Oral History Interview with

Paul Kocher

June 29, 2023

Video Conference

Conducted by Jeffrey Yost

Charles Babbage Institute

Abstract

This oral history interview is sponsored by and a part of NSF 2202484 "Mining a Useable Past: Perspectives, Paradoxes, and Possibilities with Security and Privacy," at the Charles Babbage Institute, University of Minnesota. It is an interview with Paul Kocher by videoconference.

The interview begins with Kocher's interest and experience programming prior to attending Stanford University, his interests in math and biology, and his goal to be a veterinarian. He relates summer jobs he had while at Stanford, first at software company Symantec and then at RSA Data Security. He discusses meeting Hellman at Stanford in his second year, support and encouragement from Hellman, and his participation as a student in a group at Stanford of Silicon Valley cryptographers.

Hellman referred consulting opportunities to Kocher during the early the growth of the Internet and Web, which enabled to Kocher to pursue cryptography as an early career. Kocher formed Cryptography Research Inc. in 1995, initially with just him doing consulting but soon adding others and branching beyond consulting.

Kocher discusses various projects, including his pathbreaking work with Taher Elgamal on Secure Sockets Layer (SSL) 3.0/Transport Layer Security (TLS) 1.0, a protocol to protect communications over the Internet. He relates how his knowledge and exposure to many areas like statistics without a focus in one contributed to his discovery of timing channel attacks and power analysis attacks (both categories of side channel attacks). The interview also explores the growth of the company, the variety of technical projects it did for clients, and how consulting led to opportunities to also explore other security research. He recounts the context of the Spectre paper. He also reflects upon the field of computer security broadly in terms complexity adding to vulnerabilities/risks and the economics of computer security. He highlights that he was able to work with many great people who together achieved impactful new technologies, techniques, and understandings in the field of computer security. Kocher tells of how, as the company grew larger, it needed to internally expand more of the infrastructure typical of larger corporations or be acquired by another corporation. The latter made more sense and Cryptography Research, Inc. merged with Rambus in 2011. Finally, he mentions how the success of the company and the merger allowed him to become more involved in philanthropy.

Keywords: Secure Sockets Layer (SSL) 3.0; Transport Layer Security (TLS) 1.0; Stanford University; Computer Security; Cryptography; Encryption; Cryptography Research, Inc.; Martin Hellman; James Bidzos; Side Channel Attacks; Timing Attacks; Power Analysis Attacks; Side Channel Attacks; ValiCert; Rambus; Spectre (paper).

Yost: It is June 29, 2023. My name is Jeffrey Yost of the Charles Babbage Institute. I'm here today, on a video conference, with Paul Kocher to conduct an oral history as part of CBI's National Science Foundation-sponsored project, "Mining a Usable Past: Perspectives, Paradoxes, and Possibilities with Security and Privacy." So, I'll begin with just some basic biographical questions. I see that were born in 1973, and you grew up in Oregon. Where in Oregon did you live?

Kocher: I lived outside the town of Corvallis. My father was a physics professor at Oregon State University. My mother didn't practice law, but she had a law degree, and I have two brothers, Scott and Zan, who live in Portland.

Yost: And pre-college, what were your hobbies, your interests in growing up, both in school and outside of school?

Kocher: We lived outside of the town, and prior to having a driver's license, there wasn't a lot to do. No cell phones obviously. This was back in the digital dark ages. But my dad brought home various computers, or pieces of computers, that he had at his lab or bought, including a Timex Sinclair ZX81. It was a little, tiny computer with a horrible membrane keyboard and 2 kilobytes of RAM (with a 16KB expansion). Later, we got an IBM PC/AT with a 6MHz 80286 processor that was spectacularly fast in comparison. I learned to program there from the bottom up. Except for an assembler and the built-in BASIC, we didn't have any software, just the hardware. So, using that I learned to program games, draw the Mandelbrot Set, or do things that seemed interesting. I ended up with a fairly strong low-level understanding of how these early computers worked from that.

Also, I did the usual sort of high school things. Swimming, tennis, and chess were the three activities I can remember doing in high school. I had some really great teachers in high school, including a physics teacher named Pat Canan, two really strong English teachers (Blou Carman and Judy McIlvenna), and a great biology teacher named Paula Minear. So, I had a handful of super inspiring teachers and got a solid education in school, but I also learned pretty well on my own, and my parents had lots of books around.

I got into Stanford, probably through a combination of luck and having done fairly well in high school with a fair number of kids that weren't doing really great.

Yost: That's great. I'm curious, did you have an interest in mathematics, or did you think at all about secret communications, or anything? Any thoughts about cryptography at all, prior to, going to Stanford?

Kocher: No. My plan was to be a veterinarian. I had worked in vet clinic when I was in high school and found the intellectual aspects of veterinary medicine really interesting and, for me, it has the emotional upsides of medicine, but not the downsides. So, at Stanford, I studied biology which was the standard thing to do before becoming a veterinarian. I took exactly one computer class, which was CS1U, which was an introduction to UNIX. I think had to take something to meet the computing requirement. So, at least from a formal education perspective, I'm completely unqualified for just about everything I've ever done.

Yost: So, I understand you met Martin Hellman in your sophomore year. Correct?

Kocher: Yes. I that's right.

Yost: Had you also worked summers at RSA Data Security prior to that or after that? Which came first?

Kocher: Let me think for a second. The first summer of school, I worked at Symantec. That was the summer after my freshman year. I learned a lot there, but maybe not the things I expected to learn. The CEO then, Gordon Eubanks, got in a bunch of legal trouble, and I was intrigued by how the organization responded. That summer they replaced all the carpets with a different color, which made the whole place smell really bad due to the carpet glue. And lots of other little things like that went wrong.

They had a database project that was supposed to be transformative for the company, and as a summer student I was tasked with doing builds. On the technology side, the code was being written for multiple compilers which created all sorts of challenges and was probably a role that should have been doing by someone with more continuity than a summer intern.

They also laid off a bunch of people when I was there. So, it was partly lesson in corporate dysfunction, if that's the right word. Still, watching it all come together was super interesting.

During that summer I also had some time to think cryptography as an intellectual problem. I had come across the cipher used in a program called PKZIP. You're probably familiar with zip files, and there was a cipher the designer had designed himself. It was the perfect learning tool in the sense that it was widely used, nobody had really looked at it, it was strong enough that it was hard to break, and it wasn't so strong that you couldn't break it. It struck me as a really interesting problem.

I still remember thinking about it, because it was very clear to me that I had enough information to find the secret — which in this case was a password that was converted to a key. But I didn't immediately see a way to find it that wouldn't take impossibly long to run on my computer. In the math classes I'd taken, once I had enough information to solve the equation and had gone through the process of solving the equations, actually doing the computation was trivial. So, for me it was a new kind of problem to have the ability to check the answers, but not a path to a computable solution. And I ultimately broke the cipher. The cryptographer Eli Biham helped improve the attack, and we co-authored a paper that was published in 1994.

Remember, this was pre-internet era. So, if you wanted to go read stuff, you went to library. Stanford's libraries were terrific and included the proceedings of various conferences in cryptography. During the summer, in my spare time, I read a bunch of these and found them really interesting and started learning about the theory and computer science underpinning computationally hard problems like breaking ciphers.

After school resumed, I reached out to Marty [Stanford Professor Martin Hellman] with a note saying that I was interested in cryptography. He referred me to one of his graduate students, Susan Langford. I met with her. Marty later has told me that that when he got a random email from a sophomore who wasn't in any engineering classes, he figured I wanted to solve cryptograms or something like that and had me meet with one of his grads student. And, apparently, I passed that test and I soon got to meet Marty.

So, Marty invited me to participate in a regular meeting at Stanford of people interested in cryptography. The regular participants were Marty, of course, and other people who were in the Bay Area who

were working in cryptography. So, Jimmy Upton, Peter Neumann, Tom Berson, and a handful of grad students. Drew Dean was also one of the people who participated regularly. He just passed away about a year ago, which is a big loss. I'm trying to think of who else, Susan Langford. So, there was a set of people who were working on various aspects of cryptography—it was a combination of social discussion, sharing notes, and ideas.

Beyond the technical discussion, that group was also really transformative because it gave me the realization that there are important things you can do with cryptography. It's not just this intellectual problem.

The summers of my sophomore and junior years, I worked at RSA Data Security Inc., which later was renamed to just RSA. Jim Bidzos was running the company, and Burt Kaliski was the head of RSA Labs. I don't think I first encountered Burt through the Stanford group. I think it might have been through on Usenet, where I'd mentioned at the bottom of posts that I was a student looking for summer work and Burt reached out.

I still remember interviewing with him, and he made an offer. He did some really complicated math thing in determining how much he would pay me for the summer, which he computed wrong – it came to just some tiny amount per hour, far below minimum wage. I remember asking him nervously whether that was intended—if that was what he intended or not. And then he realized he made a mistake. Then it was clear to me how he had done a very interesting short cut, but with an error. It might've been the only error he made the entire time I worked with him. He's a super, super nice, friendly, thoughtful guy.

Yost: What was the culture like at RSA Data Security Inc. at that time?

Kocher: So, there's often a difference in an organization between the person that you're working most closely with and the person at the top. This was particularly true at RSA — both Burt and Jim were super interesting and very different. Jim was like a bulldog. It's a fair description. I like Jim. But I also fully understand why some people don't like him. I've always found him a little bit intimidating. He's the alpha male in a room and will try to fill that role in a way. But he was the right person for the job - if there was anybody else running RSA, it would've failed. So, I have a respect for somebody who can step up and get a job done, but it was a job that involved stepping on a bunch of toes. And when Jim steps on a toe, he steps on it. I learned a bunch about patents and about how

small businesses work from watching him navigate challenges, and crises, and work to build the company.

Burt Kaliski is his opposite in some ways. He's very soft spoken. He's the kind of person when you're having a conversation with him, he thinks before he speaks. He's someone who might pause for ten seconds while thinks, then say something super interesting. So, there were very much contrasting personalities with Jim and Burt. At this point, RSA was just a handful of people. Burt's role was running what was called RSA Labs, and I think when he wrote me the offer for the first summer, he was the only person in labs, though Lisa Yin and Matt Robshaw joined about the same time as full-time employees.

The two summers I spent at RSA were interesting work. I found a flaw in the RC4 cipher that I wasn't allowed to talk about but got publicly rediscovered later. I found it when I became curious whether RC4 could be used as a cryptographic hash function if I provided the message as the key and used the first bytes of the keystream output as the hash. I ended up finding that the cipher had problems with related key attacks. I also worked on implementing various algorithms in assembly language so that we could help make their software faster. That was mostly x86 assembly, though I wrote code for some embedded processors too. So, it was really kind of a practical intersection between engineering and cryptography.

During my sophomore year at Stanford, I had connected with a guy named is Dale Christensen. He worked at Microsoft in the marketing department, and I did projects for him as a contractor (without ever meeting in person). He would send me CDs with try-before-you-buy, or pay-to-unlock software, on them. At this time, the CD format was relatively new. The amount of storage you could put a CD was orders of magnitude above floppy disks, which were how software was mostly distributed. One of the things Dale was responsible for was figuring out whether there were ways that Microsoft could sell software without having to ship somebody a box. The basic idea was that they'd pre-ship a CD with encrypted software, then let people buy activation codes to enable the programs they wanted. Try-before-you-buy was also attractive because there were a lot of people who didn't know what Microsoft's software could do and might be excited to buy it after they tried it.

Working from my dorm room, I probably evaluated 30, or 40, or 50 different of these designs and I think I broke every one of them.

And each one averaged several thousands of dollars of work. That paid my college tuition, between that and summer work.

So, this also involved, which we'll touch on a little later, the intersection between some cryptography, some engineering, some reverse engineering, and some low-level computer architecture. A try-before-you-buy system, especially if it's something which is applied in an automated fashion after the software is developed, is probably not going to be cryptographically strong. This is simply because the software has to work during the trial phase. In practice, there would often be some tricks to obfuscate the decision about whether the trial phase was over, so the breaks I came up with usually involved figuring out a way to modify the binary in a way that would reactivate the trial indefinitely or disable the restrictions that were put on it.

Pay-to-unlock could be stronger, so for these I got to do some cryptanalysis in addition to reverse engineering. As I mentioned earlier, I'd learned about PCs and working without manuals or documentation, so I was comfortable with reverse engineering code, and including taking a binary apart, figuring out what it did, and making it do something different.

Anyway, I digressed from your question which was about RSA. Do you want to ask more about RSA experience? You said you interviewed Jim Bidzos. I don't know if my description of him matches up with yours—

Yost: You get different perspectives from different people. In my interview with him I was pretty much trying to get at the early years of the company and the security software industry. Asking him about coming to a young company from IBM. A company that had developed some problems in terms of early leadership with the RSA group. I was asking him about how he installed some stronger and more defined managerial practices and how he drew on his prior experience at IBM. So, it, overall, was more of an oral history that focused on management rather than the technical side. In terms of his personality, forceful alpha male is my impression as well. I enjoyed interviewing him. He is a nice guy with a big personality, and an interesting and quite successful leader.

You remained a biology major. You graduated with a bachelor's in biology. As you were in your junior and senior years, were you still thinking you wanted to be a veterinarian after you completed your degree?

Kocher: Yes, that was the plan. But when you're at Stanford, and the internet is suddenly taking off, and you're young, and there aren't older people that have the experience to compete with you, and you're connected to people who are doing interesting things, well, plans have a way of changing. They should change.

The year I graduated from Stanford was about the same time that Marty Hellman decided to stop working on cryptography and focus full time on trying to reduce the risks of nuclear war. When he was approached about consulting projects, instead of doing them, he referred them to folks. I was one of the people on the list that he referred people to. And because I had this background in both programming as well as cryptography, and also was cheaper, I think, and probably had more time available than other people who might've been on the list, I suddenly had this stream of really interesting projects. And those projects led to other projects as well.

So, the idea of vet school went on hold. I told myself at the beginning that, "Okay. I'll get these interesting things get done first, and I can always push vet school off a little bit." I don't think I ever had a moment where I decided that I am definitely not going to vet school. But I think at this point, I can probably decisively say that I've deferred that long enough that I'm not doing that.

Yost: Okay. So, you graduated in 1995, and you formed Cryptography Research in the first year. Was it just you doing consulting work initially?

Kocher: For about a year, yes. Before I graduated, I had registered the domain name cryptography.com because there were these things called domain names, and I was interested in what it took to register one. And crypto.com was taken by Matt Blaze, but cryptography.com hadn't been taken, so I registered it. I had done this a few years before. When it came time pick a name for a company, I picked one that worked with the domain.

First it was just me doing consulting projects. I had some student loans and no capital but made more than enough to pay the bills. The first person I hired was Josh Jaffe, who had been a friend of mine in middle school. Part of the rationale was, okay, I have more projects than I can do. Also, the model for a consulting company is your revenue is your billing rate times your number of hours. At first, I was raising my rate, but it was pretty clear that if I had somebody else working with me and also contributing, I could bill two people onto these projects just as well as one, and that would

certainly cover his salary. And I kept the level of billing moderate, maybe about half of our time was billed projects, which left also quite a bit of time for interesting R & D projects and thinking about things that we might do.

The billed and unbilled projects were both really interesting. One of the volunteer things was the Deep Crack project, I forget quite what the timing was, but that was probably in the first year or two when a conversation with John Gilmore—

Yost: Before we get to that, I wanted to ask you about a project that you worked on with, a 1984 doctorate of Hellman's. Taher, is it Elgamal?

Kocher: Yes.

Kocher: He's also a great person to interview, if you ever want to—

Yost: Great I will keep that in mind. He was with HP Labs for about a decade, and then he became Chief Scientist of Netscape in 1995. And the two of you coauthored Secure Sockets Layer 3.0. Can you tell me about that project and how it came about and how it evolved?

Kocher: Yes. So let me think. So those are two or three things that lead to this. So, obviously Netscape was the unicorn of the time. Everybody knew what Netscape was. It was hugely impactful both in terms of the technology they were building as well as shaping the mental view of what the internet could be. The company, at that point, didn't have a clear revenue source, and I was told, a little bit before Taher called me, Gordon Biersch [a famed Palo Alto brewpub] had been giving free beers to engineers in exchange for an ad on their homepage.

So, Netscape was going from being an idea to becoming a real company. For this to be possible, one of the things that was very clear to Taher, and others was that they needed to have a way of instilling trust in the online world. An Internet with no sense of security or trust is fine for casual things, but not for real business or commerce. Netscape internally built a protocol called SSL Version 2, which they rolled out in the web browser, but it was getting broken in different ways on a regular basis because it wasn't very carefully architected. To be fair, it was better than many other protocols of the time, but still wasn't nearly as robust as it could be. There had been a *New York Times* article or two

talking about the dangers on the internet, partly in response to people finding flaws in the protocol and its implementation.

So Taher gave me a call. I actually don't know if Taher got my contact information through Marty, or if he knew me from when I was a contractor at RSA Labs. I'd also met him a couple times there through Stanford because he'd been one of Marty's grad students. So, there are a bunch of connections there. But he called me up and said, "We've got this problem with this protocol. Can you come and help us fix it?" Taher understood the cryptography really well and had a clear idea of what the protocol should and shouldn't do from a cryptographic perspective. While it was clear that everything that was trying to be done was feasible, there were still a whole bunch of things that needed to be designed or reworked to get the protocol to be something that would avoid criticisms in the *New York Times* or elsewhere.

In thinking about what should this protocol look like, it was clear to me that I was designing the first step in an evolutionary process, and that I wasn't going to be building a protocol that would last unchanged forever. I was trying to build a protocol that would evolve.

There were three of us working on the project. It was me, Phil Karlton, and Alan Freier. The deliverables were specification documents—there were two versions of it. One that was a text RFC for standardization, as well as a slightly prettier version that contained the same information. Also, in parallel, Phil was interfacing with work being done on the Netscape implementations of it, and helping make sure that we met the needs of the various Netscape groups and processes.

It was a fast project, maybe a few months. I don't remember quite how long the timeline was from when we started to when the document was done, but it was really fast if you compare to the time that is taken to make new iterations of that protocol, though obviously now there are a lot more stakeholders weighing in. There were a lot of decisions that got made, many by me, without any formal process. Phil's strength was on the engineering side, and he knew TCP/IP better than anybody else I've ever met. Alan was primarily doing the documentation. So, he was the RFC document and IETF guru but didn't have a cryptography background. So, we moved fast. I could choose an approach and say, "Hey, why don't we do it this way?" And sometimes, Phil would say, "Oh, no. Our software can't do that. We don't have a

license for that thing." Or "yes. Okay. Let's try that. Let's see how it looks."

Beyond the cryptography itself, a lot of the questions I was thinking about related to how to make a protocol that could be updated with backward compatibility, and how to enable evolution going forward. Part of what made this hard is that I didn't have a great model to work from – a lot of protocols follow the aspect of Postel's law that says to "be liberal in what you accept", but this is generally a really bad idea for security protocols. From a cryptographic perspective, the high-level concepts of starting with a public key handshake and then using symmetric crypto for the fast data exchange were well understood at the time, so a lot of the cryptographic pieces were simply following what was then known practices. Obviously, the question of what the algorithms are "best practices" has evolved quite a bit, especially on the symmetric side. We don't use RC4 and MD5 anymore. But that was the main set of algorithms at the time.

So, we came up with a protocol. There were a couple things that I missed that people caught later. And, of course, various implementations had bugs that were found. Looking back on that project, and how the protocol was able to evolve and keep moving forward, it worked better than I could have hoped. So, that was what I think was the biggest contribution that I made.

I also want to highlight Taher's role. Even though he's not listed as an author of the protocol, he had the vision for what that protocol could accomplish. He also really understood the importance of trust and the business side of the internet. He found the money. He assembled the team. I was one of the people that got brought in to do it.

Just after the protocol was finished, Microsoft introduced a competitive alternative that used the acronym PCT. I don't remember what it stands for, but there was a real fear that the internet was going to have the Netscape protocol and the Microsoft protocol, and they would be incompatible and different. So, there were big challenges to navigate getting the standardization process together due to the rivalry between Microsoft and Netscape. While it could have gone really badly, it didn't. I remember working with Barbara Fox at Microsoft, who was super pragmatic, though I don't know if she was the one who ultimately decided that Microsoft could agree to standardize on the Netscape design. While I think SSL v3 was the better protocol largely because of its

ability to evolve, Microsoft's willingness to have a common protocol was really important for Internet security.

The name TLS got adopted as a compromise between Microsoft and Netscape because Microsoft didn't want Netscape's acronym being used on the standard protocol. The text of the protocol the IETF [Internet Engineering Task Force] standardized was SSL v3, minus a US government requested Fortezza mode that almost nobody ever used. (Fortezza was a hardware care the NSA designed that included a key escrow system that had to be supported in Netscape's software, apparently due to an agreement between Netscape and the U.S. government — it wasn't something the broader Internet community wanted.) So, Netscape kept using the name SSL and the IETF standardized name was TLS. I don't think anybody to this day quite knows whether you're supposed to call it SSL or TLS or SSL/TLS—

Yost: Interesting.

Kocher: Being confused about the name is way better than having the internet fracture into two worlds that are incompatible with each other. So, that was a small price that we've all paid for that compromise.

One other anecdote from the standards process —I can recall one dinner where I took everybody who was involved from Microsoft and Netscape out and spent some astronomically large amount on wine for the group. Then the next day, Microsoft agreed to go with the protocol. So that was maybe the best investment in wine that I have ever made…

Yost: Was it clear from the start that there needed to be one standard or was there a time when it was seen—I thinking of the battles in the browser wars—that this was an area of competition, and the more secure and better protocol will win out in the browser competition.

Kocher: Well, the security of the protocol wasn't necessarily the main thing that would cause one design or the other to win out. What was more important was the question of what browsers support, what servers support, and if they work with each other. And we see this still on the internet. We see trends toward fracturing and compatibility happening now, where compatibility and having a common experience for everybody has both big advantages and disadvantages. If you're Microsoft, for example, and you can give your browser users access to some services that don't work with Netscape browsers, that would be an advantage. That drives a

pressure to fragment, especially if you can fragment where your users get the superior experience that other people don't. To their credit, Microsoft didn't try to do this with security. There are also good reasons to not fragment.
Metcalf's law basically says that the value of a network grows as the square of the number of users. If you have everybody having a common experience and being able to share information, this larger connectivity adds value. When we see countries try to partially or completely split themselves off from the broader Internet, they lose a lot because their isolated networks are smaller. Still, local control is attractive for some good reasons, though often it's just entrenched governments wanting to hide their corruption from their citizens.

I think it's interesting to think about the question of what would've happened if Microsoft had gone the other way. You could've easily had an internet, for example, where web servers would offer up a one port that was Microsoft browser compatible with Microsoft security, and a different port with different protocols that would work with Netscape. We'd have had a painful coevolution of features and capabilities, where users would have a different experience based on, in this case, different browsers. You could imagine a lot of splintering that might happen. That splintering would have transferred costs into other areas, with friction and probably no benefit to end users.

And I think folks realized this. I think everybody involved was aware that there were reasons why, tactically, a company, like Microsoft or Netscape, might well want to go in their own direction. Both had enough market power at that point.

But there was also a recognition that the pie for everybody would be bigger if everybody got along. The willingness to pick one protocol, even if it is named it in a way that wasn't Netscape's favorite, would lead to a much greater adoption for the Web. And the SSL/TLS story is one of many decisions that helped shape the formation of the Web.

I think, in this case, we look back at it, the decision that got made was the right one for Microsoft. If Microsoft had gone down that incompatible route, they would've not been as successful as they've been. So, I'm glad that worked out the way that it did, though I still wished we had one set of acronyms that everybody could settle on.

Yost: In 1996, you discovered timing channel attacks. Can you talk about the context and what led to that work and that discovery and its impact?

Kocher: Yes. I'll also talk about other side channel work as well, since power analysis is also closely related.

In cryptography, there are a lot of papers that have been written looking at how to break algorithms. Often, what you're trying to do is to find some kind of statistical non-randomness that you can identify, often with a tiny bias that's only visible from a great deal of input/output message pairs and computation effort. Linear cryptanalysis and differential cryptanalysis are both examples of these kinds of attacks, where you need far more data and computer power than a young person in their 20s like me working in their bedroom can come up with. So, I would read these papers but couldn't actually do the things they were describing because I didn't have access to the compute power to be able to actually implement them for anything other than reduced algorithms or toy examples.

In parallel, just from having done a lot of low-level programming and implementation work, I was aware of the timings of different computing operations. When you're trying, for example, to write a fast software implementation, the number of clock cycles each step takes is something that matters. So, it was clear to me that multiply instructions took different amounts of time on some processors, that different code paths took different amounts of time, and that access times were affected by caches and memory systems. When writing programs, I would think about these things to get better performance. And as I mentioned, I was also aware of what kinds of correlations were needed to implement non-side channel cryptanalytic attacks.

And I don't remember exactly how I connected the dots, but in hindsight it seems like a fairly obvious thing. I didn't have enough computer power to attack algorithms with normal cryptanalysis alone, but I saw that timing variations provided me with a great correlations that I could collect. I could get enough information to find keys from just dozens, or hundreds, or thousands of operations — as opposed to needing billions to trillions of messages.

I had taken a statistics class at Stanford, which was probably the most useful class I ever took. In fact, the textbook from it is on my shelf right there. I still use that book because it's the one that I know. So, I had a good understanding for how to take weakly

correlated values and tease out an answer. Again, it wasn't something where I had to invent any new math. I implemented the attack against a variety of algorithms. I think I only put a few of them in the paper in the end because there had been a page limitation, but it was clear that timing variations were a general problem that wasn't specific to any particular algorithm.

A bit later, I had just finished my undergraduate degree and was establishing Cryptography Research. One of the kinds of projects that I would often get hired to do was to and evaluate a product that somebody had designed, often just before it was about to ship into the market. For evaluation projects, so long as I could find weaknesses, clients didn't mind that I was young and had minimal credentials. These projects were also a great way to build relationships. Anyway, some of those projects were to evaluate hardware devices, like smart cards. I didn't have the equipment needed to go and take the chips apart and find secrets by imaging or probing them, and there were people doing that kind of work anyway. I found cryptographic flaws in some of them, but I was interested in more general attacks.

I went (to) go down to Fry's Electronics and bought the cheapest oscilloscope they had. It was analog, with a cathode ray tube and such a dim phosphor that if there was any light shining, I couldn't see the screen. But it was enough that I could see both timing variations (which I already knew were exploitable) as well as information being revealed in the power consumption.

In cryptography, there's an intuition that some information attackers are allowed to have, such as the plaintext and the ciphertext. If you can get access to any information you're not supposed to have — for example, any information about intermediates involved in the computation process that converts the input to the output — you can likely break the algorithm. There's this algorithm-level brittleness. Even today, algorithm designers often just assume that computations will be done in a completely sealed black box that reveals nothing else. Because I had mild competence with electronics, plus low-level programming and cryptography experience, this assumption felt wrong to me. And when I tried to check the assumption, I immediately saw things that I think had been missed by a lot of folks who were more used to thinking in a compartmentalized way.

Even today, the usual process is that an algorithms person writes a paper describing an algorithm. A programmer takes the algorithm paper and writes some software. A hardware person might design a

chip. An end user puts those together and suffers the consequences for whatever goes wrong. And because I had at least of little bit of experience in each of those areas, I was able to ask, "What are the implications for the hardware on the software running this algorithm?" I had a little bit more of a perspective than somebody working on just one of those. So, anyway, I published the timing paper in 1996 and followed a couple years later with the differential power analysis (or DPA) paper.

While I published these, there were a lot of other things that I was working on in parallel but didn't publish just because, not being an academic, there was never any incentive for me to write papers. So, I've probably written less than I should, and I feel a little guilty about that.

Yost: Can you please do go into some of the other work you were doing at that time that you didn't publish.

Kocher: Well, let me think. There were some other fun projects. There was one worth mentioning called the Deep Crack project. I was talking with John Gilmore, who was associated with the EFF, about a statement that was communicated to Congress where the NSA had argued for key length restrictions because of how many years it would take using Cray computers to break the DES standard. I don't know if laughable is the right word, but a Cray was the wrong tool for breaking DES. As a result, while it was maybe not exactly untrue, it was deceptive to measure the cost using a high-end supercomputer with really strong floating point to break a cipher which requires a very simple and parallelizable non-floating-point calculation to be done over and over again.

So, our conversation went from complaining that the NSA was being misleading, to discussing what would it really take to do to it, to John saying, "I can come up with $250,000, if you can build it." And I said, "Sure. Let's do it." There's a company I had done an evaluation for called AWT, which had experience and board design and working with chip foundries. I had just hired Ben Jun, who was also from Stanford a year behind me — a super smart guy. So, with him and me, we figured out the architecture for a chip, and through AWT we got a couple thousand chips made using a horrible semi-custom manufacturing process. The chips that came back were flaky, and there were a whole bunch of engineering challenges that I won't go into, but at the end of the day, we did come up with a machine that could break the DES cipher by brute force.

DES has a 56-bit key, so there are 2 to the power of 56 possible keys. That's about 72,000 trillion. It's a big number but not too big for semiconductors at the time, which is probably intentional because when NSA was working on the DES standard, they presumably didn't want something that was beyond their capability to break. A number of years had passed, so what NSA could have done when DES was proposed was, by that point in time with Moore's law and so forth, more accessible to somebody with some money but not a huge amount.

So, we built this machine. It was cobbled together. John found some chassis from old SUN workstations that provided power supplies, and AWT got the boards made. We ran them at AWT's office and caused the air conditioning system to ice up because we were drawing so much power and the cooling forced the AC to run all the time. So, it wasn't a super well-engineered process, but it was enough to demonstrate that on a shoestring budget you could break ciphers with a 56-bit key length. The broader intent was to show to people that breaking DES didn't require huge budgets or many Cray computers, but rather it was something that malicious actors, or in our case non-malicious people, could manage to do — and certainly any country could do it with no particular difficulty. So that played a role in demonstrating why we needed to have standards like AES with longer keys, or at a minimum use triple DES. It also highlighted the security risks resulting from export rules that limited key lengths.

Yost: That's fascinating and great context to the history of DES.

I'd like to move to the fixes side of the equation on vulnerabilities you were finding. So, were you, shortly after coming up with what you did with timing channels and power analysis, were you also working on fixes or design criteria so that they would not be susceptible to these types of attacks?

Kocher: Yes. In fact, there are two or three threads to that history. Especially with the power analysis work, these were happening in parallel.

First, early on, I filed a bunch of patents on techniques for protecting against power analysis. I had learned about how patents worked broadly from being at RSA. Also, one of the consulting projects I had done was for with Jay Walker, who was running an organization called Walker Digital, and he was also enthusiastic about patents. So, when I saw what I had found a problem, the question of "can we patent the fix?" certainly did occur to me, and

I did file for those. Patent licensing is an awkward business – chip companies deployed the countermeasures we'd patented but didn't want to pay us. Getting the licensing business going involved suing Visa and spending huge amounts on lawyers for several years to show that we were serious. Eventually, though, we settled with Visa, and worked out licensing arrangements with the major chip vendors. After a few years, they were making around 10 billion licensed chips a year.

Going back, there was also an interesting set of ethical and practical questions around when and how to disclose the power analysis work. Today, there is a general sense that when someone finds a vulnerability, it's responsible and appropriate to give companies 60 to 90 days, or maybe as much as 180 days, to fix it. At the end of that embargo, you can and should write a paper describing the results. The idea is to give companies enough time to roll out fixes while not keeping problems hidden indefinitely from users or other researchers.

But the DPA work took place before people had a sense for what the responsible disclosure process should look like. But even if we had known, it was super messy because you had lots of different smart card and chip manufacturers, you had lots of companies, like payment brands, that were rolling these technologies out. You also had, presumably, lots of hardware being used in government systems. So, who do you pick up the phone and call when there is something like this? It really wasn't clear. If you tell everybody who ought to know, you'll tell so many people that it'll leak. The appropriate timeline for public disclosure is also unclear, since this was hardware issue that wasn't easy to fix, and the security consequences for some systems were pretty severe.

In this case, the question of when to disclose got solved for me. Several of the companies that I was consulting with knew about it. And a reporter for an Australian newspaper, I think it was the Australian Financial Review, wrote an article about this attack that Visa knew about in San Francisco, and described enough detail that somebody who was capable would be able to figure it out. He didn't have the full story, but it was enough that the cat was out of the bag. So, right after that came out, we put up information on the Cryptography Research website describing how power analysis works, then proceeded to submit the academic paper.

Even in retrospect, it's still not clear to me how the embargoes and disclosure process for widespread hardware vulnerabilities ought to work, either in terms of who should be notified or what

timelines are best when there aren't straightforward patches to an issue. We'll probably talk about Spectre a little later, but the that disclosure process certainly gave me a feeling of déjà vu.

Yost: In the early years, was there any pressure from the government regarding disclosure? I know that early on, with public key cryptography that Diffie and Hellman and, in the early years of RSA Data Security, there were some run-ins with NSA people and some serious pressure if not intimidation on the academic crypto community by the NSA, and perhaps other defense and intelligence agencies also.

Kocher: Great question. I didn't run into any issues, though I also wasn't engaging with government much. And I think, probably, by the time anybody in government would've known what I was doing, that there would've been enough other people who also knew. Because of that, it's not clear what the government could've done if they wanted to do anything, but they didn't. More generally, I've tried to maintain friendly relationships with folks that I've met in government, but also never spent a lot of time focused on government. And I think one reason I was able to do that is that people like Marty and Jim Bidzos had been blazing the trails, in terms of defining what was okay, for example, to publish academically. And by the time 1995 rolled around, it was pretty clear that academia was going to publish openly about cryptography. The so-called crypto wars were going on, but they were mostly centered on questions of export control. And while there was always a possibility that Congress might do something like try to ban cryptography, but I never encountered any intimidation or attempts to classify any of my work.

That said, I did also tried to follow the rules. For example, I remember doing a project for Andersen Consulting (which later melted down), trying to help them build a system for sharing information. For that project, I applied for the export licenses for a floppy disk containing the software I'd written, going through the same process for exporting physical munitions, which all seemed a quite surreal. It wasn't my style to go get a tattoo with RSA source code on it and try to go through customs and say, "Hi. Am I allowed to go through here?" That said, I'm appreciative of the people who challenged the rules, and I also have some sympathy for people on the government side. But I've always gravitated more to the engineering aspects of security. So, while I didn't encounter issues, I do remember Jim Bidzos telling me a story about having a run-in with some NSA in a parking lot or whatever. You know, that—

Yost: Yes. He recounted that threat to me.

Kocher: —those were to me stories. They weren't the things that I experienced. So, I don't—

Yost: That's in my interview with him. So, I understand, concomitantly to the early years with Cryptography Research, you started ValiCert, Inc, as well for managing digital certificates. Can you talk about that?

Kocher: Sure. I met a super smart guy, Chini Krishnan, who was trying to create a company. If I remember correctly, it involved automotive supply chains. I remember spending a bunch of time with him trying to figure out the security problems that would have to be solved to make his idea work. Often, when somebody has an idea, it turns out there are a dozen things that have to get built to get there, and they can't build all 12 of these. However, sometimes one of those things might be broadly useful, and becomes the right thing to focus on. And so we ran into the problem of what to do when certificates need to be revoked. The approaches that had been proposed then were slow and unscalable.

I came up with an approach using Merkle trees to generate compact proofs of validity. We wrote software, filed patents, and Chini ran with it, including raising money and building the team. Ultimately, it went public. The idea was well before its time, plus Verisign was the dominant certificate issuer and they weren't really on board and didn't want to provide the revocation data needed to make it work well. However, it's fun to see that many years later there's now a lot going on with using Merkle trees for certificate transparency and other aspects of tracking certificate issuance and revocation.

ValiCert's IPO happened in an era when companies with no revenue and questionable prospects were going public. After ValiCert went public, suddenly in my 20s I had stock with a paper valuation of more money than I could imagine. I've never been particularly motivated by money, which sounds funny for somebody who's gone out and started companies, advised companies, and done a lot of things where money is pretty central. Still, I think it was good that I had to figure out the question of what money means to me. I realized I could go sit on a beach and not do anything, but that wasn't appealing. So I kept working on projects. And over the next while, ValiCert's stock went up and down, then eventually the company got acquired.

And that was an interesting learning experience, one where the insanity of some of technology bubbles doesn't seem to ever get learned. Cryptography, for whatever reason, has been a piece of a number of them, as we see now see with the cryptocurrency bubble—and maybe bubble is the wrong word. Mania perhaps. But I assume there will probably be another one in the future. And ValiCert's stock price was a little tiny one amongst others at the time.

Yost: Generative AI, of course, now—

Kocher: Yes.

Kocher: And with each of these bubbly technologies there's the question of whether it could actually be a huge thing in the future or is it a thing that can't possibly pan out. While some of the math in cryptocurrencies is intriguing, I'm confident the ones being touted now will never remotely live up to the hype, and nobody should be putting their money into them. In contrast, AI can clearly do some useful, and sometimes scary, things…

Yost: Yes. I think the question with generative AI is more how it should be regulated and controlled, perhaps—

Kocher: Yes, or if will it be regulated and controlled—

Yost: Right. There can certainly be many negative consequences to generative AI, especially differentially experienced across the population.

I'd like to ask just a broad question. In prep for this interview, I came across you mentioning that the real problem is complexity leading to vulnerabilities. That is something that came up and was a core point of emphasis in an interview I did probably in 2014 or 2015 with SRI's Peter Neumann. And he emphasized that strongly as well that, we needed to think about the problem of complexity. If we are really going to have more secure systems and reduce vulnerabilities, reducing complexity is critical. So, can you talk about your ideas on complexity?

Kocher: Yes. I think the first is what gets taught in a normal engineering class. You get an assignment describing some functionality you're supposed to create. But security is not functionality. It's more often the absence of something, specifically the absence of certain kinds of functionality that you don't want. For example, if there exists a

set of packets that someone can send to your computer that will cause it to send out all of its information and then wipe your machine, that is functionality that you almost certainly don't want.

The thought processes around preventing unwanted functionality are fairly different from the ways of creating functionality. Complexity, when you're trying to create functionality, is almost always helpful. It lets you build layers of abstraction where you can provide a simple interface to something that's enormously complicated, and somebody who understands this interface can get the functionality's benefits without having to understand all of the details inside. If we didn't do this, we would go crazy as engineers because we would be constantly thinking about minutiae. If you're trying to build the dialogue box and thinking about every transistor in a PC, and your brain would explode.

But the difficulty from a security perspective is that complex logic behind these simple interfaces really can have some enormous implications for security. And there are different ways that can manifest. A very simple but important example is software bugs, like buffer overflows. If there is an error like this in a piece of code that's doing some tangentially minor thing in a big complex system, an adversary can often take advantage of the bug to modify memory arbitrarily within the process address space. That bug is a piece of bad functionality that can compromise the whole system. You get a scaling effect where, as the number of transistors, or lines of code, or whatever metric you want to use, increases, the chance of there being bugs goes up, or you could say the average number of bugs goes up.

One mental model for security is the absence of exploitable bugs. This is a probabilistic notion. If you've got a certain bug density, more code complexity means more chances to have bugs. Complexity also tends to increase the consequences of bugs. So, if put 10 times as much important stuff into a system, which is 10 times more likely to have bugs in it, and attackers will get 10 times the reward for breaking it, then far more people will try to attack it — and so on. You have many dimensions of scaling that all are going against the outcomes that users want, such as to have their data not be exploited and to have their systems operating reliably.

But the benefits provided by Moore's law also drive huge economic and user wins, so you can't just say, "Okay. Let's do everything with physical, mechanical computing machines. Let's get rid of all these transistor-based things and go back to the good old days" because the old days weren't good either. So, we have to

grapple with the hard problems that keep getting harder. Figuring out what do we need to build today in order to cope with tomorrow's risks is super difficult.

And so far, I'm just talking here about the complexity of individual pieces of a single system, but you add opacity when you pull pieces from different sources together. I don't know exactly what Intel has done in their chips. Intel doesn't really know what Microsoft is doing. Neither of them knows what Adobe is doing, and yet each is relying on each other and making assumptions about the other.

Often the resulting failures are fairly uninteresting when you look back in retrospect. You say, "Okay. This programmer made a bug here while they were writing this line of code, which affects the program's security in this way, and this bug needs to be fixed." I've been involved in lots and lots of those. While the systemic drivers of these mistakes are important, and finding the specific bugs is hard, most of the actual bugs are not really interesting individually. So typically, they aren't worth writing papers about and hopefully just get fixed.

Occasionally, though, I've found issues that are more academically interesting or non-intuitive. These usually go beyond cases where a programmer made an error, and instead there's an issue that involves conflicts across layers of abstraction. The timing and power analysis side channel work are examples of this, such as where effects at the transistor level are causing variations in the power consumption of the transistor. This transistor is a piece of microprocessor. This microprocessor is running some software. The software is implementing an algorithm with a secret key that is protecting some business function. And, through this all these layers of abstraction, these transistor effects are leaking information that ripples all the way up and compromises the security needs of the people using the system.

All this is breaking down these layers of abstraction that we depend on for our sanity, which is academically interesting and has practical implications as well. If you would like to make a chip for a payment card, for example, you clearly don't want a malicious terminal to be able to pull all the keys out and steal people's money. So, there's a practical piece but also an interesting academic one.

The Spectre paper that I co-authored a few years ago was also an example where security issues arise from the interactions between

layers. The components in a processor can all be working the way they're supposed to work. You have speculative execution in processors. You have caches. Individually these are well understood, but if you put these together, the combination is a system that doesn't have the security properties that you intended.

As compared to the usual complexity-induced bugs, issues like Spectre are interesting to think about academically. But whether the issue involves a localized bug or an unexpected interaction, the issues ultimately manifest in a failure to achieve the security objectives that people had in mind.

As we think trying to achieve security, bug densities are important to think about along with complexity. I also should mention that, as you add complexity, the number of bugs may scale faster than linearly. The reason is that bugs often come from interactions between components. If you double the number of components in a system, you've got approximately 4 times as many pairs of components. If those components have some unexpected or improperly analyzed interplay between them, there's often a security problem. You may well find that the risk is not scaling linearly, but instead increases with the square of the the number of lines of codes, functions, logic blocks, or other measure of system complexity.

This doesn't mean that we're going to face a sudden crisis that will appear, say, next year. People have been coping with this for a long time. But it also doesn't mean that you can assume that the things that have kept the wheels on the bus so far are going to keep the wheels on in the future. We have to step back and rethink about every order of magnitude in complexity growth. We have to consider what's working, what's not. In some cases, the things that people have been spending a lot of money on really don't scale.

And to pick an example where, right now, we've got a lot of unscalable security processes as organizations try to comb through a flood of alerts. However, if the difficulty of getting through a day's data becomes a million times harder, you can't hire a million times more people to do this. Even if you could, or if AI agents can do the work we well as people, you may potentially have a million times as many false positives that annoy users, and a million times as many false negatives where attacks get missed. So, one problem organizations face is that their coping mechanisms aren't scaling. The other issue is that, instead of spending to fix a problem, they're spending a great deal of money on things where, the next

day, they have to begin the process again. But we don't really have a road map for how to deal with these challenges.

We aren't doing a good job of fixing issues. We haven't even fixed some of the really old problems, like memory safety. If you've got buffer overflows or use-after-free bugs in your software, it doesn't even matter what fancy cryptography you do — you're insecure and easily exploitable. To be able to find solutions that scale for some of the bigger challenges lies ahead, we've also got to first find ways to really address and solve some of the major, well-understood problems that we've been struggling with for a long time. The unscalable coping mechanisms and band-aids that worked for a while are a real problem.

Yost: A couple weeks ago, I interviewed Jean Camp, and about a decade ago, Ross Anderson, and they, along with Hal Varian were central in launching the academic field or specialty of computer security economics. In 2000 or 2001, I believe. I'm wondering, as you're talking about complexity, and risk, there's organizations making decisions, there's sometimes externalities where you don't face the consequence of the problems you cause. And I'm wondering if you could just broadly talk about computer security economics, and is it something that you've given a lot of thought to and how your thought has evolved on the theme of computer security economics over time?

Yost: So, I'm not an economist, though there are clearly information asymmetries and both microeconomic and macroeconomic pieces to this puzzle. One set of challenges are clear when you're trying to run a company, or fund a project, or accomplish something that requires paid labor and people to go work on something. Unless everyone is doing it purely as a charity project, you need to convince funders or customers to give you money. However, security involves complex probabilistic issues that are hard to explain or quantify. In practice, you're likely to be competing against companies that are lying or misleading. For example, if you go online and look at security claims from even big reputable companies, you'll typically see a concerted effort to convince the people they are trustworthy and to give an impression that their systems are robust when we know from the history of bugs that they're not. The cynical – and often correct – interpretation is that this is a deceptive sales tactic.

There's also a piece of this—I'll come back to the economics in just a second—that even experienced security engineers are often

too optimistic. You see this all the time where people will go build a system that they don't know how to break, and then assume that, if they don't know how to break it, nobody else will know how to break it. At some level, this is the way we have to work because we can only protect against the things that we know about. At the same time, this can go very wrong, especially if you have somebody who has a very poor understanding of how to break something.

When I was doing security evaluations, this optimistic thought process was also useful. If you know something other people don't know, or as if you have tools in your toolbox that are more sophisticated than the customers have, you'll find vulnerabilities that they don't expect. And one of the fastest ways to gain credibility is to show a security team in a tactful way that they missed something and help them fix the issue.

One of the big economic questions at a macro level is the one of how regulation ought to work. Because of issues like information problems I just talked about, Bruce Schneier and others have described the security market as a "market for lemons". From a regulatory perspective, I sometimes think almost anything would be better than the status quo in many ways.

Although we still have a lot of innovation that's happening and you don't want to break that, I would love to have more sharing of information needed to understand, prevent, and mitigate risk.

We now have software update mechanisms for fixing bugs that are pretty widely deployed and relied upon. The internet as we know it would not function without regular software updates to fix and mitigate vulnerabilities before the adversaries scale up the attacks. But this reliance has an unhealthy side, since it means people can always kick the can down the road rather than doing the harder engineering thing to address or prevent risks.

A number of the projects I've worked on have been hardware chips. Unlike software, you don't have this ability to say, "Oops. I screwed up. Let's push out a software update." Instead, you'd be looking at a multi-month redesign and then a colossally expensive recall and replacement of hardware in the field. Even in applications where replacements are anticipated, like pay TV systems with removable security smart cards, it's colossally expensive. So, learning to make things work robustly upfront is something that I have put a huge amount of my life into trying to figure out. And while it takes more up-front work, particularly to

keep complexity under control, it's a lot cheaper in the long run. So, for certain kinds of problems, I don't subscribe to the idea that "it's all broken so don't bother".

Back to the economic question, another piece of the puzzle is that the preventative technologies that are important for society often aren't necessarily the things that make much money. Technologies, for example, that eliminate classes of bugs are hugely impactful, but often don't make a ton of money for the people that build them. I'll pick SSL 3 as an example, since it's something I worked on. I got paid for some consulting hours by Netscape, but it wasn't very much. If I were to go back and say, "Per hour, where did I deliver more value to society — that or working on security mechanisms for pay TV companies?" Clearly, SSL has the much bigger societal impact, but many of orders of magnitude less money. Which for me is totally great. There's no complaint here at all. If I got paid too much for one thing and perhaps too little for another, I should consider myself deeply lucky. But from a broader perspective, big disconnects between impact and funding are a problem.

For example, the efforts to build memory safe programming languages like Rust tend to be charity projects on the side. Vastly more is spent by corporations scanning their networks trying to observe the consequences of people having not used memory safe programming languages to begin with. So, we do a very poor job, as a society, of allocating resources to the things that would ultimately be effective. "An ounce of prevention is worth a pound of cure" is the old adage, but we're not even spending an ounce of our money on these high upstream high-impact efforts.

Hardware is another neglected area. The security properties of hardware provide the foundations for absolutely everything. If the hardware is bad, often, there's not much you can do in software. Yet if you look at the number of companies that are doing hardware that were funded by VCs in the last year, especially outside of AI acceleration, it's a small number. It's a really small number, and it might even be zero, for security hardware. I'm trying to think if can name one, and none are coming to mind. So, we're not investing very well, and that's partly an economic question again — of what are the risks and needs vs. rewards and revenue. The same sort of disconnect happens after something goes wrong, when the penalties companies pay for exposing user data is allocated for credit monitoring services instead of much more useful things like better programing languages to prevent breaches in the first place. As a society, we'll spend more on building one physical highway or bridge than we will on open-source projects

trying to tackle big security issues. I hope some of those things may change over time.

That said, even though the allocation of resources is pretty poor, there are still are a lot of great projects that are helping even if their budgets are tiny percentages of the overall total. It helps that, due to the scaling and complexity issues, we are spending hundreds of billions of dollars a year on security. So even if 99 percent of that's wasted, people can do some really neat stuff with the other 1 percent.

Let me pause there. You asked a question about economics, but I went in six directions. Do you want me to try to answer the question a little more?

Yost: No that is great. I meant for it be fairly open-ended just to see what you wanted to talk about and major part of this project and the book we will do is to look bigger picture issues.

I have done some research on early on with access control systems, like RACF and Top Secret, a number of organizations were acquiring these but not even using them. There was a piece of regulation in 1977 as part of the Foreign Corrupt Processes Act that if an enterprise or organization was doing international financial transactions, they had to have an access control system. So, there were a number of organizations that would purchase RACF and either not use it all, or they would only protect five percent of their data with it. So, in doing that research, and also in interviewing Jean Camp recently and Ross Anderson a while back, the decisions that corporations as well as government make with regard to security, what economic factors they consider are of great interest to me.

Kocher: Yes. And you touched on the question that regulations, how they come with a cost and hopefully some benefits too?

Yost: Yes.

Kocher: There are some good examples of what not to do. There's a standard called FIPS 140 that this has been around for a long time, mandating a set of requirements for cryptographic systems used by government, and it requires a bunch of documentation and processes. And companies had to go through the process of getting certified. They'd build a system, hand it over to a different set of people who would go produce a bunch of paperwork, then still other people would read the paperwork and approve the product.

Money gets spent, the system gets approved, and the product didn't get improved. Afterward, the product doesn't get updated because that would force re-certification. So, you end up with a process where no code gets changed, and it becomes harder to fix the bugs afterwards. That's a high cost, often with little or no benefit.

There are other sorts of metrics that might be more useful to focus on. When I use a piece of code in a security product, one of the first things that's going through my mind is "What's the complexity and bug density of this software I'm integrating?"

As a starting point, I would love it developers would self-disclose information about their understanding of the risks in their software. Even voluntary self-disclosure would be helpful. Even if a vendor says, "Here's my million lines of code. There are no bugs in it. I'm sure it's perfect." You immediately know that it isn't. You've already answered the question that you wouldn't trust it, right? We have much better disclosures on companies' financial documents than we do about their software. Maybe someday we'll have a GAAP accounting type process for software disclosures...

Anyway, I'm on a tangent here. The broad challenge is that we will need to swing more toward the prevention side as complexity grows, because with scaling cost monitoring and response procedures becomes unmanageable. I don't quite know what the exact technologies will end up looking like, but it won't take long for complexity to scale by another order of magnitude. Spending on monitoring and response approaches can't scale to handle that, so something has to change, which is exciting and scary at the same time.

Yost: In 2011, Rambus acquired CRI, and I'm wondering if you could tell me about the context and the decision there and how many employees did CRI have at that time?

Kocher: Yes.

Yost: What was your decision process, in being acquired at that time and can you go into the contexts of that acquisition?

Kocher: Well, we reached this awkward stage where we needed to build a bunch of corporate functions that we didn't have or join somebody who had them. There's a scale where accounting is a computer running QuickBooks and a person who knows how to pay bills, and there's a scale where you need a department to oversee internal budget allocations and such. So, there's a scale where IT is

engineers making sure the systems they need stay up, and there's a scale where you need a 24-hour help desk. There's small-business HR that involves payroll and health insurance, and a scale where you have employee handbooks, and processes, and rules, and compliance. So, we didn't have much corporate infrastructure, but were scaling our revenue and size to point where we needed for it, and we were building products and systems that would make this need even greater. So, the question then, was should we scale, or should we be acquired by somebody who has these functions?

There was also the question of scaling growth. We had been doing hardware projects, but there was more demand than we were meeting. We also wanted to handle than just the on-chip logic, for example systems for programming keys in factories, and helping secure manufacturing more generally. So, there were a whole bunch of things that needed to be done and we had lots of things happening.

One of the options was to get acquired, and we got an offer from Rambus. The details of it are public, and it was a large amount of money. And also, they were an interesting company, having both expertise in patent licensing, which was a big chunk of our business and theirs, and they also we're trying to get into offering products and branch out in new growth areas. We had another company that was also interested in buying us as well, and they had an offer that was pretty comparable.

At that point, the team was about 30. The process was one where we ran a little bit unusually. Everybody in the company was informed and trusted to know what was going on. I shared with each person what an acquisition would mean for them financially, and polled everybody in the organization, asking, "If this happens, this is economically what it means for you. Do you think, personally, you'd like to do this?" I gave people three options — yes, support whatever decision Kit, Ben, and I (who were the management team) favored, or no. Everybody voted either yes or to go with our consensus.

I tried to structure the acquisition also in a way where it was good for everybody, and also did a couple of things that were a bit unusual. For example, we negotiated with Rambus to put a fairly significant amount of the deal proceeds into what we called a retention pool. An amount was designated for each person on the team. If the person stayed employed, they would receive the money over a period of time, but the unusual aspect was that if they left voluntarily or were let go, their money would go to

charity. So, there was no incentive to fire anybody, but there was also no incentive to keep anybody around who was not pulling their weight. And I wish lots more agreements said, "If we can't agree on X, the resources that we're arguing over go to the Red Cross, or the US Government debt relief fund, or some other neutral entity that doesn't benefit either party."

Yost: Yes. That's a great idea. I haven't heard of that before. Is that something that you saw somewhere else, or did you come up with that?

Kocher: I came up with it, and Laura Stark, who was the person driving the process from Rambus side, was like, "Yes. That sounds great. Let's do it."

Yost: That's terrific.

Kocher: Even though it was unusual, the team largely stayed together and remained engaged even after the retention period, though one a researcher did leave to pursue a PhD.

Yost: Right.

Kocher: In that case, the charitable portion resulted in about a million dollars going to the IACR, the nonprofit that organizes academic conferences in cryptography, which created a fund to support students. The approach meant that, when Rambus did some layoffs shortly after we were acquired, there was no economic incentive to factor in the retention funds. Otherwise, you might worry that someone in an acquiring company might say, "Person X in the newly acquired company will get a big paycheck but, if we lay them off, we'll save that money." Well, there was no incentive to pick, or not pick, anybody in particular. I think it worked great. I think I would use that structure again, but I don't know anybody else that's done that.

Yost: Yes. That's fantastic.

Kocher: So—

Yost: Go ahead.

Kocher: Yes. So, we were acquired by Rambus.

I like to work on technical challenges, but there were a lot of other things that needed attention. I did that for a few years, and learned

about a lot of things that I hadn't encountered before. For example, internal budgeting was really different from what we did as a small company. Before, everybody would spend money on things pretty rationally. There wasn't a process where the manager of a division would ask for more money than they need in order to build their department, or protect their people from layoffs, or pay their group better than other groups. In contrast, budgeting in a larger organization is a lot more complex.

In retrospect, it's the another aspect of some of the scaling problems I talked about earlier. At some size, fighting internal battles becomes important, but in a small organization fighting the external challenges is the important thing.

I don't mean to be critical of Rambus. They were, overall, pretty good with a lot of really capable people. Still, Rambus went through a couple CEOs — the CEO at the acquisition was replaced fairly soon after, and the person who replaced him actually got replaced after I left. So, there was some politics, which were an interesting learning experience for me. But it's not where my passion is. And I learned that I will never run a division of a publicly trading company again.

Yost: I understand your division grew to about 200 people. Can you talk about your, style and principles and approach to managing people?

Kocher: I should probably begin by saying I'm not necessarily a very good manager. I'm mildly competent at a lot of things, as opposed to really good at any single thing. And my style works pretty well with 15 people. If you are working with me, and you run into a technical problem, or there's something else going on, I'd like to be able to sit down, spend a few hours to go to lunch, connect, and figure out what can be done. When teams were spread around the world, with more people than I could get to know, lots of things happening, and too much information for it to all be shared super well, I didn't enjoy that, and I'm not sure I know how to manage that. I don't feel like I'm in my element if I have more people than I can check in with over a couple days. I don't know. The Peter Principle that says people tend to get, get promoted beyond your level of competence, and sometimes I felt that way. I don't actually know whether others felt that way or not, but there was definitely a realization for me that this isn't where my strengths are.

Finance was another area that changed more than I expected. I don't love accounting, but I have a deep appreciation for the need to have revenue for a project to be successful. With a publicly

traded company, it's a lot more complex -- you've got different categories of money and different ways of presenting things. Prior to be acquired, I don't think I knew when one quarter started, and the next one began. It wasn't a date that was of any particular significance. Even year boundaries didn't matter much. But to suddenly I was in a world where the end of the quarter was a big deal -- that's when things start, that's when things stop. And quarters are surprisingly short. If I told you that you were going to win the lottery six weeks from now, or six weeks and a day from now, you probably would be like, "Great. I'm winning the lottery." You wouldn't be saying, "Oh. Well sorry. The six weeks and a day is after the quarter boundary, so you have to make sure that you get it done the day before." So anyway, a lot of attention had to be paid to about things that were important, but fairly far removed from my strengths in technical problem solving. So while I stayed well past the retention period, eventually I left Rambus in, I don't know, 2016, maybe, 2017. I forget exactly the year.

Yost: 2017, I think.

Kocher: Okay. I haven't done anything since with full-time employees. I've had contractors that worked with me, and collaborators on projects, but I've enjoyed not having as many responsibilities. Which probably says something more about me than anything else.

In terms of management, I some really strong colleagues. There were two key early hires that I made at Cryptography Research. One was Ben Jun, who was a Stanford engineering student a bit younger than me. He had lot of enthusiasm and knowledge and talent, but initially not a lot of domain experience in cryptography. I learned a lot in parallel with him.

Also hired Kit Rodgers, who had played football for Stanford and handled the non-technical things for the company. When I say "non-technical things", it sounds like I'm denigrating it, but that meant building relationships, and figuring out office space, and dealing with all of the sales, marketing, and handling that. And he also is a fantastically capable person.

So, we had the three of us who had radically different personalities and skillsets that were together running the company. Management books would usually say having three people trying to run a company at once is a recipe for disaster, but each of us had different areas of strength. Which is a nice way of saying each was basically incompetent at the things that the others were good at doing. I couldn't build the customer relationships Kit created. Ben

was making sure that the engineering deliverables were of fantastically high quality. And I was charting out where we could go and driving the innovation forward.

We might have different ideas of how to solve a problem, but that usually meant we hadn't thought through the solution well enough. Still, we have really different personalities. Kit is very extraverted, and people oriented. Ben is very worried about making sure things come out right, and I'm super excited about the possibility on the technical side. So, it worked really well.

Of course, I think there was a lot of luck in that we were so different. When I advise—I do a lot of advising of companies now—I'll see companies where people with the same skillset have gotten together and decided to go create a company doing the thing that they're all really strong at. And they generally all have the same blind spots and the same strengths. Any decision that comes along, they're either all unable to make the decision, or all equally capable of making it. In contrast, for any decision that came along, one of us would know what to do or have a good idea, and the others would be like, "Kit, you know this. Great. Let's do that." And there wasn't much overlap.

Yost: That's great. So, in your incredibly distinguished career to date, you have many honors, including being inducted to the Cybersecurity Hall of Fame, the National Academy of Engineering. What have these things meant to you, and can you reflect a bit?

Kocher: That's a really tricky question. Somehow, it's the hardest one you've asked because I'm not somebody who sought that. An example is the recent RSA Mathematics Award — but I don't think of myself as a mathematician. It'd be more accurate to say I'm a failed veterinarian. What am I supposed to say when a bunch of mathematicians decide to give me an award, but yet even in the papers that I've written, the math was, well, the math from my college statistics class. So, I don't know.

So, while neither money nor awards are things that interest me very much, I ended up with both of those, which is slightly puzzling. On the money side, I do put energy into philanthropy and giving it money away, and that's great. I enjoy doing that, though it's surprisingly hard to do well. And money is a useful tool for accomplishing things. Awards aren't as directly useful. I can't go and say, "Okay. Thank you for this award. Now, I'm going to go

give it to somebody else who I think is really deserving." So, I'm not sure. The things that I've worked on, I've done them because I found them interesting, or because I like the impact they would have, or I thought there was problem that needed to be solved.

And this conversation has been a lot about things I've worked on, but I should highlight that almost all of these were team efforts. If you think of projects as pushing a boulder up a hill, my role has often been to say, "Oh. Here's a boulder, and there's the top of a hill. Wouldn't it be nice if that boulder was on top on the hill?" And other folks have agreed and helped push the boulder up. I've enjoyed getting teams together and working with the folks who have done the real pushing.

I think awards would have mattered more when I was young, but generally awards are given too late in careers. Right now, if there is a security project I want to do, I have a pretty good presumption of credibility. I can call somebody up I don't know and say, "Okay. Here's who I am. Will you go spend a few hours with me listening to this idea?", or "Will you take the meeting if I come to visit you?", or "Will you respond to my email." And they often do. Even early in my career, my connections to Stanford opened key doors. There are lots of people who don't have that access. I wish there were more ways that we could give that to people earlier in their careers.

I've also had a lot of luck. In addition to the more obvious sorts of luck, I had the good fortune that, of all of the tools in the toolbox for computer security, it turns out the cryptography it is one of the few that are deeply effective. For a certain set of important problems, cryptography basically eliminates whole categories of attack. If I give you a cipher like AES, it does what it's supposed to do. It's really, really unlikely that somebody will break it with a head-on mathematical attack. That enables cryptographers to have high confidence in certain aspects of the security. In contrast, with other kinds of tools, there are a lot of soft areas and gaps. So as a result, cryptographers seem to get more awards than people working in other areas.

I was also fortunate to be being a person with knowledge across a bunch of areas, entering a field at a time when it was embryonic. This meant that there was a lot more under-explored territory for interesting results than there might be for somebody who's studying cryptography now. And someone today with a biology degree who planned to be a veterinarian and who started looking for consulting work in cryptography would find a different world

that I did. It doesn't mean it's not possible that somebody might make it work, but now there are lots of PhD and undergraduate programs, online courses, textbooks, and just a lot of stuff that wasn't there then.

And it's great that it exists. If you want to find somebody who can write code, who knows a little bit about hardware, and has kept up on the cryptographic literature, your menu of people when I was starting out you could probably count on one hand — outside of government anyway. Nowadays, the Real-World Crypto conference has maybe 500 people who are looking at these kinds of problems. But, by being early, I sometimes came across things that I assume other people would have found later. Or, in the case of the Spectre paper, I found the issue independently from Jan Horn at Google Project Zero, then we each found out about the other's work during the embargo process. So, I drifted off your question a bit...

Yost: No. It's a useful response. Before we conclude, are there any topics or themes that I haven't brought up that you'd like to discuss?

Kocher: I don't know. We covered a lot of territory here. I was about to turn that question back at you, but you've been asking the questions so far, so I assume you've gotten through yours.

Yost: Yes.

Kocher: Let me think for a moment.

Kocher: I should highlight again the importance of all the people that I've worked with. For most of the projects I played a part in, other people did most of the work. If I look at the SSL/TLS ecosystem today, the percentage of the work that I did is a rounding error that rounds to zero, probably even if you go out to several decimal places. If I look at everything that I've worked on, there's lots and lots of super capable people who've done so much — and for different projects, it's often been different people. So, I would like to highlight the importance of that.

For anybody who is thinking about trying to build a career in security, I've found it's been really helpful to have familiarity across a lot of areas. I know enough to have a conversation about accounting, or venture capital, or business, or hardware engineering, or cryptography, or mathematics, or software engineering, or debugging, or hardware architecture. A lot of the people that I spend time with are experts in one of those but

haven't been exposed to the other areas. I wish we had more people able to know four or five areas somewhat, rather than pushing everybody to be the deepest in one topic but unfamiliar with the other ones. If I look back, almost everything I've worked on has usually pulled from three or four of those areas that I listed.

Let's see... We covered most things. I mean, there were a lot of technical projects that I worked on that we didn't dig into, like the BD Plus project that's part of the Blu-ray format, or security for various pay TV systems, or work on anti-counterfeiting chips. These were interesting, had their own impact, brought money in, and kept the company going and funded.

There have also been projects I've worked on that failed. I'll pick one, we called Fax Archive. The idea was to build a service where you could fax a document to the service, and it would store it for you, time stamp it, and fax you back a receipt as a companion to traditional notarization. That was one of the first non-services ideas that Cryptography Research explored. We never launched it. I'm laughing no, realizing, wait, fax machines. And we thought this was good to be a business even though email was taking over?

Part of why I mentioned that project is there's often a survivorship bias when people talk about their work. I wouldn't want somebody to read this interview and assume I was drifting from one thing that worked out to the next. Nobody sees the things that didn't launch. If you were to look at the whiteboard of ideas that I've floated and even started pursuing, there were so many bad ones. And maybe we were a little better at filtering some of them out sooner than others, but most of the time it hasn't felt like path was clear. There was an awful lot of muddling and backtracking and errors, and screw ups, and bugs caught at the last minute, and discoveries about things like how chip foundries' processes differ from what people assumed they did. And each of these surprises led to rearchitecting something, scrambling, work late at night, and that kind of thing.

And I didn't begin to list all the ways that luck mattered. Between having awesome educated parents and being at Stanford at the right place in the right time with the Internet taking off, with hobbies that happened to become relevant, and people willing to take a chance on a young guy who didn't have any qualifications. So I can give the world's most useless advice, "Just make sure you're at the right place at the right time and catch the right wave."

Yost: Make it past the graduate student to Marty.

Kocher: Exactly. But I'm super fortunate—

Yost: Right.

Kocher: Lots of luck in it.

Yost: I greatly appreciate your generosity with your time today, and your insights and participating in this NSF-funded project. I'm so grateful. In about probably a month or two, I'll send you a transcript. And we will have it transcribed, and then I go through and correct what I see, in terms of transcription mistakes and then, you'll have 45 days to make any deletions or edits you want.

Kocher: Thank you so much for doing this project. It's tickled a lot of old neurons that have not been fired for a long time for me. So, I don't tend to reminisce, so this is my, maybe, first time ever doing this, thinking about a lot of these things in years. I super enjoyed it, and deeply appreciate that you're capturing these interviews. I will go back and look at some of the interviews you did with Marty, and Jim, and some of those folks that I have enjoyed spending time with, but never heard their histories told in a thoughtful way. So, thank you very much, and super enjoyed this. I really appreciate you taking the time.

Yost: Yes. Well, thank you and have a great rest of your day.

Kocher: Thanks, Jeff, bye.

Yost: Bye.

[End of Audio]