The Challenge of Tamperproof Internet Computing

Edmund M.A. Ronald, Ecole Polytechnique, Paris
Moshe Sipper, Swiss Federal Institute of Technology, Lausanne

The Internet is the largest distributed computer ever built—at least, in theory. Writing on the actual role of this worldwide network of tens of millions of computers, Terry Winograd, director of the Stanford Human-Computer Interaction Consortium, observed, "the computer is not a machine whose main purpose is to get a computing task done. The computer ... is a machine that provides new ways for people to communicate with other people." (See "The Design of Interaction," *Beyond Calculation: The Next Fifty Years of Computing*, Peter J. Denning and Robert M. Metcalfe, eds., Springer-Verlag, New York, 1997, pp. 149-161.)

The prevailing tendency to exploit computation to achieve better communication is swiftly redefining the global information ecology by making every published item instantly visible in any part of the world. However, the inverse trend—harnessing global communication to achieve more powerful computation—is also developing before our eyes.

CONVERGING ON THE INTERPUTER

Researchers have proposed various schemes to transform the Internet into the "Interputer." Several companies are creating applications, tools, and protocols to harvest cycles from idling CPUs around the world, while compensating their obviously industrious users with offline and online gifts (David Pescovitz, "Power to the PC: Distributed Computing over the Internet Goes Commercial," *Scientific American*, Apr. 2000, pp. 15-16). Pescovitz notes that the largest distributed computation in history is SETI@home, a project that aims to discover life on other planets. More than 1.7 million volunteers have participated in this project thus far.

Internet computing—harnessing global communication to increase computational power—is now possible. But will it ever truly be secure?

Although the potential benefits of a universally accessible Interputer are undoubtedly extensive, a fundamental problem lurks backstage: How can you guarantee the accuracy of the results you receive from a remote computing node, which has just *purportedly* run the program you sent it?

Suppose Arnold writes program P and sends it to Bernice over the Internet, along with input I. Bernice runs program P on supplied input I and sends back to Arnold output O, which she claims to be the computation's result. How can Arnold be sure that O is indeed P's output run on I? That is, how can we achieve *tamperproof* Internet computing, or "cyberputing?"

TWO KEY QUESTIONS

The problem of checking that $O = P(I)$ is undecidable in its most general form—as so many questions concerning program dynamics tend to be. However, if we assume Bernice returns an answer after a known, finite amount of time—thus placing an upper limit on the number of computational steps—the problem becomes theoretically possible to solve (that is, the problem is formally decidable—though not necessarily tractable). Obviously, we'd like a shorter verification procedure than Arnold's running P on I and comparing his output with Bernice's. Therefore, we aim for an efficient tool or procedure that would assure Arnold of O's veracity.

A few questions regarding the tamperproof-computing scenario come to mind:

- Is Bernice malicious? Did she intend to send Arnold the erroneous output, or was this just an accident?
- Does Bernice have a standard computer, or does she use special, tamper-resistant hardware?

If Bernice is malicious, the second question may be irrelevant. As Ross Anderson and Markus Kuhn note, "trusting tamper resistance is problematic; smartcards are broken routinely, and even a device that was described by a government signals agency as 'the most secure processor generally available' turns out to be vulnerable. Designers of secure systems should

consider the consequences with care" ("Tamper Resistance—A Cautionary Note," *Proc. 2nd Usenix Workshop Electronic Commerce*, Usenix Assoc., Berkeley, Calif., 1996, pp. 1-11).

CRYPTOGRAPHY TO THE RESCUE

Cryptographic techniques can help. A public-key scheme, for example, can render tamperproof a simple one-line program containing a two-operand mathematical operation such as addition; in this case, $P = a + b$ and $I = (a, b)$. Arnold and Bernice first agree on the public-key scheme they will use. Arnold then sends Bernice a lookup table containing all possible combinations of a and b, along with their resulting sums, $c = a + b$. Each row comprises a pair (a, b) as the index, along with a triplet (a, b, c) containing the original input and result c. Table 1 shows this 16-bit addition table.

Arnold's public key, E_A, encrypts both P and I, cloaking them while still providing an index into the lookup table. He then sends these to Bernice, who looks up the answer in the table and sends Arnold the output $O = E_A(a, b, c)$. Arnold can immediately decrypt O using his private key, D_A. The encrypted lookup table gives Bernice access to only an obfuscated version of c and prevents her from tampering with computations. When Arnold receives a message containing a, b, and c, he can determine whether the result line belongs in the lookup table. To foil attacks, Arnold can randomize and modify the lookup table as often as he pleases.

WELL, NOT EXACTLY...

Although this simple scheme demonstrates a possible use of cryptographic techniques in tamperproof Internet computing, it is neither all-encompassing nor necessarily practical. How do you make multioperand operations ($a + b + c + d + e$) tamperproof? How do you protect program constructs such as assignments and conditional jumps? And, probably the most difficult, how do you handle loops and unconditional jumps (the constructs ultimately responsible for rendering a language universal)? Some form of *cumulative encrypted checksumming*, providing the final output O with an authenticity certificate, might be possible, but remains a challenge.

Table 1. A 16-bit lookup table listing indexing pairs (a, b) and triplets (a, b, c), where c is the result of adding inputs a and b. Public key E_A encrypts both entries. (The comma operator used in the encrypted entries is simple textual concatenation.)

Index	Indexing pair, (a, b)	$c = a + b$
1	$E_A(a_1, b_1)$	$E_A(a_1, b_1, c_1)$
2	$E_A(a_2, b_2)$	$E_A(a_2, b_2, c_2)$
·	·	·
·	·	·
·	·	·
2^{32}	$E_A(a_{2^{32}}, b_{2^{32}})$	$E_A(a_{2^{32}}, b_{2^{32}}, c_{2^{32}})$

Is reliable tamperproof Internet computing truly possible? The problem is doubtless harder than standard cryptography because it involves dynamic programs rather than static data. Of course, the whole notion of cryptography is rather slippery, anyway. As Terry Ritter said, "Despite the frequent cryptography articles in IEEE journals, cryptography is an art, not an engineering discipline: The property we seek to produce—strength against unknown attack—is not measurable, and so it is literally out of control." (See "Cryptography: Is Staying with the Herd Really Best?" *Computer*, Aug. 1999, pp. 94-95.)

Though elusive—and possibly illusive—the challenge of tamperproof Internet computing is still worth facing. And if, as Prince Otto von Bismarck declared, "politics is the art of the possible," then perhaps it is also possible, through the art of engineering, to render the nascent Interputer more secure. ✷

Edmund M.A. Ronald is an affiliate researcher in the Center for Applied Mathematics at Ecole Polytechnique, Paris. He is currently also a visiting researcher at the Swiss Federal Institute of Technology, Lausanne, Switzerland. Contact him at eronald@cmapx.polytechnique.fr.

Moshe Sipper is a senior researcher at the Swiss Federal Institute of Technology, Lausanne. Contact him at moshe.sipper@epfl.ch.

Editor: Ron Vetter, University of North Carolina at Wilmington, Department of Computer Science, 601 South College Rd., Wilmington, NC 28403-3297; voice +1 910 962 7192, fax +1 910 962 7457; vetterr@uncwil.edu