Public Key Cryptography

Description: Having discussed symmetric (private) key ciphers during the last two weeks, this week Leo and Steve examine asymmetric key cryptography, commonly known as "Public Key Cryptography." They begin by examining the first public key cryptosystem, known as the Diffie-Hellman Key Exchange, invented in 1976. Then they describe the operation of general purpose public key cryptosystems such as the one invented by RSA.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-034.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-034-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 34 for April 6, 2006: Public Key Cryptography. Bandwidth for the TWiT Network is provided by AOL Radio at AOL.com/podcasting. Financial support is provided by Astaro, makers of the Astaro Security Gateway, on the web at www.astaro.com, Astaro.

Well, now we get to the meat of the matter. Hello, Steve Gibson.

Steve Gibson: Hey, Leo. Great to be back.

Leo: We've been covering cryptography. This is, what, our fourth podcast on the subject?

Steve: I think it's our fourth. We talked about first sort of the overall social implications, what does it mean for citizens to be able to encrypt things that even their own governments would not be able to decrypt. Then we've had two episodes so far talking about two sort of basic early forms:; first. symmetric stream cryptography and symmetric stream ciphers; and then last week was block ciphers, talking about how those things work, symmetric block ciphers. And today we talk about public key cryptography.

Leo: And really this is kind of where I come in because while I've done, you know, secret decoder rings and everything, of course, I really didn't start using crypto – and I think this is probably true of most of us – until public key cryptography was widely available.

Steve: Well, actually it was a revelation which occurred in 1976 when two researchers, Whit Diffie and Martin Hellman, were working at Stanford University in the math department. And they first really conceived of this concept of a cipher where one key would be used for encrypting and a different key would be used for decrypting. And they understood what that

meant. What was really interesting was when they introduced the technology the NSA said, oh, we had that ten years ago. Literally, they said, oh, we've known about this since 1966, which was a decade before this was all made publicly available and public knowledge.

Leo: The NSA is the super secret spy agency, and of course it would make sense that, if anybody would know it ahead of time, they would.

Steve: Yup. But before we get into...

Leo: And would not tell anybody. That was the other side of it.

Steve: Well, exactly. So, you know, and so it has raised some concern. They were never able to prove that they had it before because of course there was no documentation for that. They didn't talk about whatever else they may have had. But it's like, okay, fine, well, we'll presume.

Leo: I don't think they would be boasting. I mean, but it does raise the issue what else they might have in the lab there, yeah.

Steve: Yes.

Leo: They may have cracked it, for all we know, but they wouldn't tell us.

Steve: Well, what these guys came up with is actually different than what I want to talk about. I want to talk about what are called "asymmetric ciphers," which is sort of the fancy term for public key cryptography. But first I want to talk about what it was that Diffie and Hellman did because, in the same way that we like to talk about sort of conceptual things that people can visualize through an audio podcast, what's called Diffie-Hellman Key Exchange is the concept they came up with. And it's perfect for us to talk about because it follows on what we've been talking about with the one-time pads and locking the box and sending it back and forth. You know, you've mentioned many times – and this is very appropriate – you've mentioned the problem of being able to communicate securely, but needing to exchange a key between two parties. And, you know, that being the real problem with cryptography is how do you initially communicate that key.

Leo: Yes, because we have unbreakable cryptography with one-time pads, but they have this little flaw that you have to pass it along.

Steve: Well, and of course we also have virtually unbreakable cryptography with a symmetric block cipher, where we use a long key, like maybe 256 bits. And we know that, as far as we know, there's just no way to reverse engineer that. So that if you use symmetric cryptography to encrypt your message and send it to somebody else, if they have the same key they can easily reverse that process. The problem is that key is gold, and you want to be able to somehow get it to them without anybody being able to intercept it because, if it's intercepted, then you've got no protection at all.

Leo: Right.

Steve: So in general these crypto systems involve sort of one-way functions. They involve doing something which is easy to do in the forward direction, but is difficult to do in the reverse direction. Of course we've glanced on the concept of factoring primes. And that's the way the most famous crypto system, the most famous public key system, RSA, that's sort of its trick is you take two really big primes, and it's easy to multiply them together to get a huge result, but it's virtually impractical, I mean, it's impractical in any given length of time to determine which two primes made up that result. That is, it turns out there's no – we've never found a fast way to do a prime factorization. And so that's the so-called "one-way function" that RSA uses.

But that actually came later than the so-called "Diffie-Hellman Key Exchange." They used a different one-way function, which is interesting. They used simple exponentiation, that is, when you raise one value to the power of a second value, so, like, A to the power of B. That, it turns out, is very simple to do, that is, to do an exact exponentiation is simple because you just multiple A by itself B number of times. That's raising A to the power of B. It turns out that getting the reverse process, that is, an exact logarithm, is incredibly difficult. And it turns out that, if you define that operation properly, there's something called in number theory a "finite field," which is sort of a fancy term for a field of a given size. If you pick your numbers correctly, and you do an exponentiation with a modulus – you know how we talk about our Mod 4 episodes is every four – the idea being that you raise one number to a power, and you only keep the lower digits, basically, of the result. Well, that prevents you from being able to divide backwards and do an exact logarithm of the result.

Okay. So here's how this all works. Say that two people want to communicate. They agree upon the base number that they're going to raise to a certain power. And that doesn't have to be private. I mean, they could publish it on their website. They can communicate that easily. They say, here's a big number, this is going to be what we're both going to use as our base. Okay, now, one other cool thing about exponentiation is, say that we take 2 to the power of 3, and then we take the result to the power of 4. You get the same answer if you take 2 to the power of 4 and then take that result to the power of 3.

Leo: It's commutative.

Steve: Exactly. It's commutative. And in fact it's the same as 2 to the power of 12. That is, if we said A to the power of B, all of that to the power of C, that's the same as A to the power of B times C.

Leo: Right. I remember that from eighth grade.

Steve: Exactly.

Leo: That much I can follow.

Steve: Well, get a load of this. This is the key. It's such a simple and beautiful idea. So we both agree upon A, and we share it publicly. It doesn't have to be private. We each have this number that we've chosen to be A. And there are some, actually, when you get down to the details, there are some requirements on primeness and what the number has to be to be secure. But then I choose a big random number for my B. You choose, separately, your own big random number, which we'll call your B. And we both raise this agreed-upon number to that big power. Now, we send that intermediate result to each other. And now I take your intermediate result and raise that to my big power. You take my intermediate result and raise that to your big power. We both end up with the same answer.

Leo: Because those raises are commutative.

Steve: Exactly. But nobody listening, nobody eavesdropping who sees that can do anything about it. They see both of our intermediate results. But because they cannot do what's called a "discrete logarithm in a finite field," technically, because they're not able to reverse that process that we did simply, that simple exponentiation, they can't get a logarithm that is discrete in any reasonable amount of time. They can't figure out what our powers were that we made up. So we end up being able to choose random numbers, use those to raise a common value to that power, exchange those results, do it again, and we both end up with exactly the same value, which we can then use as a key.

Leo: So we're not exchanging the message, we're merely determining our keys at this point.

Steve: Exactly. And in fact that's why it's called the "Diffie-Hellman Key Exchange," is it's a way for us to basically create what's called a "session key." We don't even have to keep that key around for long. We could do it easily every time we want to have a secure communication. We'd just say, okay, let's generate an agreed-upon key. So this is how we agree upon a key, which we will then use for securely encrypting our actual conversation. And nobody listening in who sees any of that transaction is able to figure out what our final result is.

Leo: Professor Gibson, can I ask a dumb question? This sounds a lot like this double padlock thing that we did a while ago. What makes this different from the stumper that we talked about?

Steve: Well, the real reason is that the XOR was not hard to undo.

Leo: Ah.

Steve: Because it's symmetrical, it was trivial for someone to re-XOR the message versus, you know, the unencrypted versus the encrypted to get back the pad. So there was an example of a process that was as easy to undo or reverse as it was to go forward.

Leo: So this is very similar, though. I mean, it's a commutative process. In other words, it can be done in different directions, and it still works, like the XOR. But unlike the XOR, it's not easily reversed. And that's the key, if you will, to the whole thing.

Steve: Yes. It involves what we're going to be seeing a lot of in the future, in the next couple of weeks, as we're talking about more of this crypto, a so-called "one-way function," something you can easily do in a forward direction that there just is no practical way that anyone has found – and a lot of people have pounded on it for years – there's no practical way to undo it, to reverse the process.

Leo: Now, did Whit Diffie discover these non-reversible things, or are these well known by mathematicians for some time?

Steve: Those kind of functions, yes, those kind of functions were well known. They just realized, hey, we can do something really cool and useful with these properties. And, you know, they are heavy-duty math guys. And so they knew that doing an exponentiation in a so-called "finite field" is simple to do, but they were smart enough to recognize that taking the discrete logarithm in a finite field, you just can't do.

Leo: This is where the beauty and elegance of math really shines. Unfortunately, you have to like math to even get far enough to appreciate the beauty and elegance of it. Because I'd love to turn on my kids to math with stuff like this. I mean, this is very elegant, very beautiful.

Steve: Yeah. I just wanted to sort of describe that first because it's an easily visualizable – I can't even say the word.

Leo: Easy to visualize, hard to say, apparently.

Steve: Exactly. Example of how we could have a public communication that somebody could listen in on and still not get the key. So, and it's just a very cool thing.

So now, with that said, let's talk about asymmetric ciphers. We've talked about symmetric ciphers so far, where the symmetry is that the same key is used to encrypt as decrypt. And we've looked carefully at how those work. Now, I'm not going to go into the math because it does get hairy. And in fact, we don't even really need the math in order to understand all the cool things we can do. But an asymmetric cipher is – basically it's a cryptographic process where you have separate keys for encrypting and decrypting. And, importantly, there is no way to figure out one key from the other. And it's sort of a misnomer to call this "public key cryptography." I mean, it's what we do call it, but that sort of implies that one of the keys is public and the other is private. Well, there's nothing to differentiate these two keys from each other except that they're different from each other. In other words, if you had an agreed-upon cipher, for example RSA's famous public key cipher, you could encrypt something with one key, and it can only be decrypted with the other. But you could also encrypt it with that other key, and then that message can only be decrypted with the other key from the one you encrypted. But it's not like one of the keys has to be used for encryption and the other has to be used for decryption. You just have to use the other key, no matter which one you used.

Leo: So that's why it's symmetric.

Steve: Well, actually that's why it's asymmetric.

Leo: I'm sorry, asymmetric, right.

Steve: Exactly. Because you don't use the same key, as we have been using when we looked at the previous symmetric ciphers before.

Leo: Right.

Steve: Okay, now it turns out that this is a huge win because it means a few things. For example, say that I wanted to send you a message, and again over a channel which might be

eavesdropped on. Well, you're able to publish publicly one of your keys. So you go through a process to generate what's called a "key pair." This is the first time we've ever used the phrase "key pair" because with asymmetric ciphers keys are generated in pairs. One can be used to encrypt, the other to decrypt, and vice versa. Doesn't matter which one you use. But a pair of keys are generated at once, and they're sort of like the mirror images of each other.

Leo: So I have an A key and a B key, and I can use the A key to encrypt and the B key to decrypt. If I reverse it, and I use the B key to encrypt, will the A key decrypt?

Steve: Exactly, yes.

Leo: How interesting. Okay. That's why it doesn't matter which is public or which is private.

Steve: Exactly. So you generate a pair of keys, and you publish one on your website. And you say, this is my public key. What it really says is this is the one of the two keys which I have chosen to make public. I'm keeping the other one private. So what this allows is, it allows me to get a copy of your public key, that is, the one of those two keys that you have made public. I can encrypt a message using your public key and send you the result. Now, anybody else can, too. So we begin to get a little tricky with things like authentication, which we'll be talking about. We'll be talking about how those mechanisms work in upcoming weeks. But the concept is I can encrypt something using your public key, and nobody other than you, that is, nobody other than the person who has the matching private key can decrypt it. So I get your public key, which is right there out in the open for anyone to use, encrypt something using it, and send it to you. I know that it can be intercepted, it could be interfered with in any way. If it were changed, it would not decrypt properly. So when you receive it and apply your key to decrypt it, the only way it will properly decrypt is if, first, it was encrypted with your public key, that is, the other key; and second, if it has not been modified because that would cause it not to be decrypted correctly.

Leo: Got it.

Steve: So it's interesting, that's sort of the fundamental concept here that can be used in many kinds of ways. For example, now, if you received this message, you'd have to take it on faith that I sent it to you because anybody could receive your published public key. Anybody could send you a message. But look how we can add one step to this in order to begin to provide some authentication. I have my own pair of keys. I have my own private key and also a published public key. So now I want to send you a message which I know that only you will be able to read. And I also want to be able to prove that I'm the one who sent it to you.

Leo: Yeah, because I do that. I sign all my emails, you've probably noticed.

Steve: Well, and signing, we talk about signing, and that's what we're coming to, essentially, is I take my message, and I encrypt it with my private key. And here again, this is why it's so cool that it doesn't matter which key you use, as long as you do the reverse process with the other key.

Leo: Ah.

Steve: So I encrypt my own message to you with my private key that nobody else has. Then I encrypt that result with your public key.

Leo: Or decrypt it. No, encrypt it.

Steve: No, no, I encrypt, yeah, exactly.

Leo: A second time. Okay.

Steve: I encrypt it a second time. And now I send it to you. You receive it, and you have to know what it was that was done to it in order to reverse the process. So the first thing you do is you decrypt it with your public key.

Leo: Right.

Steve: Because the last thing I did was to encrypt it with your – I'm sorry. You decrypt it with your private key that only you know because the last thing I did to it was to encrypt it with your public key.

Leo: Encrypt it with a public key, right.

Steve: Yes. So you receive it. You decrypt it with your private key. Then you decrypt it with my public key.

Leo: Which you have because it's public.

Steve: Exactly. Which you're able to get from me. So the only way the message is going to completely decrypt properly is if the first thing that was done to it was it was encrypted with my private key, because your decrypting it with my public key will only work if that's the case.

Leo: Right.

Steve: And so what's so cool about this is, by double encrypting with various combinations of public and private keys, we're able to basically create an authenticatable transfer.

Leo: Because only I could have sent it because it's encrypted with my private key, and only I have access to that. And only you can read it because I encrypted it with your public key, and only you can decrypt that because only you have your private key.

Steve: Exactly.

Leo: So you've verified that I sent it, and only you can read it.

Steve: Exactly.

Leo: Wow.

Steve: Now, there are still some problems with this. For example, the biggest problem is that public key cryptography, due to its nature, it requires much longer keys than symmetric. If the keys are too short, that basically means that the prime numbers that they are composed of are too low in value. So it's feasible for them to be cracked. Consequently, the way to think of it is that the public key system requires much longer keys to get the same level of security that symmetric algorithms give us with a much shorter key. So, for example, 128 bits is considered very good security for a symmetric algorithm; but 1,024 bits, you know, eight times more, is considered sort of the minimum now for strong public key. And you probably even want longer public keys in the future.

Leo: And the good news is bandwidth is so cheap, disk space is so cheap, and processor power is so great that you can have a 4,096-bit key.

Steve: Ah, but one thing that happens is, not only do the keys have to be longer, but this asymmetric technology, the asymmetric ciphers are about 1,000 times slower.

Leo: So decryption takes a while.

Steve: Well, that's the problem. In fact, it takes so long that the asymmetric ciphers, the so-called "public key ciphers," are not used to encrypt messages. They're used to encrypt keys.

Leo: Ah. Interesting.

Steve: And so what I'm actually doing when I'm encrypting a message to you is I use a cryptographically strong random number to give me a key, like 128-bit or maybe 256-bit, that is, a relatively short key, which I use as my symmetric cipher because symmetric ciphers that we've talked about in the last two weeks are extremely fast.

Leo: Right.

Steve: So I choose a random number to get that key, that is, to generate the key. Then I encrypt the bulk of my message very quickly with the key. Then I use the public key technology to only encrypt the key, not the message. So then what I'm sending you is sort of an envelope that contains the public key-encrypted symmetric key, and then the bulk of the message, which has been encrypted with a symmetric key.

Leo: That is so cool.

Steve: Yeah, it is so cool. When you receive it, you then use the public key technology only to decrypt the symmetric key, which restores the 128-bit or 256, whatever it is, bit random number, which I just generated out of the air just for this one particular transmission. That allows you to recover the symmetric key, which you then use with a symmetric key algorithm to decrypt the bulk of the message.

Leo: So basically it's I'm sending you a one-time pad. That symmetric key is like a one-time pad.

Steve: Well, yes, exactly. You're sending me a key that is very long and very random that nobody can figure out.

Leo: Right. And I'm protecting it using public key cryptography because otherwise it would be dangerous to send it to you.

Steve: Well, you would need some sort of a secure channel to send the key.

Leo: Right.

Steve: And the beauty of this public key technology, because separate keys are used to encrypt and decrypt, that's really all public key cryptography is. Separate keys are used to encrypt and decrypt. As a consequence of that, you're able to keep one of them private. You're able to let anyone who wants to know the other one. They can either use that to encrypt something to you or to decrypt something that you sent. And the only way that works is if it matches, the two keys match, whether one is private or public or vice versa.

Leo: And this whole system allows things like key servers, where people can go and say, look up Steve Gibson or look up Leo Laporte and get your public key – it's published, it's widely available – so that they can send you a private message.

Steve: Exactly. And in fact the use of this so-called, you know, the 128-bit or 256-bit key, that's called a "session key," which is normally generated on the fly just for that session. So it's a random number created just for the purpose of encrypting the message. It's used once, and then it's thrown away and never used again. So it's extremely strong because you're not stuck using the same key over a long period of time.

Leo: Very interesting. Very elegant.

Steve: Now, one last thing to talk about is that there is an attack on any public key system, which is another sort of a cool thing. If you know what someone's public key is, then you would be able to generate messages, your own messages, encrypt them with that person's public key to see what the encrypted result is. The point is that, if you tried all possible plain text messages, encrypted it with the person's public key, you could look for a collision between the encrypted result and what they actually sent.

Leo: And that would give you a clue as to what the key was.

Steve: Well, it would give you...

Leo: It would give you the key.

Steve: It would not give you their private key, but it would tell you what the plain text was. This is called...

Leo: Oh.

Steve: It's called a "chosen plain text attack." Because think about it, in symmetric encryption this is not a problem because the symmetric key is never known. There's no way for you to, like, try all possible combinations of input to get an output because you don't know what the key is. But in public key cryptography, you do know what was used to encrypt. So you can use it yourself to encrypt a whole bunch of different things to see if the result matches with the encrypted result. Isn't that neat?

Leo: Yeah, another kind of brute-force attack, really.

Steve: Well, it is. And in fact it's easily defeated. All you have to do, first of all, anyone using public key technology needs to obviously be aware of this. You just mix in some random stuff. You put some random stuff, like padding, at the end of your message. And then there's no way anyone could ever figure out what the random stuff was, I mean, because all they're doing is they're trying to guess at all possible input combinations to see if they're able to get the same output. But they, you know, they'd be using common words and phrases and starting with A and B and C, doing, as we've talked about before, a brute-force attack. But by padding it with random stuff, they're never going to get a collision. I mean, and it's so commonly done now because it's an obvious weakness of public key cryptography when anyone can use the public key to perform the same kind of encryption you did in order to see if they get the same result.

Leo: Yeah, that would be a bad thing. How much random data has to be included at the beginning to be secure?

Steve: Actually, the blocks that are encrypted with public key technology are so much bigger than the keys they're normally actually being used to encrypt that you just fill the result, I mean, typically...

Leo: You pad it.

Steve: You pad it. But normally it's like three quarters random and one quarter is the key. So it's, like, way impossible to get any kind of a collision.

Leo: Because you don't know where the real data begins and ends.

Steve: Right.

Leo: Well, do you want to talk about signing in this context, or should we wait to another episode?

Steve: We need a little more – we need to talk about cryptographic hashing in order to talk about signing.

Leo: Okay.

Steve: And that happens to be next week.

Leo: Good, good. And here's another question for you. Do you want to give out your public key? Do you have a public key?

Steve: I've never been a PGP user.

Leo: Well, I'll tell you what. I'll put my public key in the show notes.

Steve: Oh, cool.

Leo: A link to it, anyway. And if anybody wants it, then they can send me a private message, and I'll just read it to you over an encrypted VoIP line or something. How about that?

Steve: That sounds perfect.

Leo: You know, every time somebody sends me an encrypted message I go, ooh, there must be something really juicy in here. And there never is. It's fun, though. You feel like you're a spy. So I do have a public key because I don't use PGP anymore, I use the GNU Privacy Guard, which is essentially a PGP, open-source PGP.

Steve: Very cool.

Leo: Fascinating stuff. I had no idea really that there was that much going on. I had a kind of more simplistic understanding of it. So it was really great to hear how that actually works.

Steve: Well, we're going to add another piece to the puzzle, probably next week. And I think that'll pretty much – oh, no, there's also certificate signing chains. We'll probably...

Leo: Oh, we'll never run out here, right? Crypto goes on and on and on. You know, at some point we should talk about DRM in this context, too...

Steve: Yup.

Leo: ...the copy protection, because it uses crypto, too, of course.

Steve: Well, and in fact it's why I wanted to start out the year with laying this kind of foundation because these concepts are key to all the things we're going to be talking about coming down the line.

Leo: If you want to know more about this, and I know you do, and maybe you'd like to read the transcript so you can get all of that, plus a 16KB version for the bandwidth impaired at Steve's site, GRC.com/securitynow.htm. It's also the home of Steve's fantastic SpinRite program, the ultimate disk maintenance and recovery utility. Steve's even made a special URL just for you, SpinRite.info, for more information about SpinRite.

Steve: Yeah, in fact that's the URL that takes people basically to our customer feedback page for the testimonials because it's so effective for people to look at, you know, what other people think about the product and what their experiences have been.

Leo: Steve's so cute. He comes in with these letters every once in a while, look at this, look at this. You really – I think that that really almost in a way is your payment for this stuff. You just love reading how people are using your program.

Steve: Well, yeah. In fact, you know how I have a WAV file that plays whenever a copy of SpinRite sells on my server? What I love is that sometimes I'll hear three of them in a row. And the reason is we have a site license for SpinRite. And the deal is, you know, we tried to figure out how we could be fair to people. So what we decided was, if somebody wanted to be able to use SpinRite like on anyone's computer that they wanted to fix, not only just all of their own machines, but like, you know, in a small business or maybe somebody who's, like, a consultant, how could we do that fairly? So we decided, okay, if you own four copies of SpinRite, that is, you own licenses for four copies, then you have a site license, and you can use it, you know, don't leave it anywhere, but you can use it wherever you want to. So what I love is when I hear three being purchased. Because I'll hear, you know, the WAV file, the WAV file, the WAV file, three times in a row. Because what that means to me is somebody bought one, and they tried it out, and they saw what it did, and they came back and said, I want a site license for this, so they bought three more.

Leo: Three more. That's nice.

Steve: And it just makes me feel great because it means, you know, they proved it to themselves.

Leo: So let me ask you a question. You don't have to answer. What sound does it make? Does it go ka-ching?

Steve: No, actually I have a ka-ching sound as people are filling out the form and moving through my ecommerce system. When all that's done and everything clears, I have a WAV file of Fred Flintstone saying "Yabba dabba do."

Leo: That's really cute.

Steve: That just makes me smile every time I hear it.

Leo: I don't know why, but that's what I exactly was picturing. That's hysterical.

Hey, we're out of time. But before we go I really do want to thank our brand-new sponsor. You might have heard it at the beginning of the show. Yeah, we have a sponsor for the show. You know, people have been very generous all along, supporting the TWiT Network, and of course AOL providing bandwidth and everything. But we actually have somebody who's giving us cash money to do this show, which is pretty neat. And it's Astaro. They invite you to protect your business network against spam and hackers and viruses quickly and easily with Astaro Security Gateway Appliances. Open source-based, they're really a good company, making great products. In fact, home users are invited to download a free trial version of the Astaro Gateway software, ASG software at www.astaro.com. Thanks, guys, for your support of Security Now!.

And that's it for another edition of Security Now!. Steve Gibson will talk more about crypto next week. We didn't talk anything about the Internet Explorer problems and everything. But, you know, I think people have had enough of that.

Steve: And they're going to go – it'll all be fixed next Tuesday anyway.

Leo: Well, we hope. There's a new one, you know, there's a new phishing exploit that just came out today. So, you know, don't use Internet Explorer. Are you still using it?

Steve: Uh, yes. I am because I have it completely locked down. And once we get through talking about crypto, it's on the top of my list because many people have written, they've seen the list of things we're going to be talking about on the SecurityNow.htm page. And it's like, I want to find out how to lock down IE. It's like, okay, we'll be telling you soon.

Leo: Or just use Firefox, for crying out loud. I don't know what's so hard about that. Steve, thanks a lot. We'll talk again next week. Have a great day.

Steve: Thanks, Leo.

Leo: That's it for Security Now!.

Copyright (c) 2006 by Steve Gibson and Leo Laporte. SOME RIGHTS RESERVED

This work is licensed for the good of the Internet Community under the Creative Commons License v2.5. See the following Web page for details: http://creativecommons.org/licenses/by-nc-sa/2.5/