Addressing The Trust Asymmetry Problem In Grid Computing With Encrypted Computation

Peter A. Dinda
pdinda@cs.northwestern.edu
Department of Computer Science
Northwestern University

ABSTRACT

Trust asymmetry is a core, albeit rarely discussed, problem in scalable computing. Techniques for protecting a host's operating system (and other processes) from a user's process are well understood and widely deployed. However, there is currently no way to protect the user's process from the OS. Hence, while the host's owner need not trust the user at all, the user must trust the owner completely. This, we argue, leads to practical limits to scalability for computation that, because of encryption, simply do not exist for communication. We argue that it is imperative for the grid computing community to address this problem using encrypted computation techniques. We then propose a simple mechanism for encrypted computation of Boolean circuits and show how it can likely be generalized for use in an object code translator.

1. INTRODUCTION

The goal of parallel computing is the creation of algorithms and toolchains that provide performance that scales cleanly as we add machines. In practical terms, however, there are a limited number of machines available to a user that are under his control or under the control of his organization. Grid computing [14, 15] has the potential to make many more machines available. Early projects such as SETI@Home [30], Folding@Home [21], and Entropia [6] have demonstrated that it is possible to successfully scale some applications to hundreds of thousands of machines. Research in computational economies [32, 5] suggests that it is at technically feasible to sell resources in a distributed computing environment. A future in which loosely coupled applications scale to the number of machines that the world is willing to volunteer, or that the user has the money to rent, does not seem unreasonable, both in terms of need [2] and technology [4, 12, 20].

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449, and a gift from VMWare. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

As his application scales to more and more machines, the user has less and less control over the machines on which it runs. Consider any one of the machines on which the user's code is running. At a high level of abstraction, that machine has been sent an input data set and an algorithm, it will send and receive messages to/from other machines, and it will produce an output data set to send back to the user.

Using well understood and widely deployed techniques pioneered in the 1960s, the machine's operating system (and other processes) are in principle well protected from the user's process. In practice, bugs exist and so the machine may well have been hacked. The user's process is in an entirely different position, however. It is completely at the mercy of the machine's OS (hacked or not); the OS (or virus) can straightforwardly

- read the input data sets and messages,
- read the user's algorithm,
- read the output data sets, and
- write bogus output data sets and messages.

Interestingly, those operating systems that provide the most isolation and protection in theory and practice, virtual machine monitors, are in an ideal position to observe the application's computation [17], and communication [18].

Trust asymmetry—the user must trust the machine completely while the owner of the machine need not trust the user at all—is the most significant reason why grid computing may not scale to meet the needs of many possible applications. As the application scales, the user must trust more individuals and sites, of whom he is likely to know very little. There certainly exist *low stakes applications*, such as SETI@Home and Folding@Home, that can make effective use of untrusted machines. In these applications, the datasets, messages, and algorithms are open, and well known techniques can be used to protect against even Byzantine failures.

It is important to note that low stakes applications can be extremely important and have substantial funds applied to them. It is a low financial impact of information or algorithm disclosure that is important. For example, the high energy physics applications that have driven much of grid computing research represent substantial investments on the part of the U.S. and the E.U. However, the disclosure of data or algorithms would at worse be linked to scientific misconduct. The effect of trust asymmetry in this context is small, and is further ameliorated by the fact that it is largely governments or scientific labs that are the resource providers for these applications. Given this, the trust chain-based security model that is widely deployed is sensible.

High stakes applications, those where disclosure of datasets, messages, and algorithms would result in significant *monetary* loss, are, on the other hand, greatly affected by trust asymmetry, but their users are precisely who are needed to make scaling of grids to Internet levels necessary: paying customers. If we do not fix the trust asymmetry problem, we simply may never create a large scale demand for computational grids, and many applications, constrained to live within a single organization, will go hungry for cycles. Alternatively, a few highly trusted service providers might emerge that are themselves constrained in how many machines they can field, and who will most likely exact a high price for signing their trusted names on legal contracts that indemnify the user.

High stakes messages traverse untrusted Internet paths every day and can be trivially observed. However, because of fast encryption [23] and key exchange protocols based on asymmetric cryptography [24, 9], trust is required only between the sender and the recipient. The result is that communication scales. The Internet is composed of tens of thousands of autonomous systems and we need not consider the path a message takes through them in order to be confident that the message was not read or corrupted by nefarious actors.

We advocate addressing the trust asymmetry problem through the use of encrypted computation. We envision a post-compilation step in which input and output keys are chosen, the object code is transformed given the keys, and a stub based on the keys is generated for encrypting input datasets, calling the object code, and decrypting output datasets. This will meet the following requirements:

- The remote machine will be unable to read the input data sets,
- The remote machine will be unable to reconstruct the algorithm implemented in the object code,
- The remote machine will be unable to read the output data set, and
- The user will be able to verify that the output has not been tampered with.

It is necessary that we be able to prove that these requirements are met. If so, then the user need not trust the remote machine at all, making scaling of high stakes applications to arbitrary levels possible.

Notice that as the standards put forward by the Global Grid Forum move towards a web services model [31, 13], which is effectively an RPC model of distributed computing, the abstract model described above will match increasingly closely with what is expected to become practice.

We begin by describing several approaches to secure computation to explain why general purpose encrypted computation is necessary. Next, we describe an algorithm for the encrypted computation of arbitrary combinational logic circuits that we believe is new, fast, and difficult to break. However, we do not yet have a formal proof of the difficulty. We next describe how this algorithm could be used in an object code translator to meet the requirements laid out above. We conclude by discussing the next steps and the likely performance overheads of our approach to encrypted computation.

2. SECURE COMPUTATION APPROACHES

The following discussion has been, in part, informed by Sarmenta's excellent treatment [27, Appendix B], which is written from the perspective of his Banyanihan volunteer grid computing system. The opinions are my own. Banyanihan itself provides "sabotage tolerance" (sabotage here means that the remote machine returns the wrong answer) through techniques related to voting [28].

Trust chains

The trust chain model essentially provides no direct protection of the inputs, algorithm, and output. Instead, it formalizes the notion of trust; you may send code and data to a machine because it has presented you with a secure document that essentially says "A says that I can be trusted". If you know A, directly or indirectly, then you may be willing to use the machine.

Trust chain models are widely used in today's Internet, particularly for the World Wide Web. Digital certificates (see [10] for an in-depth discussion) are used to authenticate and authorize users and web sites. A certificate authority, such as Verisign, signs a certificate for a web site. The certificate binds the identity of operator of the web site (Microsoft, say) with the specific web site (www.microsoft.com). It is an assertion of trust: the user trusts that Verisign can identify Microsoft and then, assured that he is in fact talking to Microsoft, determines whether to trust a particular web page. The practice in the grid computing community is similar [3].

The problem with trust chains is that they are likely to be complex to understand and evaluate. If a human must be in the loop, or must write a policy about what is to be trusted or not, then it is likely that errors will occur. This possibility obviously precludes using trust chains with high stakes applications. Another problem is that formal reasoning about trust chains is in its infancy. Trust chains in web systems are typically very short, potentially leading to limited scalability. Revocation is also a problem. When a machine goes bad, it might take a long time before everyone knows it.

Attestation

Attestation, as in Terra [16], essentially provides a the user with a certificate chain that describes the software stack on the machine. This makes it possible to detect software stacks that have been tampered with. The user can then choose to run his code and provide his data to only to appropriate and unmodified software stacks.

The problem is that a software stack is a massive and complex entity, so even the unmodified stack may have features or security holes that leave the user's code and data vulnerable. Complexity is growing, which suggests that the risk associated with attestation is also growing. Furthermore, as with trust chains, the verification of an attestation chain or policy creation may fall heavily on the user, making errors likely. High stakes application users are likely to be wary.

Another issue with attestation is that a software stack can often change. In many cases, these changes are not optional; security patches must be deployed promptly. The certificate chain would need to be regenerated and redistributed on such changes, limiting scalability. One response to this problem has been a proposal for semantic attestation [19] in which a component of the stack would attest to its interface and operation. Since a patch would not likely change this interface, new certificates would not need to be generated. However, determining the semantics and developing a proof that the code implements them is a hard problem.

Obfuscation

Compiler techniques have been developed to make it difficult to understand, and thus reverse engineer object code [8, 7]. In some cases inputs and outputs may also be obfuscated. Performance impacts are often acceptable.

While this approach provides some promise in making the remote machine unable to reconstruct the algorithm, there are no proofs of difficulty for these methods. The lack of proofs makes it unlikely that users who need to run high stakes applications will be satisfied with obfuscation. There is no way to quantify the risk of the remote machine determining the inputs, algorithm, and output.

For the compiler community, that compiler toolchains have been developed that implement practical obfuscation suggests that it is likely that the techniques that we describe in the next section are implementable.

Encrypted computation

Encrypted computation seeks to meet some or all of the requirements laid out in the introduction, including proofs of the difficulty of circumventing its safeguards. The combination of a simple model (the remote machine is simply not trusted at all) and proofs is very powerful because it is very simple. Of all the techniques described here, it is arguably easiest for a human being to understand and thus the most likely to be trusted with a high stakes application.

Unfortunately, encrypted computation is quite complex. There are two essential approaches. The first is to develop encrypted algorithms for specific problems. This has recently seen success in polynomial evaluation [25] and pattern matching on strings [29]. For obvious reasons, this is unlikely to be an approach that developers or users of high stakes applications are likely to take.

The second approach to encrypted computation is general. The earliest work in this area Abadi and Feigenbaum's seminal work on the secure evaluation of Boolean circuits [1]. That work has not been deployed in practice because of the need for considerable communication back to the user's machine. It is often referred to as "interactive" encrypted computation, meaning that the evaluator must frequently ask questions of the submitter.

Fast, non-interactive encrypted computation would be much preferred in our high stakes application scenario. Sander and Tschudin's work [25] may be extensible to general computation. Of particular interest is recent work by Loureiro et al [22] that looks at non-interactive encrypted computation of Boolean circuits using encryption techniques drawn from coding theory. Our independently developed scheme, as described in the next section, is potentially related to Loureiro's approach.

Mobile agents

The mobile agent community has given considerable thought to the security components of their work. Although a large part of this work is concerned with the path an agent takes, either avoiding certain hosts or being able to determine the path a posteriori, mobile cryptography has also been a significant concern [33, 26]. It seems logical that the high performance computing community should be able to leverage this work.

3. A SIMPLE APPROACH TO ENCRYPTED COMPUTATION OF BOOLEAN CIRCUITS

Encrypted computation of a Boolean circuit to meet the requirements laid out in the introduction appears to be a difficult problem. However, looks may be deceiving. In the following, we describe a technique for doing so which, as far as we are aware, is new. However, we do not yet have a proof of the difficulty of circumventing this technique.

Consider a Boolean function f,

$$Y = fX$$

In which X is a vector of n input bits, numbered x_1 through x_n and Y is a vector of m output bits, numbered y_1 through y_m.

The function f can obviously be considered in a number of equivalent forms, such as sum of products (disjunctive normal) or a truth table. Let's write an example f in these two forms, for $n = 3$ and $m = 2$:

$$\begin{aligned} y_1 &= (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \\ y_2 &= (x_2 \wedge \neg x_3) \vee (x_1 \wedge x_3) \end{aligned}$$

x_1	x_2	x_3	y_1	y_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

Now, suppose that we want to execute f remotely for an input $X = (0, 0, 1)$ such that the remote site cannot determine

- the input X,
- the output Y corresponding to the input X, and
- the function f.

For now, we will let the remote site cheat (it can return the wrong answer), record (it can store inputs and outputs as well as f), and play back.

We claim that the three constraints can be met using one-time pad encryption [11] folded into the function.

Suppose we have a one-time pad for the input and one for the output. These are just randomly generated bit vectors of length n and m, respectively. To encrypt with a one-time pad, we just exclusive-or the bit vector with the input. To decrypt, we just exclusive-or the encrypted value with the bit vector again. Suppose we have a one time pad for our 3 input bits that is $E = (1, 0, 1)$,

$$\begin{aligned} EX &= (1, 0, 0) \\ EEX &= (0, 0, 1) \end{aligned}$$

We can think of using a one-time pad with the circuit f as well. Suppose we again use $E = (1, 0, 1)$. Now we can interpret this bit vector as the template for inversion on the circuit inputs. $E = (1, 0, 1)$ means invert the first input (if it is zero, it will clearly become 1, and if it is one, it will clearly become zero in the exclusive or step), leave the second input alone, and insert the third input:

$$\begin{aligned} x'_1 &= \neg x_1 \\ x'_2 &= x_2 \\ x'_3 &= \neg x_3 \end{aligned}$$

Of course, if we add these inverters, we are also changing the output of the circuit. However, recall that we can apply the one-time pad twice to get back the original input. Similarly here, we can clearly insert two inverters in a row wherever the one-time pad tells us and be left with the same circuit:

$$\begin{aligned} x'_1 &= \neg\neg x_1 \\ x'_2 &= x_2 \\ x'_3 &= \neg\neg x_3 \end{aligned}$$

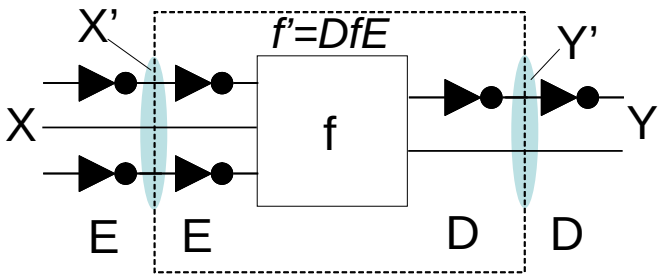

Figure 1: Encrypted computation of logic circuit

Clearly, we can play this same game at the output of the circuit. Suppose that D is the one-time pad for the outputs. Then:

$$Y = DDfEEX \tag{1}$$

Now we can play the following trick. The client generates a new function:

$$f' = DfE \tag{2}$$

and its input:

$$X' = EX \tag{3}$$

which are given to the server. The server computes

$$Y' = f'X' \tag{4}$$

which it passes back to the client, which computes the actual output as

$$Y = DY' \tag{5}$$

Figure 1 illustrates this process graphically.

Notice that X' is a one-time pad encryption of the inputs. It is very hard to derive X from X' without knowing the pad E. The same is true for Y and D. Notice, however, that we have given both D and E to the server. Even worse, it looks like we gave f to the server. But have we?

The key claim is that by folding E and D into f (i.e., $f' = DfE$), we have made it very hard to "factor out" E (which is needed to decrypt the input), f (the function we want to apply), and D (which is needed to decrypt the output). A proof of this claim is needed. Boolean functions can of course describe any computation, so, if the above claim is correct, then we now have a way to do remote computation in which the remote site, the server, cannot know the input, the output, or the structure of the computation.

Consider an example. Here we continue to assume that $E = (1, 0, 1)$ and $D = (1, 0)$. Now, let's look at what we have for

f':

$$\begin{aligned} y'_1 &= \neg((\neg x'_1 \wedge \neg x'_2) \vee (\neg\neg x'_1 \wedge \neg x'_3)) \\ &= \neg((\neg x'_1 \wedge \neg x'_2) \vee (x'_1 \wedge \neg x'_3)) \\ &= \neg(\neg x'_1 \wedge \neg x'_2) \wedge \neg(x'_1 \wedge \neg x'_3) \\ &= (x'_1 \vee x'_2) \wedge (\neg x'_1 \vee x'_3) \\ &= ((x'_1 \vee x'_2) \wedge \neg x'_1) \vee ((x'_1 \vee x'_2) \wedge x'_3) \\ &= (\neg x'_1 \wedge x'_2) \vee (x'_1 \wedge x'_3) \vee (x'_2 \wedge x'_3) \end{aligned}$$

$$\begin{aligned} y'_2 &= (x'_2 \wedge \neg\neg x'_3) \vee (\neg x'_1 \wedge \neg x'_3) \\ &= (x'_2 \wedge x'_3) \vee (\neg x'_1 \wedge \neg x'_3) \end{aligned}$$

x'_1	x'_2	x'_3	y'_1	y'_2
0	0	0	0	**1**
0	0	1	**0**	0
0	1	0	**1**	**0**
0	1	1	1	**1**
1	0	0	**0**	0
1	0	1	1	**0**
1	1	0	0	**0**
1	1	1	**1**	1

The highlighted outputs are those that have changed from f to f'.

Now consider what happens for a possible input $X = (0, 1, 0)$:

$$\begin{aligned} X' &= EX \\ &= (1, 1, 1) \\ Y' &= f'X' \\ &= (1, 1) \\ Y &= DX' \\ &= (0, 1) \end{aligned}$$

Other attacks

We could deal with replay and storage attacks straightforwardly: use a different set of one-time pads E and D each time, recomputing f' and shipping it as we go. If we reuse the pads, we increase the chances of the encryption being broken by, for example, frequency analysis. Of course, if the inputs themselves are uniformly distributed, even this would seem not to be possible.

We could deal, at least probabilistically, with the server returning a false answer to us by embedding a test circuit in f'. In other words, we would add an unencrypted test circuit g to f' that shares in the inputs (additional unencrypted inputs could also be added) and produced unencrypted outputs. The test circuit's inputs and outputs could not be distinguished from the "real" inputs and outputs, and the test circuit would fold into f' without being extractable. Then, we could add a test sequence to the input that would produce known output if the remote server was not lying.

Concerns

It is possible that our process could cause the expected size of the encrypted circuit to be much larger than the original circuit, thus requiring much more time to compute it. In the worst case, one could imagine an exponential explosion in the size of the circuit. We do not yet know what the average and worst case behavior of this approach is.

A second concern combines practical and theoretical aspects. In an implementation of this idea, the circuit, after being encrypted, would likely be re-optimized. If the simple version of the circuit f' with the inverters at the inputs and outputs is the optimal representation, then it is certainly possible that the optimizer would choose it. In other words, optimization of the circuit could potentially turn out to be an attack.

4. ENCRYPTED COMPUTATION OF A BASIC BLOCK

Boolean circuits are fine, but we really want to ship around programs that target some instruction set architecture. Therein lies a rub. Consider a basic block of instructions:

```
ADD R1, R0, R3
SUB R3, R5, R5
ST R1, A
```

Clearly, the basic block has simple straight-line flow control (ignore exceptions for now) and we can easily describe it as a data flow graph with the instructions being nodes, and dependencies being edges. Data (X) flows into the graph from memory and registers and flows out to memory and registers (Y).

Suppose we wanted to ship the basic block to a remote server to be executed. Naively, we could apply the analogue of the $Y = DDfEEX$ approach of the previous section, creating a basic block f' that looked like:

```
XOR R1, E (or part of E)
XOR R0, E
XOR R5, E
ADD R1, R0, R3
SUB R3, R5, R5
ST R1, A
XOR R3, D (or part of D)
XOR R5, D
XOR A, D
```

However, this has obvious problems: We can clearly see E and D and we have not changed the basic block in any way. All this would do is slow us down.

To apply our encrypted computation scheme to a basic block, we would need to do the following:

1. Generate the above "encrypted" basic block.
2. Generate a data flow graph for the block.
3. Drop a level of abstraction: replace the data flow graph nodes with the logic design of the data path implied by the instructions. For example, the `ADD` instruction might turn into a carry lookahead adder implementation in NAND gates. Notice that the `XOR` instructions will turn into selective inversions of the inputs and outputs.

4. Resimplify the Boolean function, folding in the inverters as in the previous section.

5. Generate a new instruction sequence that implements the new folded Boolean function. This would use techniques such as those used in code generation for digital logic simulators.

It is not clear how much overhead this technique would introduce or how the overhead would vary based on the instruction sequence.

Control flow

Beyond basic blocks, the primary challenge is control flow. To meet this challenge we would have to produce a Mealy or Moore machine, combining the combinational logic of the basic blocks, with state that flows from block to block as well as that enables, via a multiplexor, which block to use for the current step.

Concerns

Beyond the concerns of the previous section, here we would have to understand the ramifications of the initial instruction sequence of the basic block. In a sense, this sequence represents an optimized representation of a Boolean circuit, factoring out functionality into RTL-level constructs like adders. It is conceivable that the instruction sequence we generate will be very similar—that the optimal representation of the intermediate Boolean circuit form is, in fact, the original circuit augmented with `XOR` instructions on entry and exit, and that the optimizer finds that sequence.

5. CONCLUSION

Solving the trust asymmetry problem is essential to developing truly scalable grid computing, and, in particular, to attracting much needed high stakes applications. While there are many possible approaches, we have argued that general purpose encrypted computation is the one that is most desirable because its combination of simplicity for the user (no machines need be trusted at all) and proofs of the difficulty of subversion.

We have also described what we believe to be a new, simple approach to the encrypted computation of Boolean functions and shown how it could be used to transform object code into secure object code. This work is in its very early stages. In particular, we have no proof of security at this point, and our transformation scheme applies only to a basic block, although we know how to in principle extend it for control flow.

We are currently working on a proof of the difficulty of breaking our encryption scheme. We also plan to develop a proof of concept within the .NET CLR framework.

6. REFERENCES

[1] Abadi, M., and Feigenbaum, J. Secure circuit evaluation. *Journal of Cryptography 2*, 1 (1990), 1–12.

[2] http://www.ncsc.org/news/pr/surabiogrid.html.

[3] Butler, R., and Genovese, T. Global grid forum certificate policy model. Tech. rep., Global Grid Forum, June 2003.

[4] Butt, A. R., Zhang, R., and Hu, Y. C. A self-organizing flock of condors. In *Proceedings of ACM/IEEE SC 2003 (Supercomputing)*.

[5] Buyya, R., Giddy, J., and Abramson, D. An economy grid archicture for service-oriented distributed computing. In *Proceedings of the 10th IEEE Heterogeneous Computing Workshop* (April 2001).

[6] Chien, A. A., Calder, B., Elbert, S., and Bhatia, K. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing 63*, 5 (2003), 597–610.

[7] Collberg, C., and Thomborson, C. Watermarking, tamper-proofing, and obfuscation - tools for software protection. Tech. Rep. 2003-03, Department of Computer Science, University of Arizona, 2000.

[8] Collberg, C., Thomborson, C., and Low, D. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Principles of Programming Languages 1998, POPL'98* (January 1998).

[9] Diffie, W., and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory IT-22*, 6 (1976), 644–654.

[10] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., and Ylonen, T. Spki certificate theory. Tech. Rep. RFC 2693, Internet Engineering Task Force, September 1999.

[11] Ferguson, N., and Schneier, B. *Practical Cryptography*. John Wiley and Sons, 2003.

[12] Figueiredo, R., Dinda, P. A., and Fortes, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003).

[13] Foster, I., Berry, D., Djaoi, A., Grimshaw, A., Horn, B., Kishimoto, H., Maciel, F., Savva, A., Siebenlist, F., Subramaniam, R., Treadwell, J., and von Reich, J. The open grid services architecture version 1.0. Tech. rep., Global Grid Forum, July 2004.

[14] Foster, I., and Kesselman, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[15] Foster, I., Kesselman, C., and Tuecke, S. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications 15*, 3 (2001).

[16] Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., and Boneh, D. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM symposium on Operating systems principles SOSP 2003* (2003), pp. 193–206.

[17] Garfinkel, T., and Rosenblum, M. A virtual machine introspection-based architecture for intrusion detection. In *Proceedings of the 2003 Network and Distributed Systems Symposium (NDSS 2003)* (2003).

[18] Gupta, A., and Dinda, P. A. Inferring the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing (JSPPS 2004* (June 2004).

[19] Haldar, V., Chandra, D., and Franz, M. Semantic remote attestation—a virtual machine directed approach to trusted computing. In *Proceedings of the 3rd USENIX Virtual Machine Research And Technology Symposium (VM 2004)* (May 2004).

[20] Hand, S., Harris, T., Kotsovinos, E., and Pratt, I. Controlling the xenoserver open platform. In *Proceedings of the Sixth IEEE Conference on Open Architectures and Network Programming (OPENARCH 2003),* (April 2003).

[21] Larson, S. M., Snow, C. D., Shirts, M., and Pande, V. S. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. In *Computational Genomics*, R. Grant, Ed. Horizon Press, 2002.

[22] Louriero, S., Bussard, L., and Roudier, Y. Extending tamper-proof hardware security to untrusted execution environments. In *Proceedings of the 5th Smart Card Research and Advanced Application Conference* (November 2002), USENIX.

[23] National Institute of Standards and Technology. Announcing the advanced encryption standard. Tech. Rep. Federal Information Processing Standards Publication 197, November 2001.

[24] Rivest, R. L., Shamir, A., and Adelman, L. M. A method for obtaining digital signatures and public-key cryptosystems. Tech. Rep. MIT/LCS/TM-82, Massachusettes Institute of Technology, 1977.

[25] Sander, T., and Tschudin, C. Towards mobile cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy* (1998).

[26] Sander, T., and Tschudin, C. F. Protecting mobile agents against malicious hosts. *Lecture Notes in Computer Science 1419* (1998).

[27] Sarmenta, L. F. G. *Volunteer Computing*. PhD thesis, Massachusetts Institute of Technology, June 2001.

[28] Sarmenta, L. F. G. Sabotage-tolerance mechanisms for volunteer computer systems. *Future Generation Computer Systems 18*, 4 (2002).

[29] Song, D., Wagner, D., and Perrig, A. Practical techniques for searches on encrypted data. In *Proceedings of the IEEE Security and Privacy Symposium* (May 2000).

[30] Sullivan, W. T., Werthimer, D., Bowyer, S., Cobb, J., Gedye, D., and Anderson, D. A new major seti project based on project serendip data and 100,000 personal computers. In *Proceedings of the Fifth International Conference on Bioastronomy* (1997), C. Cosmovici, S. Bowyer, and D. Werthimer, Eds., no. 161 in IAU Colloquim, Editrice Compositori, Bologna, Italy.

[31] Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maquire, T., Sandholm, T., Snelling, D., and Vanderbilt, P. Open grid services infrastructure (ogsi) version 1.0. Tech. rep., Global Grid Forum, June 2003.

[32] Waldspurger, C. A., Hogg, T., Huberman, B. A., Kephart, J. O., and Stornetta, W. S. Spawn: A distributed computational economy. *IEEE Trans. on Software Engineering 18*, 2 (February 1992), 103–117.

[33] Yee, B. S. A sanctuary for mobile agents. In *Secure Internet Programming* (1999), pp. 261–273.