Can We Trust the Chips of the Future?

The IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2011) was colocated with the 2011 Design Automation Conference earlier this year. As the topic of cybersecurity has drawn more attention, especially in mainstream press publications, several participants from the symposium were asked to give their thoughts on the area in this roundtable. The design, verification, and attack detection or prevention of secure systems present real challenges in the development of computing platforms, beginning at the IC and extending to the system architecture, including application software.

IEEE Design & Test thanks the roundtable participants: Mohammad Tehranipoor (Univ. of Connecticut), our moderator; Srinivas Devadas (Massachusetts Inst. of Technology), Kevin Gotze (Intel), Farinaz Koushanfar (Rice Univ.), Miodrag Potkonjak (Univ. of California, Los Angeles), and Ingrid Verbauwhede (KU Leuven). *D&T* gratefully acknowledges the help of Kathy Embler (MP Associates) for arrangements and David Yeh (SRC), our Roundtables Editor, who organized the event with the moderator. Special thanks go to the IEEE Computer Society for sponsoring the roundtable.

D&T: Hardware trust problems include hardware Trojans, counterfeiting, side-channel attacks, and a lack of CAD tools for secure designs. What are the major problems in hardware security and trust?

Srinivas Devadas: There's an intellectual gap between hardware designers and security designers, for want of a better term. There are opportunities in building hardware security mechanisms into processors and ASICs, but we're missing the people and students being trained who have the intellectual capabilities to understand how to implement security functionality in hardware properly and securely, and to know which security mechanisms are appropriate for a particular security application.

Ingrid Verbauwhede: Product providers and users will need hardware security because of threats such as hardware Trojans and side-channel attacks. People will defend against these particular attacks only if they see an economic benefit—e.g., counterfeiting could be considered strictly a business issue. The development of CAD tools for secure design is one of many potential countermeasures, but counterfeiting also involves the human factor. The weakest link must be addressed first: most counterfeiting is done by people just stealing designs.

Farinaz Koushanfar: A lot of problems are being worked on and may be driven by some of the hypothetical attacks out there, like the early hardware Trojans. Five years ago, DARPA had a basic model for the IC Trust program to detect structural or parametric changes. However, the likelihood of a sophisticated attack by such a broad model wasn't clear. This model has driven a lot of research work in the past few years but it seriously needed to be refined. The recent programs at DARPA (such as IRIS) and other agencies, and the recently funded hardware security projects by NSF, are attempting to improve the state of the art by looking into more intelligent and sophisticated functional attack models, and by studying the functional behavior. Hopefully, much-improved results will stem from the more-recent research. Side channels are also important since, apparently, people have gained economic benefit from making side-channel attacks on embedded systems. The problems are complex: take, for example, the counterfeiting problem. Certain problem aspects can be addressed by policy; others, like active prevention of counterfeiting

0740-7475/11/$26.00 © 2011 IEEE Copublished by the IEEE CS and the IEEE CASS

or detection, need much more research and can be fixed by construction and with better technology.

Verbauwhede: I agree. Companies have numbers on how much they lose to counterfeiting, not only in the IC design space but in general.

D&T: Someone asked the CEO of a Massachusetts company, "What is the single biggest problem you have?" The answer was "counterfeits."

Miodrag Potkonjak: Let's look at the problem in terms of a chess game. In IC testing, it's as if we're playing with an opponent who makes random moves. So even if we're not much of a chess player, we have a reasonable chance to win. In hardware security, we're playing against an opponent who is human, with intelligence like us and who could be well trained. The situation is much worse than in a game because the attacker can make additional types of moves that didn't exist when the game began. Now we have to try to anticipate what these additional moves might be, and to develop our defense with this in mind. So the first major difficulty in hardware security is that the problem itself is complex.

The second major difficulty is in trying to become knowledgeable in all possible areas from which attacks might originate. Somebody who is creative, capable, and hard-working can develop a defense against a given attack: the problem lies in becoming smart in other areas where we might have little, but an attacker has extensive, knowledge.

Verbauwhede: Isn't that the same in other security applications: engineers building systems for reliability in general?

Potkonjak: I see two types of problems. One type is reactive, in which typically there's a Trojan horse or a side-channel attack. An attacker does something, and I must respond. But for me, the key problems are those in which we use our creativity to invent something new. A prime example is the PUF—physical unclonable function, which is an elegant new concept and has much potential application. In practice, these PUFs will remain in place regardless of how nasty the attacker is. The key problem is to develop new concepts that let us leverage design degrees of freedom, and knowledge and innovation, to bring novel ideas to security.

D&T: Is it possible that we consider defense ahead of offense in the sense that we'll develop some mechanisms that could not only address what the attackers are able to do now but also basically eliminate some of the attacks they might make later?

Verbauwhede: Perfect security does not exist.

D&T: That's true. But could we eliminate some of the potential attacks that might happen?

Potkonjak: I don't envision eliminating all possible attacks; that would be too expensive. I'd like to prevent the possibility that an attacker gets control of my design, or to make sure that I can create mechanisms that control anything that an attacker may have in store for me.

Devadas: One of the things I think Miodrag is talking about is inventing new primitives, such as hash functions. Public-key cryptography is another powerful primitive which just changed the game three decades ago. Integrity verification allows us to use untrusted memory, so the question is, in hardware security and trust are there new primitives we can invent that would change the game in our favor? You asked, Can we have defense ahead of the offense? Well, if we invent a new primitive, that's one way to have a defensive mechanism in which the hacker must now do much more to subvert that mechanism.

Kevin Gotze: I work as a security validator, so I try to take on that role of an attacker, and typically I won't look exclusively at security primitives. I'll say, "Okay, it's likely they have a PUF or random number generator here." But I know they're using it for something, so I'm going to evaluate the entire system. I'm going to evaluate the caches, the buses they use. I'm going to find some part of the design that's not particularly security conscious where somebody made a mistake and which I'm going to try to exploit. I don't try to run right into the wall—I try to go around it or find some way through it.

Verbauwhede: I agree with that approach, and I think there's something to learn from the design and test flow. For instance, we can learn from the CAD community about design methods because many security features are done in an ad hoc way, and while the modules may be correct and secure,

composing the modules into correct and secure combinations is still a challenge. Security partitioning, for instance, is something worth investigating systematically.

Devadas: That's a valid point, and to extend that idea: some primitives can allow components to be untrusted. If we implement encryption, we can allow eavesdropping on the components. Then what an attacker would do is try to break the mechanism corresponding, for example, to the secret key storage or something that gives us access to the decrypted content.

Gotze: Right. And then we would have to redo our secret key storage, right?

Devadas: That's right. With respect to primitives, as I was describing them: if we invent something like information flow tagging, for example, what does that do? That limits the attacker's capability in the sense that the tags of the spurious data (data that is potentially sensitive or spurious) are going to limit where the data resides. That's what I really meant by primitives that narrow the scope of the attacker.

D&T: My understanding is that when we're talking about the information, the data, and data exchange, we're now talking about hardware cryptography. But when we talk about some of the other attacks like counterfeits or Trojans, for example, then it's a design problem. Can we consider a new design method as a new primitive? I do understand the primitives, but some of the things we're discussing probably are more of a design problem, and many solutions we have are simply patches. We apply the patch and say, "Okay, this problem is fixed." But then someone asks, "What about the cache problem?" And then we decide to fix that. So are we talking about a completely different way of designing ICs?

Potkonjak: To give an example: what Srini has developed is the PUF, which is a security primitive. But what Farinaz developed was how to use the PUF for hardware-metering enabling security, and that's a security protocol. Of course, these primitives and protocols are wonderful, and essential to build the cryptography field. However, hardware security is much broader and harder to address than cryptography. It's broader because new applications are emerging: applications that we never had before in cryptography, such as anonymity, trusted remote operations, and trusted remote sensing and actuation.

Also, regardless of what we conclude about hardware or system security, the conclusion may not be meaningful, depending on the technology being used. For example, if there's no process variation, then PUFs disappear because the PUF depends on process variation to operate correctly. We don't know if process variation will exist in future technologies. Also, process variation may be replaced by device aging. In addition, all of our most effective hardware Trojan detection techniques use leakage energy, but in nanomechanical devices, the leakage energy is on the order of 10^{-18}, 10^{-19} J. That can't be measured, so all current leakage-current-based hardware Trojan detection techniques disappear.

It is very tricky to figure out the technological system issues in building a two-billion-transistor chip. However, I agree with Srini that one thing we can make progress on is that we identify some small classes of problems—primitives, signal amplifiers, protocols, side channels, and so on—and then try to address them more systematically. What is now a major research focus is addressing the inefficiencies that ad hoc problems are being addressed in an ad hoc manner, at an ad hoc point in time. There is no leveraging of one problem solution to another.

Verbauwhede: But we still can learn lessons from other fields. IC design is not the first area to suffer from counterfeiting. Other areas with this problem include designer-labeled bags or paper currency money. In the case of money, paper fibers may be checked to determine validity, and that's analogous to a component in our discussion. We may consider a PUF to be a component, and that is like biometry in other applications. Certain protocols, or procedures, are used that are similar. But the question here is, What is unique to this field that others do not have?

Potkonjak: What we have to realize is that the whole semiconductor market is worth about $300 billion. The financial industry market losses due to security breaches are about $1 trillion. So if we really want to have impact, we should do something with these hardware techniques for the financial industry or other system-level techniques in high-value industries (e.g., health, energy, social interaction). If we were to

create a solution for secure, trustable, and anonymous financial transactions using mobile phones, that would be much more beneficial than the most amazing Trojan horse detection.

Verbauwhede: True. That's the kind of problem we should be working on, and perhaps in this application we can find solutions. We should provide the primitives and protocols, possibly using variation-based or PUF-based techniques, to enable these types of features on smart mobile phones.

Gotze: But probably even with that system we'll still end up with a lot of system-level complexity. As an attacker, you're going to look at a system, and you're going to say, "Wow—there's so many different applications and threads running, lots of different things going on. I have a lot of opportunity to get in here, a lot of attack surface to exploit," and so you may move the problem by doing that.

Verbauwhede: Exactly. That's one of the characteristics of this field: the complexity of these chips. They're systems on chips or they have many components, including IP blocks coming from everywhere and then we have to put them together and all that that entails. The complexity is very high.

Potkonjak: Another point about these Trojan horses: is Intel, for example, scared of Trojan horses? I can't imagine that because they are manufacturing their own products. The only place that's really open for attacks would be the contract manufacturers or foundries, and even then it's unclear how vulnerable they really are.

Devadas: We're all familiar with hardware-software codesign and looking at performance, right? In codesign, we decide we want a certain level of performance for a particular application, and then we ask, what do we need to put into hardware and what do we need to put into software? Perhaps trade-offs are associated with this: for example, software is easier to build; hardware is harder to build. So we want to minimize the amount of hardware we build to get the performance we need.

What I mean is that if we have an application with a security requirement, we need to think about it from the standpoint of "Do we want to use hardware mechanisms, or do we want to use software mechanisms, to achieve the security?" There could be a performance conflict issue, which might mean we end up building a hash function in hardware rather than in software, as a simple example. What's more interesting to me is if there are unique hardware mechanisms, such as tagging information at an instruction level or potentially at the bit level for the data that's residing in the BIST [built-in self-test] structure and then comparing that with trying to do the same thing in software. It often turns out that the software will be easy to break or just be incredibly inefficient. Anyway, that's actually a security problem to decide how we're going to build this hardware-software system with a minimum trusted-computing base, with minimum complexity for verification to get the desired security for the given application.

D&T: We're basically building a trusted system from untrusted components.

Koushanfar: I must contradict Srini: so often when I hear about this software-or-hardware discussion, it's definitely true that more people attack software than hardware, and that just could be because more people know how to attack software than hardware. Whenever this discussion comes up, the context is to assume that hardware is secure, that software is insecure. This reminds me of Bertrand Russell who spent a lifetime finding a foundation for all the mathematics; but then it was realized that the primitive proposition itself contains certain assumptions that could not fit into a foundation.

People have already worked for years on building hardware that supports insecure software. Shouldn't we focus now on thinking about the next level? What happens if more people are trained in attacking hardware—won't we want to protect hardware in its own right, as opposed to just assuming that hardware is more secure than software because fewer people know how to attack it?

Devadas: Well, I won't contradict you because we have an intersection in terms of what you said and what I said. My point was that for a given application or a given direct model, there may be a more efficient way of using a hardware mechanism versus a software mechanism. What you're saying is—and it's true—at this stage, it's easier to attack software. More people attack software than hardware, and I see that as a side benefit of pushing functionality

into hardware. I agree that the benefit will disappear as people become more skilled at attacking hardware, but I still believe what I said earlier is what we should do, which is that we should find a mechanism (appropriate for a given application) that is most convenient and which is easiest to implement, and we should then maintain that threat model, which could be either a hardware or software attacker.

Verbauwhede: People know how to optimize for performance and they're going to trade off hardware-software for performance. Now I want to optimize hardware-software for security. One of the problems I have is, how do I measure my security? I want to ask my Intel security evaluation expert, "When are you happy? When do you conclude this thing is secure?" In the performance space I can say, "Okay, I'm at 90% of my target. Let's push the critical path a bit."

Gotze: To measure performance we have MIPS, power, timing, gates, gate count area—but in security there's no strong metric, and that's a big problem. I'd look to the research community to see if there's something that would help. Typically, though, we just hold architecture and design reviews and have a team member try to attack the design—the idea being that someone who is intelligently attacking something will naturally look for the lowest bar which we can then try to raise. Independent attackers will find the easiest way in; a designer might have considered one possible way but not all possible ways. So by having someone intelligently attack a design—they'll find the easiest way, and then we can score how secure a design is based on the lowest attack that it's vulnerable to.

Verbauwhede: I was on a panel recently discussing security and part of the discussion was on attacking smart cards or credit cards. One of the metrics used was how much does it cost to break the card's security? Apparently, one of the companies' levels is $25,000. Whether or not the equipment and the time investment an attacker needs to break the security is above or below that amount is considered a security metric.

Gotze: I've heard of similar metrics being used.

Verbauwhede: But this is measured in money: it's more like an insurance measure.

Potkonjak: But in this situation, the bottom line is that you cannot be a little bit compromised. Either you are compromised in hardware or you are not. That's very different from the metrics we use to optimize for performance in terms of area, power, and speed.

Verbauwhede: That's true. I think we agree: you're broken or you're not broken, but what we're trying to do is minimize the risk of being broken into.

Gotze: Counterfeiting is a good example of minimizing security risks. If it costs someone three or four times the amount to counterfeit a specific part than what they will receive from selling that part, then clearly it's not a good business. We can say we're worried about someone who rationally and purposefully conducts this attack as a business, and so from a business perspective, we might draw a line that way to say that we'll accept being vulnerable to some attack if it requires someone to irrationally spend more than the worth of the part being counterfeit.

D&T: But isn't that true for many other security problems such as hardware Trojans, and side-channel attacks? There's a cost element in every one of the defense mechanisms that we insert into a design.

Devadas: Yes, I'm a little more concerned about counterfeiting in the sense that I think it's a one-time cost, so I want to be careful in using those metrics. I'm not sure I agree with the counterfeiting example completely, because it may be that it costs $25,000 or $100,000 to take a network router, clone and duplicate it, but that cost may be amortized by the number of fake routers someone sells, so we've got to be a little careful in saying the counterfeiting costs significantly more than just one part and therefore we don't have to worry. In the case where someone steals a credit card, say, and clones it, and the victim discovers there are false charges on it, there's a time window. So the thief spent $1,000 to clone the card and then puts maybe $500 worth of purchases on it before the victim discovers that the card's been stolen. That makes sense. There's a couple of different ways we need to measure this; I don't think there's one single way.

D&T: I guess that it requires a statistical analysis to figure out: if a victim finds out the card was stolen, and if the victim never finds out.

Verbauwhede: And, again, we have primitives to address that. If I clone one of these credit cards I should only have a copy of one card; I'm not going to be able to copy and paste it. It's the same situation for network routers. If somebody copies one, they should only have one, and so even if they have one design, with the proper security, whether it has PUF-based or ID-based security, they own the key of just one and not a master key to unlock all cards or devices.

Koushanfar: But that's variable. Sometimes that one piece of hardware could be very valuable. For example, a recent virus, Stuxnet, targeted cyberphysical systems (consisting of physical actuation/control systems integrated by computing and networking) in nuclear facilities, and that's the first example of huge cyberphysical systems being attacked—the virus authors knew the physical details and configurations of the systems they were targeting. They knew everything that was mapping from software to hardware. If they had applied it to another type of cyberphysical system, the virus wouldn't have messed anything up; the virus was designed to target only certain classes of cyberphysical systems and nothing else. But they were targeting a few targeted costly systems, and it made sense to spend the time and effort to study all the details of those systems.

Verbauwhede: We are trained as engineers to make something to certain specifications, not to something unknown, such as an attack that we don't know from where it will come or to what it will target.

Potkonjak: It's well-known that up to half the electrical energy in a South American country is stolen, and if somebody calls the police to report the problem there's little that can be done since the thieves stealing the energy will kill or at least attack the police if they interfere. Solving the secure energy delivery problem would be a great contribution to many countries. Energy manipulation on higher levels of abstraction such as pricing may also be devastating. The current California budget deficit is greatly impacted by a $40 billion state loss due to manipulations in electrical energy delivery after deregulation. Many systems everywhere are increasingly interactive with us. Say you're in your car—if I do something bad to your car, you're in trouble. If you have a pacemaker and I do something bad to you, you're gone, not just in trouble. There are many ways to attack a system, and some might be done without our knowledge. For example, I can just put high radiation on your IC and cause it to stop working.

Koushanfar: So, Miodrag, I have a question. How many of these physical attacks are addressable by hardware methods that are of interest to the design and test community?

Potkonjak: It is an open research issue, but that's only a judgment, not hard data. So, I can imagine it's not that difficult to prevent some attacks using current hardware security techniques. However, much broader and deeper approaches are required.

Gotze: It's interesting that when people talk about hardware attacks, they're immediately thinking of physical system attacks, and on the security side, the response model has been mainly the smart-card model, where a single chip with a single thread running in that chip provides that function. From a processor perspective, such as mine, we're building large computing chips. We've got many different execution environments running in a system, and they all could represent different interests or different roles that need to be isolated through hardware, and so there's a concept that they potentially exploit, or attack the hardware—not through a physical presence, necessarily, but through a bug or some other weakness.

D&T: Srini mentioned a scenario in which trusted systems use untrusted components. One concept that I wanted your opinion on is commercial off-the-shelf (COTS) components. If we talk about our own design—yes, we can do something about it. We can have some security mechanisms in place to make sure that it cannot be easily attacked: a Trojan isn't going to be in there. Counterfeiting—yes, we can embed some PUF circuitry there. So we can do something about the design that *we* have, but if 90% of a system's COTS, then what? What kind of security problems do COTS components pose to system integrators?

Devadas: In an ideal world, if we can minimize trust in the computing base and have, for example, a COTS component that's a smart-card chip, we can build secure systems. TPM [Trusted Platform Module] is an

example of an effort to do that. We had a case where we tried to localize all the security functionality into one chip, and it obviously had issues in terms of deployment. The problem is that the amount of security functionality based on a TPM is tens of millions of lines of code, and we're trusting that. We have no way to truly verify that, verify the upgrades, and so on, and that's why people don't use TPMs in their laptops very much. If we view this situation as a COTS problem, one of the COTS components happens to be a smart-card chip. We need to do better than a TPM to solve the security problems, so the question is, can we create a COTS component that is secure or has security features in it such that it doesn't matter what other insecure, untrustworthy COTS components we use in building our systems?

Potkonjak: There are at least two answers to that currently. One is to push for the use of formal verification tools such as proof-carrying code (PCC) and its generalization. The other is what I consider a more practical technique: trusted simulation and trusted synthesis using untrusted tools. The idea can be easily understood by considering the following situation. When I was in mandatory military service back home in Slovenia—you can guess how that is, a bunch of 19-year-olds suddenly deprived of what they consider essentials, in a country about to disappear, doing all kinds of stupid things nonstop. Four people got killed in the first three months, but not in my unit and I wondered why. Then I realized our officer in charge had a simple algorithm, which was to make the soldiers do something—anything and often completely useless—nonstop. By just keeping us busy, we wouldn't have a chance to do anything bad.

So what I'm advocating is use of this security paradigm in hardware and system security: take a component and make sure all its resources (functional units) in each clock cycle are used by me so there aren't any available resources that somebody else can use to launch any type of attack.

Koushanfar: But what happens to efficiency?

Potkonjak: If we look at speed, there is no overhead since critical parts of the design are not affected. If we look at energy and power, we make sure that the inputs are changing only by one bit at a time so there's little switching and therefore little power overhead. We can further decrease the overhead if we can power down sections securely. The question is, can we make industrial tools to make this happen? We have several ways where we can start to fight attacks. One is during design; one is when the item is manufactured; and one is when it's in the field. The cost to fight attacks increases by several orders of magnitude as security measures are pushed toward the field. So the most cost-efficient way to fight attacks is when we design and assemble the system.

The key is that we must use current CAD tools. Current design tools are very complex, but we can't give them up since they increase designer productivity and software productivity by such a huge margin. What we have to do is to write checkers to verify if the tools' output is meaningful and secure. We need not try to make sure, for example, that the hardware Trojan horse is present or not. But we should make sure that each piece of silicon is used by us, or at least tested from time to time, and then nobody can insert anything to change the functionality.

Verbauwhede: These are two examples addressing the original question on security using COTS components. I believe in the same model. We must have a few components that control our rules of trust. That can be TPM, it can be a smart card, or some secure hash function, or another one of these special techniques, and we must have design methods in security. Then we need protocols to put components together, and we just heard about two examples: one is proof-carrying codes; the other is a technique such as keeping the components busy. But there are likely many more design tools and composition techniques for putting these components together securely.

Potkonjak: When we're doing development, the key is to minimize surprise. When we're doing research, the key is to maximize creativity and surprise. Design automation for a long time has largely been in development mode. A typical research approach has been to develop a better algorithm for optimizing a well-defined performance metrics or to apply a new synthesis flow to demonstrate a better design for a popular application. These are certainly valuable contributions; however, they rarely bring much surprise. That's good for CAD vendors, but not for new research directions. Hardware and system security research is now in a phase where we not only can

About the Participants

Srinivas Devadas is a professor of electrical engineering and computer science at the Massachusetts Institute of Technology and served as associate head of the EECS department from 2005 to 2011.

Kevin Gotze is a security validation engineer in Intel's Security Center of Excellence (SeCoE).

Farinaz Koushanfar is an assistant professor of electrical and computer engineering and the director of Texas Instruments DSP Leadership at Rice University.

Miodrag Potkonjak is a professor of computer science at the University of California, Los Angeles.

Mohammad Tehranipoor, our moderator, is an associate professor of electrical and computer engineering at the University of Connecticut, Storrs.

Ingrid Verbauwhede is a professor of electrical engineering at Katholieke Universiteit Leuven and an adjunct professor of electrical engineering at the University of California, Los Angeles.

David Yeh is director of Integrated Circuit and Systems Sciences at SRC, Research Triangle Park, No. Carolina.

afford to look for surprises, but where in fact surprises are mandatory for actual contributions.

Koushanfar: This question has surprised me since we started the discussion about out-of-the-box ideas for hardware security and trust. It's hard for a student who's starting to do research to think about out-of-the-box ideas. It's easy to say that we should come up with a surprising idea and a new primitive, and then everything is bright and sunny, but what do you think would be a good research direction, in particular for the D&T community interested in hardware security and trust? Doesn't coming up with new primitives require systematic effort?

Potkonjak: We can be driven either by new applications popping up now like, say, cell phone payment or smart grid, or we can be driven by new technologies and their properties, such as process variations and device aging. We can also receive our research impetus from actual hardware and system security attacks. Also, we can try to refine emerging hardware security paradigms and protocols so that they address new types of security tasks.

D&T: Let's change direction and discuss metrics. If we have tools to help us develop a secure design, what kind of metric are we going to use? If we want a tool to address some of the issues and we are dealing with certain attacks, what level of confidence are we going to have?

Gotze: Clearly, there are no guarantees. We need security as a business requirement. There's some risk and we need to mitigate it, so we engage in activities such as reviewing architecture, reviewing design, reviewing code, doing penetration testing with the idea to find as many issues as possible and fix as many issues as possible. This is the best method we have for doing this—and in trying to get lots of smart people with different backgrounds to work on it.

Devadas: We need a host of different metrics. The metrics for cryptographic primitive strength are well understood, and if a new primitive is proposed, typically it goes through a battery of tests. A good example is the SHA-3 competition. But when we get to the system level, it's much harder given the heterogeneity and variety of system architectures and threat models. I agree with Kevin that it becomes a business requirement and is a lot like software testing.

Potkonjak: Any security technique should have low impact on performance metrics such as power and speed, and more importantly, should have high probability or generate proof that some class of attacks is detected, diagnosed, and resolved. However, this is relevant only if the attacks are known. Some generalized metrics from testing such as observability, controllability, or their minimization may be fine starting points.

D&T: Thank you, all, very much.