix are stable, the elector has no freedom at all!

If the elector is correct, a satisfactory combination of inputs must exist, as long as all delays are shorter than δ. This is because any output must be the result of the BGP decision process, and all inputs to the process must have been valid at some point within the last δ seconds. To find such a combination during verification, the proof generator can simply choose the actual output and, for each other producer, the first input from $[T-\delta, T]$ that would not have been preferred over the actual output. Such an input must exist because otherwise that producer would have offered the elector a better route during the entire window, so the elector's output could not be correct.

E. Logging, Checkpointing, and Replay

The recorder maintains a log of all announcements, withdrawals, and acknowledgments that its AS has sent or received. This log is used to produce evidence in case a verification is triggered. To prevent the log from growing without bounds, verification is limited to commitments that are at most R days old; thus, ASes can safely discard log entries older than R days, since they are no longer needed for verification. We refer to R as the *retention time*, and we expect $R = 365$ to be a typical value.

SPIDeR also uses the log to save space for MTTs. Recall from Section IV-E that electors must provide bit proofs on demand, which requires access to any MTT within the retention time. However, the MTTs are generated from the AS's routing tables at commitment time, and the log contains all the information needed to reconstruct the MTT. This can be done as follows: the recorder maintains a full checkpoint of its routing state at the beginning of the log, and optionally some additional checkpoints at various commitment times. Then, when verification is triggered for a commitment at some time t, the proof generator loads the most recent checkpoint prior to t, and replays all messages with timestamps up to t. This reproduces the same routing state that was used to create the MTT originally. Thus, the recorder can discard the MTTs after each commitment, and the proof generator can reconstruct them on demand.

However, the MTT contains not only the routing state, but also some random bitstrings. In principle, the recorder could store these bitstrings explicitly, but there is an easier way: it can generate the bitstrings using a cryptographically secure pseudorandom number generator (CSPRNG) with a secret seed. The proof generator can thus later reconstruct the bitstrings from the stored seed. As long as a new seed is chosen truly at random for each commitment, other ASes (who do not have knowledge of the seed) cannot predict the generated bitstrings or distinguish them from hash values, which is sufficient to maintain privacy.

F. Verification of Withdrawals

Similar to S-BGP [20], SPIDeR uses signed route announcements to enable the consumers to verify that a received route actually exists in the underlying topology. However, as in S-BGP, the consumer cannot easily verify whether the route *continues* to be valid once it is announced. If the route is withdrawn by the producer and the elector fails to propagate the withdrawal to the consumer, the consumer has no way to detect this. Therefore, SPIDeR allows ASes to trigger an *extended verification* for a given commitment, which works exactly like a normal verification, except that the AS must also re-announce all routes it had announced originally, with timestamps equal to the commitment time. To prevent the re-announcements from being used in place of the original ones, they contain a special RE-ANNOUNCE message type.

Since the elector cannot sign messages on behalf of the producers, the producers must send a set of RE-ANNOUNCE messages to the elector when extended verification is triggered, one for each route they were exporting at the time of the original commitment. (If a producer fails to provide an acceptable RE-ANNOUNCE message, the elector can use evidence of import to demonstrate that the producer is misbehaving.) The elector then selects the RE-ANNOUNCES that correspond to the routes it had originally chosen, and distributes those to any consumers that had originally received the routes. Note that the elector cannot avoid asking for all messages, even though it must discard most of them; if it asked only for RE-ANNOUNCES that corresponded to chosen routes, it would reveal to the producers which routes had been chosen, which would compromise privacy.

G. *Incremental Deployment*

SPIDeR is a local protocol: all interactions occur between the AS that is being verified and its direct neighbors. Thus, SPIDeR does not need global adoption, or a global PKI, to be effective—it merely requires some local collaboration between ISPs, similar to what is done for route debugging today. Moreover, SPIDeR's guarantees strengthen gradually with the size of the deployment: if only a subset of the neighbors of an AS deploy SPIDeR, they can still detect and prove violations of promises that involve inputs and outputs from that subset. Thus, an initial deployment could be as small as three adjacent ASes: one AS that has made some of the promises from Section III-B, and two customers or peers of that AS who would like to verify that the promises are being kept. Once such ‘islands’ of deployment exist, they enable the ASes on their perimeter to easily offer verification to their own neighbors, so the ‘islands’ can grow incrementally.

VII. Evaluation

Next, we report results from an experimental evaluation of SPIDeR. Our goal is to answer two high-level questions: 1) is SPIDeR practical?, and 2) how expensive is SPIDeR?

To provide a baseline for comparisons, we aligned our experiments with those from the NetReview paper [17]. NetReview is a good baseline for SPIDeR because it also verifies promises about interdomain routing policies, and can also be deployed as a companion protocol to BGP. However, NetReview requires ASes to disclose sensitive information, whereas SPIDeR is designed to provide strong privacy guarantees.

A. *Prototype Implementation*

For our experiments, we built a proof-of-concept implementation of SPIDeR, including a recorder, a proof generator, and a checker. For the recorder, we reused some code from NetReview [17], specifically the component for mirroring BGP routing state from existing routers and the component for maintaining a tamper-evident message log with signatures and acknowledgments (but not the code for auditing, which is different in SPIDeR). We added code for the MTT and for generating commitments; the proof generator and checker are written from scratch. Overall, we added or changed 8,012 lines of C++ code.

We chose RSA-1024 signatures and the SHA-512 hash function, but we use only the first 20 bytes of each digest to save space. The CSPRNG is implemented by encrypting sequences of zeroes with RC4, discarding the first 3,072 bytes to mitigate known weaknesses in RC4. Our recorder implementation uses separate threads for handling messages and for generating commitments; this prevents the message handler from blocking while MTTs are being labeled. The number c of commitment threads can be varied to take advantage of multiple cores; when $c > 1$, we break the MTT into subtrees that are each labeled completely by one of the threads.

B. *Methodology and Experimental Setup*

Our goal was to estimate the cost a typical Internet AS would incur by running SPIDeR. Since it was not feasible to replicate the Internet's entire AS topology in our lab, we decided to set up a small, synthetic topology (shown in Fig. 5) using 36 Quagga BGP daemons in 10 ASes. However, as in [17], we injected BGP messages from a RouteViews trace into one of the ASes. Thus, the conditions in our synthetic topology were approximately as if the ASes had been a part of the global Internet: the routing tables contained routes to every reachable IP prefix, and the number and the arrival pattern of the BGP UPDATEs were similar to the conditions at the RouteViews collection point.

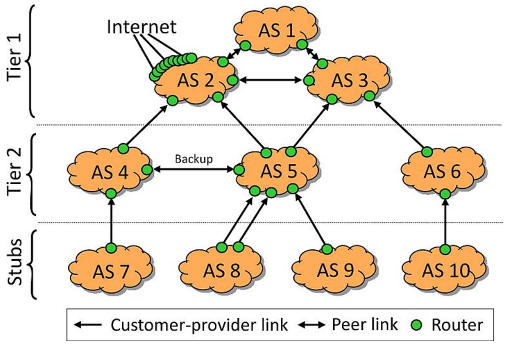

Fig. 5. AS topology for our experiments (from [17]). A RouteViews trace is injected at AS 2.

Specifically, we used a 15-minute RouteViews trace that was collected by a Zebra router at Equinix in Ashburn, VA, on January 18, 2012 at 10 am. This trace contains 38,696 BGP messages, and the corresponding RIB snapshot contains 391,028 distinct IP prefixes. In our experiment, we first populated the routing tables by slowly announcing the prefixes from the snapshot over a period of 30 minutes; then we replayed the 15-minute message trace. We refer to the first phase as the *setup period* and to the second phase as the *replay period*. Unless otherwise specified, we report data collected during the replay period, and we focus on the AS in the middle (AS 5).

Each AS was configured with a simple routing policy based on Gao-Rexford [11], defined 50 indifference classes based on the number of hops, and promised to choose the shortest route to *all* prefixes in the BGP routing table. These simple choices are sufficient for measuring overhead because the cost depends mostly on the size of the MTT and thus on the *number* of prefixes and indifference classes. Recall from Section III that only very few ASes support more than five local-pref classes, and consider that promises could be made for a specific set of prefixes (“I will give you my shortest route to Google”); hence, 50 classes and all prefixes are both conservative choices.

We ran our experiments on a cluster of 11 machines that were connected by a 1 Gb Ethernet network. Each machine had a 2.4 GHz Intel X3220 CPU with four cores and 4 GB RAM, and ran Fedora Core 10 (Linux 2.6.27.41). For the BGP daemon, we used Quagga 0.99.20 with a 100-line patch that enables the daemon to bind to a specific IP. Commitments were generated every 60 seconds. Unless otherwise specified, we used $c = 3$ cores for commitments and the fourth core for message handling.

C. *Microbenchmarks*

We first ran a set of microbenchmarks to measure the size of a typical MTT, and the time for generating and verifying proofs.

MTT size. The MTT from AS 5's last commitment in the experiment contains 22,333,767 nodes, including 389,653 prefix nodes, 950,372 inner nodes, 1,511,092 dummy nodes, and 19,482,650 bit nodes. This is expected because there is one prefix node for each IP prefix that is reachable at that point in the trace, and each prefix node has 50 bit nodes. In total, these nodes required about 137.5 MB of memory.

Labeling time: With $c = 3$ cores, computing the label of this MTT's root node took 13.4 seconds. This seems unproblematic because the computation is done by the recorder (and not by a border router!) and because it is asynchronous, i.e., does not block BGP. Thus, an AS could use our implementation to make a commitment every 15 seconds and catch any promise violations that last at least that long. For comparison, the same computation took 38.8 seconds with only $c = 1$ core, so the speed-up for $c = 3$ is 2.9. This is expected because MTT labeling is highly scalable. Because of this, shorter intervals could be achieved by adding more cores or machines.

Proof generation and proof size: When verification is triggered, the proof generator must reconstruct the MTT at the time of the commitment and then generate a set of bit proofs for each neighbor. For AS 5's last commitment, it took 13.4 s to reconstruct the MTT and 70.2 s to generate the proofs for the five neighbors. The average size of a proof was 449 MB. As a rough approximation, each bit proof with k indifference classes contributes k hashes, or $20 \cdot k$ bytes, plus potentially some hashes of dummy nodes.

For comparison, we also generated the proofs for an alternative promise about just one prefix: "I will give you the shortest route to Google." This took only 0.431 s to generate (after MTT reconstruction), and the corresponding proofs were 2.1 KB for the producers and 2.1 KB for the consumers. If an AS wants to make promises about very many prefixes and proof size is a concern, its neighbors could trigger verification for smaller subtrees, e.g., all prefixes in 32.0.0.0/8.

Proof checking: With $c = 1$ core, verification of a single proof takes 27 s on average; we observed times from 8.6 s to 40 s. As a first step, the checker needs to rebuild the part of the MTT that is included in the proof and re-label it to verify the commitment; this step took 26 s on average. Then, the checker must verify that all the required bits are present and have the appropriate value; this step accounted for the remaining 1 s.

D. Functionality Check

Next, we performed a number of sanity checks on our implementation. First, we ran the experiment to completion and then triggered verification; as expected, no broken promises were reported. Next, we re-ran the experiment, injecting different faults into AS 5 that caused its promise to be violated:

- **Overaggressive filter:** The AS incorrectly filters out a good route from an upstream AS.
- **Wrong export:** A received route is marked as 'not for export' (with a promise where some routes are worse than the null route) but the AS passes it on.
- **Tampered bit proof:** The AS attempts to hide a good route by changing a bit in the bit proof sent to a downstream AS.

After each run, we triggered verification, and in each case the fault was detected by one of the ASes. In the first run, the upstream AS raised an alarm because it did not receive a bit proof for the route it had supplied. In the second run, the downstream AS noticed that it had a bit proof for the null route, which was better than the route it had actually received. In the third run, the downstream AS detected that the proof did not match the hash value from the commitment. These examples complement the proofs in Section IV-F to give us confidence in our implementation.

E. Overhead: Computation

The SPIDeR recorder performs two kinds of operations that are computationally expensive: It signs messages and ACKS, and it generates and labels a MTT for each commitment. To quantify this overhead, we used the `getrusage` system call to measure the computation time the recorder's threads spent overall, and we separately instrumented the code to measure the time spent on generating and verifying signatures and on labeling MTTs. We excluded the first and the last minute to avoid startup/shutdown effects.

We found that, during the replay period, the recorder spent 634.5 s of computation time overall. 9.75 s were spent on generating and verifying 3,913 RSA-1024 signatures; note that this is lower than the total number of BGP updates in the trace because, when updates arrive in bursts, the recorder can batch several of them together and sign the entire batch. Generating 13 MTTs required 519 s. All other operations, e.g., BGP RIB maintenance, account for the remaining 105.75 s. Averaged over the entire 13 minutes, a single X3220 core would have been about 81.3% utilized. Since we reuse its messaging code, NetReview would have incurred exactly the same costs, except for the MTT generation; thus, NetReview's CPU utilization would have been about five times lower.

SPIDeR's computational cost increases with the commitment generation rate, and with the number of routing updates that need to be sent (which in turn depends on the number of neighbors), since there are more messages that need signing. For a small AS with five neighbors, like AS 5, the SPIDeR recorder could easily be run on a single commodity workstation. According to CAIDA's topology data [1], 89% of the current Internet ASes have five or fewer neighbors.

F. Overhead: Bandwidth

SPIDeR increases the amount of interdomain routing traffic because all BGP updates need to be re-announced via SPIDeR with additional signatures and acknowledgments. To quantify this overhead, we used `tcpdump` to capture all BGP packets and all SPIDeR packets that were sent from AS 5 during the replay period. We found that, on average, BGP sent traffic at a rate of 11.8 kbps and SPIDeR at a rate of 32.6 kbps. The relative increase (176%) may seem high, but compared to the amount of traffic ASes commonly handle, 20.8 kbps is not very much—it is about 2% of a single typical DSL upstream.

SPIDeR additionally sends bit proofs to neighboring ASes for verification; the amount depends on the frequency of verifications and the number of commitments checked, which in turn depend on perceived AS needs. Verifying 1% of commitments every minute would result in about 3.0 Mbps of traffic for AS 5.

G. Overhead: Storage

Each SPIDeR recorder requires some local storage for the message log, the information needed to reconstruct past MTTs, and a number of snapshots of its routing state. To quantify the amount of storage needed, we examined the storage of AS 5's recorder after the replay period. We excluded information that was stored during the setup period.

The log contained 2.95 MB of message data, excluding snapshots; a substantial fraction (24.4%) consisted of cryptographic signatures. Thus, it grew at an average rate of about 232.3 kB per minute. Complete snapshots of the routing state were about 94.1 MB. All of this information would be stored by NetReview as well. The only addition in SPIDeR is the MTT-related data, which was comparatively small: each commitment added only 32 bytes to the log. This is because the MTT can be regenerated from the message trace; only the CSPRNG's seed needs to be stored explicitly.

Based on these results, we estimate that an AS could keep a year's worth of logs, including one snapshot per day, in 145.7 GB of storage. This data would easily fit onto a commodity hard drive.

H. Comparison to Zero-Knowledge Sets

A possible alternative approach to our problem would be to apply a more general zero-knowledge proof (ZKP) technique. Following the seminal paper by Goldwasser, Micali, and Rackoff [14], a variety of cryptographic techniques have been developed to allow one node, the *prover*, to convince another node, the *verifier*, that a certain statement is true, without revealing any additional information. The bit proofs in VPref can be seen as special-purpose ZKPs, and our tree construction resembles those used for *zero-knowledge sets (ZKS)* [26]. VPref's bit proofs are not as general as zero-knowledge sets, but they can be constructed efficiently using only hashing, i.e., without modular exponentiation.

To quantify the performance benefits of our approach, we replaced the MTT with ZKS implementations from Catalano *et al.* [4]. We tested three different ZKS implementations: the original construction on a subgroup of residue classes of integers modulo a safe prime (MRK-MOD), the same construction over elliptic curves (MRK-ECC), and trapdoor q-mercurial commitments (qTMC) over the same elliptic curves.

Our ZKS-based system works as follows. Set elements are constructed by concatenating prefixes and the hop numbers; for example, if an AS receives a route in indifference class k for prefix p, it inserts $\langle p|k \rangle$ into its set. To be fair, we use five bits for the prefix length and five bits to support 32 indifference classes; thus, the key length is 42 bits. As in SPIDeR, ASes make regular commitments to their set. When verification is triggered, each AS must prove to its neighbors that 1) its set contains all routes the AS has received from that neighbor, and that 2) its set does not contain any routes that are better than the ones the AS has propagated to that neighbor. Note that the number of proofs of the second type depends on how many better routes are possible. We did not parallelize our ZKS-based implementations, so they only support a single core.

For our experiment, we used AS 5 as an example. We constructed a set with 389,653 prefixes, as above, and we report the time needed to make a commitment, as well as the time for generating and verifying the proofs of existence and non-existence, in Fig. 6. The different ZKS schemes have different strengths, and no single scheme outperforms the others for all five operations. However, even the fastest schemes took hours to generate all the proofs and commitments, and more than a day to verify them, whereas, as we have shown above, SPIDeR can complete these operations in seconds. This holds even if, for a more direct comparison to SPIDeR, we assume $c = 3$ cores and perfect scalability, and thus divide the numbers in Fig. 6 by three.

	MRK-MOD	MRK-ECC	qTMC
Commitments	271.6 min	70.3 min	133.5 min
Generate EP	0.4 min	0.9 min	1.8 min
Verify EP	1,058.7 min	611.7 min	479.6 min
Generate NEP	740.7 min	199.3 min	962.7 min
Verify NEP	2,987.8 min	1,874.3 min	21,500.0 min
EP size	78.6 GB	9.5 GB	6.3 GB
NEP size	250.1 GB	25.7 GB	10.2 GB

Fig. 6. Cost of commitments, existence proofs (EP), and non-existence proofs (NEP) with zero-knowledge sets instead of MTTs. Values shown are for AS 5. Note that we used a single core for this experiment; for a fair comparison to SPIDeR, the runtimes should be divided by $c = 3$.

VIII. Discussion

AS atomicity. In our presentation of SPIDeR we have assumed that an elector has uniform policy towards its consumer, for a given prefix. In reality, policy (and therefore promises) may have geographic differences, and the sets of routes visible at each border router may legitimately vary: ASes are not atomic [27], [35]. Our system, operating at the AS level, treats such equivocation as evidence of misbehavior. In order to recover the ability to make and verify such promises, the protocol must be run not only for each consumer but for each consumer adjacency, so that the promise made to Alice in Europe can be differentiated from the promise made to her in Asia. This capability would also be necessary to support communities relating to specific adjacencies, such as ‘do not advertise this route at LINX’ [7]. If this were done, then additional information would be revealed to producers: we have assumed that the AS-level topology is public knowledge, but now the producers can find out how many interconnections there are between the elector and consumer. Adding extra dummy instances would conceal the true number of connections, at additional cost.

Aggregation. Aggregation of routing announcements is an important concern for network scalability [37]. Most aggregation is performed by the network originating the prefixes, and poses no problem for us. BGP also allows *proxy aggregation*, where some other AS can combine routes to adjacent prefixes in some circumstances. It would be possible to support proxy aggregation in the case of identical AS paths, with additional computational cost (but we do not propose to deal with the rare and deprecated AS_SET attribute [21]). The MTT could include aggregates as well as ordinary prefixes. For example, if p and q were two aggregatable $/24$ prefixes, then their immediate parent node in the tree, a $/23$ prefix, would contain a subtree for verifying promises about the aggregate. To achieve privacy, the elector must construct the $/23$ tree (including a 1 bit for the routes in question) and the producer must check it, whether or not aggregation actually occurred. This is in addition to the requirement to validate the individual $/24$ trees. Without this step, the producer could use the fact that aggregation happened to deduce that both of its routes had been adopted.

The addition of aggregate prefixes into SPIDeR would greatly increase the computational overhead, in terms of storage space, computation time, and volume of traffic. However, since correct SPIDeR operation requires the producer to reason about the possible aggregation, the producer would be better off simply doing the aggregation directly. Ultimately, it should be done by the originator in all cases, removing the need for the larger MTT.

IX. Related Work

BGP security: There has been a great deal of work on BGP route and origin attestation, aimed at preventing prefix hijacking and related problems; example protocols include S-BGP [20], SO-BGP [42] and psBGP [39]. These proposals provide mechanisms for ensuring that an individual route is genuine, but do not aim to validate any aspect of the BGP decision process or of AS policy, beyond this minimum level. The IETF Secure Inter-domain Routing Working Group is engaged in an attempt to strengthen and standardize such proposals, including the provision of a Resource Public Key Infrastructure [18]. Our own system aims to check a complementary set of properties about the outcome of routing decisions, and in a local rather than end-to-end fashion. It could also make use of the RPKI, and be run alongside one of these other systems.

The work [15] by Gupta *et al.* proposes a conceptually centralized routing framework, where routing decisions are calculated using Secure Multi-Party Computation (SMPC), for improved, provable privacy guarantees in interdomain routing. However, the work requires radical changes to BGP's distributed routing framework, and does not aim to validate the consistency between promises and actual routing decisions.

NetReview [17] does allow routing decisions to be checked, but it reveals the entire stream of BGP updates an AS has received from its neighbors, so it is considerably less private than SPIDeR. An earlier variant of VPref, previously published in [16], provided similar guarantees but was only able to support two simple operators, and only in a static setting. This paper substantially generalizes our earlier work, and it also presents a complete system design and an evaluation. BorderGuard [9] verifies a different type of promise, namely whether an ISP is advertising consistent routes at all peering points it shares with a given neighbor. In contrast to the promises we consider here, this can be done using information that is already available to the ISP, so privacy is not an issue.

Verifiable aggregation: In the context of distributed database queries, verifiable aggregation in the presence of adversaries has been considered by Garofalakis *et al.* [12]. Their 'proof sketches' idea allows parties to check if the outcome of a query (such as count, sum, or average) is close to the true value. The technique combines an authentication manifest—which uses cryptographic signatures to prevent input values from being forged—with a one-pass streaming algorithm for approximate aggregation; this is analogous to our combination of S-BGP and the MTT structure. They do not consider privacy of the aggregation function, or the idea of verifying the actual function against a promise.

Collaborative detection: SPIDeR is not the only system in which nodes work together to detect misbehavior; for instance, in Catch [23], nodes collaboratively detect free-riders in a mobile ad-hoc network. However, in Catch, there is no private information to be protected, whereas SPIDeR performs collaborative verification with privacy guarantees. Privacy-preserving detection of *traffic* anomalies is possible in the P3CA system [28], through a version of principal component analysis that employs secure multiparty computation. This is a data-plane counterpart to our work.

X. Conclusion

This paper has shown that interdomain routing systems do not need to make a choice between verifiability and privacy: it is possible to have both. Using our VPref algorithm for collaborative verification, networks can verify a number of nontrivial promises about each others' BGP routing decisions without revealing anything that BGP would not already reveal. The results from our evaluation of SPIDeR show that the costs for the participating networks would be reasonable. VPref is not BGP-specific and could be applied to other routing protocols, or perhaps even to private verification tasks in other domains.

References

[1] *AS Relationships Dataset from CAIDA,*, [Online]. Available: http://www.caida.org/data/active/as-relationships/

[2] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. ACM CCS '93*, Fairfax, VA, USA, 1993.

[3] O. Bonaventure and B. Quoitin, "Common utilizations of the BGP community attribute," *Internet Draft*, 2003 [Online]. Available: http://tools.ietf.org/html/draft-bonaventure-quoitin-bgp-communities-00

[4] D. Catalano, M. Di Raimondo, D. Fiore, and M. Messina, "Zero-knowledge sets with short proofs," *IEEE Trans. Inf. Theory*, vol. 57, no. 4, pp. 2488–2502, Apr. 2011.

[5] E. Chen and T. Bates, "An application of the BGP community attribute in multi-home routing," in *RFC 1998*, Aug. 1996 [Online]. Available: https://tools.ietf.org/html/rfc1998

[6] X. Dimitropoulos *et al.*, "AS relationships: Inference and validation," *ACM SIGCOMM CCR*, no. 1, pp. 29–40, Jan. 2007.

[7] B. Donnet and O. Bonaventure, "On BGP communities," *ACM CCR*, vol. 38, no. 2, pp. 55–59, Apr. 2008.

[8] P. Faratin, D. Clark, P. Gilmore, S. Bauer, A. Berger, and W. Lehr, "Complexity of Internet interconnections: Technology, incentives and implications for policy," presented at the 35th Annu. Telecomm. Policy Research Conf. (TPRC), Arlington, VA, USA, Sep. 2007.

[9] N. Feamster, Z. M. Mao, and J. Rexford, "BorderGuard: Detecting cold potatoes from peers," presented at the 2004 Internet Measurement Conf., IMC '04, Taormina, Sicily, Italy, Oct. 2004.

[10] L. Gao, "On inferring autonomous system relationships in the Internet," *IEEE/ACM Trans. Netw.*, vol. 9, pp. 733–745, Dec. 2001.

[11] L. Gao and J. Rexford, "Stable Internet routing without global coordination," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 681–692, Dec. 2001.

[12] M. Garofalakis, J. Hellerstein, and P. Maniatis, "Proof sketches: Verifiable in-network aggregation," presented at the 23rd Int. Conf. Data Engineering, ICDE 2007, Istanbul, Turkey, Apr. 2007.

[13] S. Goldberg, S. Halevi, A. Jaggard, V. Ramachandran, and R. Wright, "Rationality and traffic attraction: Incentives for honestly announcing paths in BGP," presented at the ACM SIGCOMM 2008, Seattle, WA, USA, Aug. 2008.

[14] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989.

[15] D. Gupta *et al.*, "A new approach to interdomain routing based on secure multi-party computation," presented at the 11th ACM Workshop on Hot Topics in Networks (HotNets-XI), Redmond, WA, USA, Oct. 2012.

[16] A. J. T. Gurney, A. Haeberlen, W. Zhou, M. Sherr, and B. T. Loo, "Having your cake and eating it too: Routing security with privacy protections," presented at the 10th ACM Workshop on Hot Topics in Networks (HotNets-X), Cambridge, MA, USA, Nov. 2011.

[17] A. Haeberlen, I. Avramopoulos, J. Rexford, and P. Druschel, "NetReview: Detecting when interdomain routing goes wrong," presented at the NSDI '09, Boston, MA, USA, 2009.

[18] *IETF Working Group on Secure Inter-Domain Routing*, IETF [Online]. Available: http://tools.ietf.org/wg/sidr

[19] A. J. Kalafut, C. A. Shue, and M. Gupta, "Malicious hubs: Detecting abnormally malicious autonomous systems," presented at the IEEE INFOCOM San Diego, CA, USA, 2010.

[20] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (S-BGP)," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 4, pp. 582–592, 2000.

[21] W. Kumari and S. Kotikalapudi, "Recommendation for not using AS_SET and AS_CONFED_SET in BGP," in *RFC 6472* [Online]. Available: https://tools.ietf.org/html/rfc6472

[22] H. V. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane Nano: Path prediction for peer-to-peer applications," presented at the NSDI 2009, Boston, MA, USA, Apr. 2009.

[23] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Sustaining cooperation in multi-hop wireless networks," presented at the NSDI 2005, Boston, MA, USA, May 2005.

[24] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," presented at the ACM SIGCOMM 2002, Pittsburgh, PA, USA, Sep. 2002.

[25] R. Merkle, "Protocols for public key cryptosystems," in *Proc. IEEE Computer Soc. Symp. Security and Privacy*, Apr. 1980, pp. 122–133.

[26] S. Micali, M. Rabin, and J. Kilian, "Zero-knowledge sets," presented at the FOCS 2003, Cambridge, MA, USA, Oct. 2003.

[27] W. Mühlbauer, A. Feldmann, O. Maennel, M. Roughan, and S. Uhlig, "Building an AS-topology model that captures route diversity," presented at the ACM SIGCOMM, Pisa, Italy, Sep. 2006.

[28] S. Nagaraja, V. Jalaparti, M. Caesar, and N. Borisov, "P3CA: Private anomaly detection across ISP networks," presented at the Privacy Enhancing Technologies Symp. (PETS), Waterloo, Canada, Jul. 2011.

[29] J. Nagle, "Congestion control in IP/TCP internetworks," in *RFC 896*, 1984 [Online]. Available: https://tools.ietf.org/html/rfc896

[30] O. Nordström and C. Dovrolis, "Beware of BGP attacks," *ACM CCR*, vol. 34, no. 2, pp. 1–8, Apr. 2004.

[31] W. B. Norton, "A Study of 28 Peering Policies," Technical Report Dr-Peering International. [Online]. Available: http://drpeering.net/white-papers/Peering-Policies/A-Study-of-28-Peering-Policies.html

[32] BGP Community Guides, One Step Consulting, Inc., 2012 [Online]. Available: http://onesc.net/communities

[33] N. Patrick, T. Scholl, A. Shaikh, and R. Steenbergen, "Peering Dragnet: Anti-social behavior amongst peers, and what you can do about it," in *North American Network Operators' Group (NANOG 38)*, Oct. 2006.

[34] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," in *RFC 4271*, Jan. 2006 [Online]. Available: https://tools.ietf.org/html/rfc4271

[35] M. Roughan, W. Willinger, O. Maennel, D. Perouli, and R. Bush, "10 lessons from 10 years of measuring and modelling the Internet's Autonomous Systems," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1810–1821, 2011.

[36] R. Sherwood, A. Bender, and N. Spring, "DisCarte: A disjunctive Internet cartographer," presented at the ACM SIGCOMM, Seattle, WA, USA, Aug. 2008.

[37] P. Smith, R. Evans, and M. Hughes, "Recommendations on Route Aggregation," RIPE Routing Working Group, Technical Report RIPE-399, 2006.

[38] P. Traina and R. Chandrasekeran, "BGP communities attribute," in *RFC 1997*, Aug. 1996 [Online]. Available: https://tools.ietf.org/html/rfc1997

[39] P. C. van Oorschot, T. Wan, and E. Kranakis, "On interdomain routing security and pretty secure BGP (psBGP)," *ACM Trans. Info. Syst. Secur. (TISSEC)*, vol. 10, no. 3, 2007.

[40] C. Villamizar, R. Chandra, and R. Govindan, "BGP route flap damping," *RFC 2439*, Nov. 1998 [Online]. Available: https://tools.ietf.org/html/rfc2439

[41] F. Wang and L. Gao, "On inferring and characterizing Internet routing policies," in *International Mechanical Code 2003*, Oct. 2003.

[42] R. White, "Securing BGP through Secure Origin BGP," *Internet Protocol J.*, vol. 6, no. 3, pp. 15–22, 2006.

[43] E. L. Wong, P. Balasubramanian, L. Alvisi, M. G. Gouda, and V. Shmatikov, "Truth in advertising: Lightweight verification of route integrity," presented at the PODC 2007, Portland, OR, USA, Aug. 2007.

[44] J. Wu, Z. M. Mao, J. Rexford, and J. Wang, "Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network," presented at the NSDI 2005, Boston, MA, USA, May 2005.

[45] M. Zhao, W. Zhou, A. J. T. Gurney, A. Haeberlen, M. Sherr, and B. T. Loo, "Private and verifiable interdomain routing decisions," Univ. Pennsylvania, Philadelphia, PA, USA, Technical Report MS-CIS-12-10, 2012.

Mingchen Zhao received the B.S. and M.S. degrees from Shanghai Jiao Tong University, China, in 2007 and 2010, respectively. He is pursuing the Ph.D. degree in the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA.

Wenchao Zhou received the B.S.E. degree in computer science and engineering from Tsinghua University, China, in 2006, and the M.S.E. and Ph.D. degrees in computer and information science from the University of Pennsylvania, Philadelphia, PA, USA, in 2009 and 2012, respectively.

He is an Assistant Professor in the Department of Computer Science, Georgetown University, Washington, D.C., USA.

Alexander J. T. Gurney is a distributed systems engineer at Comcast Cable, Inc., and a previous postdoctoral researcher at the University of Pennsylvania, working on network routing theory and related topics in network design and management.

Andreas Haeberlen received the Ph.D. degree in computer science from Rice University, Houston, TX, USA, in 2009.

He is an Assistant Professor in the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA.

Micah Sherr received the B.S.E., M.S.E., and Ph.D. degrees in computer science and engineering and computer and information science from the University of Pennsylvania, Philadelphia, PA, USA, in 2000, 2005, and 2009, respectively.

He is an Assistant Professor in the Department of Computer Science, Georgetown University, Washington, D.C., USA.

Boon Thau Loo received the Ph.D. degree in computer science from the University of California, Berkeley, CA, USA, in 2006.

He is an Associate Professor in the Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA, USA.