Light at the middle of the tunnel: middleboxes for selective disclosure of network monitoring to distrusted parties

Nik Sultana
Cambridge University

Markulf Kohlweiss
Microsoft Research

Andrew W. Moore
Cambridge University

ABSTRACT

Network monitoring is vital to the administration and operation of networks, but it requires privileged access that only highly trusted parties are granted. This severely limits opportunities for external parties, such as service or equipment providers, auditors, or even clients, to measure the health or operation of a network in which they are stakeholders, but do not have access to its internal structure.

In this position paper we propose the use of middleboxes to open up network monitoring to external parties using techniques from privacy-preservation research. This would allow distrusted parties to make more inferences about the network state than currently possible, without learning any precise information about the network or data that crosses it.

Thus the state of the network would be more transparent to external stakeholders, who would be empowered to verify claims made by network operators. Network operators would be able to provide more information about their network without compromising security or privacy.

Keywords

network monitoring, security, privacy, measurement, performance

1. INTRODUCTION

Interconnected networks can be mutually opaque. As long as requests are serviced by the connected network or its end-hosts, one does not have to understand how the network is organised and operated. Datacentre networks are a good example of this: they are open to serve clients, but closed to inspection of their internals. Many services are being virtualised and outsourced to cloud systems managed by third parties such as Amazon, Microsoft and Google. This brings economies of scale to tenants, but it diminishes their control. Should a problem arise in the network, tenants are usually unable to diagnose it directly; instead they must wait for a support technician to diagnose the problem, and the tenant must trust the technician's evaluation. Datacentre operators employ reliability engineers around the clock to ensure that problems are detected and addressed quickly.

We envision a more open environment, where more *stakeholders* in a network get more (albeit limited) visibility of its internals, to be able to verify the claims of the network's operator, and perhaps participate in the diagnosis of problems. Stakeholders include service or equipment providers, auditors, clients, and users of a network.

Implementing this idea requires middlebox support. At the very least, there needs to be an on-path appliance that can interpret and answer queries from interested stakeholders. In this article we consider issues encountered when designing such a system. We explore the threat model, outline the current state of the art in relevant privacy-preserving technology, and discuss how this might be applied to allow external stakeholders to monitor a closed network. Our contributions: formulating the problem and its requirements, outlining existing relevant proposals of solutions, and proposing a more comprehensive solution for this problem.

2. CONTEXT AND MOTIVATION

Many networks have *external distrusted stakeholders*, and we want to enable them to make more measurements of the networks in which they are stakeholders.

For example, (i) a home network could grant limited visibility to an ISP that can help diagnose malfunction or scan for malware [9]; (ii) companies could grant auditors limited powers to ensure, for instance, that a Chinese Wall policy is followed [6]; (iii) network operators could grant a regulator insight on whether "net neutrality" principles are being respected, (iv) sensors placed in a private network can be queried by an external party without leaking information about specific data that crosses the network.

More advanced examples of scenario (iv) could involve telemetry from IoT, healthcare, industrial devices on both traditional and "fog" networks [5], "smart" meters [21], lawful interception, or applying research tools like RIPE Atlas [25] to closed network.

Currently, network operators are, however, reluctant to provide external stakeholders with visibility inside the network, since this risks privacy breach, losing trade secrets to competitors, or aiding an adversary carrying out reconnaissance. Similarly, network operators are wary that opening up their network might make it easier to exploit hidden vulnerabilities in their network [17].

While corporations might do full packet capture, to be used for forensic analysis in case of a breach of security or trust, datacentre operators only report the health of their systems via a simplified status dashboard on the Web, and post updates about outages to blogs or microblogs. In some cases, network operators might allow forms of active monitoring to customers that have service-level agreements (SLAs) [24].

We conjecture that further improving this state of affairs is beneficial to both network operators and their stakeholders. If a stakeholder could verify the claims of a network operator then this is likely to improve stakeholder satisfaction and customer retention. And stakeholders would benefit from increased autonomy and transparency. Operators are more likely to want to be *more* transparent to improve their competitiveness. Network transparency complements trustworthy cloud processing [3] and provenance tracking for data [7]. Ideally, stakeholders could *both* trust and verify a network's operation.

We believe that privacy-preserving technology can help address this transparency problem. Such technology is often intended to protect the privacy of the *users* of a service. Here we use it to protect the privacy of the *service*, as well as that of users, from other users and external parties. Using this technology, the operator could allow stakeholders to make queries about the network state, to learn things that they would not have been able to learn otherwise, without the secrets of the operator, or the network's security, being compromised. We call this *network cryptometry* (§4).

3. REQUIREMENT SPACE

At the very least we need (i) to obtain network readings that we can trust and ascertain the source of the readings, (ii) ensure that the readings are not tampered or otherwise faked in transit, (iii) ensure that the readings are sanitised to not leak sensitive details about the source network.

3.1 Threat model

There are essentially three non-colluding parties at play. For brevity, and consistency with the security literature, we refer to the network operator as the *prover*, and the distrusted external stakeholder as the *verifier*. We call the third party the *croupier*, who is trusted by both parties, and who supplies equipment used by the verifier on the prover's network. This equipment provides a trust anchor for both.[1]

The prover might deviate from its expected behaviour by fooling the stakeholder, by making it seem like the network is performing better than it really is. For instance, it might replace the verifier's query with one that results in a more favourable answer, or it could mirror traffic to a system intended to answer queries quickly, rather than service content *and* answer queries. The verifier on the other hand might want to extract more data from the prover, to learn more about the network's internals than the prover intends to reveal.

We assume that the verifier can authenticate the prover, and thus the network it is making queries about. Furthermore, the verifier is authorised to make queries to the prover. The prover can authenticate the verifier, but not necessarily know its identity. Subsequently rate-limiting or other DoS-reduction measures can be applied. Similarly, the verifier can authenticate and differentiate, but not necessarily know the identities of, middleboxes provided by the croupier.

The prover, verifier, and the middleboxes can communicate over channels that preserves confidentiality and integrity. Thus there is no risk of tampering by, or leaking to, third parties via such channel.

Finally, the network has other stakeholders in addition to the prover and verifier—such as other tenants in a datacentre. We assume that other stakeholders can act maliciously (for instance, by overloading a shared resource to disadvantage the verifier) thus it is in the interests of both the prover and verifier to recognise such behaviour.

3.2 Network visibility and access

The threat model is linked to the degree of visibility and access that the verifier has to the prover's network. We assume that the verifier does not know the network's composition or topology, but could learn about this through its queries.

The prover should have full control over what the verifier can learn, as long as the verifier only learns true measurements.

We assume that the verifier has (limited) control over one or more nodes in the prover's network—such as a VM in a datacentre, workstations on a corporate network, or *customer premises equipment* in a corporate or home setting. This control must be acknowledged and supported by the prover—this is usually established by a contractual arrangement or legal requirement, but as for croupier devices, this could be enforced using hardware.

In our model, the verifier trusts computation done on the nodes over which it has control. Furthermore it trusts the middleboxes (provided by the croupier) that interpret the query and answer it.

[1]Technologies such as Intel's SGX are intended to help deliver this: `https://software.intel.com/en-us/isa-extensions/intel-sgx`

Query		Idealised method	Example of information	
			Disclosed to verifier	**Hidden** from verifier
Reachability		Ping	RTT between monitor and node.	Precise position of the monitor.
Path		Traceroute	Path length.	Addresses of all nodes.
Traffic	summary	SNMP lookup	Value is in a given range.	Precise value.
	detail	Packet capture	A payload satisfied a property.	Exact payload details.
Activity		App.-specific	Time taken for database to reply.	(N/A: under verifier's control.)
Resource or Load		Env.-specific	VM/memory ratio.	Quantity of VMs and memory.

Table 1: Examples of queries that could be made by a verifier, and the prover's choices for disclosure.

3.3 Queries and replies: expressiveness and privacy

What kind of queries should the verifier be allowed to ask? And whose privacy should be preserved in queries and their answers? What should be hidden from whom? The *scope* of queries is largely up to the prover—it owns the network, and has the ultimate authority over it. Our goal here is to ensure that if the prover commits to answering a query then the verifier learns an answer together with evidence for its correctness from the prover (otherwise the verifier learns that the prover is being untruthful, or it has failed to uphold service). For example, the verifier might learn that there is a path of length 3 from its gateway until the verifier's node of interest, and total latency consists of $h_1 + h_2 + h_3$ for the three hops, for specific values of h_i, but would not learn the specific internal address of each h_i.

We must also ensure that the integrity of the answer is preserved. We might optionally require that the answer is confidential from all other parties.

We briefly outline the space of queries that the prover might commit to answering in Table 1. The *idealised method* of making a query is usually very different indeed from the equivalent cryptographically-protected method.

Take *reachability*, for example. The trusted counterpart consists of a *verified ping*, where we need to verify that we are pinging the right machine, and protect against relay or replay attacks. A simple protocol might involve an interactive session of challenge-response computations over nonces. In the next section we describe our proposal for how to implement network queries without disadvantaging network throughput, security or privacy, yet providing the verifier with adequate proof. Furthermore we describe how ping would work in this setting in §4.3.

An *activity* query involves checking an application-level feature, such as the time it takes to execute a specific function, or the value of a specific variable. Since this is assumed to be under the verifier's control, the prover has limited hiding ability. Notwithstanding, we still want to preserve the integrity of the query and its corresponding answer. Activity queries can be used to check if the network operator has swapped network functions on the end-host for cheaper ones—such as a packet filter for an Intrusion Detection System (IDS) [16, 11].

4. NETWORK CRYPTOMETRY

Our design needs to achieve three things: (1) receive and interpret valid queries from verifiers, (2) make the required measurements, (3) provide valid answers without disclosing more information than the prover intends.

We propose a design that reduces network measurement to the lagged accrual of a global audit log [22] of network measurements. The audit log is merged from disparate measurement-making vantage points in the network we call *cryptometros*. Furthermore, we separate between *measuring* the network (and appending to the log) and *querying* the measurement database. This allows the two operations to take place on different timescales, to avoid carrying out expensive privacy-preserving cryptography on the dataplane. Instead, the dataplane involves cryptography that has been shown to scale well and is amenable to hardware implementation [19]. Privacy-preserving querying of the audit log for the result of an earlier *network* query needs to take place after a time window. There has been tremendous recent progress in related research, as described in §4.4.

4.1 Architecture

Verifiers will make queries to the system through a proxy that combines a normal service *request* (e.g., an HTTP GET) with a network *query* into a single packet, and send both to the target network.

Figure 1 outlines our setup. We are inspired by FlowTags [12] to add contextual information to flows; in this case, contextual information consists of queries. A similar idea was used by Naous et al. [19] in their verified network primitive, described in §5. Tags will be ignored by everything other than cryptometros. This is also similar to the *tiny packet programs* approach [15], in which a small number of instructions can be embedded in normal network traffic, to be executed by in-path network elements.

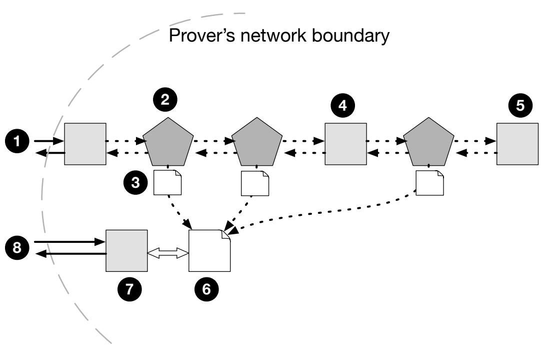

Figure 1: (1) Normal *request* from client is combined with a *network query* at the verifier side, to form a composite that is forwarded to the network. Composites are preserved as they traverse the network. (2) Cryptometros act on composites by (3) logging them, and making the required query on the network, as instructed in the query. (4) Queries are preserved by end-hosts or other middleboxes. (5) End-hosts may include an *answer* to a query, together with their *reply* to a request. Eventually cryptometro logs are gathered into a (6) global audit log, which may be queried via (7) privacy-preserving protocols by (8) external distrusted stakeholders.

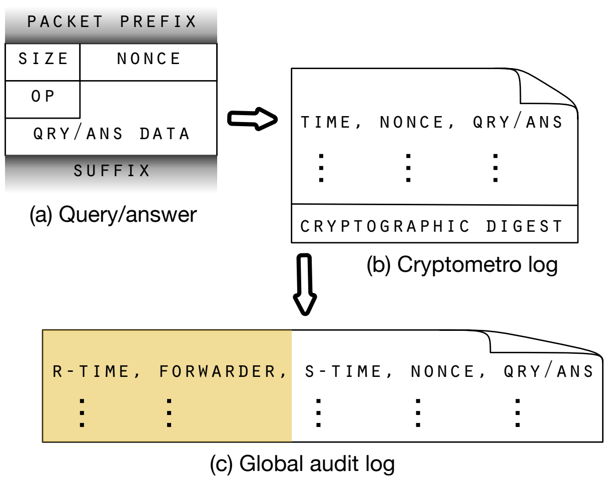

Figure 2: (a) Queries and answers are embedded in packets as described in §4.2. (b) Cryptometro locally log all queries and answers they observe in traffic that traverses them. (c) Eventually all cryptometro logs are collected in a global log, which is then cryptographically queried. Log entries are timestamped. In (c), `R-TIME` is the time the entry was received by the audit log, and `FORWARDER` is a pseudonym for the cryptometro that recorded the entry. `S-TIME` is the time the entry was originally recorded by that cryptometro.

Our queries can be regarded as programs that instruct in-path middleboxes to carry out network monitoring activities, and log both the queries and the requests, as described further later. We call these middleboxes *cryptometros*. In addition to performing normal network function, these middleboxes interpret and transform queries gathered from normal traffic.

Cryptometros serve to attest a trusted internal view of the network state, such that private data is not disclosed outside the network. A network needs to include at least one such appliance, which must be in the path of requests to backends. Placement of cryptometros is chosen by the prover, but more cryptometros (and higher probabilities of replying) will produce higher-resolution answers.

We envision that some tags may be encrypted and that most requests could be tagged with random dummy tags, to provide cover traffic for encrypted tags and thus prevent preferential treatment for tagged requests.

Cryptometros accumulate their observations in local logs. Observations consist of *queries* and *answers*, as described in §4.2. Depending on the nature of the query, a cryptometro might execute it from its vantage point, otherwise it will log the query, and log any answers it observes.

Cryptometro logs are merged to form a global append-only audit log of network events. A publicly accessible cryptographic digest for this log guarantees that past events cannot be tampered with, and that proofs of validity are verified with respect to the same audit log.

Data about queries and their answers is written to this audit log, with sufficient identifying and linking information. For example, a query needs to be linked to the node that answered it. This ensures that a ping query was not changed to a ping to 127.0.0.1 for example. Cryptometros are assigned unique pseudonyms by the croupier (since we do not necessarily have access to the naming used inside the network), which in theory could be fresh for each query. Nodes in which the stakeholder has an interest have fixed pseudonyms however, since we expect to reliably refer to them from outside the network.

A cryptometro needs to have a precise time source [8], since all query replies are stamped before sending them to the log.

A query is answered by the cryptometro or some other node depending on the query (e.g., an activity query might only be answerable by a node). For cryptometro-answerable queries (such as traceroute), as each cryptometro forwards a packet, it might answer a query with some probability. This serves both for scalability, and to provide the operator with further statistical means to hide information about their network.

4.2 Packet and log format

We follow the approach used in FlowTags [12], tagging network traffic with additional data, as shown in Figure 2(a). There the packet *prefix* and *suffix* consists of the network-level encapsulation and service request (e.g., TCP containing an HTTP GET) respectively. *SIZE* consists of a 2-byte field specifying the total size of the tag. *NONCE* is an 8-byte value used once by the verifier, that is used both for freshness (to prevent replay attacks) and to link answers to queries. *OP* is either *QRY* (indicating that the *DATA* body is a query) or *ANS* (answer). Finally, *DATA* contains a specific query or answer, and optionally a signed hash of the entire header for added integrity.

The answer is query-specific. The set of queries is under the croupier's control, while the set of *allowable* queries is under the prover's control. We are compiling an API of queries that could be supported, based on Table 1. Anything that produces observations that can be appended to the cryptometro's log can be queried. We give an example for ping in the next section.

Our approach is similar to TPP [15]. The main differences are: (i) we include nonces, (ii) do not require in-packet memory, (iii) instructions are restricted to specific queries, not low-level operations.

4.3 Example: Ping

Here we elaborate the example outlined in §3.3 using the architecture described in this section. Let qry be the 'ping' network query to a machine externally identified as X. We send this to the prover, encoded as described in the previous section. After waiting for the expiry of the time window required by the prover (for the global audit log to eventually accumulate the cryptometro logs), we query the audit log for a pair of entries (following the tuple format described in §4.2): $(\mathrm{rt}_1, \mathrm{f}_1, \mathrm{st}_1, \mathrm{n}_1, qry)$ and $(\mathrm{rt}_2, \mathrm{f}_2, \mathrm{st}_2, \mathrm{n}_2, \mathrm{ans})$, where qry is our query and n_1 is our nonce. We expect to find that $\mathrm{rt}_1 < \mathrm{rt}_2$ and $\mathrm{st}_1 < \mathrm{st}_2$ (the query occurred before the answer), and $\mathrm{n}_1 = \mathrm{n}_2$ (the answer links to the query via the nonce). Furthermore, $\mathrm{f}_2 = X$ (our intended machine answered the query).

Note that $\mathrm{rt}_1, \mathrm{rt}_2, \mathrm{st}_1, \mathrm{st}_2$ appear as symbols: we do not learn their specific values in this answer. Should the prover allow, we could further learn that, say, these values are bounded by certain constants. For instance, $\mathrm{rt}_2 - \mathrm{rt}_1 < Y$, where Y is some quantity in milliseconds. Thus we might not learn the specific latency, but an upper bound.

4.4 Privacy-preserving measurements

We need cryptographic evidence for standard network measurements (via ICMP, SNMP, etc). Our solution is based on two cryptographic tools.

First, an authenticated cryptographic log of distributed network events that accumulates detailed logs of all participating nodes, gateways, cryptometros, and end-hosts.

Second, a way to prove without leaking prover secrets that a network measurement can be observed in these traces. For this we rely on succinct non-interactive arguments of knowledge (SNARKs). Research on SNARKs has seen amazing progress in recent years [13, 20, 2, 10]. For instance they are already used as a privacy preserving tool for block-chain technology [18, 4]. We propose to produce similar proof systems for a network log by extending research into cryptographic queries to databases [14] and graphs [26], to carry out cryptographic queries over graphs (for SDN), hierarchical structures (for SNMP), and streams (for packet traces).

4.5 Performance

Our design carefully separates dataplane operations carried out by cryptometros, from the privacy-preserving querying of the global audit log. This is designed to separate the two timescales, allowing the first to include as few complex operations as possible. Expensive privacy-preserving cryptography is carried out outside the network's critical service path.

To improve performance further we use rate-limiting (to throttle queries) and sampling (which also helps with privacy-preservation). In the first case, we prevent too many network queries being made in quick succession, to further reduce pressure on the dataplane. In the second case, we might only answer network queries with a given probability, to further reduce load.

5. RELATED WORK

Verifying computations done on an opaque system is not a novel problem, and this is receiving all the more attention in cloud system research. We single out research on the verification of the following qualities. (i) The *result* of computations [20]. This involves providing evidence that a remotely-executed user-defined function was executed faithfully. (ii) That the *resources* used were adequately quantified (and billed) [23]. This research can feed the kinds of measurements made in the "Resource or Load" row of Table 1. (iii) The *forwarding* done by a network. Argyraki et al. [1] propose a method for doing this for externally-visible forwarding by making observations as traffic crosses boundaries between domains. Naous et al. [19] propose a system called ICING to verify that a packet follows each node in a pre-established path. This is intended to enforce a forwarding path, while we are interested in collecting information about how a closed network handles a stakeholder's traffic. (iv) The routing and processing done within *virtual networks*, by Keller et al. [16]. This relies on external visibility into the network in order to make measurements. (v) The functionality, performance and accounting of *outsourced network functions* [11]. These ideas complement those described in this paper, since they assume knowledge of the network topology, while we focus on making privacy-preserving yet trusted observations on the internals of a closed network.

6. CONCLUSION

We described the problem of remotely making measurements on a network over which we have no direct visibility. We call this problem *network cryptometry*. We propose that this problem could be tackled by using on-path logging appliances that feed a global audit log that is queryable in a manner that prevents the leaking of certain details, through proofs of knowledge. This enables distrusted external stakeholders to make measurements on the network without jeopardising the security, privacy, and quality of service of the network or its stakeholders.

Acknowledgements. Partly funded by EPSRC grant EP/K034723/1 on "Networks as a Service". We thank Alastair Beresford, Richard Clegg, Richard Mortier, Marcin Wójcik, Noa Zilberman, and the anonymous reviewers for their feedback.

7. REFERENCES

[1] K. Argyraki, P. Maniatis, et al. Verifiable Network-Performance Measurements. In *CoNEXT*. ACM, 2010.

[2] M. Backes, M. Barbosa, et al. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *Security and Privacy*, pp. 271–286. IEEE, 2015.

[3] J. Bacon, D. Evans, et al. *Middleware 2010*, chap. Enforcing End-to-End Application Security in the Cloud, pp. 293–312. Springer, 2010.

[4] E. Ben Sasson, A. Chiesa, et al. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy*, pp. 459–474. IEEE, 2014.

[5] F. Bonomi, R. Milito, et al. Fog Computing and Its Role in the Internet of Things. MCC '12, pp. 13–16. ACM, 2012.

[6] D. F. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Security and Privacy*, pp. 206–214. IEEE, 1989.

[7] L. Carata, S. Akoush, et al. A primer on provenance. *Commun. ACM*, 57(5):52–60, May 2014.

[8] J. C. Corbett, J. Dean, et al. Spanner: Google's Globally Distributed Database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, Aug. 2013.

[9] M. Costa, J. Crowcroft, et al. Vigilante: End-to-end containment of internet worm epidemics. *ACM Trans. Comput. Syst.*, 26(4):9:1–9:68, Dec. 2008.

[10] C. Costello, C. Fournet, et al. Geppetto: Versatile verifiable computation. In *Security and Privacy*, pp. 253–270. IEEE, 2015.

[11] S. K. Fayazbakhsh, M. K. Reiter, et al. Verifiable network function outsourcing: Requirements, challenges, and roadmap. HotMiddlebox '13, pp. 25–30. ACM, 2013.

[12] S. K. Fayazbakhsh, V. Sekar, et al. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. HotSDN '13, pp. 19–24. ACM, 2013.

[13] C. Fournet, M. Kohlweiss, et al. ZQL: A Compiler for Privacy-Preserving Data Processing. In *USENIX Security*, pp. 163–178. Citeseer, 2013.

[14] M. Fredrikson and B. Livshits. ZØ: An Optimizing Distributing Zero-knowledge Compiler. In *USENIX Security Symposium*, pp. 909–924. 2014.

[15] V. Jeyakumar, M. Alizadeh, et al. Tiny packet programs for low-latency network control and monitoring. HotNets-XII, pp. 8:1–8:7. ACM, 2013.

[16] E. Keller, R. B. Lee, et al. Accountability in Hosted Virtual Networks. VISA '09, pp. 29–36. ACM, 2009.

[17] M. Lennon. Cisco Reviewing Code After Juniper Backdoor Hack. Securityweek.com, Dec 2015.

[18] I. Miers, C. Garman, et al. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy*, pp. 397–411. IEEE, 2013.

[19] J. Naous, M. Walfish, et al. Verifying and Enforcing Network Paths with Icing. CoNEXT '11, pp. 30:1–30:12. ACM, 2011.

[20] B. Parno, J. Howell, et al. Pinocchio: Nearly Practical Verifiable Computation. In *Security and Privacy*, pp. 238–252. IEEE, 2013.

[21] A. Rial and G. Danezis. Privacy-preserving smart metering. WPES '11, pp. 49–60. ACM, 2011.

[22] B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2):159–176, May 1999.

[23] V. Sekar and P. Maniatis. Verifiable resource accounting for cloud computing services. CCSW '11, pp. 21–26. ACM, 2011.

[24] J. Sommers, P. Barford, et al. Accurate and Efficient SLA Compliance Monitoring. *SIGCOMM Comput. Commun. Rev.*, 37(4):109–120, Aug. 2007.

[25] R. N. Staff. RIPE Atlas. *The Internet Protocol Journal*, 18(3):2–26, Sept 2015.

[26] Y. Zhang, C. Papamanthou, et al. ALITHEIA: towards practical verifiable graph processing. In G. Ahn, M. Yung, et al., eds., *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pp. 856–867. ACM, 2014.