Cryptographic Hashes

Description: Having covered stream and block symmetric ciphers and asymmetric ciphers, this week Leo and Steve describe and discuss “cryptographic hashes,” the final component to comprise a complete fundamental cryptographic function suite. They discuss the roles of, and attacks against, many common and familiar cryptographic hashes including MD5 and SHA1.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-035.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-035-lq.mp3

Leo Laporte: This is Security Now!, with Steve Gibson, Episode 35 for April 13, 2006: Cryptographic Hashes. Bandwidth for the TWiT Network is provided by AOL Radio at AOL.com/podcasting. Security Now! is brought to you by Astaro, makes of the Astaro Security Gateway, on the web at www.astaro.com.

Time once again for our Crypto Primer with Steve Gibson. I have to say, Steve – welcome, first of all.

Steve Gibson: Hey, Leo.

Leo: I have to say I just have been really enjoying, and I know the listeners, too – the cryptography has been fun.

Steve: Been getting really good feedback, yeah.

Leo: Yeah. You know, and what’s neat is I’m getting – you know, I published my public key on the show notes on TWiT.tv. And I’ve been getting a lot of people sending me encrypted email. They’re, you know, “My first encrypted email, it’s so cool.” It’s just really fun.

Steve: Well, and in fact I’ve been preparing the questions for next week’s Q&A, our Mod 4 Q&A. And there have been some great questions and feedback that’ll be appearing in next week’s podcast, with people writing in with some questions they’ve had.

Leo: Good.

Steve: So we're on a roll.

Leo: Every fourth podcast we answer your listener questions, so get them in so we can talk about them next week. Meanwhile, this week we continue with something called "hashing."

Steve: Yeah. This is another one of sort of our basic components of a comprehensive cryptographic suite. In the prior weeks, of course, to review very quickly, we've discussed the two types of symmetric cryptography, stream ciphers and then block ciphers, where you use relatively short keys, that is to say, 128 bits is typical, sometimes 256. And you use a single key to very quickly encrypt or decrypt some content, where the same key is used to go both directions.

Then in last week's episode we talked about asymmetric cryptography, also known as or popularly known as public key cryptography, the idea there being that you have different keys, one which you can let the world know about if you want to, and one which you keep private. The cool thing there is that many very powerful concepts become possible where you're able to say, okay, anyone who wants to send me something, use my public key. Only I, who have my private key and keep it private, will be able to decrypt it. And the reverse is true. I could encrypt something that I wanted to send someone, and they would know it was from me if they used my public key to decrypt it.

Leo: That's what we've been doing. In fact, that's what people have been doing. They've been sending me encrypted email because they have my public key, and I decrypt it and read it. And then, you know, what's fun is, because these key exchange servers, PGP and many other institutions and universities run these servers, I can go out, search for their key, and encrypt it back to them. And then once we've exchanged encrypted email, it'll be encrypted from then on, which is great. It's just very transparent and easy to do.

Steve: Very cool.

Leo: Yeah, I really like it. You don't use this, though.

Steve: I just haven't the need. It's funny because there was some discussion in our newsgroups about the fact that I had said, yeah, I've never been a PGP or other user. And some people were saying, wow, I find that really strange for, you know, a big security guy like me. But the fact is, I don't know why, but I just haven't had electronic conversations. I mean, I have a corporate attorney, but I go to his office, and I sign paper documents. I'm not doing this stuff over the air.

Leo: Well, I don't, either. I have really no practical use for this. But it's just – well, there is one practical use, and that's the cryptographic signing, which we haven't really talked about yet. But, you know, because people will spoof my name – and it's very easy to spoof email...

Steve: Oh, yeah.

Leo: ...and as a public figure sometimes people will spoof your name, I sign all my mail. So if you get mail from me that's not signed, it's not from me.

Steve: Okay.

Leo: So there's a good reason to do it.

Steve: And in fact signing is one of the things that we cover today because signing uses hashes, so-called "cryptographic hashes."

Leo: Right.

Steve: Okay. So getting on to today's topic, cryptographic hashes. They're known by many different names, sometimes called a "compression function" or a "contraction function," popularly known as a "message digest," sometimes called a "fingerprint" or a "cryptographic checksum," also known as "message integrity checks" or sometimes "manipulation detection codes." Many different names.

Leo: These things have more pseudonyms than Mata Hari. I mean, this is very – but it's all the same.

Steve: Well, yes. Those are all sort of the same. Some of the terminology, like "fingerprint" or "manipulation detection code," sort of talks a little bit about – it implies the application that you're going to be using. But in every case these are reducing the amount of information typically to a much smaller size, which is really why "message digest" is one of the more common terms. A digest, as we know, like "Reader's Digest," that takes a long book and reduces it to a shorter form, or a digest that appears at the beginning of a lengthy paper that just explains what it's about, sort of, you know, a capsule summary.

The idea is that a cryptographic hash takes a long, typically long object in plain text – and in this case actually the term is "pre-image." In hashing terminology it is normally the same as the plain text, but technically it's called the "pre-image" versus the "hash value." So you take this pre-image that, for example, is email that you're sending to someone, and you run it through an algorithm, much as we've been talking, I mean, this is all about algorithms, of course. We had symmetric algorithms and asymmetric algorithms. Those were ciphers. And in all of the cases so far they've been reversible, that is, the only point to encrypting something is to decrypt it later.

Well, now we're running across something that is not reversible, and it's specifically the non-reversibility of it that is key. So we take our plain text or so-called "pre-image." We pump it through this cryptographic hash, and we get out something smaller, basically a token. And these are normally 128 bits, in some cases 160 bits, and even more recently longer tokens, 256 or 512 bits or more. But the idea is that that represents a fingerprint of what you fed in. That is to say that, if you change anything about the source text or the source content that is fed into the hash, you get a completely different output. And in fact, well-designed hashes, if you change – say that you had a 10K blob that you were going to hash. If you change one bit of that input anywhere, one bit, on average about half of the bits will change in the result. So, I mean, it basically takes a really large object, much larger than the result, and scrunches it down. And the point of this is, this is not reversible. Now...

Leo: Is this somewhat like a CRC?

Steve: Well, it's a CRC except – CRC, by the way, stands for Cyclic Redundancy Check. The

problem with...

Leo: It's a way of generating a number, a fairly large number, but a number that represents the data that you're looking at.

Steve: It is, but it's a special number. And a CRC is a good example of what a hash is not because it's very easy to deliberately create some input which will result in a deliberately chosen CRC, that is, a CRC is not cryptographically strong because you're able to say this input will give me this output. So if you wanted, for example, to create a different input that will produce the same output, that would be easy to do.

Leo: Okay. So you could change a message in such a way that it would give you the same CRC, but of course would be different.

Steve: Exactly. And that's where a cryptographic hash is special. A cryptographic hash has three main focuses. First is, it is very easy algorithmically to come up with a hash of an input text. It is very hard – and by hard we mean in cryptographic terms, you know, millions of millennia kind of hard – to go the other direction, that is, to design an input that will give you a chosen output. That is, I mean, basically it's like it's a random number, but it's really not. I mean, it's a deterministic function. You put in the same text; you're going to get the same hashed value. But it is cryptographically difficult – meaning really, really, really, really difficult – to design a text input that will give you an output that you have deliberately chosen. It's just you put stuff in, and you get out this magic token, but there's no way to design what you're going to put in to get a specific token.

Leo: Right.

Steve: And then the third principle of a hash is that there is no collision, that is, given two inputs, the probability of them hashing to the same result is really, really low. You're not going to get what's called a "hash collision" with, like, a trivial change to an input is going to give you a huge output change. So...

Leo: I mean, hashing is well known, has been around in computer science for a long time. How is a cryptographic hash – or maybe you're going to get to this – different from just a regular hash?

Steve: Well, okay, here's a perfect example. Say that I wanted to take everybody's IP address – we know that IP addresses are four bytes, and we've talked about the XOR function before. Say that I wanted to hash an IP address from four bytes down to one byte so that, for example, I wanted to, like, sort users to my website into 256 categories based on their IP. I could simply take the four bytes of their IP and XOR them together to give me a hash. And that would actually be a very nice hash because it would generally spread all the 4 billion IPs equally across just 256 results. So simply XORing the four bytes together. But it's not a cryptographic hash because I could look at someone's IP and trivially see what it's going to be. And I could easily design or choose IPs that would hash into a given result.

So hashes normally take some larger value and reduce them to fewer number. And good hashes will evenly distribute that reduction across their smaller space. What's special about a cryptographic hash is that you can't choose what you're going to send it in order to get a known result. And you can't go backwards. For example, if I hashed – just using the little XOR

example I gave, if I were to hash that from an IP address down to a result, I could easily spit out all kinds of IP addresses. That is, I could easily reverse that process and come up with IPs that are going to give me my final resulting hash. You cannot do that, that's specifically what you cannot do with a cryptographic hash.

Leo: Okay. Okay.

Steve: Okay. Now, one of the ways this is used is signing documents. And this is what we've talked about before is you take a piece of email. Now, we know that we could encrypt the document in order to make it secure and send it somewhere. But say that we didn't want to encrypt it, we just wanted to say, you know, as in the example you've given earlier, you know, I'm Leo Laporte, I really wrote this piece of email.

Leo: And it hasn't been changed since I wrote it.

Steve: Well, exactly.

Leo: This is very important.

Steve: Exactly. So what you do is, the PGP system you're using, for example, it will literally do – it will take a hashing function exactly like we've talked about, run your email through it, and produce basically a fingerprint of the document. So it says this is the result of hashing this longer email down. And it'll be 128 bits or 160 bits, depending upon what the hash is. And it says, you know, this is the fingerprint. Now, the problem is, if you just put the fingerprint at the end, somebody else could, in fact, change your document in some way, do a hash and change the fingerprint.

Leo: Just paste a new one in.

Steve: Just paste, exactly, replace the fingerprint as they have changed the document. So the final key here is, this is where our public key cryptography comes in again. You encrypt just the fingerprint, using your private key that nobody else has.

Leo: So not only does this say this is the hash for the message, but only I could create it.

Steve: Only you, using your private key, would have been able to encrypt that fingerprint. That's what gets appended to the message. Now the message gets sent to somebody else. And anyone who wants to can read it because the message itself was not encrypted. But to prove the authentic signer of the message, they would take your public key and decrypt the appended fingerprint using your public key. And that would give them the hash for the message. Then they perform the same hashing function you did and compare the hash they get with the hash that they decrypted from what you provided. Only if these documents are identical, that is, the one you originally signed and the one they received, only if they are absolutely identical will the hashes match.

Leo: Just so people understand from a practical point of view, if you get email from me or

anybody who's using this, it'll be plain text email. You could encrypt it, but that would be another thing to do. This is just signed. You'd get a plain text email, but at the beginning and the end it says, you know, "PGP signed" at the beginning, and at the end it says "Begin PGP signature," and there's this gobbledygook which is the cryptographic hash. Now, you could just see that and say, okay, fine. But in order to really verify that that came from me you would need to have PGP or what I use, which is the GNU Privacy Guard, an open source free one, installed. And then you'd need to get my public key to verify. Right?

Steve: Correct.

Leo: And it does that automatically. But if you don't have my public key, you can't verify it.

Steve: Right. So I don't mean to imply that these are complex things that the user has to manually do.

Leo: You know, the software's very good at this, yeah.

Steve: Exactly. If they were all set up to be doing this automatically, then it would just be, you know, it might just say "authenticated," you know, automatically in their system.

Leo: Well, I mean, I have to say it is automatic, but that's one of the complaints people have about this whole thing is it is so – it's not as easy as it ought to be, you know.

Steve: Right.

Leo: Mom's not doing it. It does require a little geekiness. You have to install the software, which is not always easy. You have to know about keys and know that you have to get a key, and you have to be able to search the key server and all of that stuff.

Steve: Well, and my complaint, frankly, is that I'm not worried that somebody's sending me stuff that you haven't sent me, yet every piece of email from you has all this extra crud in it, you know...

Leo: Well, it's not that much crud.

Steve: ...that's visible in your email. So...

Leo: Right. Well, there's another way. If you certificate sign, then it would just send an attached certificate. Same idea, but it's not visible in the body of the email.

Steve: Right.

Leo: I prefer to use the text method because it's more universal.

Steve: Well, now, one place that many people who are Internet users may have seen a cryptographic hash is in download sites, which will often post the MD5 and/or the SHA hash for the files that are being downloaded. And this is sort of an interesting, you know, real-world practical example that people may have run across before because the idea being that – and a lot of open source sites will also do this. They will say, here's our files, and here's the MD5 of it, or the MD5 hash, or whatever they're going to call it specifically. The idea being then that this is them saying that this is – basically it's the fingerprint for the file. You download the file and then, using your own MD5 or SHA-1 hashing tool – and all operating systems have these as freeware, they're easy to find on the 'Net – you then give your own hashing tool that file, which will quickly produce, basically, the fingerprint. And there you just do a visual comparison. You look at the website, and it says, you know, 5A30FE1...

Leo: It's long, yeah.

Steve: ...and you just visually compare that the result these people got when they produced the fingerprint is the same thing that you got when you produced it. And that tells you beyond, you know, with absolute certainty at a cryptographic level that that is, bit-for-bit, the identical file that they intended you to receive.

Leo: Now, you know, a lot of programs do it. But I'll tell you, the time when it really is important is when you are first installing PGP or GNU Privacy Guard, to make sure that you're not installing some Trojan horse that's posing as those. Because that would then compromise your security from then on.

Steve: Sure.

Leo: Yeah.

Steve: Well, now, there is one really interesting trick to this whole thing, and that is about the bit-length because, you know, these hashes are long in terms of bits. I mean, obviously we're used to talking about long bits. But there is an interesting and, I think, really sort of fun and fascinating attack on signed documents. If the person generating the document has no control over what they're signing, then they're not vulnerable to something that's known as the "birthday attack" on hashing. But say, for example, that somebody wanted to trick somebody into signing a contract that was not in their favor. It turns out that it's easier to do than you might expect.

First of all, let's step back a little bit. There's a well-known sort of statistical surprise known as the, what is it, the birthday puzzle or...

Leo: Oh, yeah, yeah.

Steve: ...birthday – the idea is, the question is, how many people would have to be in a room for one of them to have the same birthday as you?

Leo: No, two people to have the same birthday. Not necessarily as you.

Steve: I know, I know.

Leo: Okay, okay.

Steve: I'm doing this on purpose.

Leo: Oh, all right.

Steve: So how many people have to be in the room to have the same birthday as you? Turns out that's a lot of people. And remember, of course, we have 365 days in the year. So it turns out that you need 253 people in the room for the chance to be greater than 50/50 that one of them will share your birthday.

Leo: That makes sense. That makes sense.

Steve: It does. It sounds about right.

Leo: And if it were 365 people, it would be almost a certainty. Well, maybe not.

Steve: Well, no, because they might have a common birthday.

Leo: They might have duplicates; right. That's right, yeah.

Steve: But it turns out that 253 is the magic number of people where it's more than 50/50 that they're going to have the same birthday as you. The surprise is when you ask the question, how many people do you have to have in the room for any two of them to share the same birthday, and there the number is 23. And the reason is, if you have 23 people, you have 253 pairs. So it's the same 253, but instead of me insisting that someone have my birthday, it's, okay, what's the chance of a birthday collision?

Leo: Right.

Steve: And that's the key, the birthday collision between any two people. It turns out that the same kind of thing can be used in order to create fraudulent fingerprints using one-way hashes, using cryptographic hashes. The way that would work is, say that I was hashing, and I had a hash that only produced a 64-bit key. Well, I'm saying "only." But, I mean, 64 bits, that's a lot of keys. Since 32 bits is 4 billion, which is to say four with nine zeroes, 64 bits is that many squared. So that's going to be 16 billion billion possible hashes, which is to say 16 followed by 18 zeroes. Okay, I mean, that's just a lot. So the chance – say that you only had a hash with 64 bits. The chance of two different contracts having the same fingerprint, the same cryptographic hash, would be one in 16 with 18 zeroes. I mean, just diminishingly small. There's just no chance you're going to have, like, a good contract and an evil contract that are

going to have the same hash.

But if you're able to prepare both contracts, you could make small changes to each of them, like adding a space at the end of a line or putting two spaces after a period. That is, you make tiny changes to both of them, checking to see whether they have the same keys. That is, it's very much like the birthday attack, where you're not saying I want my evil contract to match the fingerprint of my good contract. Instead you're saying I'm going to make a bunch of different evil contracts and a bunch of different good contracts. And I'm going to look for any collision of the hashes between them. And it turns out that is far more probable.

Leo: Really.

Steve: Just like having many people in the room comparing their birthdays, where anyone who has a birthday collision gives you success. What this allows you to do, for the same reason of much lower probability – I'm sorry, much higher probability of the collision occurring, this allows you to prepare two contracts, a good one and a bad one, where they're going to have the same fingerprint. So say that I produce two contracts like this. I then get someone to cryptographically sign it. Later I can come along and claim that they signed the evil contract because it has the same fingerprint. And any judge who understood cryptography would have to accept the fact that the evil one was signed.

Leo: Wow.

Steve: And it turns out that the number you need is half of the bit length, that is, 2 to the 32.

Leo: That still seems a lot.

Steve: That's 4 billion. But it turns out that, since you're just doing a forward hash, and you're making small changes each time, it doesn't take that long to produce that many different contracts. And the chance is greater than 50/50, if you do that with both of them, you're going to find a collision and be able to produce basically a fraudulent hash. And that's why hashes are much longer than that. That's why they're 128 bits or 160 bits and even more now because it turns out, if you have control of both sources, then there is this chance for being fraudulent. Now, this is, for example, not a problem downloading a file because no hacker will have produced the good file and a bad file unless, you know, they were really, really clever. But in any case these hashes, even the smallest hash in common use today is 128 bits, which means you would have to produce 2 to the 64 sets of hashes of the good file and the bad file, 2 to the 64. Now we're back up at, you know, 24 zeroes on the end of the number. So we're really safe.

Leo: People might say, well, even the 4 billion sounds like a lot. But with computers – it's not 4 billion, first of all. But with computers you can do this quite quickly, I would imagine.

Steve: Right.

Leo: It's a form of brute-force attack.

Steve: Right.

Leo: Wow, that's interesting. I didn't know that.

Steve: So, yeah, it's an interesting way that what looks like a very secure hash can be defeated, if you have control over generating both the good and the bad. There is, you know, because of this birthday collision there is a chance to, I mean, still it's a huge amount of work. But if the hash is too short, you end up needing less work than you would expect. So you just need to make sure you're using long hashes. And everything today does.

Leo: So, Steve, that's why there's so much gobbledygook at the end of my messages. And I don't want to hear any more complaints about it. It's for security. Does it really bug you that all that stuff's in my messages?

Steve: No, no. I tune it out.

Leo: You tune it out.

Steve: I tune it out.

Leo: But fortunately, if you have the right software on your computer, it doesn't tune it out, and it says "Verified. This came from Leo."

Steve: So this is the final piece of our puzzle. There's really no need to spend a whole episode talking about cryptographic random numbers. We've talked about the idea of the need for generating really random numbers. For example, we mentioned it last week when we were talking about choosing a really good random number or pseudorandom number, unpredictable, unguessable at least, and then using that as your symmetric key for your bulk encryption, and then using public key cryptography to encrypt it, in order to allow a non-secure channel to deliver that encrypted symmetric key and then decrypt it. So basically we've got symmetric block ciphers. We've got asymmetric block – I'm sorry. Symmetric block ciphers, symmetric stream ciphers, asymmetric ciphers which are block ciphers, cryptographic hashes, and cryptographic random number generation. That's sort of the suite of our fundamental algorithms that allow us to do all kinds of cool stuff. And week after next we're going to look at all the different ways these five building blocks can be pieced together in order to solve all the common problems of cryptography.

Leo: It's really fascinating. You know, it's kind of spy stuff, and yet it's very practical. And it's kind of elegant and mathematical, too. It's really been fun.

Steve: And anybody who uses the web, you know, now you know what the MD5 is when you see them on download sites. And anyone who establishes a secure connection to a server is using symmetric and asymmetric ciphers. In fact, those certificates are signed – and we're going to talk about certificate signing week after next, after next week's Q&A episode. And, you know, it's using hashing in order to put all this stuff together.

Leo: So cool. So neat. Wow. And so next week we will do the question-and-answer session.

Steve: Yup. Number 36.

Leo: And then we'll be back and talk a little bit about practical applications, which is always fascinating, too. But now you have the fundamentals.

Steve: We have all the building blocks in place.

Leo: I want to thank our good friends at Astaro for – and I mispronounced their name last week, and I apologize. Astaro.com, Astaro Internet Security, their hardware and software solutions. And by the way, don't forget there is a free version for home use to protect up to 10 IP addresses. And this is very cool. Basically it's open source. You can add it to an old PC, put a second Ethernet card in, and you've got an industrial-strength, stateful packet firewall. You've got VPN for both PPTP, the Windows VPN, but also IPSec. You get intrusion detection and protection. And you can even add a very powerful AV and content and spam filtering for as little as 79 euros a year. It's all free, but then the antivirus on top of that. For more information, www.astaro.com. And thanks to our friends at America Online for providing us with the bandwidth for this podcast, of course AOL.com/podcasting. For more information about the topics we cover on Security Now!, visit Steve's site, GRC.com/securitynow.htm.

Somebody said, "I'm confused, there's two show notes." I put some links sometimes on my page. But really the main show notes are on your page, Steve. That's where you'll find also transcripts, 16KB versions for the bandwidth-impaired, and let's not forget SpinRite, which really is Steve's sponsor.

Steve: It is. It is the sponsor of GRC and my website and our bandwidth and all my time, yeah.

Leo: Keeps him in ham sandwiches. So if you have a problem with your hard drive, it's the ultimate disk maintenance and recovery tool. SpinRite.info for testimonials. GRC.com is the other place to go. And that's a good place for ShieldsUP!. And you're still developing, I know, your freeware, your security freeware like DCOMbobulator, UnPlug n' Pray. You've got a new one in the works, don't you?

Steve: Actually I'm working right now, I've been coding all week. GRC will shortly have a brand new feature, the first brand new feature sort of service that we've offered in years. It's something I've wanted to do. And in fact, Leo, I'm getting it ready because I want to introduce it on the "Call for Help" show up in Toronto next week.

Leo: Oh, cool.

Steve: Or week after next, rather.

Leo: Good. So watch "Call for Help," if you're lucky enough to be able to get it. Otherwise we'll mention it.

Steve: We definitely will in two weeks, yeah.

Leo: Yeah, yeah. Thanks, Steve. We'll see you next week for our Q&A segment.

Steve: Great.

Leo: I'm Leo Laporte. Thanks for joining us. We'll see you next Thursday on Security Now!. Security Now! is brought to you by Astaro, makers of the Astaro Security Gateway, on the web at www.astaro.com.