Research Statement

Riad S. Wahby

The way we build computer systems leaves them—and us—vulnerable to complete disaster. Each layer of our technology stack—algorithm, implementation, OS, CPU, datacenter operator, etc.—implicitly trusts the ones below: your CPU's manufacturer, for example, has total control of your machine. More insidiously, today's chip design software can inject tomorrow's hardware back-door, because a system implicitly trusts the tools that were used to build it [29]. Over time, the growing chains of implicit trust connecting layer to layer and generation to generation will give attackers chance after chance to break security—and since we need the old computers to build the new ones, a sufficiently corrupted technology stack is nearly unrecoverable.

Breaking the chains of implicit trust requires a dramatic shift in the way we build systems. Fortunately, theoretical computer science gives us hope of securing the technology stack in spite of corrupted layers: powerful tools to control "a herd of supercomputers working with . . . unreliable software and untested hardware" [6]. Yet, these tools are often too costly or too difficult to use, or they entail unrealistic assumptions. Distilling hope into progress requires closing these gaps: finding correspondences between theory and practice, identifying opportunities to slash concrete costs, and leveraging domain knowledge that bridges layers of the stack.

Accordingly, my work attacks implicit trust by taking a holistic view of the technology stack, scouring the connections between layers to excise trust and extract performance, and building, measuring, and deploying real systems. Giraffe [34], for example, defends against malicious chips, first, by devising new theory that reduces asymptotic and concrete costs while exploiting hardware's natural parallelism, and second, by innovating in systems and hardware to automatically generate circuits tailored to both the algorithms and the available chip area. As another example, my work on fast hashing to elliptic curves [31] addresses implementation, security, and performance issues using tricks from number theory and algebraic geometry; this work is being widely deployed, including in IETF standards that I am co-authoring [9, 17].

The systems I've built to tackle implicit trust relationships across the technology stack have yielded powerful and sometimes unexpected insights. As examples: defending operating systems against malicious peripherals [4] showed me that systems often inherit extra, unnecessary trust assumptions from their creators; reading the *Hacker News* discussion about my work on protecting the privacy of cryptocurrency users [1, 32] was a clear lesson on the power of human (mis)trust in technology; and creating the first hardware implementations of general-purpose proof systems [33] allowed me to explore issues of supply-chain security that I first encountered during my ten years designing analog and mixed-signal integrated circuits. In that role, I led engineering teams responsible for writing a dozen patents and building products that yielded hundreds of millions of revenue dollars—and I gained first-hand experience with the dangers of implicit trust pervasive in the technology stack.

My work up to now attacks implicit trust in three areas: probabilistic proofs, applied cryptography, and systems. After describing each, I briefly discuss future directions.

■ Probabilistic proofs

If we can harness untrusted infrastructure to produce trustworthy outputs from security-critical computations, we can cleave entire generations of technology from the chain of trust. To that end, the theory of probabilistic proofs has produced a cluster of landmark results—IP=PSPACE, MIP=NEXP, the PCP theorem, and the theory of zero knowledge—that allow a powerful but untrusted entity—the *prover*—to convince another, weaker entity—the *verifier*—that a statement is true. The last ten years have seen an explosion of built systems that apply these results to practical problems in two broad categories: *verifiable outsourcing*, in which the prover runs a program and proves it did so correctly; and *zero-knowledge proofs*, in which the prover convinces the verifier that it knows a secret satisfying some predicate, without revealing the secret.

These systems implement a pipeline that transforms a description of the program or predicate of interest into executables implementing a prover and verifier. Conceptually, this transformation happens in two stages: the *front-end* compiles the program to a circuit-like representation (e.g., an arithmetic circuit or a system of arithmetic constraints) whose satisfaction is tantamount to correct execution of the program. The *back-end* is a proof protocol that guarantees (with high probability) that an accepting proof establishes satisfaction of the circuit or constraints, and therefore that the prover faithfully executed the program.

Unfortunately, this high-level description conceals enormous concrete costs at each step. Many back-ends entail one or more public-key cryptographic operations per gate or constraint in the circuit—so large circuits mean high proving cost. Meanwhile, front-ends often transform even modest-sized programs into large circuits, because they must encode every possible execution path: both branches of all conditionals must be expanded, and likewise all loop iterations must be unrolled. Moreover, expressions that a CPU might evaluate in one cycle—say, a 64-bit XOR—are much more costly when expressed as an arithmetic circuit. All told, the prover's overhead is four to eight orders of magnitude or more compared to the cost of executing the original program.

My work brings probabilistic proofs to bear on real-world problems of trust in two ways. First, it expands proofs' general applicability, slashing costs and eliminating assumptions via theoretical and practical improvements in the back-end, and cryptographic and compilers techniques in the front-end. Second, it devises new applications by identifying settings where it is possible to reduce proofs' high costs, and where the remaining costs are justified.

Defanging malicious chips. Zebra [33] and Giraffe [34] tackle the dangers posed by malicious chip manufacturers. Building an integrated circuit factory (a *fab*) is expensive, so many chip design companies outsource manufacture to a third party (e.g., TSMC or SMIC). A large body of research shows that these third parties can add extremely hard-to-detect back-doors into chips (e.g., [7, 40]; for a survey, see [28]). One response is to arrange stringent fab security measures [2]; another is simply to purchase a fab and abandon outsourced fabrication altogether. Unfortunately, because of the economics of semiconductor manufacturing, chips built in a trusted facility are more expensive while likely suffering orders of magnitude worse performance.

Our key insight in Zebra is that we can use verifiable outsourcing to defeat malicious chips by taking advantage of the cost- and performance-scaling characteristics of CMOS manufacturing; we call this model *Verifiable ASICs*. In this arrangement, a designer creates *two* chips, one that is trustworthy but low performance, and one that is untrusted but cutting edge. The trustworthy chip outsources work to the much faster untrusted chip, which provides the result plus a proof of correctness. All told, Zebra allows a trustworthy but slow chip to reduce per-computation energy costs by $3\times$ using an untrusted but fast co-processor; it addresses hardware back-doors under a strong threat model; and it is the first implementation of a general-purpose proof system in hardware. Zebra received a Distinguished Student Paper award at IEEE S&P 2016.

Giraffe takes Verifiable ASICs a few steps closer to practicality. Compared to Zebra, Giraffe scales to dramatically larger problem sizes, reduces both prover and verifier costs, and enables a greater degree of design automation. In addition, Giraffe supposes a more realistic operating model: in contrast to Zebra, which requires a trusted party to run a setup whose cost must be amortized over many computations, Giraffe requires no setup and no trusted party.

These improvements required several innovations. First, we implement a *design template* that automatically generates realistic and highly efficient hardware designs across a wide parameter range; this required us to devise a new digital circuit tailored to the proof protocol that makes it easy to precisely control parallelism. Second, we design a new interactive proof protocol for data-parallel, layered arithmetic circuits with asymptotically optimal and concretely favorable prover cost. Third, we describe algorithmic improvements that reduce verifier cost by $3\times$ and apply to all systems in the line of work building upon GKR [20] and CMT [16]. Finally, we apply program analysis techniques to automatically transform programs into representations that are amenable to outsourcing. Giraffe scales to $500\times$ larger computations than Zebra and—for the first time in the literature—allows a verifier to save work despite paying for *all* costs.

Improving expressiveness and cutting costs in the front-end. Compiling programs into circuit-like representations is challenging because natural program constructs can become extremely expensive when written as circuits. Front-end improvements are therefore crucial to expand the types of computations to which we can apply probabilistic proofs.

Buffet [35] dramatically reduces front-end costs for programs that use RAM or complex control flow, marrying two lines of prior work. In Pantry [10], straight-line programs are cheap, but RAM is expensive. Control flow with nested loops is extremely costly, because inner loops must be unrolled into outer loops, which must themselves be unrolled. In contrast, BCTV [8] translates programs into instructions for a simple CPU, which it simulates in constraints. This approach has good asymptotics for any program, but simulating a CPU is concretely expensive.

Buffet leverages techniques from parallelizing compilers to get nearly the best of both worlds. Buffet adapts BCTV's efficient RAM primitive, and it automatically translates a rich subset of C into a program-specific state machine, which it compiles to constraints using techniques from Pantry. Since this state machine does not use nested loops, it avoids Pantry's high cost for complex control flow; since it encodes a specific program's logic rather than a general-purpose CPU's, it sidesteps BCTV's high per-simulated-cycle cost. Buffet's techniques reduce costs versus Pantry and BCTV by one to four orders of magnitude, depending on the program.

Pantry's RAM primitive, unlike BCTV's, allows state to be transferred from one proof to another using a Merkle tree. The cost of each access, however, scales logarithmically with the total state size. In recent work [25], we instantiate a different cryptographic primitive, an RSA accumulator [12, 22], which yields lower costs in settings where a proof interacts with state many times. This requires several new technical tools, including a method of hashing to prime numbers that is optimized for the proofs setting. This primitive can be used to verifiably outsource state updates, saving cost compared to Pantry across a range of parameters.

Reducing cost and relaxing assumptions in the back-end. Hyrax [36] constructs a new zero-knowledge proof back-end that builds on the theoretical innovations from Giraffe while addressing practical issues of trust inherent in many prior systems: first, several lines of back-end work specify a setup phase that must be executed by a trusted party; if this phase is executed incorrectly, a cheating prover can "prove" false statements. Second, many back-ends rely on strong, nonstandard cryptographic assumptions for security.

Hyrax needs no trusted setup and builds on standard cryptographic assumptions while achieving a good balance among prover cost, verifier cost, and proof length. To achieve this, we devised a new, concretely efficient polynomial commitment scheme, and designed and optimized a protocol that transforms the Giraffe interactive proof into a zero-knowledge proof. Hyrax is secure under the discrete log assumption, and can be made non-interactive in the random oracle model. Careful evaluation against five other systems with similar goals shows that it strikes a pragmatic balance among competing costs. It has also inspired several follow-up works, and its polynomial commitment's proving costs are among the concretely cheapest known [27].

■ Applied cryptography

Applied cryptography fills in the crucial details that help condense theoretical cryptography into practice—details that can so easily go wrong. My work applies a blend of rigor and pragmatism to problems of security and performance, and is being deployed to secure real-world systems.

Hashing to elliptic curves [31]. Many useful cryptographic tools—BLS signatures, VRFs, PAKEs, and IBE, to name a few—require a cryptographic hash function whose codomain is the group of points on an elliptic curve. A large body of research focuses on constructing suitable hash functions, because naïve methods lead to real-world attacks [30]. In spite of this, until our work, the best hash function to many important elliptic curve families had the unfortunate property that fast implementations were complex, while simple implementations were slow.

In this work, we use tricks from number theory and algebraic geometry to generalize and optimize the *SSWU map* [11], yielding a hash function that applies to essentially any elliptic curve; admits fast, simple, side-channel–free implementations; and compares in speed to the fastest prior work, which applies only to a much smaller class of elliptic curves.

Our work is being deployed as part of the recent adoption of BLS signatures by several blockchain-related projects. In addition, it is being standardized by the IETF as part of a new document I am co-authoring on hashing to elliptic curves [17]. This in turn is being used by several other draft standards, including the one for BLS signatures [9], which I am also co-authoring. Ushering research from the academy to the broader community is valuable in part because it brings practical problems into sharp relief: even small details like choosing a branch of the square root can have big effects on implementation complexity—and thus on security.

Private airdrops [32] protect user privacy with efficient, purpose-built zero-knowledge proofs.

An *airdrop* is a way to give away free money; applications include streamlining direct stimulus payments and, somewhat more commonly, attracting new cryptocurrency users. Airdrops reach the widest audience when they don't require explicit opt-in, so potential recipients are often identified by harvesting public keys from public key infrastructures like Bitcoin. Unfortunately, existing airdrop mechanisms reveal which public keys have claimed money—so airdrops that use harvested public keys allow anyone to learn the off-chain identities of claimants.

To address this problem, we define a new cryptographic primitive, a *private airdrop*, that allows recipients to claim funds without revealing their identities; we give practical constructions for RSA- and Diffie-Hellman–based public keys. We also design a *private genesis airdrop* that enables efficient private airdrops to millions of recipients at once. Our main technical tool is a new special-purpose zero-knowledge proof of knowledge of factorization of an RSA modulus, which is far cheaper than prior work but requires a stronger cryptographic assumption. I helped deploy this work as part of the Handshake airdrop [1], which targeted more than 200,000 users.

■ Systems

Cinch [4] tackles the problem of defending an operating system against malicious peripherals, which are a serious and growing threat: they can, for example, exploit bugs in kernel code, or impersonate or snoop on honest devices. Cinch starts from the insight that peripherals have much in common with remote, potentially malicious hosts on a network—meaning that we can apply decades of experience securing hosts against network adversaries to protect operating systems against malicious devices. To do so, Cinch leverages virtualization hardware to isolate USB devices and borrows networking concepts to detect and defeat malicious behavior.

Embedded RNGs [21]. Trustworthy randomness is crucial for cryptography, but embedded systems make its generation difficult. Some vendors provide a hardware random number generator (RNG), but give no way to audit its design or operation. This work describes a RNG circuit that is easy to audit and cheap to build using simple parts available from many different vendors.

ExCamera [19] is one of the earliest works to observe that "serverless" infrastructure like AWS Lambda can be harnessed to run massively parallel computations. ExCamera proposes a new architecture for video encoding that enables fine-grained parallelism by avoiding the serialization bottlenecks inherent in existing approaches. To deploy ExCamera's massively parallel video encoder, I developed the first general-purpose framework for running massively parallel computations on platforms like AWS Lambda. This open-source framework, called *mu* [23], was the basis for further research on serverless architectures [5]; it will also inform my future work leveraging powerful but untrusted infrastructure.

Other work. I have worked in a variety of supporting roles on several other projects, including: balancing the security of Internet of Things devices against device owners' right to see what their devices report to manufacturers [37]; coupling video encoding and network congestion control to improve video streaming in spite of network failures [18]; measuring and emulating real-world Internet links (Best Paper, USENIX ATC) [39]; and designing a domain-specific programming language for writing cryptographic code that is resistant to timing side channels [14].

■ Future work

Foiling malicious hardware. The correspondences between hardware and probabilistic proofs go well beyond the ones that Zebra and Giraffe exploit, and they have enormous practical potential. For example, multi-prover interactive proofs (MIPs) can in principle reduce verification cost exponentially—and, perhaps surprisingly, the multi-prover model seems well-suited to hardware: MIP soundness requires that the provers do not communicate, which might be enforced simply by powering down one prover at a time! Bringing Verifiable ASICs to practice will also take a blend of circuit design and proofs experience, and I am uniquely well-prepared for this task.

Making proof systems more practical. In the past decade, proof systems have improved rapidly. Sustaining that record will require new theory, new systems work, and new front-ends.

Recent advances in proof systems show that polynomial commitment schemes are almost always the bottleneck. Designing new schemes that reduce costs and remove cryptographic and trust assumptions is therefore a path to improving almost all back-ends.

Distributed systems show promise for increasing reachable problem sizes [38], but an important open question is how to efficiently handle Byzantine (or even just random) worker failures. This presents an opportunity to find improvements in the connections between layers. As a first attempt: since the computation's output is a proof that can easily be checked for correctness, we might proceed optimistically and ignore errors. But this is wasteful: it cannot detect intermediate errors, so recovering from one requires generating a full proof at least twice. Instead, we should create new proof protocols that are designed to be distributed, allowing workers to detect and correct intermediate failures locally and with low communication overhead.

Finally, to reduce front-end costs and handle complex applications, we will need compilers and analysis techniques tailored to proofs. I am working with Fraser Brown and Alex Ozdemir on a new approach for compiling to circuit-like representations [24], which will serve as a springboard for developing and applying program analysis techniques that dramatically expand front-end efficiency and expressiveness, and therefore the practical applications of probabilistic proofs.

Connecting secure compilation with cryptography. Bringing together disparate research areas creates new opportunities to bridge layers of the technology stack. *Secure compilation* studies the properties of programs and compilers, for example, asking how to compile programs while preserving source-level properties—even when the resulting executable runs in an adversarial environment. Recently, my collaborators and I began exploring connections [26] between new results in secure compilation [3] and Universal Composability (UC) [13], a framework for specifying and analyzing cryptographic protocols. We believe this connection will let us synthesize results and tools from the two disciplines, say, to make writing security proofs easier.

In a separate effort [15], several other collaborators and I are working on a new domain-specific language (DSL) for writing high-level, high-performance cryptographic code. One of this project's long-term goals is to exploit the connection discussed above in order to scale up formal verification of cryptographic implementations: first, the DSL will capture the UC notion of ideal functionalities in its type system, connecting implementation to specification via mechanical proofs. Second, we will exploit the connection between UC and secure compilation to guarantee that the DSL's compiler securely composes verified implementations.

⁂

As the scale of security threats escalates—yesterday, Equifax; today, the US Government—the world is turning to computer science for help. And although we have made great progress, implicit trust remains a serious blind spot: today, we have no choice but to trust our CPUs' manufacturers. Theory gives us that choice asymptotically, and the last two decades have seen twenty orders of magnitude improvement in concrete costs. My work has helped carry this theory to the threshold of practice, and the next order of magnitude will be the most exciting yet: it will let us build trustworthy hardware from untrusted parts, design systems that make trust explicit and narrow, and, ultimately, secure our technology stack—and the ones that follow.

References

[1] Handshake airdrop. https://github.com/handshake-org/hs-airdrop.

[2] Trusted foundry program. http://www.dmea.osd.mil/trustedic.html.

[3] Carmine Abate, Roberto Blanco, Deepak Garg, Catalin Hritcu, Marco Patrignani, and Jeremy Thibault. Journey beyond full abstraction: Exploring robust property preservation for secure compilation. *CSF*, July 2018.

[4] Sebastian Angel, Riad S. Wahby, Max Howald, Joshua B. Leners, Michael Spilo, Zhen Sun, Andrew J. Blumberg, and Michael Walfish. Defending against malicious peripherals with Cinch. In *USENIX Security*, 2016.

[5] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. Sprocket: A serverless video processing framework for the cloud. In *SOCC*, 2018.

[6] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, May 1991.

[7] Georg T. Becker, Francesco Regazzoni, Christof Paar, and Wayne P. Burleson. Stealthy dopant-level hardware Trojans. In *CHES*, 2013.

[8] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security*, 2014.

[9] Dan Boneh, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, and Zhenfei Zhang. BLS signatures. IETF CFRG Internet-Draft, 2020. https://datatracker.ietf.org/doc/draft-irtf-cfrg-bls-signature/.

[10] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *SOSP*, 2013.

[11] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In *CRYPTO*, 2010.

[12] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, 2002.

[13] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[14] Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad S. Wahby, John Renner, Benjamin Grégoire, Gilles Barthe, Ranjit Jhala, and Deian Stefan. FaCT: a DSL for timing-sensitive computation. In *PLDI*, 2019.

[15] Jonathan Cogan, Fraser Brown, Alex Ozdemir, and Riad S. Wahby. High-level high-speed high-assurance crypto. In *PriSC*, 2021.

[16] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, 2012.

[17] Armando Faz-Hernández, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher Wood. Hashing to elliptic curves. IETF CFRG Internet-Draft, 2020. https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/.

[18] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *NSDI*, 2018.

[19] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zheng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In *NSDI*, 2017.

[20] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for Muggles. In *STOC*, 2008. Full version: *J. ACM*, 62(4), Aug. 2015.

[21] Ben Lampert, Riad S. Wahby, Shane Leonard, and Philip Levis. Robust, low-cost, auditable random number generation for embedded system security. In *SenSys*, 2016.

[22] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In *ACNS*, 2007.

[23] mu: A framework to run general-purpose parallel computations on aws lambda. `https://github.com/excamera/mu`.

[24] Alex Ozdemir, Fraser Brown, and Riad S. Wahby. CirC: Infrastructure for constraint compilers. In *Submission*, 2020.

[25] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. Scaling verifiable computation using efficient set accumulators. In *USENIX Security*, 2020.

[26] Marco Patrignani, Riad S. Wahby, , and Robert Künnemann. Universal composability is secure compilation. In *PriSC*, 2020.

[27] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO*, 2020.

[28] M. Tehranipoor and F. Koushanfar. A survey of hardware Trojan taxonomy and detection. *IEEE DT*, 27(1):10–25, January 2010.

[29] Ken Thompson. Reflections on trusting trust. *Communications of the ACM*, 27(8), August 1984. Turing Award Lecture.

[30] Mathy Vanhoef and Eyal Ronen. Dragonblood: Analyzing the dragonfly handshake of WPA3 and EAP-pwd. In *IEEE S&P*, 2020.

[31] Riad S. Wahby and Dan Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR Trans. CHES*, 2019(4):154–179, 2019. Full version: `https://eprint.iacr.org/2019/403`.

[32] Riad S. Wahby, Dan Boneh, Christopher Jeffrey, and Joseph Poon. An airdrop that preserves recipient privacy. In *Financial Crypto*, 2020. Full version: `https://eprint.iacr.org/2020/676`.

[33] Riad S. Wahby, Max Howald, Siddharth Garg, abhi shelat, and Michael Walfish. Verifiable ASICs. In *IEEE S&P*, 2016.

[34] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, abhi shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *ACM CCS*, 2017. Full version: `https://eprint.iacr.org/2017/242/`.

[35] Riad S. Wahby, Srinath Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS*, 2015. Full version: `https://eprint.iacr.org/2014/674/`.

[36] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *IEEE S&P*, 2018.

[37] Judson Wilson, Riad S. Wahby, Henry Corrigan-Gibbs, Dan Boneh, Philip Levis, and Keith Winstein. Trust but verify: auditing secure Internet of Things devices. In *MobiSys*, 2017.

[38] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In *USENIX Security*, 2018.

[39] Francis Y. Yan, Jestin Ma, Greg Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for Internet congestion-control research. In *USENIX ATC*, 2018.

[40] Kaiyuan Yang, Matthew Hicks, Qing Dong, Todd M. Austin, and Dennis Sylvester. A2: Analog malicious hardware. In *IEEE S&P*, 2016.