Catching Lies (and Mistakes) in Offloaded Computation

Michael Mitzenmacher* Justin Thaler†

Consider a client who wants to run a computer program on some dataset, but lacks the processing power to do so. The client, or verifier, thus accesses a powerful but untrusted prover, who must not only run the program and return the output, but also provide a formal guarantee that the output is correct. This framework captures a wide variety of real-world scenarios. The verifier and prover may model a client and a commercial cloud computing service, or a CPU and a fast but potentially faulty coprocessor, or a peripheral device and a mainframe computer.

How can the verifier obtain a guarantee of correctness, without just ignoring the prover and executing the program locally? It is not obvious that this is even possible, at least without making strong assumptions about the behavior of a "cheating" prover. Indeed, researchers have proposed many solutions that rely on such assumptions, including replication, auditing, and the use of trusted hardware. In contrast, the research area known as verifiable computing (VC) is more ambitious: it seeks solutions that make no assumptions about the behavior of a cheating prover.

Theoretical computer scientists discovered surprisingly efficient VC protocols in the late 1980s and early 1990s. While these came in several flavors (known as interactive proofs, probabilistically checkable proofs, and argument systems), they all provide the following guarantee: if the output returned by the prover is incorrect, then the verifier will catch the prover in a lie with high probability, no matter how hard the prover tries to convince the verifier otherwise. Moreover, asymptotically speaking, these protocols ensure that the verifier does little more than read the input, and the prover does little more than execute the program.

These discoveries had a transformative effect on the the theory of computational complexity, with many consequences still being explored today. But despite their remarkable asymptotics, all of these protocols were thought to be wildly impractical, and with good reason. Naive implementations would have comically high concrete costs – the prover would need millions of years to prove correctness, even for small computations, and the verifier would save time relative to local execution only on truly enormous inputs.

But the last few years have seen this viewpoint challenged, as several research groups have developed VC protocols with drastically reduced costs. These groups have pursued several different tacks, following the various theoretical approaches. The resulting collection of implementations combine algorithmic improvements with systems work to achieve costs that approach genuine practicality.

The Pinocchio system described in the following paper refines an important theoretical advance by Gennaro et al. [1]. Together, these two works represent a dramatic improvement in speed, generality, and functionality. Pinocchio provides a non-interactive argument system that supports programs written in a subset of C, and automatically executes the program in a verifiable manner. Pinocchio's verifier does a one-time pre-computation to construct a public key based on the

*Harvard University, School of Engineering and Applied Sciences.
†Yahoo Labs.

computation to be performed; if the same computer program is run on multiple inputs, the same key can be used for them all. The proofs produced by Pinocchio's prover are short (288 bytes) and quick to verify. The authors demonstrate the system's capabilities with several test programs, ranging from matrix multiplication to a lattice gas simulation.

It is worth placing Pinocchio in a broader context. The various implemented VC protocols offer a wide range of tradeoffs between expressiveness, features, and efficiency — Blumberg and Walfish provide a detailed comparison of these tradeoffs [2]. Broadly speaking, interactive proofs are the least general, but have the lowest costs when they apply. Argument systems, particularly non-interactive ones like Pinocchio, are more expensive: several of Pinocchio's costs remain very high, particularly the prover's runtime. In addition, the one-time pre-computation for the verifier can be orders of magnitude more expensive than local execution, meaning that many inputs are required for the verifier to save work. But these expenses come with a substantial increase in generality and features. In particular, a critical feature supported by Pinocchio is zero-knowledge, best explained with an example.

Suppose that a computer program takes two inputs, one from the verifier and one from the prover, and that the prover's input is sensitive. One might want the prover to run the program and provide the answer to the verifier, without revealing any extra information about the prover's input. A standard example is a person who wants to compare his or her salary to others', yet the actual salaries of others must be kept private. Pinocchio is the first implemented system to provide such zero-knowledge proofs.

Continued improvements in the cost of VC protocols may render them genuinely practical for a wide range of applications, thereby offering a new way to deal with issues of trust and correctness in real systems. But a crucial point is when multiple parties hold sensitive inputs, there may be no alternative to zero-knowledge proofs (in contrast, local execution is always an alternative, however unattractive, to outsourcing computation when the input is not sensitive). Thus, even if efficiency improvements tail off, there are important scenarios where systems like Pinocchio are the only option. In these settings, the efficiency improvements achieved by the authors are crucial, and the kinds of overheads we see in systems like Pinocchio may already be acceptable.

References

[1] R. Gennaro, C. Gentry, B. Parno, M. Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. *EUROCRYPT*, pages 626–645, 2013.

[2] M. Walfish and A. J. Blumberg: Verifying computations without reexecuting them. *Commun. ACM* 58(2): 74-84, 2015.