Harvesting Entropy

Description: After catching up with an interesting, though not dramatic, week of security news, Steve and Leo examine the practical size of randomness and the challenge of collecting entropy in a client that may not have any built-in support for providing it, and may also be surrounded by active attackers.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-456.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-456-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. The latest security news, plus he looks at the very challenging problem of how do you get enough randomness into your random number generator? We'll talk about securing entropy, next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 456, recorded May 20th, 2014: Harvesting Entropy.

It's time for Security Now!, the show where we protect you and your loved ones online, your privacy, your safety, your security, with this guy right here, the Explainer in Chief himself, Mr. Steven "Tiberius" Gibson. Hey, Steve. He's pointing at us.

Steve Gibson: Hey, Leo.

Leo: He's got his Harry's blade. Did you shave today? I notice no more goatee. You're going to be mustached only, yes?

Steve: Actually, I shaved before Jenny's daughter's wedding because I thought she'd...

Leo: How long ago was that?

Steve: That was this Saturday. So I have a nice little - the best thing about Jenny is she doesn't mind the little...

Leo: She doesn't mind the scruff?

Steve: Oh, she doesn't. She says, "Oh, don't shave, don't shave." It's like, okay. Where did you come from? You are just like - that's just too perfect.

Leo: You're the perfect woman.

Steve: Yeah. And so she didn't want me to shave before the rehearsal dinner and wedding. I said, "Oh, no, no. This is for all of them, not for you, Jen. I appreciate the way you are, but...." But, boy, I tell you, I hate shaving. But, well, I'm not going to...

Leo: We'll explain this a little later.

Steve: We'll do a commercial a little bit later.

Leo: Steve, it's so funny because, when Steve likes a product, he doesn't hesitate. It's like, "I'm going to tell you about this." Whether it's a program on TV, a razorblade, whatever, I'm going to tell you about this. So that's good.

Steve: Yeah, well, and my success has been, I mean, I get all this feedback from people saying, oh, thank you, thank you, thank you. So it's like, hey, I want to represent what I find and help people to know.

Leo: We thought, you know, you're a trusted curator for us, and so we like to know. Speaking of which, what are you wearing in your ears for headphones these days?

Steve: I think they're...

Leo: Are those the Shures?

Steve: They're the Shures, E5c or something like that. I can't really remember. But I love them. They're just great.

Leo: Yeah. They're really good. I'm wearing them, too.

Steve: So, as promised, we're going to talk this week, our main topic is where the code meets the pavement, real-world challenges to securely harvesting entropy, even on platforms, as our last question of last week's Q&A brought up, where the platform may not have good sources of entropy. How do we solve that problem? The code is all written. I posted the source through my Twitter feed a couple hours ago, and it's also down lower in our show notes. You can bring it up at some point when I'm talking about this, if

you're curious.

But we have some, not any dramatic news, but some things worth talking about. Of course, Firefox and Mozilla announced their support for DRM video, which caused a lot of controversy, at least in my Twitter feed. So I wanted to talk with you a little bit about that. The interesting news that China has banned the use of Windows 8. A Swiss-based end-to-end encrypted email solution everybody is asking about. Ladar Levison has maybe said the last thing he's going to say. And while it seemed very repetitious to me, I really liked the way he summed it up. Then there was the news of a disturbingly effective second-factor authentication bypass...

Leo: Uh-oh.

Steve: ...and the somewhat disturbing reactions of the various major industry players, reactions to it. The news that everyone who knows I'm interested in entropy sent, which was how a smartphone camera could be used to harvest entropy. And then a little quick update. We have some miscellaneous stuff, update on SQRL and SpinRite. And then we'll get into our topic. So lots of, you know, one of our big potpourri episodes. Let's see how much we can cram in.

Leo: Yeah, it strikes me that people who've never heard this show before are listening, they start listening, and it sounds like English, right up to a point. And then all of a sudden you sound like you're speaking English, but it's kind of, what? Harvesting entropy? I don't - what? What are we talking about? What's going on here?

Steve: It's funny because when I first came back into Jenny's life, she would have the podcast on, on her smartphone. And her yoga girlfriends were standing around, and they were saying, "What is that?" And she'd say, "That's Steve."

Leo: My boy.

Steve: And they said, "Well, you have no idea what he's talking about. Why are you listening to him?"

Leo: It sounds like English words. In fact, I'm pretty sure they are. But there was this point, we were talking about shaving and headphones, very common, mundane things everybody understands. Then suddenly Steve says, "And then we're going to talk about harvesting entropy." And it's like, I'm hearing words. I understand the words. I don't understand what he's talking about. So the reason you harvest entropy is to make a better random number generator. And this turns out to be a very important thing for encryption and security of all kinds, not just video gaming. And so he's going to talk about a clever little thing he's discovered, I take it.

Steve: Yeah, you don't want any - if you're going to have mist in your video game, you don't want any lines running through it.

Leo: Right. It's got to be truly random, yeah.

Steve: Blows the whole illusion.

Leo: That's a good example. Misty stuff has to be totally random. Noise.

Steve: Yeah, because we are exceedingly good at picking out patterns.

Leo: Patterns, yeah.

Steve: One of the things we do. And so some simple tests for entropy, for randomness, is just to plot it, put it on a 2D or a 3D plot, and see if it looks like static from when the aerial used to get disconnected on your old analog television. Or do you see, like, the famous example is the picture of the penguin. Even though it's all obscured, you can look at it and go, oh, well, the penguin is still there. So it's like - and once again, we're talking about something, people are going, what? What penguin? Where?

Leo: It was English. It really was. And then it wasn't. But it was. But it's strange. All right, Steve. Let's launch right in here.

Steve: So it was a controversial move, and I know from having read a lot about this that the Mozilla folks did not do this cavalierly and casually. I mean, they really are working to keep their users' best interests in mind. But they essentially had no choice. When the World Wide Web Consortium, the W3C, decided to add digital rights management video into the HTML5 spec, which is really the big change, Firefox could say, obstinately, no, we refuse to have anything to do with digital rights on the web. But all that would do is lose them users. I mean, if people wanted then to watch content-controlled video, they'd go to Chrome, or they'd go to IE, or to Safari, or the commercial browsers.

And so I saw a real reaction to this news, it was last Wednesday, so just after last week's podcast, from people who were sending me tweets with basically their outrage. And I understand the position that Mozilla was in. And there's one other thing. And in Cory Doctorow's piece, where he wrote about this, I mean, he was similarly, of course, disappointed. But he made the point that - and I have to mention, too, the way this is done is an Adobe, of all people, DRM plugin is loaded via the browser in order to do the decryption. And of course the really annoying thing is this is all - it's all DMCA-protected. So researchers get in trouble if they talk about vulnerabilities which are found in this.

So whereas we spent the first four or five years of this podcast talking about PDF flaws and Flash flaws from Adobe products, and now they're going to be the source of the digital rights plugin that the various browsers are going to use to bring this in. And the significant feature of this for Firefox users is that the sandbox, the all-important containment for this digital rights management plugin, is open source and completely inspectable, which is not the case for any of the closed-source commercial browsers. So Firefox users get something that, admittedly, if you're going to watch DRM-locked video through Firefox, this is going to be the way you're going to have to do it. If you want to watch Netflix content, for example, this is the way it's going to have to happen.

Leo: That uses Silverlight. Now, in the past you would, if you wanted to watch Flash or Silverlight, I know if I wanted to use Mozilla to watch Netflix I would go to Microsoft's site, download and install a plugin. And that took it out of the hands of Firefox. And it doesn't come with that plugin. So this is different.

Steve: No, no, it's actually very, very similar. Firefox still doesn't come with it, but it dynamically fetches the plugin from an Adobe server on the fly as needed. So, I mean, they're really doing everything they can to distance themselves from this, to provide the functionality. I mean, and also the World Wide Web Consortium, I mean, essentially they were the first people to cave because they, under pressure from the industry, said, okay, don't we want web browsers to be able to deliver copy-protected content? And, I mean, yes, if everyone in the industry refused to do that, then I don't know. Maybe we'd get capitulation from the movie industry, but I don't think so. I just think that web browsers would not be allowed to play content.

Leo: But of course Internet Explorer, Safari, and Chrome all do support DRM. But they're commercial enterprises. Mozilla is a not-for-profit organization.

Steve: Right.

Leo: And so I guess you could say, well, we're going to take the high road. But I think the upshot of that is, I don't know, only kind of...

Steve: Abandonment. No, I mean...

Leo: By everybody but the DRM hardcore people like Cory Doctorow, hardcore DRM haters.

Steve: Right, right. And I have to say, Cory came off saying, well, I mean, he understood it. It wasn't the way he hoped to have things evolve. Earlier today, as I was thinking about this, I was sort of thinking, to sum up so much of what we're seeing is the commercialization of the Internet. And it was inevitable. This, when you look at it, this was what was going to happen. Yes, we all created it. We were there in the beginning. I mean, yes, all the techie pioneer types. But it wasn't - it just wasn't going to stay ours. I mean, we're going to have battles over Net Neutrality and over Digital Rights Management and how this all gets carved up and who pays whom to do what, you know, as this wonderful network of ours gets commercialized.

Leo: I have mixed feelings. Like Cory, I have mixed feelings about it.

Steve: Yeah, yeah. From my standpoint, I'm a Firefox user. I don't know that I'm watching any DRM content now. But I suppose, if that becomes the thing to do, I would rather stay in Firefox than be driven off of Firefox to IE or Chrome or Safari. So, yeah. So, I mean, I guess, on balance, I'm glad that it's there. And I would imagine there's no doubt a setting you can set to turn that off, to neuter Firefox of the ability to play that.

So it'll be as if Mozilla did what the hardcore people wished, if they wish to just say no, I absolutely refuse to view digital rights managed content in my browser. I'm sure there's a setting for that.

Leo: It raises issues in Linux and other open source operating systems.

Steve: Oh, of course.

Leo: It means you can't, well, see, it's merely a mechanism to install DRM. It doesn't come with DRM, just the mechanism. I don't know. This might raise an issue with Linux distros, as well.

Steve: Yeah, I mean, you're right. The licensing requirements of the, what, the GNU Project can prevent this kind of thing...

Leo: The GPL and - yeah.

Steve: ...from happening. Well, and remember that the Mozilla guys had the same problem with, was it MP4 or OGG? I mean, we've already had problems with the whole MPEG consortium and the idea that just video compression is not...

Leo: OGG was created because it was unencumbered, and MP3 was encumbered. And so purist Linux distros did not include MP3 players. They used OGG. That's why Mint came along, which included all of the stuff like Flash and MP3 players. It was a version of Ubuntu with all of this stuff. And it became very popular very fast because you didn't have to install anything.

Steve: Right.

Leo: I don't know.

Steve: Well, speaking of not installing anything, Reuters this morning reported that it, in some weird document about energy conservation, it's not clear to anyone why it was in that particular document, but China's central government procurement center is now - they have issued a ban on installing Windows 8 on any government computers, which the only thing anyone can figure is that it's in protest over Microsoft's decision to stop supporting Windows XP, which still has a 50% share of Chinese desktops. So Reuters said, quote: "The official" - how do you pronounce that news agency?

Leo: Xinhua.

Steve: Xinhua. "The official Xinhua news agency said the ban was to ensure computer security after Microsoft ended support for its Windows XP operating system, which was

widely used in China." So, okay. It's not clear to me, unless they're saying, well, we don't want to make the same mistake again, because now we're smarting over the fact that we have no more updates for XP, so we don't want to continue with Windows 8. I mean, and arguably, the whole idea that China is using Windows sort of seems like a fundamental problem.

Leo: If I were them, I'd be worried about backdoors more than anything else.

Steve: Exactly. You know? Why...

Leo: They do have a version of Linux that is created for and used by the Chinese government. It's Red, it's called Red Linux or something like that.

Steve: And so maybe that will be what they switch to is they're saying, no, no Win8. We want to go - it's time for us to cut the cord and start using our own stuff.

Leo: It's just a mess, yeah.

Steve: Crazy, crazy. I just got a kick out of that. It's like, okay, no more Win8. We're going to enhance our security by not installing Windows 8. It's like, wait a minute. Windows 8 is being supported. So the only thing I can figure is they're, like, saying, we're not going to continue down this road. We're going to force ourselves to a more difficult path, but bite the bullet.

ProtonMail. Now, I wrote .com, but I think it's actually .net. Everyone's been talking about this. And unfortunately, it's mostly a consequence of the headlines...

Leo: By the way, it's .ch because it's Swiss.

Steve: Oh, okay. Try ProtonMail.net and see if that works.

Leo: All right. Yeah, maybe they did...

Steve: Because I think it did for me.

Leo: Yeah. The company is .ch.

Steve: Yes, and you're right, it is absolutely Swiss.

Leo: No, .net does not work. It's .ch.

Steve: Oh, okay, good. So the headlines are what caught everyone's attention because of course the various stories talked about the only NSA-proof email. And it's like, oh, okay. Here we go. So I guess what I wanted to say was that end-to-end encryption is not hard. I mean, it's not difficult. We talked about this a couple weeks ago in the context of the rumors that Google was experimenting with this idea. And you and I, Leo, discussed the quandary that an advertising-based email service would have if in fact, I mean, the only way they could meaningfully provide privacy is if they, too, cannot see into the content of your email.

Yet we know that Google's cleverness is that they're able to read your email, present you with ads sort of in context, in vitro, right there along with the email, tied in to what the email is talking about. But if Google's going to do end-to-end encryption, then what that means is you encrypt in the browser, and all you're doing is they're the carrier of noise, a blob of random noise that they have absolutely no visibility into, if this is to be meaningful.

So I guess, I mean, I like the idea that this is being popularized; that everyone is saying, hey, this is what we need, end-to-end encryption. Though it does lock you into their system to some degree. Although this is also something that they've thought through. They said, for example, we support sending encrypted communication to non-ProtonMail users via symmetric encryption. When you send an encrypted message to a non-ProtonMail user, they receive a link which loads the encrypted message onto their browser, which they can decrypt using a decryption passphrase that you've shared with them. So that's sort of a, I mean, that's the necessary sort of half-step you need if you're going to try to bootstrap basically a new proprietary email system into existence.

Now, if you're going to send to non-ProtonMail people, then you need to decide, do I keep the email encrypted, and the recipient gets a link which then allows their browser to retrieve this, and then they have to decrypt it themselves? Or do I tell them get a ProtonMail account, and we can exchange privately, point-to-point? So this is not advertiser-supported. It's got all the buzzwords, all of the things that seem right. They have got an Android and iPhone apps expected by the end of the summer. Right now it's browser-based. So I'm sure you could use the Android or iOS Safari browsers and Firefox in Android in order to do the same thing.

So basically this is using JavaScript, which is downloaded on the fly, which is an increasing - it's something that we see more and more. That's, after all, the way LastPass operates is the way they provide Trust No One encryption is that JavaScript gets loaded into the browser. The encryption occurs in the browser. All that goes out is encrypted. It's just not difficult to do.

But that's not the only problem. For example, claims that this is completely NSA-proof, and there's no way the NSA can get in there, well, okay. First of all, there's an issue with metadata. That is, it may be that your blobs are encrypted. But the fact of blobs going back and forth is not being concealed. So you need to do something entirely different, like route through Tor and a bunch of hops through The Onion Router network, if you wanted to obscure the fact that you were connecting to ProtonMail. And doing things like comparing incoming and outgoing blobs, maybe there is a way to associate traffic surrounding the ProtonMail server farm. So that's a concern.

And then there's authentication because the person you're connecting to still has to prove that they are who they say they are to ProtonMail. Well, now that becomes a weak link because, again, if the NSA wanted to pretend that they were them, they probably could. They target some malware, get it into the computer, catch the person logging in. Now they have their credentials, and end-to-end encryption is broken. So I just sort of

wanted to acknowledge the fact that these guys existed, but that this is, first of all, not difficult. End-to-end encryption is acknowledged as what we're going to have to have. But it's only one part, one piece of the puzzle. You also have to have - there's a concern over metadata, that is, the fact of who you're communicating with still could leak, and authentication. Without absolutely knowing that no one but your intended target is able to decrypt this, it's just not safe.

And so if you really want security, it means you write something in Word, something not online, and then you do offline encryption using something like AxCrypt, which is a perfect, clean, simple encryption tool. Then you email the blob to its recipient, who takes it to a non-Internet-connected machine that the NSA has a much harder time getting into, and they decrypt it and read it there. I mean, it's not convenient. But it's secure.

And so, again, we see all these efforts to give us the benefit of security and the convenience as if it was all just transparent. And it's very, very difficult to get that. We need really good authentication mechanisms as part of that in order to make it happen. And then there's still the "did you just send email to that person," the whole metadata problem, even when the content is not known.

Also today, in The Guardian, Ladar Levison had a relatively short piece. Their headline said: "Secrets, lies and Snowden's email: Why I was forced to shut down Lavabit." And I double-checked the date. It's like, wait a minute, you know, how many times is Ladar going to tell us this, because I'm sure he's told us already several times. But I did like his summary where he said: "The problem here is technological. Until any communication has been decrypted and the contents parsed, it is currently impossible for a surveillance device to determine which network connections belong to any given suspect. The government [in this instance] argued that, since the 'inspection' of the data was to be carried out by a machine, they were exempt from the normal search-and-seizure protections [provided by] the Fourth Amendment."

Well, that's horrifying. The idea that a court can say, well, search-and-seizure is only if people do it, not if machines do it. So obviously this didn't go to any Supreme Court challenge. But sooner or later, I mean, if this kind of reasoning is used, you can imagine the EFF just blowing a gasket and saying, wait a minute. The Fourth Amendment certainly covers automated search-and-seizure. But in this case Ladar as standing in front of a court, and that's what they decided.

And then he finishes, saying: "More importantly for my case, the prosecution also argued that my users had no expectation of privacy, even though the service I provided - encryption - is designed for users' privacy." It's like, again, how can the court say that? Just they declare it, and so that lets them off the hook? I mean, the prosecution said it because of course they want that to be the case. But the court should have said, wait a minute, of course there's an expectation of privacy. That's the whole reason they were using a service that specifically provided encryption.

And finally Ladar sums it up, saying: "If my experience serves any purpose, it is to illustrate what most already know: Courts must not be allowed to consider matters of great importance under the shroud of secrecy lest we find ourselves summarily deprived of meaningful due process. If we allow our government to continue operating in secret, it is only a matter of time before you or a loved one find yourself in a position like I did - standing in a secret courtroom, alone, and without any of the meaningful protections that were always supposed to be the people's defense against an abuse of the state's power." So I thought that was a great summary. And, I mean, it really does...

Leo: Wow.

Steve: It is the chilling aspect.

Leo: Yeah, it's very chilling. Speaking of chilling, this next one is really scary.

Steve: Oh, boy. Okay. So this one also got a lot of attention. I'm sure it was the most tweeted to me story of the week, titled "How I bypassed two-factor authentication on Google Facebook, Yahoo!, LinkedIn, and others." So this tells the story of a clever young hacker. He said when he first encountered two-factor authentication when he was 16, two years ago. So now he's 18. And he's thinking about this, and he realizes there's a fundamental vulnerability in the implementation of just about everybody's two-factor authentication. It's a feature, and it's a bug.

So what's the problem? We already well understand, the listeners of this podcast, that security is about a chain of links, and that obviously the chain's net strength is only as strong as the weakest link. We run across this analogy because it fits and suits so many different instances where really fancy encryption is, like, being communicated over a string with two tin cans, and you can tap that. And so it's like, well, okay. Sorry about that. Here, that's exactly what we have. What he recognized is that, in all of these systems, the second-factor code - now, I should mention these are systems where you're being asked to prove ownership of a second factor by them sending you a code.

So texting is one technique where you've preregistered your cell phone's number. And so they text you a code which presumably only you can receive because you're holding the phone in your hand. And then you enter the code that was received in order to close the loop back to the web browser and server to prove that, yes, I'm in possession of this device. So that's the loop. The problem is that all of these services allow that code to be delivered via voice to a user's cell phone. So they will phone you and speak it to you. And then you either have a good memory, or you write it down, and then you enter it. Or maybe you type it in as you're listening to it.

But he recognized the way cell phone voicemail works is that, if the caller is busy, it immediately goes to voicemail. And voicemail was never really designed with security in mind. And remember we covered the story about all the celebrity voicemail hacking that happened a few years back because it turns out that they left their PINs defaulted to 0000, or there wasn't one, and you just entered their number into the voicemail box and out came their messages. So if the second factor is made to get stored in the user's voicemail box because the attacker has phoned them first so that they're on the phone when the attacker attempts to log into the website and say, yes, send this second factor to my phone, that gets diverted to voicemail, and the voicemail box is not secure. The attacker is able to dump the voicemail, get the second factor, enter that, and defeat second-factor authentication. So that's how it works. And it turns out he did it over and over and over with all of these systems.

Leo: But he has to have the login, the password, and the phone number before this attack works.

Steve: That's absolutely true. So...

Leo: Your password has to have already been compromised, plus the associated voicemail number has to have been compromised. So that's a significant amount of effort before you could do this attack.

Steve: Well, the assumption, of course, of needing a second factor is that your username and password are compromised, that you're the victim of a phishing attack.

Leo: Well, no.

Steve: Yeah. That's the only reason you need another factor is that...

Leo: It gives you two things you have to do. So you could have the second factor and not have the password. That's no good. It doubles your requirements for logging in.

Steve: Correct.

Leo: I wouldn't say that it - I think it's not exactly correct to say it's designed to back up the password. It's mutual. It just makes it twice as hard.

Steve: Right.

Leo: But so it's not insignificant to get the password. It's not, I mean...

Steve: Right. But Leo, we're talking about security. And if it was impossible to get the password, then we wouldn't need a second factor.

Leo: Right.

Steve: The reason we need a second factor is there are all kinds of ways, like a keystroke logger, for example. These are designed to defeat a keystroke logger.

Leo: And the phone number.

Steve: Yes, but it's often, and he makes the point, it's often very easy to get somebody's cell phone number. You Google them. They posted it on Facebook, or it's listed in a directory somewhere. Or, I mean, and it can be someone you know. So, yes, it's not as if this defeats login completely. But this guy was able to demonstrate that the second factor can be defeated through this clever hack.

And what he got was a disturbing array of responses. He contacted Facebook security

and told them they had this problem, and their response was: "We've temporarily disabled sending login approval codes via phone while we investigate further. Our plan is to re-enable the system when we can prompt users for interaction as part of the phone call, which should prevent us from sending codes to voicemail boxes." Perfect response. So it's like, yes, instead of just dumping it into a blind voicemail box, just add an interaction requirement so that you prevent this completely.

LinkedIn was told. They said: "Thanks for notifying us of this issue before publicly disclosing it. While the potential impact for our members is limited, we have made the decision to temporarily turn off the voice option in our two-step verification setting. We are working with the third-party vendor we use for this service to implement a fix. After the fix is in place, we will evaluate turning the voice option back on." Once again, bravo.

Google's response: "Hey, thanks for your bug report. We've taken a look at your submission and can confirm this is not a security vulnerability in a Google product. The attack presupposes a compromised password, and the actual vulnerability appears to lie in the fact that the telcos provide inadequate protection of their voicemail system. Please report this to the telcos directly. Regards, Jeremy." So Google blew him off and is still vulnerable, presumably, today.

Leo: Well, but you can remove the backup voicemail solution, which I just did.

Steve: And I would argue, anybody who's got that turned on, who's worried about this, should turn it off because Google does not have your back. Google, everybody else, well, except Yahoo!. Yahoo! never responded. This hacker confirmed, he says: "Yahoo's main services which allowed for two-factor authentication were also vulnerable to the exploit I document above. In fact, the exploit to get into Yahoo! accounts with 2FA enabled is even more severe as the attacker does not fully risk the victim knowing about account access to login. Fourteen days from disclosure, Yahoo! still hasn't replied, and hence they are still vulnerable to the 2FA bypass." So Facebook and LinkedIn did exactly what we would hope. Google blew it off and said, sorry, go talk to your telephone company. Of course that's all the telcos in the world we're talking about, not one.

Leo: Well, and some do it better than others.

Steve: And Yahoo! never responded.

Leo: So you could turn off this in Google. I think Google, and I've said this for a long time, has a much more, a much bigger vulnerability. Even if you have two-step turned on, of course, because not everything supports two-factor authentication, they do application-specific passwords. If I somehow - if somebody got my application-specific password, until I revoke it, it can be used again and again. It's not my real password. It requires no second factor. And it could be just used all the time. It's not a one-time-only password. Basically those are very dangerous, those application-specific passwords. The only protection is I can revoke it. But in order for that to work, I'd have to know it was being used. So I think that's a much bigger flaw. I mean, that's terrible. That's like 16 alphabetic characters, case-insensitive alphabetic characters. If somebody should see that or find that, then I'm really screwed.

Steve: Yup.

Leo: I don't know.

Steve: So also in the headlines - and I couldn't really understand this, except that it has the magic word in the headline. So the headline is "How to make a [and here's the magic word] quantum random number generator from a mobile phone." And I realize "quantum" gets everyone excited, as if some exotic physics are in use. It's like, oh, quantum technology, nobody understands that. And so I wanted to say, okay, wait. Let's have a little bit of a reality check here. Everyone, especially our age, Leo, but I would imagine even younger kids, are all familiar with audio hum and hiss.

Leo: Yeah. You don't hear it in digital, but in analog you do.

Steve: Right. And so hum and hiss have historically been the boogiemen of audio systems. Hum [humming], that's essentially induction of 60-cycle noise, or 50 if you're in a 50-hertz area of the world, in which case it's [lower humming]. And so in the old days you'd set up a really cool stereo system or a hi-fi, as we used to call it.

Leo: Ah, yes.

Steve: And you'd turn up the volume. And the question was, what did you hear? Was there [humming], well, actually the frequency wouldn't change, but it would get louder. And if so, you'd go around moving wires, and you'd rearrange things, and there were things like ground loops that caused this problem. And so there was a whole black art to, like, getting the hum out of your stereo system because you wanted to be able to really crank it up. But when you came to a quiet part of the song, or between tracks, you didn't want to have this hum.

The other thing that was also the bugaboo is hiss. Hiss is quantum noise. You don't need fancy stuff to have quantum noise.

Leo: In fact, that fog you were talking about.

Steve: Yes. We've always had it. I mean, hiss is there. And so hum is the pattern, and we perceive it as a pattern, therefore a tone. Hiss is toneless. It's white noise or pink noise. They have different spectrums depending upon where, like, what is carrying it. But it is quantum noise. So anything that is hissy, anything that is noisy, the static that was on the screen of "Poltergeist." Or after, in the old days, they would raise the flag, and the jets would fly by, and the station would go off the air at 2:00 a.m., and it would go [hissing]. That's quantum noise.

So the reason that we use the term now is that we want computers always to add two numbers and get the same result. We want them to absolutely be unnoisy. And in fact the whole change from analog to digital was essentially to get us out of the noise. That's what it was for. Instead of having signals that roamed around in value where we had a problem with thermal drift and components aging and power supply putting out different

voltages, where the actual voltage itself, the variable voltage carried the signal, we said, wait a minute. We're going to come up with a system called "digital" where we have amplifiers at every single stage: an AND gate, a NOR gate, a NAND, every one of those things is an amplifier. And there's some small voltage range, normally around 0.7 or 0.8 volts, where when the input crosses that, the output slams all the way the other direction, from, like, zero to five.

So right there is this massive amplification effect. And so essentially this amplifies out the noise at every stage in a logical system. That's digital computing is these amplifiers that just squeeze out the noise so that there's no chance for any to come in. And in which case, obviously, we can no longer represent data with a variable voltage or a variable current, anything analog. Now we represent, we approximate with a series of ones and zeroes which are maybe zero and five volts; or, it turns out, anything above 0.8 or below 0.8, meaning that the noise is completely ignored as long as it doesn't get near that threshold. And so digital systems are designed to keep us away from that noise threshold and give us perfect results, perfect approximations as a function of how many bits we have quantized that analog value into.

So this was all about using smart - this article that I saw so many times. because everyone knows I'm interested in entropy, was the idea that a couple dollars' worth of smartphone camera could produce an amazing amount of noise. Which is not at all surprising because a smartphone camera is now a grid of noise producers. Every single one of those little pixels on the camera is trying to be an analog receiver, an analog discriminator of photons coming in. But it is because it's running in the analog domain rather than in the digital domain, it's going to be subject to noise. There will be random effects. And actually, in a dim environment, where there aren't a lot of photons coming in, you're not sure whether a photon is going to trip it or not. And if it's actually counting photons, then you've got, again, a very good source of quantum noise.

Now you multiply that by the size of these smartphone cameras. I mean, they are massive now. So you've got many, many megapixels, many, many millions of pixels, each one of them an individual noise source. And sure enough, if you were to digitize that and look at the least significant bits of those pixels, what you would see is changes, random changes as a function of this quantum noise, which is present in the whole system. And now Leo is showing us a perfect picture of noise.

Leo: Random noise. You were wondering why I was showing that. It was the famous "They're here" scene from "Poltergeist."

Steve: So it's absolutely the case that something like a camera is able to provide a huge amount of noise. Now, we're going to talk about here, in a minute, about the noise system, the noise gathering, the entropy harvesting that I've designed for SQRL. One of the characteristics that SQRL has is a very, very modest requirement for entropy. But, for example, servers, web servers, have a huge appetite for entropy because you need a little bit for every secure connection you make. Remember that when we talked about the way SSL or TLS operates, each end sends the other a piece of randomness. They each generate something random and send it to the other to sort of protect their - so that they each take responsibility mutually for protecting the overall randomness of what they use, what they create from their randomness.

So what is significant about this video input is that these guys are talking about on the order of 1.25 gigabits per second of noise from this camera, which again is not surprising because you have so many bits of resolution and so many individual pixels, each of

which, every single one, is a separate entropy source. Now, it may be far from perfect. There might be, like, interpixel influence and coupling. So you don't want to just take that exactly as it is. You want to do a lot of post-processing. And we'll be talking about that here later in the podcast.

But what these guys did was they said, hey, you know, if anybody wants, if anyone has a need for huge amounts of entropy, then something now as cheap as a consumer video camera can give it to you. The article was a little off, though, in suggesting that this was something every smartphone needed to have because, frankly, smartphones don't need, have no application for 1.25 gbps of entropy. Even they have a need for some as they create secure communications and connections to the 'Net. But they're not a server that is inherently terminating thousands of connections per second, each of which requires and consumes some entropy. So it was an interesting article. But it's like, okay, I guess somebody could create a very high-bandwidth entropy source using a camera in a dim box lit by a dim LED, which is basically what these guys did. But most of us have no need for that kind of entropy.

Another little piece of news that was difficult to understand, and so I was glad to see that Bruce Schneier weighed in. Science Daily carried, again, a headline that was a little overinflated, saying: "New algorithm shakes up cryptography." And then they went on to say that "Researchers have solved one aspect of the discrete logarithm problem." And it's like, whoa, okay, hold on. This is DLP, the Discrete Logarithm Problem, which, for example, is the alternative to the prime factoring problem, which are the two hard things we've come up with in crypto. And so if, in fact, discrete logarithms had somehow been solved or substantially weakened, that would be really bad news. Suddenly all of that Diffie-Hellman key agreement goes out the window, and that's not good. It turns out that's also not what happened.

Science Daily went on to say: "This is considered to be one of the holy grails of algorithmic number theory, on which the security of many cryptographic systems used today is based. They [the researchers] have devised a new algorithm that calls into question [it doesn't] calls into question the security of one variant of this problem, which has been closely studied since 1976." And none of that is true.

Leo: Great. That's nice.

Steve: Yes.

Leo: Good reporting.

Steve: Yes. What they did was they broke a tiny aspect of the discrete log problem for fields of so-called "small characteristic." It would be like saying, oh, okay, we're going to multiply two primes, mmm, three and seven, and we're going to get 21. No one is going to be able to factor that. Wait a minute. Three and seven.

Leo: That's pretty obvious, yeah.

Steve: Oh, oh, yeah. Anyway, so Schneier weighs in, saying: "It's nice work and builds on a bunch of advances in this direction over the last several years. Despite headlines to

the contrary, this does not have any cryptanalytic application unless they can generalize the result, which seems unlikely to me." So I was glad to have Bruce's confirmation because I looked at the paper, and it looked to me like this was, again, nothing. And in fact that's the case. Basically, I mean, and this is good that researchers are doing this. It's not as if they're misspending their time. Certainly that's not the case because it is from this kind of constant pounding on discrete logarithms that we gain increased confidence that the actual size of fields that we're using are far enough huger than down where the academics are beginning to chip away that we're really, really sure we're safe. So, I mean, if anything, the fact that they couldn't do more says that what we have today we can really count on for the foreseeable future. So a good thing.

Now, there's another service came out of beta this week. And I got lots of requests for people asking me what I thought. I have added it to the upcoming cloud computing, synchronizing, cloud computing TNO storage podcast where I'm going to pull all this together. But I did want to acknowledge that I had seen it. And this thing is called - and this is the .net that I was thinking of, Leo, so that's where I was confused. It's called Syncthing, S-y-n-c-t-h-i-n-g, Syncthing.net. And I would summarize this as what I would recommend over BitTorrent Sync. That is, unlike BitTorrent Sync, which is completely closed source and closed protocol, which they refuse to document despite a huge amount of request to let us see the protocol, BitTorrent won't.

So here we have an open source, well-designed, cross-platform, interdevice synching tool. As they describe it, they say: "Syncthing replaces Dropbox and BitTorrent Sync with something open, trustworthy, and decentralized. Your data is your data alone, and you deserve to choose where it is stored, if it is shared with some third party, and how it's transmitted over the Internet. Using Syncthing, that control is returned to you." And they run through all the bullet points we would expect to see - private, encrypted, authenticated. For example, "Authentication: Every node is identified by a strong cryptographic certificate. Only nodes you have explicitly allowed can connect to your cluster." You're able also to, like, send a certificate to a friend, and they install it in their instance of Syncthing, and them having that authenticates them and allows them to connect to a folder in your Syncthing that you have shared.

Leo: You will have to open a port. You'll have to port-forward to make it work.

Steve: Yeah. I think it's 22,000 is the default port it runs on. You have to have one of your nodes public, although it understands Universal Plug and Play; so if you aren't comfortable forwarding yourself, it will do that for you. And they talk about open discourse, they've got forums; open source, it's there on GitHub; open protocol, it's all documented, nothing hidden. They have a Web GUI. So when you install it, it sets up a little local web server running on port 8080 on your machine. So then you just aim your browser at localhost:8080, and that allows you to access the platform-independent UI. Works on Mac OS X, Windows, Linux, FreeBSD, and Solaris. And I thought I saw something about mobile, but I'm not seeing that here, so I might be confusing...

Leo: Yeah, I don't see any mobile options.

Steve: ...it with the other story. And so I think what we're seeing here is one of two choices. I'm still a little more for the idea of a standalone client that uses a third-party storage provisioning, whether it be Google or Microsoft or whomever, and local encryption technology. But an alternative is for people, as in the BitTorrent Sync model,

and so now we have this Syncthing.net model, who want to set up - who don't want to use any third-party storage, who want to just interconnect a bunch of their devices and have folder synchronization happen. And you are able to exclude files of a certain characteristic. There's documentation in their forum about how you tell it not to synchronize some of the files that you've got in your folder. So anyway...

Leo: You know what I'm using right now? If you're going to review this stuff, I'd love to get a take on this. It's from the folks who did Drobo. It's called Transporter at FileTransporter.com.

Steve: Oh, I think that's on the list. Let me see.

Leo: So what this is doing - and I'm using this. I have a Transporter at home and a Transporter at work with a terabyte of storage on each. They use SSL. I sync over the network to them, as if they're a network-attached storage device. And they sync with one another using SSL. Plus the data is stored on the device, should somebody steal it, encrypted, as well.

Steve: Oh, so it is, it's a physical device.

Leo: You own physical devices. There's no cloud...

Steve: That's right, okay.

Leo: ...situation at all, although they do have mobile apps which can then log in and get your stuff. But I don't think they, no, they don't use a third-party.

Steve: So they're logging into the device itself, which has got a presence on the Internet.

Leo: Well, or they may use - yeah, I guess that's it, or they may use NAT. I'm not sure exactly how the transport is. But I do like the idea that there's no third party holding the data.

Steve: Right.

Leo: And it's synchronizing - I've actually been using it here and at home, and I keep stuff synchronized so it's backed up twice offsite, and I have synchronization and sharing, as well. And they claim - it's not open source, but they claim they're using SSL and standard - I don't know. I'd be curious what you think of it. This seems like a little bit more of - it's a command-line interface, Syncthing, and you've kind of got to know what you're doing.

Steve: Yeah, you're right. It's definitely not as turnkey. It's more for techies, more the kind of people who are wishing that BitTorrent would tell us what the sync protocol was. It's like, well, here's Syncthing, and it does all the things that BitTorrent does. And arguably, I like the idea that it produces a certificate so that you use that for authentication.

Leo: Yeah, that's nice.

Steve: That seems like the right solution.

Leo: Yeah, yeah.

Steve: In our Miscellany section, I wanted to note that this is the release date, today, May 20th, of Mark Russinovich's "Rogue Code."

Leo: Well, good.

Steve: Available now in hardcover and in Audible. And Simon Zerafa noted that Audible in the U.K. has it listed now. And of course Amazon's got it on their "Rogue Code" page. Has been available for the Kindle for a while. And I was, oh, about a third of the way in and got diverted into other research. But I remember exactly everything where I was, where I left Jeff, and so I need to get back and finish it because it was another one of Mark's fun reads. And again, super exactly technically accurate. As I've described it, sort of a fictionalized version of this podcast. So our listeners will be going, oh, yeah, okay. Oh, yeah, we know how that happens. Oh, yeah, okay, yeah. I mean, but wrapped around a really good story.

Leo: It's going to be a good sequel. I'm reading "Flash Boys," which is the story of high-frequency trading for reals, Michael Lewis's great book. And this will be a great segue into this because it's how high-frequency traders now give basically a backdoor to the bad guys into the financial markets.

Steve: Right.

Leo: Ooph.

Steve: And in Miscellany I wanted to mention that this is - I'm holding up for everyone to see, this was actually sitting at the front door, which I discovered after last week's podcast, when I was first talking about, unsolicited, because they were not an advertiser on this podcast, about what I was pronouncing as Harry's, H-a-r-r-y-'-s...

Leo: Harry's.

Steve: Harry's.

Leo: Harry's.

Steve: Hari Kari.

Leo: I think "Harry" is fine.

Steve: So it's Harrys.com. I did receive this handle that I like better than the fancy silver one. The silver one is a round cylinder.

Leo: Yeah. We sent you the Winston set. You got the Truman set.

Steve: Yes. And so this one is flat on several surfaces. And it's funny, too, because the first time I used it with the silver round one, I was very conscious of having no sense for which direction the blade was aimed in because there was no orientation coming, feedback on the handle. And it wasn't until after I was through using the orange one that I got - and, by the way, it's available in four colors: orange, black, gray, and white or something. And I like orange because it shows up.

Leo: You can't miss it.

Steve: And so there was, like, after I was done, I realized, oh, I wasn't even conscious of not having an orientation. So it instantly solved that problem for me. And again, the whole point of this is it gave me an amazingly good shave. I mean, just it's like - it's a different experience than I've had. Even though I've got the same number of edges, I've got five edges on my Gillette Fusion, these five are better. And I said it last week, just because it's true, and because you sent me one, and I tried it, and it's like, okay. And then I got some tweets from people who were disturbed that it's unfortunately only the U.S. and Canada. And then some others said, well, we're not going to buy it until they support your podcast. And it's like, okay, well, they're just...

Leo: That's silly.

Steve: They're still supporting TWiT. But then just today, in my Twitter feed this morning, Harrison Ward, who tweets from @HBomb341, he wrote, he said, @SGgrc, and he also said @harrys, he said: "First shave today and WOW [all caps] that's a good shave. Comparable to my seven-month-old baby's butt. Darn close to the same." And then he said: "Agree, slick handle." And I don't know what that meant because there is no rubber. It's a hard plastic handle. But at least the fact that it's not a cylinder for me made it much more pleasant. And anyway, I'm converted. This is now my shaver for as long as these guys stay in business. And I'm tempted to buy a whole bunch of blades and put them in the refrigerator.

Leo: Well, we're doing our best to keep them in business. You just stay tuned, okay? Meanwhile...

Steve: Also, "Halt and Catch Fire" that we've spoken of several times...

Leo: Oh, it's on YouTube, somebody told me.

Steve: Actually, the entire 50-minute first episode is available in preview on AMCTV.com. So if you go to AMCTV.com, technically there's a long URL, it's in the show notes for anyone who wants it. And I tweeted the show notes, and they'll be posted on the 'Net. But I imagine you can just find it if you go to AMCTV.com. I'm a little put off by it, frankly. I'll reserve judgment. I don't want to turn anyone off. But it just seemed a little, I don't know, a little overly dramatic, maybe.

amctv.com/full-episodes/halt-and-catch-fire/3571290828001/i-o-sneak-preview

Leo: Is it Compaq?

Steve: They don't - I don't think they identify by name. I kept waiting to see a Rod Canion show up, but there seemed to be nobody by that name. So I think it is that. Or maybe it's just a clone. Maybe it's not Compaq. But I think it has to be Compaq.

Leo: It's something kind of like it.

Steve: Yeah, yeah. So for anyone who's curious, I watched five minutes of it, and it was kind of - it was a little bit jerky playing as a media file through my web browser. So I thought, okay, well, I'll wait a couple weeks because it's going to be premiering, I think, on June 1st. But we'll see. Maybe. And I saw "Godzilla."

Leo: Oh. What did you think?

Steve: I have two friends who really wanted to see it. I went with them. Jenny was willing to go, bless her heart. And we all really liked it. I have to say, I mean, I don't think I even saw one, ever, Godzilla movie because the whole thing was so stupid to me. This was a solid movie. I mean, yes, you're going to have a Godzilla and a couple other things. But, boy, I mean, there was human interest. It was well produced. We didn't spend - it wasn't just them stomping around on things the whole time. I mean...

Leo: That was actually my problem with it. Every time they got in a battle, they'd cut away to the family. Who cares? I want to see the monsters fight.

Steve: Yeah, well, I thought there was enough of that. And boy, there's nothing left of San Francisco.

Leo: Yeah, they really demolished the - yeah.

Steve: Oh, baby. And it made just shy of $200 million over this first weekend.

Leo: Yeah, not bad, huh?

Steve: Its first weekend of release. So that's...

Leo: I remember when that was like, if a movie made $200 million total, ever, it was a record-holder.

Steve: Yes, yes.

Leo: Three days.

Steve: So anyway, I just have to say I thought it was a solid movie. If you're on the fence, "Godzilla" will smash it.

SQRL, I'm deep in and rolling forward and having a lot of fun. Of course all of the browser revocation stuff is behind me. The entropy harvester is written. I actually posted the source code to it this morning on my Twitter feed, if anyone is interested. There's also a link to the source code in the show notes of SQRL's entropy harvester that I'll be describing here in a minute. And I'm now moving forward. I've implemented the two most complex screens or pages of the dialogue. One is the main launching page, which has all buttons that take you to different things you want to do. The second one is the option settings page. And in order to do those, I had to finish adding a bunch of UI engine features to the system, which I did, and wrapped it up yesterday.

So after the podcast today I plow into the Identity Creation Wizard phase. So I'm moving through at very good speed and in the process creating something that is inherently multilingual, which is going to be fun to have something in 50-plus languages and counting. So I still have no idea when I'll have something. I just, you know, I'm unable to predict. I didn't know that revocation was going to happen, which caused me to suspend work on SQRL for three weeks, but I'm back to it. And I will try not to be distracted because I want to get it done so I can get back to SpinRite.

Speaking of which, I have a really nice story. And I, again, found something different. This is written by a Jeremy Webb, with two B's. And he said: "Dear Steve." He called this his "Remote Control SpinRite Story." He said: "I've always been my parents' tech support guy. When I joined the U.S. Air Force, they stationed me pretty far away from home." He doesn't divulge where.

And he said: "Thankfully, I've always been able to VNC into their computers to get things straightened out. This week I was presented by a rather unique challenge. My mother called to tell me that their computer was throwing a bunch of disk errors in Windows. Fixing this problem was particularly difficult for two reasons. First, VNC wouldn't help them if their hard disk suddenly crashed. Second, I'm currently deployed" - oh, he did

tell me - "to Afghanistan, where getting a good enough connection to VNC into their computer can be difficult." Boy, I guess. He says: "I knew that SpinRite might be able to fix the disk errors. But would I be able to walk my parents, who aren't the most tech-savvy people, through it over the phone?

"Believe it or not, I was able to get a good enough connection to VNC into their computer and make a SpinRite bootable image for them. I was then able to instruct my mother over the phone how to boot into SpinRite and start the repair process. She called me the next day and told me that SpinRite had fixed 12 errors, and that their computer was back to running normally. I cannot tell you how much we appreciate your product. It really saved the day. Jeremy." So for what it's worth, if you've got family members who are in trouble, you could consider emailing them a copy, talk them through making a boot disk and media, and fixing their problems remotely. That works, too.

Leo: That's quite a challenge. All right, Steve. That's entropy. It's entropy, dude. It's - I love it. So let's talk about your entropy generation engine. And just a reminder, I have one more break in the middle of that. ITProTV is going to do an ad. So just so you know.

Steve: We will do that. Okay. So we've got a client running on something, on a desktop, on a phone, on a tablet somewhere. And it needs entropy. We understand why we have to have entropy because in any kind of crypto there has to be something secret. In bad old crypto the algorithm was secret. And the problem is it's very difficult to keep algorithmic secrets. They leak out. Someone reverse-engineers it. It's like they have your algorithm.

Well, if the secret is the algorithm, you're in trouble because an algorithm is fundamentally difficult to change. So a big innovation was when we moved to keyed cryptography, where everybody could know what the algorithm was. People had it on their T-shirts. It was no big deal because the secret was the key, and the key could be changed. And so that's where the notion of an ephemeral key comes from, a key agreement where, on the fly, two parties are able to arrive at the same secret, even when their conversation might be eavesdropped on.

So for all of these things, in order to get these keys, we need them to be unpredictable. A key that comes from a counter, for example, would be bad because, if anyone got your current key, all they have to do is add one, add one, add one, add one until they catch up to where you are, and then they've got your new key. So you don't want a counter to generate your keys. You want something which is unpredictable so that even somebody knowing a lot about what you're doing, for example, your current key, has no idea, hopefully, what your past keys were, nor what your future keys will be. So you want that to be - you want essentially there to be as much randomness in the key as possible.

So when we start talking about specifically what we want is we want some number of bits of key length, and every single bit to have a 50/50 chance of being a one or a zero, and to be as independent as possible, if not completely independent, virtually independent of any other bits. So that knowing a few doesn't tell you anything about any of the others, even what their bias might be, for example.

And in all of this, when I'm sitting here saying, okay, I've got to solve this problem in a really, really robust fashion for Windows, and I'm hoping that other developers of SQRL clients on other platforms will consider my solution and consider adopting it because the worst thing to do is to leave the issue of entropy to the end and just say, okay, well, I

need a random number. Or just to call the RAND function in whatever language you're using and assume that that's useful. Many languages, even today, base it on a very simple multiply-and-add, called a "linear congruential" pseudorandom number generator, that to varying degrees of quality does a really poor job. So it may be varying, but it doesn't vary very high.

So sitting here saying, okay, I have to actually get random numbers. I mean, like, for real. Where do I get them? There are two things we need to concern ourselves with. We have no regard for an attacker, that is, we need random numbers without considering the attack side, which is we want them to be high quality. We absolutely, for SQRL and for, I mean, just in general for cryptographic protocols, as I said, we want all of the bits to be random.

Now, some people who learned about SQRL said, wait a minute, you're using a 256-bit key. What if there's a collision? Well, if you're arriving at them randomly, two people could arrive at the same thing randomly. So that's worth addressing briefly. And so let's, again, people, humans, are probably the best organisms on the planet for understanding probability, and even so we're really bad at it. We talk about, glibly, oh, 256 bits, you know, how hard could that be? And so it takes putting in and giving it some perspective to understand this.

So with 256 bits, I assert that the risk of collision, assuming that they're random, okay, so for the moment we assume all 256 bits are individually chosen to be ones and zeroes at random from a really good source of entropy. So we'll have that as a given. So we're just looking at this collection of bits itself, assuming they're random. What does it really mean to have them collide? Well, it means that two people are going to have 256 individual 50/50 decisions which are each identical. So not like overall identical. I mean, the first one, each of them makes the same first choice; each of them makes the same second choice; each of them makes the same third choice for each bit; 256 times, not one bit different. That's a collision. So how likely is that?

Good news is we've got math geniuses who have sat around and figured out what the chances are, not of a given value coming up, but within a population of people, each choosing their own number - and this is the so-called "birthday attack" - what is the chance that any two of them in this group will arrive independently at the same number, assuming all the bits are chosen randomly?

So it turns out that we can estimate the probability, if we have a number of sets of bits chosen is P, and the sets are of N bits. So in our case N would be 256. And let's just say a billion because that's way more than there are people who are ever going to use SQRL. So P is a billion, and N is 256. So the probability of a collision is about P^2 over 2^N+1. That mathematicians have figured that out. Now, that's an approximation, but it holds for situations about like this. What that means in terms of math, when you plug in the numbers, is that, with our 256 bits and a billion different keys, the probability is about 4.3 times 10^-60.

Now, again, we're bad with numbers, 4.3 times 10^-60. So someone would say, well, that could happen. Okay, 4.3 times 10^-60. So again, this is zero point, and then behind the decimal are 60 zeroes, way out there, and then a 43. But again, to give it some context, an extinction-level event, an ELE, occurs, we estimate, about once every 30 million years, on average. So that means that there's a certain probability of it occurring in the next second. Like within this, like, one second from now, okay, it just happened. The chance of us - we just survived it, oh, and again, and again. And every second that goes by we survived a one-second probability of an extinction-level event. The probability of our surviving from second to second is 10^-15.

So that gives you a sense. We're surviving, with a probability of 10^-15. The chance of a billion people, any two people in a billion having a collision of 256 bits chosen truly at random is on the order of 10^-60, or 45 orders of 10 magnitude more probable. I mean, 45 order of magnitude less probable than the collision that we're going to be all killed in the next second. So anyone who's worrying about 256 bits acquired with really true high entropy is worried about the wrong thing. Just stare up at the sky because that's 10^45 chance more likely to happen than any two in a billion people having their keys collide. So that's our need without regard for an attacker.

But the system, since it's going to be operating on Windows and on pads and phones, has to also be robust in the face of attack. It's got to be resistant to anything an attacker can do. So, and that's either a passive or an active attacker. A passive attacker may be able to somehow gain access to our secrets if, for example, they're inadvertently swapped out to the hard drive, or if the system is suspended and RAM is saved out to the hard drive for later resumption. Or if the system is suspended in RAM and then, for example, it's got a Thunderbolt or a Firewire interface that gives it DMA access to RAM. That's a problem. Or, remember, we talked about people spraying the RAM on the bottom of the laptop with Freon and then yanking it out really quickly and going and putting it in something else and powering it up before the RAM had a chance to - all the data had a chance to bleed away. So we need protection from eavesdropping.

But then we're also an application running on the operating system. So there's an interface, the so-called API, the Application Programming Interface, where we talk to the operating system. And that's a point of vulnerability. Even if our process itself, if the process's RAM is protected, there's a chance that somebody could get into the operating system kernel or stick a shim of some sort in between our interaction with the operating system and either passively eavesdrop the data that we're getting from the operating system, or perhaps actively change it.

For example, if we ask the OS for its cryptographically secure random number, and we just blindly use it, then what if the OS wasn't as secure as our app? We've gone to all these measures. We've made sure we can't get swapped out. We've locked ourselves in RAM. We sense hibernation, we sense suspension, and we deal with all that. And I've already written all that code for SQRL. There is a protected region of memory where all of SQRL's secrets are kept. And at the first sign of anything happening that would endanger it, it is wiped to make sure that it is never saved in any state that would allow someone to get to it. And even then it's kept encrypted until the instant it's needed, it's used, and then wiped. So nothing stays around just out of laziness or lack of care.

But still, what if an attacker was able to intercept our API call for a random number and just returned zeroes? Now the attacker knows that they've arranged to feed us zeroes, or a pattern. You know, A's in hex is 10101010. We might not detect that, where we might detect all zeroes and choose not to use it. Now the attacker has, by controlling what they're doing on the outside, has influenced what we're doing on the inside in a way that helps them. So we need to consider that. So we need to think in terms of how do we achieve a reliable set of entropy where we both need high-quality entropy just from a standpoint of needing randomness, but also resistant to it escaping from us to its being passively eavesdropped on, or an attacker actively interfering with our attempt to collect it.

Then we come to the question of how much do we need because this is a huge factor in the design. I talked earlier in the podcast about how a server inherently needs huge amounts of entropy because it is potentially terminating tens of thousands of SSL connections per second. Every single one of those connections it has to provide a random cookie, which it combines with the randomness that the user provided and sends back to

the user in the client-hello handshake in order to - or the client-hello and server-hello handshake in order to generate an SSL connection. So its need is massive.

One of the problems with physical processes, anything quantum is by definition a physical process. It might be an electron going against a diode's PN junction, and tunneling through, and something detects that, amplifies it, and that's a quantum event. Or it might be a random photon coming in and hitting a photo sensor, and that's a quantum event. But it's a physical process which inherently means there's a limit to the rate at which those things are happening. And that rate could be substantially lower than the appetite, the consumption rate that, for example, something that is hugely hungry for entropy has.

So the way we've traditionally handled that is we combine a software algorithm, which is a pseudorandom number generator, where we seed it with a true random number source. And that, for example, is exactly what Intel built into their latest model processors for the last couple years now, I think they started in about 2012, from I want to say Ivy Bridge. I think that was the generation of processor where it's the RdRand instruction, RdRand. And they have on chip a quantum noise source which generates noise. It is filtered and conditioned, and then it's used to generate the seed for a simple counter AES pseudorandom number generator. So that some randomness sets the initial count of a 128-bit counter, and another sets the initial value for a 256-bit counter, both which feed into the AES cipher. And the advantage of that is that those counters are able to spin at whatever rate is necessary and produce very high-quality, because the AES cipher is good, very high-quality pseudorandom numbers.

Now, the reason that doesn't work for me are a couple. For one thing, I don't need, SQRL doesn't need a huge amount of entropy in terms of time. We'll talk about what its exact needs are in a second. But a problem with the counter system is that you have to absolutely be able to protect its state from an attacker because, if an attacker were ever able to somehow capture the current key being used on the AES cipher, then if they looked at a value that was output, and they knew what the key was, they could run that the other direction through AES because remember that AES is a symmetric cipher. So they would decrypt the output. That would give them the current state of the input counter. And now they can go both directions. They can go and look at all the numbers. They can essentially compute all of the numbers that have recently been generated and will be generated in the future.

So while that kind of a counter-based, cipher-based, pseudorandom number generator is very nice if you can put it in silicon, where you can guarantee no access, it's dangerous to have it in software where you can't really, in a trustworthy fashion, make the same sort of security assertion. So it worked for Intel because they're creating silicon. They can put all this in silicon so that the only programmatic interface is at the other end of a FIFO. And that's what they've got is a first in, first out queue, which they fill up, and then they stop their pseudorandom number generator, which is seeded by a true random number generator, until the FIFO gets to a certain level of emptiness, and then they turn that on again. And that's just really for power consumption's sake. And then they fill up the buffer again. And then, in the background, the true random number generator that cannot produce true random numbers fast enough to meet the need, for example, of a high-end server application, it can at least reseed.

So what's going on again in silicon for them is it is that their true random numbers coming from a hardware process that is fundamentally limited, bandwidth limited, it is constantly reseeding the pseudorandom number generator. The reason you want that is you never want it to repeat. Not that it's going to any time soon. But you also just, in terms of security margin, you only want to produce so many pseudorandom numbers

from the same seeding source before cryptographers just get a little uncomfortable with the fact that we are using a fully deterministic source. So if anyone had a way of ever reversing this, that would be trouble. So constantly reseeding is the way they solve that problem.

Okay. But we don't have the need in SQRL for almost any entropy. Remember, one of the cool things about SQRL is the actual protocol is zero entropy consumption. We are handed a challenge by the server. So, that is, the server client that is the user is handed a challenge from the server. That does require some entropy. There's a nonce in there which essentially we sign using our private key derived from our super secret key, which we never share, and the domain name. So the domain name and our super secret key generates our identity, which we have previously registered with the website. Then it sends us something random. We sign that and send it back, and it verifies the signature with our public key, which is has. That's the whole protocol. I mean, that's what's so cool about SQRL is its elegance and its simplicity.

But notice that during that transaction all we were doing on a transaction basis was signing something we were given. No entropy needed. So that means that the SQRL client's requirement for entropy is only for creating an identity, which might happen once in your lifetime. Hopefully you never lose the identity, lose control of it. SQRL provides mechanisms for dealing with that, if you do. You might want a second identity on the same site, in which case you would use it again. Or each family member might have their own. So there are reasons to do it more than once. But typically very, very, very seldom.

Then the one other time we use entropy is when we are encrypting a user's password. Since the password itself may not be perfect entropy - as a passphrase, and it's coming from people, it's probably not going to be. Since it may not be perfect entropy, then we do need a nonce to salt our encryption in order to protect the password. So those are the only two things that SQRL needs entropy for at all. In other words, very little. So the solution I came up with, essentially it's simple, elegant, and easy to describe, and it solves every one of those requirements that I've laid out so far.

Leo: Steve Gibson is here. This is an education. He's talking about SQRL, which is his proposal for a sane way of authenticating with web servers. And you said you found a magic key.

Steve: Well, so SQRL doesn't need a huge volume of entropy. But what we want is an absolutely attack-proof, high-entropy chunk of randomness, relatively seldom. So we want a solution where, if anyone had, like, compromised a previous key, they can't figure out what the next one's going to be; or if they get the current one, they don't know what the one before was. So we want a solution which is not simply algorithmic and, that is to say, pseudorandom number. We want true randomness, but in a way which is robust against every possible kind of attack from the outside of the application.

Now, the solution is a hash. Hashing is a deliberately lossy function. And we have to talk a little bit about exactly what the characteristics of a hash are that enamor us of it for this application. A cipher, like AES, for example, is not lossy. Meaning that, if we put in, with a given key, we put in 128 bits, we're going to get a different 128 bits out where the relationship between what's in and what's out is a function of the key. So it's a keyed mapping between every possible combination of 128 bits in and some different combination of 128 bits out for each one in, controlled by the key. And that's reversible. So that if you had the output, you can go back and get the input. That's what decryption of the encryption is, is that reversibility.

A hash is completely different. It exists to reduce any input of any length into a fixed-size output, a so-called "digest." Or it's also known as a compression function because it can compress a large corpus into a fixed size, essentially a signature. Now, what is so cool about this is that, if you imagine a large block going in, say that we - in the case of SQRL, I use SHA-512. That's the obviously double-size SHA-256. These are all members of the so-called SHA-2 family of hashes. I use an SHA-512 because, in a single event, I need more than 256 bits.

When SQRL's creating a new identity, I need 256 bits just for the master key. Then I need 80 bits for the so-called "rescue code," which is your get-out-of-jail card if you really get yourselves in trouble. And 128 bits as the nonce, which is used to encrypt that. I end up with, like, what, 448 bits I need, so 512 is perfect for that. So I never need more than 512 bits at once, and I don't need them very often. So imagine that we had 100K of data behind this hash. We were going to feed 100KB of data through this hash. What is so cool, and imagine the 100K all, like, sitting out there, not yet hashed, but it's all there. And the result of hashing that reduces to 512 bits.

What's so cool about the hash functions, the really good cryptographically secure hashes, is that, if we reach back and change any single bit of that entire 100KB that is input into the hash, just one bit, on average every bit will change half the time. So that's the way to think about this. Essentially, no matter how large the input is, the effect of every single bit coming in is such that changing that one bit, any one of those bits, would invert every output bit half the time, independently, with no pattern that's detectable.

So this is, I mean, this is just - I just love this function. It is so cool because what this allows us to do is, in the background, continuously stream a diverse set of sources of unpredictable, somewhat random, not necessarily fabulously random, but unknown, unpredictable, changing stuff into this hash function. And the normal way we think of hashing is that you hash something. That is, there's a start and an end. That is, you have a file, and you want to hash the file to get a signature; or you've got a communications stream, and you hash that to get the result.

Well, in terms of the actual algorithm, there's an initialization phase. Then there's sort of a data collecting phase, and then a finalizing phase. The finalizing phase goes through some final work and is what finally produces the 512 bits or whatever width the hash is. What that means is, if we, instead of having a fixed buffer that we're going to hash, if we open the hash function and just pour data in, we don't know how long it is. We don't care how long it is. We don't care or know how much it is. We just pour it into the hash continuously until we first need a random number. And then we stop the stream, finalize the hash, and obtain 512 bits. And the 512 bits we obtain is a function of every single bit that we dumped in during the entire time that stream was open.

So that's the system that I've designed for SQRL. I have this notion of many system things which are of unknown value, and they're changing at different times in a way that no one can predict, and we just pour them all in. For example, Windows maintains a clock of processor counts which runs at 3.something, 3.2, whatever your clock rate is, gigahertz. And that's a 64-bit counter. So there's one thing, which is from the time the chip was last reset, that's the count. And that's with a resolution, sub-nanosecond resolution because it's running at 3.something billion counts per second, so many counts per nanosecond. And what we do is we just take a snapshot of it.

Now, itself, it's not very random. It's a counter. So the most significant bits aren't going to be changing much. But the least significant bits are going like the wind. And we take a snapshot. We don't know what they are. But we don't care. But that's just one of many sources. Also Windows maintains with much less resolution, a hundred nanosecond

resolution, but still that's a tenth of a microsecond, the time that all threads are spent in user mode, the time that all threads are spent in the kernel, the time that the scheduler is idle. Hopefully that's something. An instantaneous shot of the Windows global usage statistics. The time that the particular thread doing the entropy harvesting has spent in user mode, in the kernel mode, idle. And the time that it was - the instant where the hundred nanosecond resolution was created. The same information for the process, the SQRL process, how much time has it spent, all of its threads, in user mode, kernel mode, idle, and the instant it was created.

Then we also have some information which may not change at all, but it's probably unique to the process: the process ID, the thread ID. A bunch of different system handle values that are probably static while we're collecting it, but change from instance to instance, and certainly from user to user and system to system. And, for example, the instantaneous X and Y position of the mouse. Then we also have the platform's, in this case Windows, the cryptographic random number generator.

Now, that was no good in Windows 2000. It was notoriously flaky. So Microsoft got serious about it, and they fixed it in XP. And I've seen a list of what it collects in order to generate its own private internal buffer of data, which is one of the things that we ask for. I mean, it's everything in the system. It's packets, counts of network packets and timing and disk accesses, I mean, just everything that the guys in the kernel could think of, this is collecting. And much of it is tied into real-world events which are producing true entropy, not algorithmic entropy approximations. So we ask Windows for that.

Then there's also that RdRand instruction, which on all newer systems will be present, but won't be if they're older than two years. So I ask for it. And if it's available, that gets poured in, too. And then there's one last really cool thing because things like all those Windows variables, and even the cryptographic random number generator, remember I said that there was a chance that, in theory, bad guys could insert hooks around the application so that we thought we were getting good random values, but they were being messed with.

Well, anything that we don't ask outside of the process for, we're not crossing the process boundary, nobody can intercept. So the RdRand instruction is one of those things. It's us and the processor. It's us issuing an instruction saying give me a 64-bit random value, or a 32-bit random value. And we do it a bunch of times, and we suck some of the pseudorandom data out of that FIFO buffer. Because it's us directly accessing the chip, no software can intervene in that.

And then there's one thing, though, which is also useful because that's only been around since Ivy Bridge, and a lot of other people have older processors. From the dawn of time there's been something called RDTSC, which is Read Time Stamp Counter. That's a 64-bit counter which increments with the - it is incremented by the clock of the chip. So once again it's running at gigahertz speed. Now, not only is it just a blur, but again, even in the oldest still-running Intel chip, it was there in the Pentium, the very first Pentium. It's an instruction which no attacker can intercept our access to. And no attacker can know the value we got. And it is extremely random.

One of the things that's happened with processor design is, while all of the computation work that's being done is deterministic, you always add two numbers and get the same result. If you tell it to jump to a certain location, it's going to do that. But the one thing which we softened dramatically is the determination of when. In this constant quest for performance, we've gone to all kinds of lengths to increase the performance. We've got, as we've talked about in various podcasts, Level 1, Level 2, Level 3 caches. So some things, some instructions will execute quickly because they're right there in the Level 1

cache. If not, then they have to go to the Level 2 cache. And if not, to the Level 3; and, god help us, they may need to go out to main memory in order to actually get serviced.

But we also have so-called "superscalar processor architecture" now where multiple streams of instructions are being processed at the same time. Instructions themselves take varying lengths of time because, for example, if I add two things and get EAX, the result in the EAX register, and then I immediately want to OR EAX with itself to set the status to see what the result was, that ends up stalling because the processor has just - the previous instruction changed EAX, and now I'm wanting to use that value.

Well, it turns out that, if we were to stagger the instructions so that we were doing something else first, before needing the result from the previous instruction, those things can be done in parallel. So there's parallel channels through. Also there's out-of-order instructions. The processor looks at what it's being asked to do, and it looks down ahead of the instruction flow to see if there are instructions that are not dependent upon results that it has yet to compute. And, if so, it'll do those ahead of time and just hold their answers.

Anyway, the point is the actual internals of today's processors is incredible chaotic. And branch prediction, the system is trying to guess whether we're going to take a branch or not based on the history of the branches that we've been taking. So the instantaneous contents of the caches, the branch prediction, the multiple pipelines that we're executing, the history of pipeline stalls, all of this contributes to the fact that, when we issue this RDTSC instruction, we have no idea what time of day it is. We have no idea what value we're going to get. And we don't care. We take that, and we dump it into the hash.

But the point is that no attacker can possibly know. And essentially, the exact value we get is based on the entire history, or at least a local history, of what the whole system is doing, and it is itself chaotic and unknowable. And everything I've just said, we do minimal of 50 times per second. Fifty times a second there is a low-priority thread which captures all of that data and dumps it into the hash. And then I also have that inserted into the so-called "message loops" in the application so that every movement of the mouse across the UI generates messages. Every message causes a snapshot, that instantaneous snapshot of all of those values in the machine, cryptographic data, random data from the Intel chip, the instantaneous unknowable state of that time stamp counter, and just pours it into the hash.

And then finally, at some point, if we say we want to create an identity, or we want to change our password, and so that SQRL needs some absolutely high-quality randomness that nobody else on the planet will ever have, no one can affect, no one can attack, no one can influence in any way, we close the hash, and the history of everything we poured in, with every single bit being significant. Even if some of the stuff, things like processor ID or your MAC address, well, that's unique for you. That's not going to change from run to run, but it's unique compared to everyone else. See, even low-entropy things will mix in with all of the high-entropy things and combine to give us an absolutely random 512 bits result. And that's the way SQRL's entropy is harvested. And I can't think, and none of the people in the user group who have been working with me analyzing this can think, of any way that an attacker can get anything from us. So that's what SQRL does. Whew.

Leo: Two hours and 10 minutes later, that's what SQRL does. Ladies and gentlemen, Steve Gibson. Steve is the - by the way, I've been playing the whole time. People wonder, what do you do, Leo, while Steve's talking?

Steve: Blek, Blek.

Leo: You know about Blek?

Steve: Of course.

Leo: It's awesome, isn't it?

Steve: I saw it a week ago when Andy told you about it. And the good news is that I have convenient memory. I keep forgetting it, so it hasn't had a chance to have a grip on me.

Leo: Forget Blek. It's just going to get you in trouble.

Steve: Oh, my lord. Are those black holes?

Leo: Yeah.

Steve: There's no way you can get to those dots through that...

Leo: Oh, dude, I'm up to 31 now, and it's getting harder and harder. Actually 33. I just warn you. Actually, this one's not as hard as it looks because, for instance, if I do this, it goes, well, you've got to avoid those black holes; right? It shoots, and then it goes - oh, man.

Steve: Oh, look, they have little ricochets in them.

Leo: Yeah, some of them have little shooters.

Steve: The little white dot. Ah, very cool.

Leo: Yeah, yeah, yeah. But then I can't figure out how to get those. I was wondering if you knew about that because, given your interest in Rails, this is kind of like Rails.

Steve: Oh, boy, you're right. That would do me in.

Leo: Yeah, yeah. B-l-e-k, if you want to spend a buck ninety-nine and waste the rest of your weekend. Steve Gibson is at GRC.com. Next week a feedback episode, Stevie?

Steve: We're going to do a Q&A. And I got an email from Brett Glass, whom I've mentioned several times on the show, a super, super smart network-aware techie. And he said to me, he said, "Steve, I just need a chance to explain Net Neutrality in a way that..."

Leo: Oh, great.

Steve: "...that I think people will all understand." So I'm going to ask Brett if he'd like to be a special guest for part of our Q&A episode next week and give him a chance to really explain it in a way that he thinks is useful because I'm sure he does understand it.

Leo: Yeah. And Brett's great. I love Brett. So that'd be - I think that's good. There was a video I was showing, made by a woman named Vi Hart, that was really excellent. But then I realized it was somewhat of a broken analogy. And we're just trying to find the exact analogy so everybody can understand this. And of course the FCC has proposed those new rules, and there is an alternative, and they're taking comment. They're saying do you really - maybe you want us to declare broadband providers as common carriers, a utility that we can regulate. The court said that's what you're going to need to do. FCC doesn't want to do it. I think we want them to do it. But I'd love to hear what Brett has to say.

Steve: Yeah. Yes.

Leo: Next week. That'll be good. But we'll also take questions and answers on all topics.

Steve: Yup, and a Q&A.

Leo: Q&A. So go to GRC.com/feedback to leave your questions: GRC.com/feedback. While you're there pick up SpinRite, the world's best hard drive maintenance and recovery utility. And of course Steve's got lots of freebies there, including 16Kb audio versions of this show for the bandwidth-impaired, full transcriptions so you can read along as you listen: GRC.com.

We have the high-quality audio plus video at our site, TWiT.tv/sn. Of course you can subscribe to one of those in Stitcher or iTunes, or if you use our apps, and you listen every week, you won't miss a minute of Security Now!. Thank you so much, Steve. We'll talk again next week, right here.

Steve: Thanks, Leo.

Leo: He's shaving. He's shaving.