SECURITY NOW!

Transcript of Episode #117

Even More Perfect Paper Passwords

Description: Steve and Leo discuss the updated second version of Steve's Perfect Paper Passwords (PPP) system and examine a number of interesting subtle questions such as whether it's better to have fully random equally probable passwords or true one-time-only passwords; and how, whether, and why attack strategies affect that decision.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-117.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-117-lq.mp3

INTRO: Netcasts you love, from people you trust. This is TWiT.

Leo Laporte: Bandwidth for Security Now! is provided by AOL Radio at AOL.com/podcasting.

This is Security Now! with Steve Gibson, Episode 117 for November 8, 2007: Even More Perfect Paper Passwords.

It's time for Security Now! with the even more perfect, the better than ever Steve Gibson. Hello, Steve Gibson.

Steve Gibson: Hello, Leo. Great to talk to you again.

Leo: This is the Even More Perfect Paper Password episode. Steve, we can't go any farther than this. There cannot be any more perfect paper passwords than this. Because then even more is, like, it.

Steve: Well, we have had a really active two weeks of amazing, really amazing think tank work. And actually the newsgroup is called GRC.thinktank over in the newsgroups. It's been two weeks since we unveiled the Perfect Paper Passwords system. It's in Version 2. I want to talk about what's changed. And some really interesting issues have come up and been battled with. So this is going to be - this is sort of a propellerhead episode. Technically it's about Even More Perfect Paper Passwords. But I think people are going to find it really interesting because of some of the sort of edge conditions which have come up and we've addressed.

Leo: Very cool. Well, let's - where should we start? Do you want to do anything from previous episodes?

Steve: I don't really have any errata. And the one thing that Sue, my operations manager, who's been with me for 20 years, has asked me a couple times to mention and I keep forgetting it, is that relative to SpinRite licenses she keeps running across companies that aren't aware that we have an enterprise license in addition to a site license, the idea being that an enterprise license is a multi-site license. And it's really my fault because I don't have anything on our website about the enterprise license.

Leo: Well, come on.

Steve: But - yeah, I know. But we came up with it because there were big companies like IBM and others who really did want a multi-site license. And it didn't make any sense for us to ask them to buy four copies of SpinRite for every one of their sites. So I'm going to get my pages updated so that I explicitly explain that. But the idea is that any company that wants an enterprise license, that is, a multi-site license where they can use it within their entire organization, what we've done is we've divided the world into U.S. and non-U.S. And a domestic enterprise license is just 10 copies of SpinRite. And if a company is maintaining 10 copies of SpinRite at its current version, then they are enterprise licensed, and they can use SpinRite anywhere in the U.S. And then we similarly have anywhere outside the U.S. is another 10. So if it's an international company, and they want both inside the U.S. and outside, that would be a total of 20 copies. Or they could have one or the other, if not both. And...

Leo: Can you put this on the website?

Steve: I've got to put this on the website. But I just, you know...

Leo: I can't even follow it.

Steve: Sue's been bugging me because the way we have a - an individual can just buy one copy, have one license. Then we decided, okay, if someone has four, then that qualifies them as a site license, where they can use it on all the machines they've got in a single physical location. And then we said, okay, and if someone has 10 copies of SpinRite, then they're able to use it enterprise wide. So I just - I've said it now, and I've been promising Sue that I would. I will get this updated on our website so that people who haven't heard this will know, too.

Leo: Thank goodness. Even more - even more perfect...

Steve: Well, I just realized that I could have talked about something that I want to definitely talk about before and kind of put it in the errata section, although it's really not errata. I want to start first, before we get into Even More Perfect Paper Passwords, bringing people up to speed on some really cool news from VeriSign. I FedExed one to you, which you've - yesterday. There is a new form factor for the six-digit one-time password authentication solutions offered by VeriSign.

Leo: Does this replace the keychain fob thing?

Steve: Yes. And there's even better news, Leo. VeriSign's backend server system was recently updated to allow the simultaneous testing of up to five tokens, five hardware credentials. And PayPal supports it. So, and I've already done it, and it works.

Leo: So I can lose the fob.

Steve: Yes. Well, A, you can lose the fob; or you can register multiple fobs; or this new thing, and we didn't really describe it yet. What they've done is they've got, I mean, an absolutely credit card perfect form factor. It's the size of a credit card. There's no lumps or bumps. I mean, it's an amazing little piece of technology. It's got a - actually it's an eight-digit eInk window, so it uses eInk that you and I are fans of with the Sony Reader and other readers coming in the future. So it uses no power. It has a three-year warranty from VeriSign. It's more expensive than the fob from PayPal, which you remember because PayPal was supporting its adoption. That was only $5 from PayPal. But so I think it's like $48 from VeriSign.

Leo: But the convenience compared to the fob, you can stick this in your wallet.

Steve: Oh, well, that's just it. I mean, I have my wallet in my back pocket, and so it's with me whenever I have my pants on, which is pretty much all the time.

Leo: I hope so.

Steve: More often than I would like. And so, I mean, it's always with me. It's no longer a blob on my key ring. And so this, for example, works with OpenID, works with eBay, works with PayPal, and any other people who are part of this VeriSign identity protection backend. So the URL, and we'll have it on our show notes also, for people who are interested in just, like, seeing it, it's IDProtect.verisign.com. IDProtect.verisign.com will take you to the page where you see both what I call the "football," the original football fob token credential dongle thingy, and this really new credit card. But because their server can simultaneously test for five, up to five, for example, on my PayPal account I've got my original PayPal credential and also this credit card gizmo, both registered as PayPal authentication. So, for example, that allows me to keep my - because my own physical environment here where I'm working is secure, I can keep the little football guy here next to me on my desk and have the credit card in my wallet. So if I'm ever out and about or am using my laptop at a Starbucks or something, I'm still able to use PayPal using that very cool form factor, this credit card form factor, and at the same time still have access to the little football token.

Leo: So I could use either one. Either one would work.

Steve: Exactly.

Leo: Oh, neat. Now, how do I tell - well, this is a little personal support.

Steve: It's in there. I stumbled around in PayPal. But it's under Security Key and...

Leo: You can add an additional security key, basically.

Steve: Yes. And it shows you, I mean, it says on the UI, remove one, add one, I think you can disable one. It's got a whole nice little UI, like things you can do. And it'll give you up to five that you're simultaneously able to register. And this all uses VeriSign on the back end.

I also should mention that - I mentioned, I guess it was a couple weeks ago, that I never heard back from VeriSign on their pursuit of allowing me to play with their API for doing this backend authentication because I just sort of wanted to see what it was like. They have provided since all of the documentation for their API. And I have to say, I mean, I haven't yet had a chance to bring it up to speed. But I'm going to because I want to actually use it and get the experience of talking to their backend servers. But I've looked at the protocol extensively, and it is as clean as it could possibly be from a privacy standpoint. That is, they're collecting no information they don't need. There's no user identification in there. It's just essentially the serial number of the token. And you say here's a serial number of the token; here's the current display being showed by the token. Is this valid or not? So it's absolutely, absolutely clean.

Leo: Well, thank you for sending me one. I didn't realize they were that expensive. I appreciate this.

Steve: Well, they actually sent me two.

Leo: Oh, good, okay.

Steve: So, and then I got a couple more because I actually think that I will use this in addition to the Perfect Paper Password system for GRC's authentication. I'm going to implement it and give it a try and get a sense for how it works.

Leo: Well, I can't wait. I'm going to figure out how to do this right now.

Steve: Oh, I mean, you add yourself to PayPal. And now, as you said, you've got that in your wallet. And again, as this spreads, I mean, it's an OpenID authentication because it's completely compatible with the whole pip.verisignlabs.com OpenID system that we've talked about before, as well.

Leo: I have to add it, I guess, to my PIP account, as well.

Steve: Probably.

Leo: Yeah, yeah. Cool.

Steve: So I wanted to bring people up to speed on that.

Leo: We're moving ahead with this system. It's really great.

Steve: Yeah, it really is evolving nicely.

Leo: Yeah, I'm glad...

Steve: Okay. So Even More Perfect Paper Passwords.

Leo: Even more, ladies and gentlemen. It couldn't be any better, you thought. No, yes, they're even more perfect.

Steve: Well, first of all we are at Version 2, and the algorithm has changed.

Leo: Oh. Now, why is that?

Steve: Well, because I realized that the 128 bits that I was using to add to the input of the Rijndael cipher was buying us almost no additional security. It was completely dumb.

Leo: Why is that?

Steve: Well, what happened was - okay. Just to refresh users' memories, the system has a 384-bit key. 384 is 128 plus 256. The 256 bits of the key are used to directly key the Rijndael cipher. So that's the key that goes into the cipher that scrambles things. The other 128 bits I was adding to a counter, and that was the data. The result of that addition was the data which was going into the Rijndael cipher. And the output of that cipher was then where we got the Perfect Paper Password passcodes.

So someone posted - I had made a posting saying, gee, I wonder whether it would have been better to use an XOR than an add because then the bits would be scrambled around rather than just being - creating an offset. Well, the person correctly pointed out that it really didn't matter, neither of those were better or worse than the other because, since it was a cipher, if you had a collection of passcodes that are 24 bits each, you could assemble them back into one of the 128-bit blocks of the Rijndael cipher and decrypt that back into what was put in. And if you took two of those in succession, then what would come out would be the counter's value or the XOR value that was just differed by one bit. In other words, it really provided no additional security. You could essentially trivially reverse engineer what the counter value was and what those 128 bits were. So essentially it would require you to do two operations rather than one. So adding 128 bits only doubled the strength, like having one more bit of key. But here we had 128 bits more. So I thought, this guy's absolutely right. That was a dumb thing to do. Let's get rid of it now.

Leo: This is why, by the way, I think in security peer review is so valuable.

Steve: Oh, it's - absolutely. It helps so much to have multiple people looking at the same thing. And in fact, you know, that's been the way the last couple weeks have been spent is in just amazingly interesting dialogues with people over on the GRC newsgroups, really pounding on this and hashing it around. And I just, yes, I could have left it alone. But it just, you know, when I realized that 128 bits being added to the input really wasn't buying us, I mean, it was buying us one bit worth of additional security, essentially, I just thought, okay, let's get rid of this right now before this goes any further.

So we created Version 2, which is substantially cleaner. Essentially the 128-bit counter now just feeds directly into Rijndael. The sequence key is just the Rijndael key. We take the output from the Rijndael cipher, which is 128 bits, group those in sets of 24 bits, each of which is a passcode. So, I mean, it's cleaner than it was. And someone might say, oh, well, but you've made it only half as strong. And it's like, yes, but 256 bits is way more than we ever needed. I mean, you could argue, many people feel that 128 bits on a symmetric cipher is already secure enough because the number of possible combinations literally goes up exponentially. You know, every time you add a bit, you double the strength. And when you've done that 128 times, you're already up there way high. So we're going further just because why not, it's convenient, to 256 bits, which is just phenomenally uncrackable. So there was just, you know...

Leo: I like that. Phenomenally uncrackable.

Steve: Phenomenally uncrackable. There just was no reason to leave it the way it was. Okay. So the other thing that has happened is, in this last two weeks since the introduction of Perfect Paper Passwords, is we've had a really gratifying reaction from people implementing their own open source solutions. And they're all listed. And I'll be maintaining a list of this as this continues to grow over on the other PPP software page.

Leo: There's a link right at the bottom that says additional implement- or additional software or something like that, yeah.

Steve: Right. There's a set of pages that shows the algorithm and implementation, usage notes and everything. Every one of those pages has a little link block at the bottom. And so it's page six at the moment, other PPP software. But we've got, for example, open source implementations in C. I think we mentioned before that there was going to be a Mac OS X PAM, a Pluggable Authentication Module, and that's done. And also for Linux. People are using it to log into Macs and Linux machines over SSH, the Secure Shell login, and being prompted for passcodes.

Leo: Oh, that's neat.

Steve: So, I mean, it's done, and it's working, and a bunch of people have been using it. We've got an open source implementation of Windows, also several other for UNIX. There's one written for PHP. And there are some PHP test pages where you're able to get passcards and log in using PHP. All this is open source. There's a .NET wrapper around my DLL that I provided. So people who are using C# and .NET platform from Microsoft are able to use it. There's a complete implementation in Java that is in the language Java, so that if someone were Java-centric, then they could do the authentication, print the passcards, and everything. And there's one coming in Perl that's not yet available. So...

Leo: Speaking of Perl, I got an email from Randal Schwartz, who's a Perl guru and a security guy and is on a cruise right now, so he's not listening at the moment. But he asked is this anything like S/Key, which has been implemented in BSD for a while. In fact, actually it's been kind of deprecated in BSD for a while. Same idea. Are you familiar with it? Do they talk about this at all in the forums?

Steve: No, I haven't heard. But I've heard of S/Key.

Leo: It's a one-time-password system for authentication, similar, I guess.

Steve: Of course we did talk about OPIE, the one-time password - I can't even remember what the acronym stands for. But OPIE is a one-time-password system that's available over on

FreeBSD. And it uses a very different scheme where they've got a vocabulary of 2048, that is to say, 11 bits' worth, 2048 short real English words. And then they choose six of those, and that's your one-time key.

Leo: Oh, interesting. All right.

Steve: And so it's six different actual English-language words. And so, I mean, it's a nice solution, although it ends up of course being much more bulky than the PPP system. One of the things that people have really liked and reacted positively to is that we're just using these cute little four-character passcodes...

Leo: Easy to remember, but carry it in your wallet, I love that idea.

Steve: Exactly. And a little credit card-size printout will hold 70 of them. And in fact I modified the printout page since our first podcast to bring them closer together. So you could either cut them out as individual cards, or it's now easy to cut them out like a three-high card, which you then fold inwards to create just sort of a little booklet of three cards. And by folding them inwards you conceal what they are. As long as it's folded together, no one can see what the codes are. Because there were some people who said, wait a minute, you know, what if someone took a picture of your...

Leo: Right.

Steve: Well, and it's like, that's absolutely true, I mean, it's certainly the case that there are tradeoffs associated with a printed passcard that you don't have with a one-time token, which is only showing you the currently valid code every time you press the button. It's definitely the case that you have those sorts of tradeoffs. But this is zero hardware, no batteries to run down. And it's just - it's a nice solution. So we have a large array now of open source implementations. And just Googling Perfect Paper Passwords around the 'Net I've seen a lot of adoption happening. There's SSL-Explorer that's in the process of getting it. And it's all over the place.

Leo: Wow, that's really neat, Steve. You must be very gratified.

Steve: It's, well, it's cool because it's a solution that of course I came up for us. And I want to, I do plan to be offering it in some future remote authentication applications. Now, one really interesting point was raised, which became known as the "Peabody Dilemma."

Leo: I love it.

Steve: Because someone whose handle is "Peabody," he said, you know, I really like the idea and the whole system and this notion of random passwords. But if they're random, then they're really not one time because two of them could be next to each other. That is, you know, you could have - there's nothing to prevent you from, if every single one is random, and the chances of any particular one being one in 2 to the 24th power, which is to say one in 16,777,216, then there's that chance, one in 16-plus million, that you'll have two passwords the same, right next to each other.

Leo: Okay. So...

Steve: So how is that one time? And it's like, oh. Well, that's true. Our use is one-time use, the idea being that it's changing every time. But he said, okay, but it's not really changing every time. It's changing most of the time.

Leo: Like a lot of most of the time, but okay.

Steve: Oh, yeah, like, exactly. You're like, you know, there's a one in 16 million chance. But then as you increase the window, as you say not just the one just before, but maybe the one two before, or three before...

Leo: Yeah, that's true, that goes up, sure.

Steve: ...then the probability - because, well, and you sort of get into a little bit of sort of like the birthday paradox. Okay, so that sent us off on a really interesting tangent for quite a while because he had a point. And here was the point. It really doesn't matter if an attacker has no knowledge, that is, someone trying to attack the system has no knowledge of prior passcodes because every one of them is going to be random, and there's only a one in 16.77-plus million chance of an attacker ever guessing correctly. But Peabody made the point that, okay, but keystroke logging is one of the things we're trying to prevent against. That is, the reason we do this is that we know that one possible bad thing that could happen would be keystroke loggers, who don't know about the PPP system, but they see somebody log in with their username and their secret password and their PPP code.

Leo: And now they have it. And there's one in 2.6 million chances it'll work the next time.

Steve: Exactly. And so his point was that wouldn't a true one-time-password system be more secure in the presence of keystroke logging than a random password system.

Leo: Yes.

Steve: And he's right. He's absolutely right. And so then we said, okay, well, wait a minute now. What does that mean? Well, if we had a true one-time-password system where no password could ever occur a second time, then a couple things happen. First of all, in the face of somebody with perfect knowledge, as we've talked about before, somebody who, for example, was recording everything you did, they would not know what the next password was. But in knowing what all of the ones that had come before were, if they knew that they were never going to happen again...

Leo: Ah, they'd reduce the set.

Steve: Yes. Exactly. It allows them to know never to guess anything that he'd already seen because the system was designed never to allow that to happen again. So you're right. It reduces the set. And so, I mean, but not a lot. But again, it is shortening up the number of possible passwords that they could choose between. Now, again, we've got 16,777,216. So even if you recorded all of the last eight million of them, well, you'd still have another eight million to choose from. So, which, again, is a lot of strength. And I'll remind everybody that that's still eight times more strength than any of the hardware dongles have because they're just one million possible combinations because they're just six decimal digits, 000000 to 999999. But the point was an interesting one.

So the reason we have duplicates, and this is sort of an interesting aspect of this, is if we had passcodes that used all 128 bits of our cipher, remember that we have a counter which is incrementing, and it's never decrementing. It's only going upwards. So every value in that counter, we have 128-bit counter which goes into the Rijndael cipher, and out comes another very different 128 bits. And what we know is that every single counter value will map into a different output. So those never repeat. That is, the 128 bits never, never, never repeats. But the reason we get repetition is we're only using a chunk of those at a time. We're only using 24 bits out of the 128. And then we're using the next 24 bits and the next 24 bits, sort of going along those 128. And so the idea is we can get repetition because we're using 24 bits, and it might be that another output from the cipher would arrive where we would have the same little chunk of 24 bits that we'd seen somewhere else, even though all the other ones out of the entire 128-bit cipher were unique. So that's guaranteed.

So what this means is, if we wanted to have a true one-time-password system, then we would need a 24-bit block cipher. That is, we would need a 24-bit counter feeding into a 24-bit cipher, which mapped that then to - essentially permuted those 2 to the 24 into an entirely different set of 2 to the 24 combinations. That would give us a one-time-password system. But the problem with that is that 2 to the 24 isn't a huge number from the standpoint of, like, recording them all. That is, 2 to the 24 is 16 million, as we know, 16 million passcodes. Well, we can store that in RAM. I mean, we can - 16 million would be, what, 64 megabytes. Or, no, even - it would be not even 64 megabytes, 48 megabytes because it's three bytes per. So it's possible to make a table which we start filling in of these things.

The reason that, I mean, the specific reason that ciphers have larger block length nowadays, that is, 64 is considered very strong. 32 is no longer strong enough because there just aren't that - it's possible to build a table of the inputs and the outputs from a 32-bit block length cipher and begin to attack it that way, a so-called "electronic codebook attack" where you actually record the outputs, because we've got enough RAM to do that. We don't have enough RAM, when you talk about 64 bits, or certainly not 128 bits. There are just too many of them. So we really do want a large block length cipher. And but that means, if we're not going to have passcodes which use all of those 128 bits, we're going to have repetition.

And so we went back and forth, had some really great discussions about, okay, what does this mean? Well, and I proposed a couple schemes for, like, going back in time, for example. And if you don't really have to verify that you've never seen a passcode before, but because the presence of the possibility of keystroke logging, the idea that an attacker might use passcodes you had recently used, wouldn't it be worthwhile then to make sure you don't have repetitions within, sort of like within the near term, like within the last 250 passcodes, for example, that you haven't already used those.

So I said, well, okay. What if we went back, and we looked at the prior 250 passcodes. And if the next passcode was going to be a duplicate, we add one to each of its bytes, turning it into something different, just to sort of, like, brute force a nonlocal repetition of passcodes. Well, the problem with that, it was pointed out, is that if you're going to do that, then in order to know what those 250 were before, you've got to look at the 250 before them, and the 250 before them, and the 250 before them. In other words, you would always have to go from the very beginning and make sure that you had no repetitions. So it ends up sort of being recursive and not feasible to do this.

So anyway, we had some really interesting discussions of this notion of the difference between true one-time-passwords and basically random passwords, which is clearly what the PPP system is, is random passwords. But one thing that came out of this is the notion, then, that not all keys, not all sequence keys are created equal. That is, you could spend some time looking at random sequence keys until you found a better one. You might look at a sequence key and sort of basically run the codes on it for the first 100,000 codes, and if you happened to get one with lots of close duplicate passcodes, just say, eh, we don't like that one, and pull another random one and check it.

Well, it turns out that that's what I have ended up liking to do. This is not going to be part of the Perfect Password system. But, for example - Perfect Paper Passwords, rather. But, for example, when I do this for our keys for my GRC employees, and when I implement this in my own use of the system in our commercial authentication product that is coming downstream, there's no reason, it seems to me, not to spend a little time upfront, in terms of computing time, and we're only talking four or five seconds, probably, just to not grab the first sequence key that comes out of our random sequence key generator, but grab, I don't know, look at a hundred or a thousand of them and scan them to see if there's one that's better than another. Because the point is, yes, in the case of an attacker who has no knowledge of prior passcodes, then random is absolutely good. And random is what we've got. But because the world has keystroke loggers, that biases the attacks a little bit toward re-use. And re-use is something we'd rather not have. So there's no reason not to choose sequence keys which, for the first likely span of passcodes, 100,000 passcodes, don't have any repeats. And I've done the research, and it turns out it is very possible to find passcodes - actually I looked at the first 200,000. Turns out that there's lots of sequence keys that have zero repeats in the first 200,000 passcodes. And so we get the best of both worlds.

Leo: That was a good idea, yeah. Simple thing to do, yeah.

Steve: Yeah. It's very simple.

Leo: And if not, you generate a new set until you find one that does.

Steve: Exactly. It turns out you don't have to look very long. I did it by hand, and I found a bunch just by hand. Well, I didn't really check 200,000 passcodes. But I've altered my code to check it. And what I like about that is that sequence keys are issued rarely. That is, they're issued infrequently, and they're used for a long time. So it makes sense, I mean, it's economically feasible to spend some compute time upfront to develop this notion of better and worse sequence keys and choose one, for example, where in the first 200,000 passcodes there's zero duplicates. Turns out there's lots of those. And so why not use one? And then you've solved the problem of keystroke logging. You get the equivalent of a one-time-password system within a reasonable horizon of passcodes, I mean, 200,000 passcodes is going to last a long time, no matter how much, you know, how often you use them. And...

Leo: How did - go ahead.

Steve: And we still end up with this notion of their actually being chosen at random, even though, again, if an attacker had perfect knowledge of the fact that we had chosen a good key, they would know not to guess any that we had already chosen. But again, that presumes somebody has access to every passcode you've ever used. And even if they did, the possible universe of still-unused passcodes is still bigger than 15 million. So you haven't really lost any power.

Leo: And how did Peabody feel about all this?

Steve: Well, he likes it. It was funny, too, because he was making the point over and over and over, and everyone was telling him, no, no, no, they're random, they're random, don't worry about it, you know, they're random. But what I liked about his point was, he said, yes, but we know that keystroke loggers exist. And so what we're trying to prevent against is re-use. And so why not prevent them from being reused? And it's like, that's better than random. He's right, that's better than random.

Leo: And as it turned out, I mean, the simple solution that you came up with gets the job done, and there you go.

Steve: Well, I like it, again, from an economic standpoint, to me it makes sense to spend some time - and we're only talking a few seconds because all this is very fast - spend some time to find the best sequence key you can where "best" means no repetitions within a certain length of time. Or not near repetition, not two or three passcodes apart, but maybe at least 50 or some. And what I should say is that I found many where there were no repetitions within 250 passcodes in a horizon of 200,000 passcodes. That is, looking that far down in the future, you never saw any repetitions within 250. So it's like, okay, that's, you know, we're really, really safe.

Leo: Yeah, good enough.

Steve: Now, one thing that I mentioned erroneously generated a lot of mail two weeks ago. I talked about the idea of skipping passcodes, that is, skipping forward. And I don't even really remember what I said, but I must have said something like somehow the system would automatically skip forward. Because many people commented that that could allow a denial of service attack on the Perfect Paper Password system because an attacker could force you, like could force the next passcode, like, far enough ahead that the user would run out of passcodes and would no longer have them. So I misspoke when I said that. That was never really my intention. I certainly would only - my original notion was that you'd have to have the username and the secret password and the passcode all correct, obviously, in order to authenticate, and that maybe if only the passcode was incorrect, then under some circumstances you would move forward. Well, again, we hashed this around in the newsgroups a lot and came up with what I think is a very nice solution. One of the problems of four-letter words is, I mean, four-letter words is, in the English language, at least, is recognized slang for words that are not popular or should not be used in polite conversation. You know, like four-letter words.

Leo: Ah hah, yes.

Steve: Know a bunch of them. Well, so imagine a system, Leo, where you are being asked to enter a four-letter word that makes you uncomfortable.

Leo: That's right.

Steve: I mean, it could happen.

Leo: It could easily happen.

Steve: It could easily happen. Well, yes, I mean, in fact, every one of the possible four-letter words that exists could occur because we're just choosing four-letter things at random. Now, one person, I mean, this had been discussed before, before we even talked about it. Like someone said, well, what about eliminating vowels? Then that would make it hard. It was like, well, okay, but in general eliminating vowels meant that we had to then use more strange characters, which might be difficult to find or you might not have on a cell phone, for example. So I decided that there was - the system, to be user friendly, ought to not force someone to enter the next four-letter word if they just didn't feel comfortable for whatever reason typing it into a computer screen. So the idea...

Leo: You can't prevent them from seeing it.

Steve: You can't prevent them from seeing it. I don't think it makes sense to, like, have a dictionary of all possible offensive four-letter words and search even longer for sequence keys where none of those occur. That seems like overkill. So the idea would be, if you're presented with a word that you don't want to enter, or, for example, someone's got an inkjet printer that's on the fritz and they can't read one of the characters, I mean, that was the idea of allowing people to move forward, was what if for whatever reason you could not read a passcode, or your passcard of 70 passcodes went through the wash by mistake, and it's no longer legible. The idea was you want to be able to obsolete and move forward.

So the solution turned out to be simple. That is, normally you're being prompted for the next passcode. If for whatever reason you either don't want to enter it or you can't enter it, or for example, say that you think maybe that your whole set of three cards, your passcards may have been compromised, you have some reason to believe someone may have seen them, you want to be able to immediately obsolete them all.

Leo: Right, right, right.

Steve: So the idea is, normally the server says, here's the card and row and column of the next passcode, please enter it. But you always have the option of saying, no, I want to tell you which one I'm going to enter.

Leo: Ah, perfect.

Steve: That's all there is - yeah, well, see, there's my favorite word, Leo, you've just used it.

Leo: Even more perfect.

Steve: So all you have to do is say no. You always have the option of saying, no, I want to tell you which one I want to enter. Then it gives you a second form with the passcard number, the row and column as blanks that you can fill in, rather than them being prefilled for you. And you simply - the only limitation is you have to move forward, obviously. You would never want to allow the user to move backwards because that would no longer then be perfect. So they're able to skip over the next code, the next five codes, the next line, the next card. As long as they move forward from where they were being prompted before, no problem, that advances their pointer, they enter the code that corresponds to that at the same time, and that authenticates them.

Leo: Very nice. Very, very nice.

Steve: So it ended up being a really nice solution. There were a couple people who said, you know, for cell phone entry, these funny characters that we've used...

Leo: Oh, could be tricky, yeah.

Steve: Those could be tricky. Needless to say, on an iPod or an iTouch or iPhone getting some of those can be a little tricky, too. So someone said, you know, you're using 24 bits. So you're using six bits for four characters. I'm going to use four bits for six characters. In other words...

Leo: Oh, okay. So you get the same strength...

Steve: You get the same strength...

Leo: ...just eliminate some of the characters that you can't type.

Steve: Well, of course four bits is hex. So zero through nine, A, B, C, D, E, and F. And it's funny, too, because he said, you know, even though it's counterintuitive, there are people who will think that a six-character token, that is, a six-character passcode, is more secure than a four-character passcode, even though the four characters has the same bit strength because there are 64 characters per position rather than 16. And he's like, he's absolutely correct that it's the same strength. It's like, okay, well, certainly an implementation like that works the same, it's just as strong. And I could see, for example, in cell phone authentication it might make sense, or in applications where you don't have access to the full alphabet that could make sense.

Leo: Yeah, six characters isn't a big deal anyway. You do eight characters for a phone number. So I think that's great. That's a very clever way to do it. What I think people love about this is the - and this is what's great about programming is the problem and then the solution, the challenge of figuring out the solution. And it's really kind of an intellectual exercise. It's just enjoyable.

Steve: Yeah. And we've had a lot of fun over the last two weeks, and I wanted to share with our listeners the Peabody Dilemma. And it was really funny, too, because, I mean, for a long time he was taking a lot of heat because everyone was saying, look, you can't do better than random. And it's like, but that's true if the attacker had no knowledge of the past. But if the attacker has knowledge of the past, then you can do better than random by explicitly eliminating the codes that have come before. You know, why not? And so we looked at algorithms to do that and just ended up settling on this notion of, okay, not all sequence keys are created equal. There are better ones. And since you only need to create it once, and then you get to use it probably for months or years, why not take a little time to find a good one? And so that's how we ended up with Even More Perfect Paper Passwords. And 256-bit sequence keys now, down from 384, eliminating those 128 bits that really weren't buying us any additional strength.

Leo: Are the third-party implementations, do they incorporate your modifications? Or are these...

Steve: Yup, they're all up to Version 2 now.

Leo: Oh, that's cool. For 16KB versions of this podcast, for transcripts, for details, for show notes, for implementations of the Even More Perfect Passwords system, you go to GRC.com, that's the place. GRC.com/securitynow for the show notes; GRC.com for all of Steve's free programs that are so great to help you with your security, everything from ShieldsUP! to Shoot The Messenger and DCOMbobulator. And his Perfect Password implementation, too, I almost forgot. Also, of course, don't forget SpinRite. Can't forget that, everybody's favorite disk recovery and maintenance utility. That's from GRC.com. Steve, we've wrapped up another episode of Perfect Passwords, Even More.

Steve: Even More Perfect Passwords. And we will do one of our fun Q&A episodes next week and then move forward into other topics.

Leo: All right. I really appreciate it, Steve. Have a great weekend, and we'll talk to you next week.

Steve: Thanks, Leo.