SQRL Revisited

Description: Security and privacy-related news keeps coming! So this week Father Robert and I will cover the past week's many interesting events. Then we revisit the much evolved and nearly finalized SQRL protocol to see how it has grown and matured during the 92 weeks since I first disclosed its concept during Podcast 424 with Tom.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-516.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-516-lq.mp3

SHOW TEASE: Today with Steve Gibson: Hacking Team follies, UEFI nightmare. Can exceptional access exist beside strong encryption? And everything you ever wanted to know about SQRL but were afraid to ask. Security Now! is next.

FATHER ROBERT BALLECER: This is Security Now! with Steve Gibson, Episode 516, recorded July 14th, 2015: Hacking Team vs. SQRL.

It's time for Security Now!, the show that covers your privacy and security online with the one, the only, the incomparable Steve Gibson from GRC.com. That's right, the creator of ShieldsUP!, SpinRite, and soon SQRL. Steve, so good to see you again, my friend.

Steve Gibson: Well, Father, this is our third and final podcast together. And, boy, based on the feedback from the last couple, you're definitely a hit as my co-host. So anytime Leo feels that he needs to take a break from this, I know that you'll be welcome back.

PADRE: You know, it's easy to take me in small doses. But trust me, if you got too much of this, it would start to wear on you.

Steve: Well, I actually sort of designed this podcast for you because what I want to do is catch our listeners up sort of formally on what the last 92 weeks of development of SQRL has wrought.

PADRE: Very nice.

Steve: And so today's topic is SQRL Revisited. It was Podcast 464, that I did with Tom back on October 2nd of 2013, that I first told our listeners about this. And there are still a lot of people who - actually I have to say that I wasn't as expert at explaining it then as I hope I am now. But also, so much has happened. Essentially it's gone from a concept to a very well sort of finished, polished result. So I thought I would really love to walk you

through it, to go through it with you, like just, well, first of all I want to completely have you up to speed on what this is and, in the process, bring our listeners current because all kinds of loose ends have been nailed down, and the protocol really polished. So that I want to do as our main topic.

And of course we've got a week's full of crazy news again. We've got more revelations from the Hacking Team. We've got that news that dropped, it sort of straddled last week's podcast. We heard on Monday that OpenSSL had a highly critical announcement they would be making several days later, which did drop on Thursday. Got to talk about that. A really interesting letter was written by 11 of the industry's top crypto experts. Again, this is another one of these letters to the government, like, look, guys, really, really think about whether you want to destroy the Internet's encryption ecosystem before you go any further. And they make a couple very good points. It's a 30-something-page letter that I read, and I pulled just some key parts out of it.

More bad news from the OPM breach, the Office of Personnel Management that we've talked about several times, but something just horrific came to light. Some little bit of update on Windows WiFi, Adblock Plus, and miscellaneous tidbits. And we're going to catch everybody up on the last 92 weeks of SQRL development.

PADRE: Steve, I have got to ask, what is happening? I mean, I have gone long stretches for This Week in Enterprise Tech, and there's just a trickling of security news. But these last two, three weeks have just been filled. Is it just because we're getting closer to Black Hat and Defcon?

Steve: You know, what it seems to be, frankly, a lot of this has come from the Hacking Team breach. I forgot to mention that this is also Patch Tuesday. So this is the second Tuesday of July. Microsoft dropped eight, or I'm sorry, 12 bundles of patches. And among them are a large collection of IE updates, the standard IE rollup of things. But among them is another zero-day that was found among the hacking team exploits. And there's also a Java zero-day. There's Adobe, this is their big patch update. So they had Flash and Shockwave and Reader, like across the board. So I just think that a lot of this has been Hacking Team. But, you know, as you said, we do have Defcon and Black Hat coming up soon.

PADRE: Yeah. Steve, where do we start?

Steve: Well, before we start, I always do a Picture of the Week, or I try to, on the first page of the show notes. And this week's picture is the now official SQRL logo and icon, which most people have never seen. I tweeted it yesterday. For those listening, if you go to GRC.com/sqrl/logo.htm, that'll just take you to a page that I put up just yesterday. This is the result - I'm really, really happy with the logo that we got. This was the result of a month-long design, an international design competition that I ran on a site called Logosauce, which I've had success with, and some friends have had success with in the past. A Spanish designer came up with this, and we refined it and worked on it over the course of a number of weeks. And it's just exactly what I want.

So this is the official SQRL logo. Many people are breathing a sigh of relief that we didn't actually get stuck with that little chipmunk face that I have been using for the last year and a half. And I always said, no, no, no, that's just interim, that's just so that we have something. So that's the real one. And I tweeted it yesterday and got nothing but rave reviews back from my Twitter followers.

PADRE: I so like it. It's a nice combination of sort of impressionist art and recognizability. I mean, I know it's a squirrel; but then it's got, I don't know, it's got a geek feel to it.

Steve: It's got, yeah, it's got the other elements, too - a lock and a padlock and its tail and, yeah, I think it's just right.

PADRE: And if you turn it sideways, it looks like a symbol for the Illuminati. So that should attract some attention.

Steve: Ooh, I wonder if that was on purpose. Okay. So we've talked now for the last couple weeks about Hacking Team. And I have to say that I'm just - for those of you who don't know why I'm laughing, Father Robert is rotating the SQRL icon.

PADRE: No, that's all JammerB.

Steve: Oh, that's JammerB. Thank you, John. To see what it looks like in different positions. That is kind of odd. Anyway, so this Hacking Team data disclosure, this 400GB of data, one of the problems with 400GB is that it's like a flood. There's, I mean, how do you actually parse, find something amid that? Well, as it turns out, this team really did have a lot of heretofore not publicly known information. And so in the news since we last caught up with them last week is yet another zero-day, meaning previously not publicly known, but presumably sold to their government and other private entities, zero-day Flash exploit.

The FireEye guys found it amid the data. And they wrote: "The Hacking Team leak already resulted in the public disclosure of another zero-day Adobe Flash vulnerability, which was quickly adopted by multiple groups and used in widespread attacks." So they identified amid the data another vulnerability that was essentially buried in this 400GB of data, and immediately notified Adobe. So this is the second one of those.

We also, as I mentioned before, this is a Patch Tuesday for Microsoft. Actually it's everybody's Patch Tuesday. Oracle's got a Java patch to fix a Java zero-day which was not part of the Hacking Team data, but also came to light recently. IE, one of the things that IE is fixing is a zero-day that was discovered in the Hacking Team's data and presumably will be jumped on as these other Flash updates have been. Mozilla quickly moved to block, initially, all versions of Flash. But it turns out it was all versions before .209 which Adobe has released.

And when I updated my Firefox, I mean, I've got it blocked, so I had to, like, push through multiple layers of "Are you really sure you want to run Flash," but I was on the About Flash page. I always just put into Google "about Flash Player," and the first link is a similarly worded link on Adobe's site that just runs some Flash that shows you the version of the Flash Player, which is the easiest way to see what version you're using. I fired up Chrome, and it already knew about that. It had patched itself, as it does.

So 18.0.0.209 is, you know, everybody who listens to this podcast knows that, one way or another, you should not have Flash runnable on your browser. Many sites, unfortunately, still use it. And some don't work without it. I mean, the only problem I ever have with my iPad is that of course it doesn't support Flash. I have to say, though, I'm noticing that's less and less a problem.

For example, there's one really neat nutrition information site, it's nutritiondata.self.com. And they're like the only one I can think of that, because they use Flash to show, like, some graphical information of the amino acid spectrum that various nutritional foods have, and also they have a triangle where the three points of the triangle are fat, protein, and carbohydrate. And so they have that little graphic that is a Flash object. But come to think of it, fewer sites now than I think a couple years ago are still running Flash because

it's like IE4. You know, everyone knows it's sort of over, and we're trying not to use it; but boy, inertia is powerful in this industry.

So one way or another, for example, if you put into Google "about Adobe Flash," and go to that page, you don't want that little 3D cube to spin all by itself because that would tell you that your browser is running Flash just when you visit a site. You really want, like, all kinds of warnings to come up, and where you have to push your way through it in order to say, yes, I'm really sure I want to run Flash in this instance. And I had to do that several times. NoScript blocks it. Then when I enabled it on NoScript, I got another warning that came up from Firefox itself that said, uh, really? And I said yes, you know, I'm sure this particular one is safe.

So in this day and age, this is, for example, these two zero-day exploits were immediately deployed for installing file-encrypting ransomware, which is like the worst thing that can happen to you is that all the files in your drive get encrypted, and then you've got to pay, or maybe you've got a backup that you can roll back from. But nobody should be running Flash with, like, just browsing on random sites because we know that ads are being hosted by them, and Flash is being injected onto benign web pages in order to infect you.

PADRE: Steve, I have to ask, as you said, as you pointed out, this is a matter of inertia. These sites have been using Flash for such a long time, and it is a relatively easy way for you to display rich content and to hook into third-party organizations that may display ads on your site. But when do you think we finally say, okay, well, this is just too painful, we're done with this?

Steve: There is nothing today that JavaScript plus HTML5 cannot do. That is, Flash was early in as a way of giving us more media-rich content. I mean, what's the crazy game that everyone's talking about that was Flash-hosted until just recently? Minecraft; right?

PADRE: Right.

Steve: I think Minecraft was Flash-based.

PADRE: Well, Minecraft is Java, actually.

Steve: Java, that's right, Minecraft is Java. But there have been a lot of online games that have been Flash-based. And it was the way of playing videos for a long time. I mean, YouTube was originally Flash player-based playing, and then they switched over to HTML5. And JavaScript was kind of creaky and immature, and it didn't have the hooks into the object model for pages that it has now, which really, I mean, you can do everything you need to with JavaScript and plain HTML. Well, we say "plain"; but, I mean, it's dramatically enhanced over what it used to be.

So, yeah, it's just inertia. It's that there are many people who know, like web developers, learned Flash back in the day, as they say, and that's what they still do by default. They have to be pushed away from it. And things like, well, in fact there is this site, OccupyFlash.org, that was apparently just created, I'm not sure how old it is, but it came to my attention after this second zero-day in a row of really bad, you know, immediately exploited in the wild Flash vulnerabilities. OccupyFlash.org is all about just like saying, well, they call themselves "the movement to rid the world of the Flash Player plugin." And, you know, so they're cheerleaders for this.

As I said, I think we're seeing, for example, we're seeing some evolution on the web. I'm

noticing more sites telling me that I'm not running JavaScript because of course I have JavaScript disabled by default, and I turn it on when I need to. It used to be that sites would silently break, and I'd have to go, oh, this is not working because I have to let the site use script. So then I would turn it on selectively. Now I'm seeing that sites are telling me, you don't have scripting on, this is going to work better if you do. So that's an example of websites adapting to the reality that there are a population of people that are security conscious and recognizing that scripting is, while it's necessary, it's also a means for vulnerabilities to operate. So they're adapting.

And similarly, I think as Flash no longer has the, I mean, there's no advantage to it. If you were starting from scratch today, you would never use Flash because there's a huge learning curve. And if you're going to learn something, you might as well learn tomorrow's technology, which is JavaScript and HTML5, rather than yesterday's technology. So it's just - it's going to drain off of the Internet, but it's going to be slow. But it's certainly helped in terms of its drainage by these kinds of problems, the idea that this Italian Hacking Team has this repository of non-public, functional, zero-day exploits that they're selling to people. I mean, that's why they have them.

PADRE: Well, you know, Steve, there's another angle there, and that is, this is one security research firm. It's infamous now because of everything that's happened.

Steve: Good point. Good point.

PADRE: And because of what they've done in the past. But this is one. And so if this is their stockpile of what they were able to discover, you have to imagine that security firms around the world have done the exact same thing. They've got the few things that they've discovered that nobody else has, that's unique to them and their tools. And they're going to sit on it, or they're going to sell it to the highest bidder. But there is actually something else I want to bring up about this because there were some people in the chatroom who were saying, wait a minute, are you saying to get rid of all scripting? Because there are developers in there who are saying scripting's never going to go away. It's always going to exist in one form or another.

Now, I agree with you, Flash is a pig; it's a hog; it's an outdated technology. It probably should go away sometime in the very near future. But I'm wondering how much of this is the case of the movable feast. Now, right now Flash is the most vulnerable. JavaScript is the most vulnerable. And so of course we're looking at that saying this is bad; this is bad; best practices don't use it. But when we move on, the feast moves with us; and people just say, okay, well, they're not using Flash anymore. What are they using? That's what we're going to use our research dollars on.

Steve: Yeah. I think that one of the differences is that Flash, very much like Java, is from a single entity that controls it. Oracle produces releases of Java, and Adobe produces releases of Flash. These are both old technologies for the web. And they're apparently fixing them as problems are found, much less proactively than the leading technologies. We've got, and we talked about it last week, the forthcoming bytecode for browsers. I mean, I am bullish on web-based applications so long as they really get the scrutiny that they need. And JavaScript has received a huge amount of attention because it's tomorrow's technology. So its fundamental technology is being scrutinized carefully and being developed, essentially in public view, by a team of people working to make this the platform for tomorrow. Which is very different than sort of a has-been technology that no one would adopt today if they were starting from scratch.

So, and notice also that normally JavaScript is only the way these other things get run. That is, JavaScript is itself more of a means of entry for these other plugins to get them

going than itself such a problem. And unfortunately we do see a lot of laziness. When we're talking about, for example, script blocking, oftentimes it's not the script on the site that you need to block, it's the fact that the web developers have allowed 40 other domains' scripts to run on their page, just sort of out of laziness. It's like, well, you know, these people need this library, so we'll go load this couple hundred K of script that they can't make any affirmative representations about the quality of.

PADRE: Steve, what about the user who's going to say, okay, I understand this, and I keep abreast with security concerns. But I still like my sites that use JavaScript and that use Flash, so I'm going to run NoScript. I'm going to create a whitelist of sites that I do allow, and I'm going to be very careful about what sites are enabled for Flash, and that way I'll be secure. Would you say, "Well done?" Or would you kind of shake your head and say, "Well, no, you still can't guarantee security that way?"

Steve: There is no absolute guarantee. We're in a gray zone. So, for example, I have scripting enabled on Amazon because Amazon needs it. And I have it enabled for eBay and PayPal and the high-reputation sites where hopefully any problems will get found quickly. Is that perfect protection? No. So it's possible to encounter something malicious on a site with a good reputation. It was a Hugo Boss ad a few months ago that was notoriously installing some of this inscription malware on people's machines because somehow their ad got infected with something malicious. And so a reputable site could have been serving that ad, and the site and the sponsor of the ad are reputable, but still something got in. So, yeah, there it is.

PADRE: It is very well-dressed malware, though, just so you know.

Steve: Yes. So, you know, my feeling is we have multiple levels of defense. You should not run Flash just gratuitously when you go to a site. One way or another you should be using a browser which is keeping your Flash current, as Chrome does, or keeping it demand-based only, as Firefox will do and NoScript does. I would say don't run scripting if you're concerned about security. Again, it's an inconvenience, I recognize that, to just say, oh, I have to enable scripting for this site. If you don't care, fine. But you are lowering your defenses incrementally. I don't know how much because there are other things that are going to protect you. Hopefully the site is being scanned for malware. They're finding malware. The ad servers are hopefully checking to make sure that anything malicious is found. You know, we have many different approaches to making our experience secure. But the problem is stuff still gets through. So I would just say run with as much of your shields up as you're comfortable with. More is probably better.

PADRE: More is better. Scripting is fascinating. Flash, the zero-day bugs are very interesting. But what I find even more interesting, Steve, is the next story, again involving the Hacking Team.

Steve: Yup.

PADRE: But this time with UEFI. This, I won't say it terrifies me because I've actually done a little bit of research into it. But it is a very interesting approach to malware.

Steve: So we did a podcast a few months ago on UEFI. And I laid out the mechanisms - in fact, I think you also had me on your podcast.

PADRE: I was there, yeah.

Steve: On TWiET, I laid that out again. The extreme measures that UEFI goes through

for the so-called "secure boot process," where from the moment it starts, it verifies its own firmware using certificates which are burned into ROM or firmly placed at a lower level than the BIOS firmware. And every stage of the boot process is verified first, before control is turned over to it.

So you have to ask yourself, okay, how then can you have a UEFI rootkit? Well, you can have it if you are not using secure boot. It's that simple. And in fact, what was found, as you said, among this Hacking Team data, was that the Hacking Team offers to the customers of their RCS 9 - we talked about that last week or the week before. RCS is their Remote Control System, which is their so-called RAT, their Remote Access Trojan that allows - that we know that they have been selling to the U.S. NSA and the DEA and other law enforcement in the United States and globally, and unfortunately a little too globally, to some repressive governments to whom a company based in Italy should not be able to legally sell their products.

And so one of the features they offer, a piece of this is a means for that technology to survive the change of hard drive or reformatting of the hard drive. There's a popular BIOS used in laptops. HP uses it, Dell, Lenovo, Acer, Asus, and Toshiba. It's made by Taiwanese company Insyde, I-N-S-Y-D-E. And that's their UEFI BIOS. Apparently it's believed this could also function with AMI's BIOS. And what this does, you have to not be using secure boot. And it's believed that you need to have physical access to the laptop in order to install this, although people who have looked at this have not ruled out the possibility that it could be installed remotely.

When it comes up, it's got three modules. One is an NTFS file system module that allows it to read, gives it read and write access at the BIOS level - not in the OS, in the BIOS; one for hooking the OS boot process; and another that checks to make sure that RCS is present on the file system which is running in the OS which is on top of this UEFI. So together those three modules check for two software agents - one is scout.exe, and the other is soldier.exe - every time the system is rebooted. And if they don't exist, it reinstalls scout.exe and soldier.exe from a predefined location inside of itself. So even if you swap the hard drive and install a new OS, a new Windows OS because that's what this infects, this thing emerges from the BIOS and reinstalls these programs and hooks them in to operate.

PADRE: This is like the old MBR viruses that we used to get back in the '90s.

Steve: Yup.

PADRE: The whole idea of you could reformat it; but, if you didn't get rid of that one piece, it would just reinstall itself. But Steve, UEFI has a checksum; right? I mean, it knows how big it's supposed to be. So in the extraction of the firmware, this rootkit is going to install itself. Then it has to redo the checksum to match, and then it pushes it back into the UEFI memory?

Steve: Well, okay. So one of the problems with UEFI is its power. Back in the day we had, in the early days of the PC, the XP and so forth, we had a BIOS, the Basic I/O System. And it was basic. I mean, it occupied a small amount of ROM. It was originally ROM in those days, not even flashable. And it did, you know, it read the keyboard. It could put text on the screen. And, I mean, to tell you how old that was, it was able to do cassette I/O was also built into the BIOS. So, I mean, and it ran the printer port and initialized the serial ports and sort of performed the low-level hardware startup.

What we have today is a mixed blessing because the hardware platform has just gone crazy with amount of RAM, with all kinds of options and advanced interfaces. It's a much

more sophisticated platform. You even have things like a TCP/IP network stack built into the motherboard, even without the OS running on top of it. So if you're running Internet protocols on the motherboard, then you're dealing with something far more sophisticated. Well, UEFI, the EFI BIOS, is a modular BIOS in which you can install additional components in order to suit the needs of the hardware platform on the motherboard. So, I mean, it has a file system of its own.

So, yeah, I mean, it's designed to accept additional modules. And you have to register them and let the UEFI BIOS know that these modules are to be run at runtime, but that's part of what this installation process does. There's an installer that runs on a USB thumb drive that installs, that performs all of that work of installing this rootkit into an unprotected UEFI BIOS so that these modules run. And Trend Micro, that covered this story, said - they asked themselves the question, how do you prevent this? And the answer is enable UEFI secure boot, update the firmware to its latest version, and use a strong BIOS UEFI access password. And then somebody who had physical access to your laptop would not be able to access the BIOS and could not get in there because secure boot would be turned on.

And of course this is why we're moving towards secure boot being turned on by default, and maybe not something that end users are even going to be able to disable. We'll have to see how that resolves itself.

PADRE: I have to say, Steve, after I saw this story, I went through the different laptops in my lab, and I've got a current HP, a Dell, a Lenovo, a couple of Acers, an Asus and a Toshiba.

Steve: All of them. All the targets.

PADRE: All of the targets. Only one laptop, and that was the new version of the Acer S7 that I got, had secure boot turned on. Everything else had it off by default, including another one of my Acers. So, yeah, that's not yet the standard. But let me ask you this. I'm intrigued by two possibilities. One is the idea of remote execution of this exploit. How could you possibly get access to the UEFI memory space while the OS is running? Because that's the only way that's going to work, if you're trying to do this remotely. And secondly, when we're talking about secure boot, it's a software switch that keeps you from writing to the UEFI space. But how safe is that from actually being turned off remotely?

Steve: Well, we just, to answer your first question, we just covered about a month ago that flaw in the BIOS that was leaving, essentially, huge portions of the BIOS read-write when it should have been read-only. I think it was in the Mac; right?

PADRE: Right.

Steve: And so there's an example of the firmware itself, despite all efforts, the firmware itself having vulnerabilities which are not widely known. So given the clear expertise of these guys, I wouldn't put it past them to have come up with a way to circumvent these BIOS protections if the BIOS is not absolutely protecting itself from modification. And actually...

PADRE: And "Digmax" in the chatroom mentions, and I completely forgot about this, Dell actually has some machines that have built-in out-of-band access to BIOS. And it's designed for enterprise deployment so that you could update the firmware without having to go from station to station to station. That's a horrific infection vector.

Steve: And the Intel, the high-end Intel motherboards also have low-level, below the OS, out-of-band access to the hardware, which opens up all kinds of frightening possibilities.

PADRE: Okay, well, I'm not going to sleep anymore. Thank you. Thank you for that, Steve.

Steve: Okay. So OpenSSL. During last week's podcast there was something pending about which nothing was known officially. On Thursday, two days after last week's podcast, the news dropped. And the good news was, despite the fact that they considered it a high severity problem that needed fixing, many mitigations in practice were in place. For example, it was a problem that had only been introduced, like, four weeks previously. There was an update on June 11th to the two later versions which will be moving forward in the future, and that was 1.0.1 and 1.0.2. Both of those threads of development were updated only four weeks before. So many people hadn't had a chance to be affected by it. But sort of the old school versions that we've all been using, which is 0.9.8 and 1.0.0, they weren't affected. Now, those end up getting phased out, the 0.9.8 and 1.0.0, at the end of 2015. So they're still being maintained, but that ends at the end of this year. So ultimately we'll eventually be moving over to the 1.0.1 or 1.0.2.

So, but here was the deal. What was found, and this was found by Google's Adam Langley and David Benjamin, who does the BoringSSL, they were looking at the OpenSSL code. And so this wasn't something found exploited in the wild. This wasn't being used. As far as we know, nobody was ever affected by it. It never happened, essentially. And so this got fixed before it was even old enough to have gotten out into the world. So, I mean, this is all sort of like, okay, whew. We're glad we found it. We're glad we found it as fast as we did.

What they discovered was that, if the first attempt to build the certificate chain when a client is verifying a certificate from a server fails, the successive attempts didn't reinitialize themselves quite correctly, and it was possible for some of the status from the first attempt to get carried over into the second attempt. Specifically, if good certificates were encountered during the first attempt, but it was impossible for the client to complete a chain of trust from the trust anchors that the client had out to the certificate that was being presented from the server, then the client, the OpenSSL client would try again. But some of the validity from the first attempt leaked into the second attempt.

And so it turns out that there was, in theory, again, this was all sort of theoretical, that there was a way an attacker could present a certificate that had never been validly signed that for four weeks that little window of a version of OpenSSL could have believed was valid when it wasn't. So that's what it was. Yes, that's not good. I mean, it's, like, horrifying to the developers of OpenSSL because it's a way of spoofing a valid certificate. The good news is it didn't live long enough to get out into the world. Neither Red Hat nor CentOS nor Ubuntu ever had a chance to adopt it. So they didn't get it.

And of course none of our browsers use OpenSSL. IE, Chrome, Firefox, and Safari all have their own client-side crypto libraries of one form or another. So it would have only been those libraries or systems which use OpenSSL and would be authenticating against a remote server that could have had a man in the middle intercept the communications and present a spoofed certificate which could have been manipulated to cause the client to believe it was properly signed. So it's like, okay, that's what that was. Glad you guys found it. Glad it didn't last any longer than it did. And nobody had a chance to get affected by it. And, frankly, most of the people using web browsers wouldn't have ever been affected by it anyway.

PADRE: You know, Steve, I'm looking at this code. And, you know, it's available on GitHub. This change supposedly was added a couple of months ago, earlier in the year. And I'm wondering how you can make that mistake. I mean, it could just be that someone fat-fingered it. But it almost looks as if it was added to deliberately obfuscate what it was doing and then to break it.

Steve: Who knows?

PADRE: Hmm. Well, don't use OpenSSL. Or use an updated version of OpenSSL.

Steve: The problem with OpenSSL, as we discussed just last week, is that it's more than half a million lines of code. And more than 70,000 of those lines of code are about TLS. And it is just huge and lumbering. I mean, it's sort of the test armature for anything you want to add to TLS. You prototype it in OpenSSL. I mean, for those people who have their heads in it, they understand it; they know how it works; they know how to use it. And so it's like it's where they go.

But this is why I love the idea of Amazon saying we started from scratch. We've got 6,000 lines of code which implement all of the parts of TLS that we care about. And we're going to be putting it on our servers, which is like the perfect place for it because it's going to be high usage, lots of clients hitting it. And at some point it's just time for these old things to die. Like Flash. Like Java. Like OpenSSL. I mean, it got us where we are, and we salute it. But it's just too big and old. It just gets - these things get barnacles.

PADRE: Maybe Google can build a memorial wall of all the technologies that we respect for what they did, but they've been retired since, as you said, they're too old, and it's time to die.

Steve: Yeah, they're venerable, but barnacles.

PADRE: Barnacles.

Steve: So another attempt has been made by the industry's foremost security developers and cryptographers. This is a Who's Who of Internet security. Eleven authors authored a 34-page, carefully written statement, and they raised some really fabulous points. One of the things that they made very clear is that what's lacking so far from this discussion is any sort of specification of what the government wants, what the requirements would be for this so-called, you know, the government saying, oh, no, we don't want a backdoor. We want a well-designed, open-kimono front door where there's a, you know, developed in concert with the industry, a government-industry partnership to design a secure means by which the government can somehow arrange to get access to cryptographically encoded data for law enforcement purposes.

Okay. So I'll first just read the abstract, which is sort of - there's the abstract, and the executive summary, and I'm certainly not going to drag us through 34 pages. But I did pull out some key portions of this that I can't paraphrase because these guys really put it together perfectly.

So they said, they start off: "Twenty years ago, law enforcement organizations lobbied to require data and communication services to engineer their products to guarantee law enforcement access to all data. After lengthy debate and vigorous predictions of enforcement channels going dark, these attempts to regulate the emerging Internet were abandoned. In the intervening years, innovation on the Internet flourished, and law enforcement agencies found new and more effective means of accessing vastly larger

quantities of data. Today we're again hearing calls for regulation to mandate the provision of exceptional access mechanisms." That's the term of art here that's being used, exceptional access mechanisms.

"In this report, a group of computer scientists and security experts, many of whom participated in a 1997 study of these same topics, has convened to explore the likely effects of imposing extraordinary access mandates. We have found that the damage that could be caused by law enforcement exceptional access requirements would be even greater today than it would have been 20 years ago. In the wake of the growing economic and social cost of the fundamental insecurity of today's Internet environment, any proposals that alter the security dynamics online should be approached with caution.

"Exceptional access would force Internet system developers to reverse forward secrecy design practices that seek to minimize the impact on user privacy when systems are breached. The complexity of today's Internet environment, with millions of apps and globally connected services, means that new law enforcement requirements are likely to introduce unanticipated, hard-to-detect security flaws. Beyond these and other technical vulnerabilities, the prospect of globally deployed exceptional access systems raises difficult problems about how such an environment would be governed, and how to ensure that such systems would respect human rights and the rule of law."

And the biggest problem for those who wrote this paper was that no clear statement, as I mentioned before, of the requirements from the government have been produced. They said elsewhere here, they said: "The current public policy debate is hampered by the fact that law enforcement has not provided a sufficiently complete statement of their requirements for technical experts or lawmakers to analyze."

And one of the things that this paper so beautifully does, essentially I think it will successfully preempt, it will provide sort of a working forum for successfully preempting probably any further motion on this whole thing because, for example, we've often talked about the power of forward secrecy. When you go to SSL Labs and check a website, one of the things that Ivan has done is he puts a little "FS," short for forward secrecy, on all the cipher suites which support it. And, for example, GRC's cipher suites are all biased toward using ephemeral Diffie-Hellman key agreement.

What that means is that, when your browser and the remote server negotiate, they negotiate with the server's fixed certificate that has a several year lifetime, as we know. If you do not use forward secrecy, that is, a cipher suite with forward secrecy, and you did capture all of the traffic of that communication, then if that certificate was ever disclosed, even after it had been retired, then the secret key in that certificate could be used to decrypt all of the traffic that had ever been encrypted during the multiyear life of that certificate.

So forward secrecy is the key, next generation. It's the technology that is broadly being adopted because what it means is that the cipher suites which support it - and now they're widely supported by browsers, they are widely supported by servers - what it means is that, even though there's a fixed secret key and certificate in the server, the session negotiates an ephemeral, that is, a nonpermanent key which only has a lifetime of that session, such that any capture of the server's credentials has no long-term impact. You would never be able to go back, for example, in the past and decrypt everything that had been captured because that information is not part of the data that you're capturing. It involves the on-the-fly negotiation of secrets which, even when you're capturing everything going over the wire, no eavesdropper is able to obtain the key that both endpoints agree to.

So what they say here, and to explain this, is they said: "The first technical obstacle is that, although the mode of encrypting a symmetric key with a public key is in common use" - and of course we've talked about that often. That's the way this is done is you use a random number generator to get a symmetric key, then you encrypt that with your public key, and only the person with the private key is able to decrypt it. That way the endpoints are able to get a common key.

These authors go on, saying even though that's in common use, companies are aggressively moving away from it - and I would already argue have moved away from it. If you bring up any website under SSL Labs, all of the ones we're using now, they're already using ephemeral encryption. So "Companies are aggressively moving away from it because of a significant practical vulnerability: If an entity's private key is ever breached, all data ever secured with that private key is immediately compromised. Because it is unwise to assume a network will never be breached, a single failure should never compromise all data that was ever encrypted."

That's just perfectly phrased to really be a slap in the face of anyone who says, oh, well, we need the technology to decrypt it. Essentially, what we're saying is, look, the only way to give you exceptional access would be to turn back the clock on this substantial improvement in security which is forward secrecy, which hasn't been adopted to thwart law enforcement, it's been adopted to dramatically improve functional security.

PADRE: Let me play a little bit of devil's advocate. I'm with you. So I like the idea of a communications session in which the key is created and destroyed as needed. So absolute forward security, no use of a single key that could compromise an entire network or, indeed, an history of communications.

Steve: Right.

PADRE: But if we are to look at the political nature of communications, is it possible to have a forward-looking system in which keys are maintained by a trusted entity, not using a single private universal key that can be used for exceptional access, but an escrow system in which access can be decided in a legal fashion, whether or not someone needs access to a particular set of communications, that would still maintain forward-looking secrecy, while at the same time fulfilling the needs of law enforcement. Does that exist?

Steve: Okay. So it is not the case - okay. So what I loved about their argument is that it is that no one today is recording these ephemerally arrived at keys. And what it would require would be that one endpoint or the other would divulge a key that they had negotiated just for that session.

Now, these guys do talk about the problems, the fundamental problems of escrow key management. They said: "Who would control the escrowed keys? Within the U.S., one could postulate that the FBI or some other designated federal entity would hold the private key necessary to obtain access to data, and that judicial mechanisms would be constructed to enable its use by the plethora of federal, state, and local law enforcement entities. However, this leaves unanswered the question of what happens outside a nation's borders. Would German and French public and private-sector organizations be willing to use systems that gave the U.S. government access to their data, especially when they could instead use locally built systems that do not? What about Russia? Would encrypted data transmitted between the U.S. and China need to have keys escrowed by both governments? Could a single escrow agent be found that would be acceptable to both governments? If so, would access be granted to just one of the two governments, or would both need to agree to a request?"

And then under jurisdiction they say: "The greatest impediment to exceptional access may be jurisdiction. Building in exceptional access would be risky enough, even if only one law enforcement agency in the world had it. But this is not only a U.S. issue. The U.K. government promises legislation this fall to compel communications service providers, including U.S.-based corporations, to grant access to U.K. law enforcement agencies, and other countries would certainly follow suit.

"China has already intimated that it may require exceptional access. If a British-based developer deploys a messaging application used by citizens of China, must it provide exceptional access to Chinese law enforcement? Which countries have sufficient respect for the rule of law to participate in an 'international exceptional access' framework? How would such determinations be made? How would timely approvals be given to the millions of new products with communications capabilities? And how would this new surveillance ecosystem be funded and supervised?

"The U.S. and U.K. governments have fought long and hard to keep the governance of the Internet open, in the face of demands from authoritarian countries that it be brought under state control. Does not the rush, or the push for exceptional access present a breathtaking policy reversal?" And I think this just beautifully demonstrates, like, first of all, how ill-thought-out the statements from our own law enforcement agencies are, saying, uh, we want to be able to, you know, have access. It's like, okay. How? I mean, fill out the thought. How do you respond to these problems? Because these are real problems.

Either encryption does not have a backdoor, encryption supports forward secrecy - meaning that everybody, everybody is protected from breaches on their previous communications, against their communications, because every single connection that they establish uses an ephemeral key - or nobody is protected. And can we have a world where essentially forget about encryption, let's just outlaw it and go back to plaintext because I don't see how you have a middle ground.

PADRE: Well, that's the issue. I mean, one exceptional access granted means multiple exception access granted. As you said, the geopolitical factors involved in who would control keys, even if you could find the perfect agency, and I'm definitely not suggesting it's the government, but the perfect agency to maintain escrow of these keys, there's no way that you can jive that with having decent encryption. It's just they don't go together.

Steve: No. And so, for example, it is absolutely clear that forward secrecy is the solution we have found for, I mean, it's a brilliant solution, now in deployment, where endpoints spontaneously create a key for their encryption that no eavesdropper recording the traffic can get, in which they mutually destroy when they're done. We have it. And so that's the other part of this, is that they're saying they want to eavesdrop on the bad guys, on the child pornographers and the terrorists and the people doing horrible things.

But the problem is, this technology already exists. So as with so many other things that you try to outlaw them, if you do not have enforcement, and there is no means of enforcing a ban on cryptographic communication, if you don't have enforcement, then the bad guys will continue using it. The technology exists. The ship has sailed. The horses have left the barn. So only law-abiding people have technology that can be eavesdropped on because there will always be a black market or a gray market of existing crypto technology that builds an ephemeral key that cannot be eavesdropped on, where the endpoint's destroyed afterwards. It's done. It exists. You can't take it back.

PADRE: Yeah, well, okay. So wait, but let's back up for just a second.

Steve: Okay.

PADRE: At some point, even if we explained until we're blue in the face that you can't weaken encryption without destroying it - because I think everyone in the Security Now! audience understands that, they understand what happens the second you add a master key, or the second you reduce the encryption in order to make it crackable. At some point, some senator is going to stand up and say, well, if that's the case, then we just can't use it, period.

Steve: I know.

PADRE: Right? I mean, that's the only political solution, to say...

Steve: And how do you deal, how do you answer these real geopolitical questions of, okay, who's going to have the key? Our technology, our U.S. corporations - Apple, for example, is selling iPhones all over the world because they have asserted, and for their reputation, we believe that they can no longer decrypt the technology in the phones. Consequently, globally, these are in use, and they are trusted, and I believe that trust is warranted.

If the U.S. steps their foot in this and says we have to be able to decrypt it, then we have broken cryptography. But again, it's not just the U.S. We're now in a global ecosystem of communications. And as this paper clearly points out, you can't just have the U.S. with the keys. No other countries will accept that. The U.K. is apparently going to demand it. We just have to say no to everyone.

PADRE: All right, Steve. I'm going to move you off of this because, if we stay here much longer, we may say things we don't want to say.

Steve: Yeah, yeah.

PADRE: It's something you can be incredibly passionate about because the engineer in you is saying, look, you don't get it, you don't understand it. Do you not understand what we're telling you? And it sounds so ridiculous whenever someone is mandating a political solution to a technology problem. So instead, let's go ahead and mandate a technology problem to a political solution and talk about OPM.

Steve: So anyway, this is a quickie. We've talked about them several times. There was the 4.1 million record breach of a couple months ago. Then there was the news of the 21.5 million record breach, way worse than that one. And this one had all of the background information on everybody who had even applied for some sort of government position in the last 15 years. Or, wait, did it go back to '85? I remember an '85. I've seen '85, and I've seen 2000. But so, like, way back.

Now the news is a little more detail on the nature of what was in this 21 million, 21.5 million record breach. Of those 21.5 million records, at least 1.1 million of them contained digitized fingerprint images. Look at that happy face. Ha ha ha ha. Oh, lord. So, I mean, this is just - not only do you have all this background information, everybody's name, dates, Social Security numbers, personal information, family connections, friends, employment histories, everything. Now you have their biometric fingerprints that they have no ability to change.

And because today we have no standard for encoding fingerprints in a hash-like way, such that you can run the fingerprint through a hash and get something that the

fingerprint can create, that is, a digest of the fingerprint, this is the sort of thing that Apple uses to protect its users. They don't store the fingerprint. They do a feature extraction which is rotation tolerant, and they end up hashing that, essentially, to create a representation such that you cannot, from that, go backward to the image. But the image can go forward to that.

Well, that's the secure way to do it. But not only did OPM never encrypt any of this data, it never occurred to them that they had to do some sort of fingerprint hash. So they just have digitized images, high-resolution fingerprint images of 1.1 million people who no longer have their fingerprints as private information. So, I mean, they could be set up for crimes with forged fingerprints left at a crime scene. Obviously their identities can be forged using these fingerprints. Anyway, just unbelievable. Just horrifying.

PADRE: What do you with this kind of a breach? I mean, it's one thing when Target lets go of a couple of million credit cards and then offers, what is it, fraud protection alerts for five years.

Steve: Right.

PADRE: But if someone has released, not just your personal information, but your biometric information and - like, for example, I have an SF-86 with the U.S. government - let go of my security clearance interviews, credit reporting's not going to do it.

Steve: Yeah, yeah. Unbelievable.

PADRE: Great.

Steve: We talked about WiFi Sense last week. And I saw a tweet from friend of the show Simon Zerafa, who tweeted - we all updated to Windows 10. The most recent build is 10166. And Simon noted that WiFi Sense is enabled by default in the latest build. He says, "I've turned it off." And of course...

PADRE: I tested this.

Steve: Okay, good.

PADRE: So I downloaded the latest build, and I did a fresh install. So I didn't install over something that was already there.

Steve: Good.

PADRE: It was actually disabled by default. So I'm not sure...

Steve: So that must have - that's interesting. Maybe he updated, and he had enabled it before. But I'm surprised because I wouldn't have thought that Simon would have had that enabled. So, okay. But I'm glad to know that. So we still, you know, what really matters is what the final build does because they could have this enabled, for example, following from Simon's tweet, if they were still wanting developers to test it and see how it works and so forth, and offer it to people who update the final RTM once it's released, and then have it off and say, look, you can turn this on if you would like to be able to share this. But in any event, we know that at this point it is not granular enough. And we've all agreed that's the real problem is that you can't turn it on for a Facebook friend or a Twitter follower. It's like globally on for all or none, which really seems to be unfortunate.

PADRE: Yeah. At this point I'm holding off on any Win10 updates, just because I want the RTM. So as soon as they've got the version that's going to be released to manufacturer, that's the one that we can start saying this is what they did right or did wrong because the updates are coming so furiously now, I can't keep up. It's almost daily.

Steve: Right. I did want to mention real quickly, just a little blurb of the news, that a university, a Canadian university experimented installing Adblock Plus on 100 volunteers' systems. And they did a study over the course of a period of six weeks, and they measured the pre-, during, and post-bandwidth usage. And within this small cohort of 100 volunteers, the bandwidth savings varied between 25 and 40% of their network bandwidth.

PADRE: Wow.

Steve: So that gives you a sense of, I mean, we know there are many problems with the way this ecosystem has evolved. One of them that I keep coming back to is that it is trivial for a website to put links out which cause our browsers to use our bandwidth to download content that that site isn't even making any representations about. They're just saying, oh, you know, here's 40 different domains that we want you to also download stuff from when you visit our page. That's what I object to. And the fact, I mean, that, and then there's the whole tracking problem. But just the idea that they're giving me little tiny links that cause my browser to get bogged down.

I've seen some - actually iMore, unfortunately, I mean, we love iMore, they're great guys. And they even know what's happened to their website. I saw someone posted that their page load went from 11 seconds to finish loading to two seconds when they added tracking protection under Firefox on the iMore site, just two seconds to finish all web traffic versus 11. I mean, that's a huge blob of additional stuff to be downloading and slowing down people's browsers. And of course it's consuming power. And in mobile users, it's burning up bandwidth, you know, huge amounts of bandwidth.

PADRE: David Redekop in the chatroom is saying that that lines up with what he's seeing out of his gateway when they block ads. And actually, on the networks that I'm responsible for, the biggest saving has been coming from the blocking of those ads that are tied to auto-playing videos.

Steve: Yes.

PADRE: Or the ones that have videos that will start if you play. They actually preload that so that you don't have to wait. Which means, if someone leaves a browser open the entire day, it's actually downloading ad after ad after ad, even if they don't see it, which is fantastic. That's a really good use of bandwidth.

Steve: Wow. Again, it's not the website's bandwidth. They're just sending you a link and making you download this. That's what just seems upside-down about this. Okay. So a couple quickies. I did run across something of interest to cryptographers, but also just sort of everybody, I think, and it's available for both iOS and Android. It's a big number calculator, which is always fun because you want to, you know, sometimes it's nice to see what 2^256 actually looks like, rather than just saying 2^256. It's like, I want to see what that number looks like.

The product is called Ivy. It is free. That's I-V-Y. It was written in the Go language by the Go language team. And so I've got links in the show notes to the pages. Padre, you just

brought that up. And so it's an algebraic language calculator for iPhone, iPad, and Android devices that I just thought was cool. I've got one on my site that requires Java to be used. So I'll be switching to this from now on because the big numbers are fun. Sometimes I like to - you just need to see them sometimes. And this does calculation with every single digit significant.

PADRE: But will it spell out words if you turn it upside down? Like I could with my old calculator?

Steve: Back in the old seven-segment days, yes. So a frequent tweeter and source of great links and security information friend of mine through Twitter, Virgilio Corrado, just shot me a note this morning I wanted to pass on, and that is we have followed on the podcast the work over in Switzerland and France, the LHC, the Large Hadron Collider. There was a movie that we've talked about on the podcast often called "Particle Fever," which is just a fantastic documentary about the early days and startup and building of the Large Hadron Collider. The news is they have found a new particle.

The little blurb says: "Scientists at the Large Hadron Collider have announced the discovery of the pentaquark, a class of subatomic particle consisting of four quarks and one antiquark bound together. Like the Higgs boson before it, the pentaquark's existence had been theorized for years, but experiments in the early 2000s claiming to have detected the exotic form of matter were later invalidated. Many scientists had since given up on the pentaquark for good, but this time, say CERN physicists, there's no doubt it's been found."

So yay to the Large Hadron Collider, which is, you know, it was down for a couple years. They rebuilt it. And then they slowly brought it back online, and they're now running it at a much higher level of electron volts, the speed at which they're spinning photons around in counter-rotating circles and smashing them together in order to find new things.

PADRE: They're not even at their max yet.

Steve: No, they're not.

PADRE: I think they're, like, 60% of the total TeV that they could run through the LHC.

Steve: Yeah, I think you're right, they're like six or seven, and it goes to, like, 13 or something, if I remember right?

PADRE: I think it goes to 11. You can turn it all the way to 11.

Steve: Okay, 11 right. Whew.

PADRE: I do want to mention, though, there are two Jesuits who work on research teams on the LHC. I've been trying to get them on an episode for the longest time. They're very busy guys.

Steve: Have you had a chance to see "Particle Fever"?

PADRE: Yes.

Steve: Oh, good.

PADRE: Very, very good documentary.

Steve: I really, really recommend it. And speaking of what I recommend, I just wanted to say that I just watched the third episode of "Humans," which is a new sci-fi series on AMC. The U.K. started it a couple weeks earlier, so I think they're about to get episode five. But it's really turning out to be an interesting episode. When the comment was made during this third episode of something violating Asimov's laws of robotics, I thought, okay, they're on the ball here. So for what it's worth, I haven't talked it up extensively yet, but I'm really, really enjoying it. So I recommend it.

PADRE: But wait, Steve. I've only got time for one series, so...

Steve: Ooh. "Mr. Robot"?

PADRE: Or "Humans."

Steve: "Mr. Robot," that's, you know, a lot of people are a little put off by the druggie aspect of "Mr. Robot." And it's like, well, okay, so if you want something more Disney, then I would say "Humans."

PADRE: Okay.

Steve: If you're looking for something more gritty, "Mr. Robot."

PADRE: I've lived with too many programmers in San Francisco to put aside the druggie thing. It actually is pretty big in San Francisco. So...

Steve: Well, I do think it's probably authentic for some piece of the population. Speaking of recommendations, I've got one, not from me, but from one of SpinRite's users, from one week ago, on June 7th. Kevin Wilhelm, he wrote a note which either Sue or Greg forwarded to me, I don't remember whom. He said: "After listening to Steve's podcasts for years and hearing stories about SpinRite, I finally got to use it in a real critical situation. I work for an IT company that for the most part does managed services, but sometimes get called for 'break fix' work," as he put it. "On this occasion we were contacted by a large national boat manufacturing company whose headquarters are local. I was assigned with going onsite to 'revive' an old XP machine that had crashed a while back, but they wanted to get data off it.

"Once I got there and learned of the whole picture, they were actually getting audited for accounting purposes, and they realized that the only instance of a very old program resided on this one computer, and there was no support to be found anymore for this program. So it was not just a matter of getting data off the drive, but really getting the operating system running again so we could attempt to run the program and get the data from it.

"Well, I broke out my trusty SpinRite CD, which I honestly hadn't used in over four years, and within two hours, voila. The OS booted. The user could log in and open the program like nothing ever happened. Needless to say, I've lectured them on all the things they did wrong that lead to this situation and which could so easily have cost them very dearly. I was happy to be able to offer assistance with your product and will certainly recommend it to many more for years to come. Thanks, Steve. Kevin Wilhelm, senior systems engineer."

PADRE: I'm hoping that Kevin also cloned the drive onto something that wasn't going to crash out, and he unplugged the XP machine from any sort of network they may have had at the office.

Steve: Maybe they just printed out the documents they needed of their accounting and their financial information and then said, "Okay, die." But at least they got the one last piece from it.

PADRE: That story tickles me because I have gone to installations before where there's that one program that nobody knows where they got it from, nobody knows how to install it, they've long lost the media, and that's the only computer it works on. And, I mean, you can say that you shouldn't be using this archaic OS. You can say that you really need to upgrade this computer. But if that's the thing that they need, that's the thing that they need. That's why SpinRite's in my toolbox.

Steve: Yeah. Well, yours and many others. So SQRL revisited. It's funny because, in setting up the show notes for this, I thought, okay, when was it that I first mentioned SQRL? The podcast was named SQRL: Secure QR Login. And I recorded it on October 2nd of 2013 with Tom Merritt. It was Episode 424. So subtracting that from Episode 516, that was 92 weeks ago. And I thought, wow, 92 weeks. I got a tweet from Ned Griffin in response to my tweeting the news of the logo yesterday. Yesterday afternoon he sent back, he said: "Nice. I hope to one day have the slightest idea what you're talking about when you bring up the SQRL Project on the podcast."

So I thought, okay. What we've had over the course of 92 weeks has been a careful and methodical evolution from basically a concept into a mature Internet identity authentication system. And so you and I, Padre, have never had the chance to discuss this in detail. So I thought it would be perfect to sort of, now, 92 weeks later, certainly we have a lot of listeners who have joined in that period of time. And I actually think I've explained it so many times since then that I have some better ways to explain it.

So I want to sort of do a refresher on what exactly it is, and then answer the questions of how we solve a number of what at the very beginning were still outstanding problems. That is, what to do if my SQRL identity is compromised. What if I want to have a second identity at the same site, but I have an identity per site? How do we solve that? What if multiple people want to share a single identity? And there's the issue of how SpinRite - oh, SpinRite - how SQRL could be abused in a site-spoofing fashion.

Okay. So for people like Ned, who hear me talking about this, but have sort of, like, never grokked the fundamental truth of what SQRL is, there's two concepts that I'll relate in a second. The first is we've talked about cryptography and, like, how you could have a - how you have a secret key to a cipher, to a cryptographic cipher like AES or Blowfish or, once upon a time, DES, the idea being that you have this algorithm, this cipher. And you put plaintext in, and it encrypts it into ciphertext under the influence of a key, meaning that this is a keyed cipher. So everybody kind of understands that. You have the key, and that's the secret. And the algorithm is not a secret, but the key is. And so when you put plaintext in, it encrypts it differently, depending upon what the key is. So that's a keyed cipher. Okay, so that's one concept.

Now, we've also talked about, often, a hash. And a hash is like SHA-256 or SHA-1 or, earlier than that, MD5. MD5 generated, what, was it a 120-bit hash? SHA-1 is 160 bits. SHA-256, like its name, is 256 bits. And so the idea is these are hashing algorithms where you put something in of whatever length. You could hash the dictionary or hash a document or hash a message. And what happens is it turns it into a fixed-length blob which, no matter how long the stuff you put in, you always get this blob that is 120 bits or 160 bits or 256 bits, whatever it is, a fixed length.

It turns out you can also have a keyed hash. And that's, in the crypto world, we call that an HMAC. That's an acronym or an abbreviation for a message authentication code,

that's the MAC part, that uses a hash as its cryptographic function. So it's a hash-based MAC, an HMAC. And this HMAC construction, because it's a constructed cryptographic function using a couple hash functions and some other XORing of fixed material, doesn't really matter, we can kind of think of it as a black box.

But the point is that, whereas a hashing function is just like one function, SHA-256, everybody's SHA-256 is the same, yours and mine. And that's valuable for some uses. I can hash a big file and then post its hash on my website and say, after you download this, you hash it, and you should get the same result because SHA-256 is a universal function. You put the same thing in, you always get the same thing out. But a keyed hash is different, much like a keyed cipher where the key determines what the cipher does, that is, what comes out the other end. With a keyed hash, the key determines the hash function. So essentially what this means is you have an infinite number, well, not infinite, it's however many bits the key is. Those many bits of the key determine how many different hash functions you can generate. And SQRL uses at its heart a keyed hash.

So here's how it uses it. When you go to a website, the domain of the website, www.amazon.com or TWiT.tv or www.ebay.com, whatever, that domain name is hashed through this HMAC function to produce, as hash functions always do, a fixed-size blob. But the key is your SQRL identity. So everybody who uses SQRL, the very first thing they do is they create one master SQRL identity. And that is - it's a little more complicated than this. But for the first explanation, that is the key for this hash function, which means that every SQRL user has their own hash function, completely different from everyone else's, so that when everyone goes to Amazon.com with their own personal hash function, which is personalized by their master key, they get a different blob. They get a different output from their personal hash function. And so the thing that comes out of the hash function is actually your private key for the site.

So what that means is that everyone using SQRL gets, because you have your own unique SQRL identity, your master identity, when you go to different websites, on the fly, SQRL produces unique private keys. For every different site you visit, that's going to be a different web domain, you know, it's a different domain name, so it hashes to a different private key. And all SQRL users that go to the same domain, they also hash to a different private key. So what we've done is, by using this master SQRL identity and the HMAC function, we've created a galaxy of private keys such that everybody has a different key when they visit different sites, and all SQRL users have different keys because with 2^256 we've got all we need.

Consider, we were talking about how many IPv6 addresses there are, a ridiculous number, that's only 128 bits. This is twice as long. And that's not, remember, that's not twice as many, that's twice as many 128 times. And that's the same size as a bitcoin token. I mean, it is so many that you never need to worry about collision. Collision just cannot happen as long as you've got a good random number generator. And so that's one of the things we need is a good random number generator.

PADRE: I like this.

Steve: So this is the essence of SQRL, is a SQRL user has one master identity. And the SQRL system automatically creates a different private key for every site you visit. But the neat thing is, when you go back there, the same domain name, same private key.

Now, what do we do with that, that per site/per user private key? Well, this is an elliptic curve private key. And this aspect of elliptic curves was for me the aha moment that I had in September of 2013 because I was studying elliptic curve crypto, and I was looking

at Dan Bernstein's elliptic curve crypto. And it explained that the private key could just be given to it. That is, it was deterministic. We're all used to thinking, to make the comparison to RSA keys. Remember, RSA keys are where you get two prime numbers, and you multiple them together. And the hard thing with an RSA crypto is factoring. You can't - there's no - we have no known way to quickly, in a short time, to factor that back into its primes.

Well, the problem is you have to get these prime numbers by doing primality tests. You have a good pseudorandom number generator. You generate a really big random number, and then you do a primality test to see if it's prime. And so you are, at random, you are choosing these numbers which you will then mix together. And so that's the way public key crypto using prime factorization as the hard thing works. But elliptic curves, this particular family of elliptic curves allows you to specify the private key. You have to make a couple little tweaks to it. A couple of the high bits and the low bit have to be set to a certain pattern. But other than that, you get to specify the private key.

And when I saw that, it was like, wait a minute. What? Because that meant that a private key could be deterministic. It wasn't chosen at random because then you'd have to store it. You'd have to memorize it. You'd have to have, like, if you had a private key for every website, you'd have to store them somewhere, and that would be burdensome. The beauty of SQRL, because it is able to use this type of elliptic curve, is it can make them on the fly. All you have is your one SQRL master identity. And then it synthesizes, based on the domain you're visiting, your per-domain private key, any time you go back.

PADRE: And it can remake the key every single time, and it has a unique key for every domain that you ever visit, all with that single master identity.

Steve: Exactly.

PADRE: Oh. Beautiful.

Steve: So the cool thing is then the next step. There's a function in this elliptic curve crypto which converts that private key to its matching public key. You just give it your private key, and it says "Here's the public key." And it's like, wait, okay, what, really? And it's like, yes. You can't go backwards. Meaning from the public key there is no way, it's a hash-like function. There's no way to go back. So if somebody knows your public key, and this is the general property of public keys, if someone knows your public key, they cannot get your private key. Your public key.

So you go to a website. SQRL, using this HMAC function and your master identity, generates your private key for that site. It runs that through this one-way function, this generate the public key from the private key. That is your identity for the site. That is how the site knows you, is that public key. That's the token that you give the site, saying, okay, from now on, this is how you know me. You know me by this public key. And it's a public key, so it can be known publicly. The site stores it. And it says, okay, this is who you are.

So now, now here's the final piece of this that is so cool, is the site says, you just said this is who you are. Prove it. And so it gives you a random blob of data. You sign that with your private key and send it back the signed random blob of data. And so what you have done is, without exposing your private key, you've proven you have it. That is, you've proven that whatever entity gave the site this public key has the private key in its possession. So what's so cool about this is that's it. That's all there is. You give the site your public key for it to identify you. Then a week later, when you want to log in, you come back to the site, and the site says, oh, hi there. Here's a username and password.

You say no, no, no. That's old school. Besides, that's not secure. I'm not using that.

You click on this link on the SQRL link, where the site has generated a random challenge. It's just random gibberish. It's a challenge. And the SQRL client signs that random challenge with your private key that got regenerated because you're revisiting the site. And so you send it two things. You send it the signed challenge and your public key together. The site gets them both. It looks at your public key and says, ah, I know you. But let's see if you're really you. So it verifies with your public key that you properly signed the challenge that it has never given to anyone before, will never give to anyone again, has just made up for you, and your public key is able to verify the signature. So that proves you have come back, that you're the person who has that public key. And that's basically, that is SQRL in a nutshell.

PADRE: Okay. So, Steve...

Steve: Isn't that cool?

PADRE: It's very, very cool. I love the system. Very elegant. What would someone have to steal from you? What would they need to take from you in order to be able to impersonate you?

Steve: Okay. So remember I said that I hadn't fully explained what's happening with the SQRL identity. Because what we have here is a two-person system, you and all the websites in the world. You are pseudonymous to all of the sites. Every site sees a different random gibberish token, your public key. So you cannot be tracked across sites. There's no way of associating your identity between sites. No more of this problem with using the same password. In fact, what's very cool, if you think about it, that public key that you gave that site has no value to any other website or any hacker. It only is useful to that one site. Essentially, unlike with a username and password, in using SQRL, you're not giving websites any secret to keep. They don't have to keep their database secret any longer. It can be exposed by hackers. Who cares? It has no value to anybody except that one site.

But what does have value is your own identity. And in looking at this in this post-Edward Snowden era, the question is, can anyone trust a third party? Would anyone trust a third party? Is an Internet identity system going to be created where we all give our - somehow there's like a Big Brother. There's a key holder or an identity escrow or something. And the answer is no. Nobody in this day and age is going to do that. So we don't want a third party. I should mention, though, the third party would be someone for you to go to if you lost your identity. That is, that's the only advantage. If somebody else is holding your identity for you, then when you forget your password or your device - say you only had SQRL installed in one client. You can put it on all your clients. They can all share the same identity so you're known as the same unique item by every site you visit. But the problem is, what if?

And so the advantage of a third party is you have recourse. The whole responsibility is not yours. So the liability of not having a third party is there's nobody to go to. It's on you. So a large part of making the SQRL system practical was solving this problem. Basically, how do we allow people to stay responsible for their own identity, but give them the tools that they need, designed into the system, some tools that will allow them to recover from anything that could happen. Anything. Both hacking of their device, loss of their identity, anything. And so there's a couple pieces to this answer.

The first is that you don't actually use your most secret identity for SQRL because we need recourse. So what SQRL actually generates, the very first thing it does, is

something that we call in the SQRL ecosystem a "rescue code." It is a 24-digit number, 24 decimal digits. And so think of it as, like, 1.5 credit card numbers. Credit card numbers are 16 digits, so this is eight more. That is the absolute secret which must not escape. And it is so secret that it is not ever stored in any SQRL client. That's the cool thing about this protocol. Your SQRL client, the very first time you're creating your identity, it puts this out on the screen, and you write it down. You can also print it. You can print it both as 24 digits and as a QR code in order to help you move it around or store it.

But the idea is it's your "get out of jail free" code. It will get you out of any trouble that you could possibly get yourself into. But for that reason it is too powerful. So what happens is, that is never stored. It's not written to a file. No SQRL client will store it. It cannot be even - we won't let it touch any storage medium. That's the one thing, the one requirement we make of people is they write this one code down, or print it out, fold it up, and put it somewhere safe.

PADRE: And they'll never be able to regenerate that.

Steve: Yes, they cannot regenerate it. It was very random. And that's just it. Because people cannot create entropy themselves. So it's completely random. And it is your ultimate master identity.

PADRE: And for those people who are saying, well, wait, now you're putting a point of failure because now you have a password written down, well, if you really don't want a "get out of jail free" card, you could just tear that up. But now know there's no rescue. If you get locked out, you're done.

Steve: See, I mean, do the math. Either you're responsible, or someone else is responsible.

PADRE: Right.

Steve: And so what we've done with this design is to give the user every possible means of recourse. So now, once this rescue code has been generated and printed out, the client runs through a hash process. It does a number of forward hashes, XORing at each intermediate result with sort of this - technically, XORing is a one's complement addition. So basically we have a highly, I mean, even more than a hash, highly reverse-resistant process. More than a hash, it is an iterated hash to produce the working code. That working code is what is used to key the hash. And that's what your password encrypts, which is to say, notice that - so what this does is your password is encrypting the result of running that rescue code through a highly, I mean, an infinitely impossible-to-reverse series of hashes to get the key that we use for the hash function. Your password encrypts that, and that encrypted is what is stored on your computer, stored in your phone, stored wherever you use SQRL. And that is strongly encrypted with a process which is unique to SQRL that we call EnScrypt because it uses Colin Percival's Scrypt function.

But even that - and Scrypt was meant to be a PBKDF2, a password-based key derivation function, meaning that it takes your password and does a lot of stuff to it, but also takes a lot of time. This one is also memory hard, meaning that it actually uses what you give it to fill memory with a grid of pointers that point to other spots of memory. And so this jumps around through main memory, where it gets to somewhere. That contains the pointer for where it should jump to next, and that contains the pointer for where it should jump to next.

So the point is the only way to do this in any reasonable speed is to fill memory with an array of pointers based on your password, and then follow the pointers until a certain amount of time has elapsed. And this is the process that we generate so that guessing your password is incredibly impractical, more so even than, like, any other previous password guessing. For example, we were talking about in the LastPass breach you could guess, with hardware acceleration, 5,000 or 9,000 hashes per second. This, with acceleration, which you actually can't use because of the memory hardness, you can't use GPU, and you can't use ASIC acceleration because we use 16MB of memory, and none of them have access to that much memory. So it assumes a CPU, and it arranges to take five seconds to decrypt your password. Which you only have to do once, when you start using SQRL on the platform, in order to decrypt this heavily encrypted password. That then makes that available to the hash function so that SQRL can generate your identities.

And we've gone to this extreme measure because, think about it, what this does is you've given SQRL the ability to stand in for you. It is your authentication for the Internet. And notice that it's not two-factor, it is incredibly bulletproof single-factor authentication. That is, it is all you need in order to say, this is who I am, and I can prove it. And so the idea is that, thanks to the way this rescue code is then strongly hashed into the code you use, notice that the rescue code stays in a safety deposit box, securely sequestered somewhere.

What that means is that, if you absolutely ever needed, if something happened, and you changed your passcode twice, and you forgot what the new one was, and no matter how hard you tried you couldn't guess it again, it'd be like, oh, my lord, well, you could always go back to the rescue code, which is basically it forgives you from anything that could happen with you losing your identity, sort of your active field identity, or your means of accessing that identity, which is deliberately difficult because we don't want to let hackers get a hold of it. So that's how we handle the aspect of this "get out of jail free" card.

The next aspect of this evolution is what if your device got hacked? What if your identity was stolen? Because we've created essentially a single point of failure, a very strong proxy for you that presumably, as this spreads, you'll use more and more because, I mean, it makes logging in trivial. It's a matter of clicking a link or scanning a barcode in order to authenticate yourself, and you're done. But we know the reality is there's malware. Where there's value, there's going to be someone trying to get it.

So the next thing that was added to this was something called the "identity lock." And the identity lock is a protocol which I haven't described. I'm not going to try to describe it in words. It's a set of equations that I came up with over a serious bunch of coffee at Starbucks when the project was trying to come up with a way of solving this next problem, which is we want to establish an identity with a website. And, once established, we want to make it impossible to break that association without a higher level of authentication.

And so what I mean by that is, as you're going along, introducing yourself as a SQRL user to other SQRL-based websites, you're giving them your private key. There's another piece of information that you're giving them which is the identity lock information. And what's cool about this is that it is randomly generated by the client. And the protocol that cooked up allows the client to generate this identity lock information to assert your identity, but not prove it. So this is different than the regular SQRL ability to prove your identity by signing a challenge from the website.

The point of the identity lock is to deliberately have the clients unable to prove

something in the future that they asserted previously. And we did this deliberately so that, if the client got out of your control, it can't be used to change your identity on any website where you have previously established it. So a bad guy cannot come along and, even if they had your identity, if they had complete knowledge of your SQRL code and all of the software, your password, everything, they cannot remove your identity.

And that's, again, that's the second thing that the rescue code uniquely provides. It's your offline sort of final recovery code that also allows for this. And so it's able, if you provide the rescue code to SQRL, which it normally never has and is never online, it holding it in RAM - and again, it won't store it - it holding it in RAM allows it to change your identity. And it also allows it to reenable authentication if it had been disabled. Because one of the other features of SQRL is you can disable your ability to use SQRL on a website, but not reenable it without the rescue code.

And we did that for a reason. Imagine, for example, you lost your phone, or it got confiscated at the border. So you imagine that something happened so that you had reason to believe maybe your SQRL identity was in jeopardy. You didn't know it for sure, but you just don't want to take the chance. So what you can do with any other SQRL client, you could load your SQRL identity into another client, or just go to a laptop or a desktop or whatever. You could go to any of your high-value websites. And also presume, for whatever reason, you don't have access to your rescue code. We're assuming it's offline. It may be difficult for you to get. It might literally be in a safety deposit box. And because the idea is you may never need to use it.

But with any SQRL client, if you're you, you can disable further access, even by your SQRL identity. Meaning that you could deny, you can proactively deny anybody else who may have your identity, access under your identity. So this is a way to quickly go to your bank and to your high-value sites, any ones you care about, and just with one click you're able to say "disable further access," and it'll say, fine, you're now disabled. Disable further access.

So you go around, if you have reason to believe your identity is in danger, if it had been stolen, or it might be, and essentially block anybody who has that identity. Then, say, you go through whatever process it takes to get a hold of your rescue code. Since no one who has hacked your machine that had SQRL in it, or stolen your device, maybe while it was unlocked, for example - although we actually reauthenticate with a fingerprint in the case of a mobile device, so that your identity is always kept encrypted, and only decrypted just during the brief window of time when it needs to prove that you're you.

The point is that no bad guys could change your identity, and without the rescue code you can disable your identity. They can't either enable it or change it. Which means, with your rescue code, you have sort of this ultimate power. You can, if the danger turned out not to be real, the rescue code empowers you to reenable access under your current identity. And, finally, it allows you to rekey because the SQRL system can be rekeyed. So you say, I believe my existing identity may no longer be secure, for whatever reason - border guards, law enforcement, stolen device, whatever. I no longer trust that my existing identity is secure. Which means SQRL allows you to take back your identity from anyone who may have it, like presumably up to no good. You're able to rekey your identity.

And then what happens is your identity contains both the previous ID and the new rekeyed ID so that, when you then go to any website that knows you under your old identity - technically we call it an atomic process, meaning all at once. It can't be broken apart into separate pieces. There's no way to get in there and do a man in the middle. In a single event, you are asserting, I used to have this identity; I have been blessed by the

rescue code on my old identity; and I have this new identity which is also me. Fix it.

And so in one motion, your identity on the site is securely rekeyed so that anybody who may have the old one, it does them no good any longer. Since there's no central authority, you do need to go to the various sites that know you as your old identity. But you're able to keep the old and the new together for, like, actually you would typically keep it together forever. And as you visit sites that knew you under your old identity, it would just automatically - you don't even see it happening. It just is updating to your new identity.

PADRE: What I love about that, Steve, is that that same technique, the logic tree that you're using, means that you've also handled lifecycle management. If someone wants to stop using SQRL, they use that emergency code, and it's as if - it's the same procedure as if you think someone else has your key.

Steve: Yes.

PADRE: It also means that you can't be spoofed by a site that's trying to fraudulently use some other site's key. If you wonder why it's been 92 weeks since the last time you talked about it, it's because these are all the minutiae that often doesn't get figured out before a security project gets released into the wild. You've actually gone through all the different scenarios, and you've come up with, oh, okay, so we have to tweak it this way so that that doesn't work, or so that you do have an emergency code, or so that you do have recourse in case your device gets stolen, you lose your password, et cetera, et cetera.

Steve: Yeah, it is complete lifecycle identity management built into the protocol.

PADRE: And you open-sourced all this.

Steve: Yeah. Open, yes, the whole thing is open protocol.

PADRE: Wait a minute.

Steve: It's all documented on the website. I've got a Windows server running. I've demonstrated the - Jeff Arthur in the U.K. has an iOS client. There was one that a German guy, Ralf, whose last name I won't even try to pronounce, unfortunately, sorry, Ralf, for Android. But that was last summer, and we've evolved the protocol since. He's waiting for it to settle down. And we have a bunch of people working on server-side stuff, a PHP project, a Python implementation for servers. So, as you said, this is where the time has gone is in examining every single facet of this and evolving an actually bulletproof, workable, true Internet identity solution.

PADRE: I'm exhausted. Steve, I have to get you on Coding 101 just to explain SQRL.

Steve: I'd love to.

PADRE: Because that's the thought process that everyone should be going through whenever they make what I would call Internet-changing dev. Steve Gibson, of course, GRC.com. He blesses us every week, every Tuesday at 1:30 p.m. Pacific time with the latest and the greatest in security news, and of course with discussions about things like, well, how would you get rid of passwords in a modern Internet? How would you stay encrypted in every single site, every single service that you would ever want to use? Well, folks, if you've been listening, now you know. Steve, it has been an absolutely

honor and a privilege to spend the last three weeks with you. I understand that this is yours and Leo's show. But anytime I get to jump in, I geek out. I squee like you wouldn't believe.

Steve: Well, you are a great co-host for the podcast, Padre. So I'm glad we've got you as a fallback for Leo. Thank you very much. It's been a pleasure working with you, too.

PADRE: It's been a pleasure. And of course you can find the low - what version is on GRC.com? I forget.

Steve: A 16Kb version, as opposed to 64.

PADRE: If you want the 16K, exactly, if you want the lower res version of this podcast in your player of choice, make sure to go to GRC.com. You'll also be able to find there, well, ShieldsUP!, and of course SpinRite, and of course all the dev news about SQRL. It's a great place to go additionally if you just want to check out the forums, or perhaps to ask some questions that might show up in a future Q&A episode of Security Now!. That's of course assuming that we ever have a future episode of Q&A because...

Steve: If things slow down.

PADRE: Too much news. Yeah, if things slow down, maybe we'll get a Q&A episode sometime in the near future. If not, we'll just keep cranking out the security news because, well, that's what Steve Gibson does. Thanks to everyone who makes the show possible, of course to Steve Gibson, to Lisa and Leo, especially to them for going on vacation so that I could play with Steve Gibson. To JammerB, our technical director. And of course to all the fine folks who watch the show no matter where they get it and what devices they watch or listen on. Until next time, he's Steve Gibson, I'm Father Robert Ballecer, and we'll see you next time on Security Now!.

Steve: Thanks, Padre.