Security Now! #1008 - 01-14-25

HOTP and TOTP

This week on Security Now!

Meta winds down 3rd-party content filtering. Is encryption soon to follow? Taking over abandoned Command & Control server domains (strictly for research purposes only!). IoT devices to get the “Cyber Trust Mark” – will anyone notice or care? “SyncThing” receives a (blessedly infrequent) update. Government email is not using encryption? Really? Email relaying prevents point-to-point end-to-end encryption and authentication. Just because Let's Encrypt doesn't support email doesn't mean it's impossible. What Sci-Fi does ChatGPT think I (Steve) should start reading next? To auto-update or not to auto-update? — is that one question or two? And, until today, we've never taken a deep dive into the technology of time-varying 6-digit one time tokens. Let’s fix that! *(And last week’s uncaptioned picture is finally captioned!)*

Some believe that, long ago, humans roamed this beautiful planet

That caption was inspired by a suggestion from listener Steve Kangas who wrote: “Caption Contest: “Earth Abides” From a great book about life of a small number of survivors after a devastating world wide pandemic.”

Security News

Meta winds down content filtering for Facebook and Instagram

It wasn't until I encountered Matthew Green's comment about Meta's announcement last week that I decided to mention any of this today. So before we get to what Matthew Green posted, here's a brief update for those who may have been without any other source of news for the past week:

Last Tuesday, Mark Zuckerberg posted a video in coordination with a Meta news release titled "More Speech and Fewer Mistakes". Part of what they wrote under the heading "Ending Third Party Fact Checking Program, Moving to Community Notes" was...

> *When we launched our independent fact checking program in 2016, we were very clear that we didn't want to be the arbiters of truth. We made what we thought was the best and most reasonable choice at the time, which was to hand that responsibility over to independent fact checking organizations. The intention of the program was to have these independent experts give people more information about the things they see online, particularly viral hoaxes, so they were able to judge for themselves what they saw and read.*
>
> *That's not the way things played out, especially in the United States. Experts, like everyone else, have their own biases and perspectives. This showed up in the choices some made about what to fact check and how. Over time we ended up with too much content being fact checked that people would understand to be legitimate political speech and debate. Our system then attached real consequences in the form of intrusive labels and reduced distribution. A program intended to inform too often became a tool to censor.*
>
> *We are now changing this approach. We will end the current third party fact checking program in the United States and instead begin moving to a Community Notes program. We've seen this approach work on X – where they empower their community to decide when posts are potentially misleading and need more context, and people across a diverse range of perspectives decide what sort of context is helpful for other users to see. We think this could be a better way of achieving our original intention of providing people with information about what they're seeing – and one that's less prone to bias.*

And a bit lower down in the lengthy posting, Meta says: "As part of these changes, we will be moving the trust and safety teams that write our content policies and review content out of California to Texas and other US locations." So, with that preamble, Matthew Green, the well known Cryptographer at Johns Hopkins University posted:

What's interesting is that our own CISA – Cybersecurity Infrastructure Security Agency – just published a 5-page PDF titled "Mobile Communications Best Practice Guidance" and its first, #1 recommendation reads:

> ***Use only end-to-end encrypted communications.***
> *Adopt a free messaging application for secure communications that guarantees end-to-end encryption, such as Signal or similar apps. CISA recommends an end-to-end encrypted messaging app that is compatible with both iPhone and Android operating systems, allowing for text message interoperability across platforms. Such apps may also offer clients for MacOS, Windows, and Linux, and sometimes the web. These apps typically support one-on- one text chats, group chats with up to 1,000 participants, and encrypted voice and video calls. Additionally, they may include features like disappearing messages and images, which can enhance privacy. When selecting an end-to-end encrypted messaging app, evaluate the extent to which the app and associated services collect and store metadata.*

https://www.cisa.gov/sites/default/files/2024-12/guidance-mobile-communications-best-practices.pdf

And there was another related piece of news about this Telecom hack: Reporting is that source have told the Wall Street Journal the names of three additional American telcos that were breached by the Chinese espionage group Salt Typhoon last year. They are Charter, Consolidated Communications, and Windstream. As we know from previous reporting, the other four known victims were AT&T, Lumen, Verizon, and T-Mobile. Given the count of breached firms that's been shared publicly, two more telcos remain to be named.

So the clear point being made here is that no one can rely upon the security of telecommunications carriers to protect the privacy of anything that uses their bandwidth. This begs the question: Who ever **did** believe that we could rely upon anyone else's security? After all, that's the entire point of, and reason for, communicating consumers adding our own after-the-fact pre-Internet encryption to everything we really care about – specifically so that we don't need to trust anyone else. TNS – Trust No One – has been the rule of the road from the start.

So this brings us to Matthew Green's worries about encryption. And at this point that's all they are – worries – and I would suggest that it's probably not worth worrying about. No one appears to have any clear idea of what the incoming Trump administration plans to do – or will do. But I'm certain that Elon Musk, who appears to have a great deal of technical sway with president-elect Trump, certainly understands the pros and cons of any form of mandated backdoor being forced into today's exception-free end-to-end encryption. I'm certain that our incoming president is well aware that the Chinese government appears to be behind much, if not most, of the hacking into our nation's critical infrastructure, and especially the government's. Mr. Trump's feeling about China are well known, so I would be quite surprised if he wanted to, in any way, open any doors – or backdoors – into our nation's encrypted communications. I would, therefore, be **very** surprised if anything were to change along the lines that Matthew fears. Changes in content moderation are not even in the same world as changes that would weaken our encrypted communications. I think that much should be clear to everyone.

Whose Command & Control Server is it Anyway?

Here's an interesting story that everyone is sure to get a kick out of. I recall that we talked about the security research group Watchtowr Labs not long ago. They're memorable not only for what they do, but because they drop the 'e' from the "tower" of "Watchtowr". Here's what they posted last Wednesday about their latest escapade under the title "Backdooring Your Backdoors - Another $20 Domain, More Governments". They wrote:

After the excitement of our .MOBI research, we were left twiddling our thumbs. As you may recall, in 2024, we demonstrated the impact of an unregistered domain when we subverted the TLS/SSL CA process for verifying domain ownership to give ourselves the ability to issue valid and trusted TLS/SSL certificates for any .MOBI domain. This resulted in significant Internet-wide change, with Google petitioning the CAB Forum to wholly sunset the use of WHOIS for ownership validation when issuing CA-signed TLS/SSL certificates.

As always, idle hands, idle minds - it was never going to be long until our ill-advised sense of adventure struck again, and at this point, the only thing holding us back is our publishing schedule.

This time, our sense of adventure struck again, in the same vein of expired and abandoned infrastructure - but with a little twist. Today, we're taking you through our adventures through what we've affectionately termed - mass-hacking-on-autopilot.

Imagine you want to gain access to thousands of systems, but don't feel like investing the effort to identify and compromise systems yourself - or getting your hands dirty.

Instead, you commandeer abandoned backdoors in regularly used backdoors to effectively 'steal the spoils' of someone else's work, giving you the same access to a compromised system as the person who put the effort into identifying the mechanism to compromise, and performing the compromise of said system in the first place. Zero effort, same result - all for the price of a domain.

So we've been hijacking backdoors (that were reliant on now-abandoned infrastructure and/or expired domains) that themselves existed inside backdoors, and we've been watching the results flood in. This hijacking allowed us to track compromised hosts as they 'reported in', and theoretically gave us the power to commandeer and control these compromised hosts.

Over 4,000 unique and live backdoors later (a number which continues to grow), we decided this research would never be finished, and it would be interesting to share the results in its current state.

So we can report that across those 4000 unique and live backdoors, we now have access to:

- *Multiple compromised governments including Bangladesh, China and Nigeria.*
- *Compromised universities/higher education entities across Thailand, China, South Korea and more*
- *and, much much more (we've so far recorded over 300MB of logs to trawl through).*

As always, we're quick to remind everyone - we're not the first to track hackers for fun, we no doubt won't be the last. But, we have enjoyed continuing to exploit what truly appears to be a hugely underrated vulnerability class - abandoned and expired infrastructure - to basically give ourselves 'theoretical' free access to thousands of systems for the cost of a few (yet again) $20 domain names.

https://labs.watchtowr.com/more-governments-backdoors-in-your-backdoors/

I have a link to their much longer report containing much more detail. But what this amounts to is that they found that some hacker gangs had allowed the domain names used by infiltrated client malware to reach their command and control servers to expire. Who knows why? Perhaps those particular hackers are now behind bars. But for whatever reason, those domains were never renewed. This meant that the Watchtowr researchers were able to re-register those previously abandoned domain names to establish their own IP for them, Then, the next time the infiltrated malware performed a DNS lookup as the first step in reestablishing contact with the malicious hacker's mothership the IP the malware received would be the researcher's.

So the Watchtowr folks registered those domains and pointed the domain's IP address to their incoming connection-monitor. What they found was that thousands – more than 4,000 – machines scattered around the planet that had previously been infected were still, today, trying to reestablish contact with their controllers. I'm sure that the Watchtowr folks were only gathering data. But many of the incoming links were to remote web shells which would allow anyone accepting such a connection to issue commands as if they were administrators of those remote machines. Since the wayward domains were now under their control, the Watchtowr folks felt free to list 31 domains they now control:

6634596.com	flyphoto.us	jbl86.com	precision-gaming.com
aljazeera7.com	guerrilladns.com	library-ar.com	rootshell-security.net
alturks.com	h0ld-up.info	ll4best.com	shellci.biz
caspian-pirates.org	h4cks.in	localshell.net	templatez.org
csthis.com	hackru.info	locus7s.com	w2img.com
dcvi.net	imhabirligi.com	love-1-love.com	waterski21.com
drakdandy.net	nettekiadres.com	lpl38.com	yywjw.com
emp3ror.com	ironwarez.info	odayexp.com	

Since they have control over those domains, they then obtained a big wildcard TLS cert covering all of those domain roots and began accepting HTTPS webshell connections.

Just imagine how many long forgotten and unattended systems "out there" are hosting old malware that gangs have moved on from and forgotten. But nevertheless that malware is still faithfully attempting to contact home base to receive new instructions and commands. At this point the only way such malware will ever disappear will be if those machines are eventually shut down and decommissioned, reformatted and reinstalled with new software, or perhaps updated if anyone ever stops to realize that the old machine is still technically alive and definitely way way out of date – for whatever purpose it might have.

U.S. CYBER TRUST MARK

IoT devices get 'Cyber Trust Mark'

Last Tuesday the U.S. government announced the launch of the U.S. Cyber Trust Mark. This is a new cybersecurity safety label for Internet-of-Things (IoT) consumer devices. It's unclear to me whether any consumers will care or even notice. Today's IoT devices are often purchased online where any such "marks" go unseen. And with so many certifying standards bodies all weighing in with their own seals of approval, what difference is one more going to make?

But there may be reason to hope that the presence of such a seal might mean something to IoT companies that are seeking any edge they can get. So, if changing their behavior, or the behavior of their products in ways that enhance the privacy and security of the users means that they qualify for yet another seal of approval, then this new FCC award may have been worth creating.

In their announcement last week the U.S. Federal Communications Commission (FCC) said: *"IoT products can be susceptible to a range of security vulnerabilities."* [oh, really?] *"Under this program, qualifying consumer smart products that meet robust cybersecurity standards will bear a label—including a new 'U.S Cyber Trust Mark.'"*

So, as part of the effort, the logo will be accompanied by a QR code users can scan which will take them to an information registry containing easy-to-understand details about the security of the product, such as the support period and whether software patches and security updates are automatic. So this seems like something that would be of tremendous interest to the audience of this podcast... but I wonder how clued-in the typical consumer is today?

Still, the registry's information will also contain details related to changing the default password and the various steps users can take to configure the device securely. The initiative, which was announced last summer in July of 2023, will involve the participation of third-party cybersecurity administrators who will be in charge of evaluating product applications and authorizing use of the label. Compliance testing will be handled by accredited and independent 3rd-party labs.

Eligible products that come under the purview of the Cyber Trust Mark program will include internet-connected home security cameras, voice-activated shopping devices, smart appliances, fitness trackers, garage door openers, and baby monitors. The "mark" does not cover medical devices which are separately and already regulated by the U.S. Food and Drug Administration. Nor motor vehicles and equipment regulated by the National Highway Traffic Safety Administration (NHTSA); nor any wired devices and products used for manufacturing, industrial control, or enterprise applications. So, our basic consumer electronics.

The program also does not extend to equipment added to the FCC's Covered List; and products manufactured by companies added to other lists for national security reasons (Department of Commerce's Entity List or Department of Defense's List of Chinese Military Companies), or any banned from Federal procurement. So, again, basic consumer electronics.

In order to apply to use the U.S. Cyber Trust Mark, manufacturers who meet the eligibility criteria must have their products tested by an accredited and FCC-recognized CyberLAB to ensure they meet the program's cybersecurity requirements, and then submit an application to a Cybersecurity Label Administrator with the necessary supporting documents.

For their part, last week the White House chimed in, saying: "The U.S. Cyber Trust Mark program allows for testing products against established cybersecurity criteria from the U.S. National Institute of Standards and Technology (NIST) via compliance testing by accredited labs, and earn the Cyber Trust Mark label. This will provide an easy way for American consumers to determine the cybersecurity of products they choose to bring into their homes."

I searched around the various announcement pages from both last summer and more recently. There's very clearly a lot of movement on this front with various companies and individuals being selected to fill key roles. But what I was unable to find at this point was any clear specification for the criteria NIST will be setting for the behavior of complying devices. However, we've been seeing a lot of good sounding policies coming from NIST and CISA. You know, things like requiring long lifetime support and firmware updates, and in many cases requiring consumer devices to be able to keep themselves updated and requiring that a physical button on the device be pressed before any potentially dangerous configuration change will be applied. These are all really hopeful changes in the right direction and a decade from now once all of our first generation systems have been retired we could expect to see a very different terrain that we have today.

That's all of the moderately interesting security news I was able to find. But I learned something through my own recent work that I wanted to share because, though it wasn't what I expected or hoped for, in retrospect it's sort of obvious. So let's talk about the details of securing DNS by tunneling our queries for the IP addresses of remote hosts

Miscellany

SyncThing moved to v1.29.2
What we want in software is reliability and stability, with infrequent discovery and repair of the exceedingly rare obscure bug. What we don't want are daily, weekly or even monthly updates where we're on the receiving end of the ongoing maintenance of software that advertises itself as being feature complete and finished. As I've noted before, while I like the many features of the Notepad++ editor for Windows, its author's apparent inability to either leave it alone or get it right has become a source of continual annoyance. If supporting his work means encouraging him to keep changing it, I'm less inclined to do so.

All of that is a preamble to an event I can't recall ever experiencing: Sunday morning I was surprised by an instance of SyncThing, which I have open on a side monitor so that I'm able to keep an eye on its synchronization with my other location, notified me of an available upgrade. I can't recall that ever happening before. And that's exactly what you want. The bug that was fixed by the release of v1.29.2 was obscure. The person who discovered it wrote: *"By changing the contents of a synced directory it seems that Syncthing crashes when scanning a subdirectory name that contains an "ü"."* The report of the problem two days ago generated some online dialog as logs were exchanged and examined. And a resolution was produced Sunday morning, two days later.

Since SyncThing has become a favorite solution for many of us – it's what Leo and I both use extensively to keep the working files on our various platforms synchronized – I wanted to let everyone know that a tiny incremental improvement event had occurred, but I also wanted to share the observation of how refreshing it is to see a highly complex and functional open source software project, that's finished, being largely left alone because it does everything it was designed to do.

Closing the Loop

Last week's discussion of the persistence of unencrypted email-in-transit, and the fact that some 3.3 million email servers worldwide, most of them located within the United States, are still not bothering to offer a TLS certificate that would allow for email encryption, triggered a lot of feedback from our listeners. Here's some of it, with my thoughts and reactions:

Philip Pedersen

> *Steve, After your piece on the non-use of TLS for SMTP, I looked at some of the email I've received. I thought it might be small businesses that had not set up certificates, but found it to be larger companies as well. The most troublesome one I found is that TreasuryDirect.gov sends their one-time password notifications in the clear. It also seems like organizations with multiple email servers don't have all of them set up for TLS. ID.me sends the Welcome to ID.me message from a non-TLS server, although the other messages sent while setting up an account (to log into IRS.gov) were using TLS. Regards, Phil*

Philip's note is interesting because it hints at something I want to discuss in greater detail after I share another piece of feedback. But here's the part of what's interesting: Philip wrote: *"The most troublesome one I found is that TreasuryDirect.gov sends their one-time password notifications in the clear."* What's tricky about diagnosing email's use of TLS-encrypted connections is that it mirrors today's web browsing where the connecting-to server is the one that's offering to prove its identity to its caller. So, in the case of email, it's not the sending SMTP server that offers its TLS certificate, it's the SMTP server on the receiving end that does so.

So, a sending SMTP server would always have the choice of refusing to send email to any recipient SMTP server that wasn't offering to prove its identity with a TLS certificate and encrypt their conversation and any received email with a TLS connection. But, otherwise, whether or not a sending server is able to protect the email it wishes to send, is up to the receiving server. Either the sender or the receiver might elect to not send or receive messages over an unencrypted connection, but it's only the receiving server that's able to offer the use of encryption for both sides to then enjoy.

Let's look at what **Travis Hayes**, another listener of ours, has to say:

> *Hi Steve, Enjoying this week's show, as always.*
>
> *Regarding the TLS encryption of e-mail, the thought occurred to me that the reason we are where we are with unencrypted transport of e-mail between gateways is because e-mail from the beginning was always designed to be tolerant of multiple hops. Just like physical mail, if something is to be sent confidentially, it's put into an envelope rather than on a postcard for everyone handling it to read.*
>
> *This is different from the design of the relative latecomer HTTP protocol, which was designed to be point-to-point. The reason S/MIME, PGP (GPG), and the like were invented was to address that; to handle the transfer of sealed packages over a system of untrusted, unknown delivery gateways.*
>
> *So even if widespread adoption of TLS between gateways was achieved, I still have to be trusting that my mail host, your mail host, and any intermediate gateways are trustworthy. Even if the mail exchangers talk between themselves over TLS connections, the only way to*

ensure confidentiality between us is to encrypt the payload itself-- and that's the piece that is missing when all those one-time 6-digit PINs and "Forgot My Password" reset URLs are being sent to me.

Until there is some way for my bank's automated systems and me to exchange public keys so they can securely send those PINs to me, it doesn't matter if my bank's ISP and Gmail connect over TLS. I think there's some interesting things that could be done with the DKIM system; we are already digitally signing e-mail to show it's authentic; why are we not encrypting the message body as well? Thanks again for the show! -Cheers, Travis

The point Travis made about email being a multi-point relaying technology is crucial because, as he noted, TLS is only able to work with HTTP because users' web browser clients directly connect to the servers from which they wish to obtain web pages and other web application data.

If a web browser were to connect to any sort of intermediary server we would call that a man-in-the-middle attack – which we go out of our way to prevent. The point is with TLS – Transport Layer Security – we receive a certificate directly from the server we wish to trust which asserts that server's identity. There is no middleman mechanism.

One reason for this is that whereas web surfing is a real-time point-to-point activity, email was never guaranteed to be immediate. These days it tends to be, but that's mostly coincidence. Email was deliberately designed to be a store-and-forward system where someone's mail message would be dropped onto an SMTP server with the address of its destination and that SMTP server would then forward their email onward toward that destination. If the receiving server was not answered at the moment, another server might be tried if the destination's DNS MX (Mail eXchange) records offered more than one, or the email would be queued for later delivery. Having watched the delivery queue of my own email server when it's sending more than 15 thousand pieces of email weekly to those in this audience who have subscribed, I've seen that it doesn't all go through quickly or immediately. And I know that everyone has experienced the occasional delay where someone says "Hey I just emailed that to you. You don't have it yet?" And then a few minutes later it shows up. This store-and-forward system was what allowed the Internet's email delivery to be extremely robust in the early days when connectivity was far less robust and when receiving SMTP servers might be coming on or off the Internet for whatever reason.

Things have changed dramatically since those early days. One of the things that has changed is that connectivity is now pretty much "always-on" and servers are pretty much "always up". But "pretty much" is not 100%. From time to time I need to update and reboot GRC's servers. During those times, for a few minutes, GRC's visitors will receive 404 messages about the site being down, and any remote SMTP server that's attempting to deliver email will find that they need to queue it and retry in a few minutes. So the need to store and forward has not disappeared.

But as I noted in thinking about Philip's earlier note, any remote SMTP server that **insists** upon sending email to GRC over a TLS connection, or if GRC were to **insist** upon only receiving email over TLS connections, then that remote server would need to ask for a TLS connection which GRC would offer, which would allow that remote server to authenticate GRC and for them to bring up an encrypted tunnel. However, note that although we do get encryption for privacy, the authentication is only one-way. GRC offers up its TLS certificate but the incoming connecting SMTP server does not. So it is sort of a one-sided deal.

What this all appears to represent more than anything else is laziness on the part of the industry. We talked last week about how free certificates were not easily deployed using the ACME protocol because it appeared to be myopically designed for web-only use.

But encryption, if that's what we want, is easily obtained. As we've often discussed, standard generic Diffie-Hellman key agreement allows two parties to publicly negotiate a secret key which they could then use for their communication. This would protect email in transit from passive eavesdropping. But since Diffie-Hellman style key agreement does not, itself, authenticate the endpoints, this would not prevent an active attacker from intercepting email communications as a man in the middle, then negotiating separate keys with each endpoint and being able to see everything in the clear as it passes through this intercepting tap.

But we do have a potential mechanism that would solve the entire problem if anyone really cared to, because although it's not the default case for anonymous web browsing, it is possible for both ends of a TLS connection to require the other end to provide a trusted TLS identity certificate. So simultaneous mutual authentication is possible. But no one really appears to care that much and there doesn't appear to be any movement afoot to improve email security.

We do care about spam and spoofing. So those problems have been solved. SPF allows a domain to specify which SMTP servers are allowed to originate its email and DKIM allows those SMTP servers to cryptographically sign the email they send. In both cases, DNS is used to supply the SPF record and the server's matching DKIM public key. This is done to prevent others from being able to originate spoofed email claiming to come from any source that has protected itself with these measures. But even then, it's up to the recipient to care enough to check.

I'm not sure where all of this leaves us. We definitely have the tools today to bring up mutually authenticated and fully point-to-point encrypted email. But if we were to insist upon doing, before we could insist upon doing that, all email servers would need to have current and maintained certificates — just as all web servers do today.

And this brings us to our listeners who have arranged to do so. For example:

Josh Caluette in Austin, TX

> *Hi Steve, I was just listening to last week's podcast and I heard you mention that let's Encrypt does not support email services. However, I have been using Let's Encrypt on my mail servers for a few years now.*
>
> *The certbot app has some plug-ins that make this possible even without a web server. One of the plug-ins is for nginx and apache which will allow it to spin up a temporary webserver for the verification process, then takes it down again.*
>
> *The other plug-in is for DNS TXT verification. There is a RFC-2136 Dynamic DNS plugin which allows for dynamically updating a DNS zone with the necessary TXT record, waiting for propagation, completing verification and then deleting the record. This works with any servers that support and are configured to allow Dynamic DNS updates securely using private keys.*
>
> *There's a similar plug-in which I use specifically for cloudflare. It does the same thing but it works with the Cloudflare API to dynamically update the DNS zone with the correct TXT record.*

Once the certificate has been generated or renewed, it can be used in the config of anything that accepts certificate private/public keys. Because the file names do not change, you can easily configure services to point to the let's Encrypt managed files and then configure certbot with a post-script to restart the necessary service(s) in order to begin using the new certificate. I have been using this for the past couple of years and it has worked great with no intervention.

I have some monitors setup that monitor all of the certs used by services and alerts me if any of them get within 28 days of expiration, as that indicates a problem, since they should be renewed by or before reaching the 30 day mark. But anytime there has been a failure it has been due to my own errors (firewall changes, bad configuration changes, etc.)

Thanks for all you do. I look forward to the podcast during my 2-days-a-week commute to and from work.

I think Josh's note illustrates two things perfectly: First, yes, it's possible and second, no, it's neither automatic nor easy. And, not surprisingly, many of our listeners who are technically sophisticated and capable of rolling their own kludges have similarly done so. And a kludge it is. That's the proper term for arranging to create a temporary web server to satisfy a port 80 only certificate-issuing service, or dynamically edit DNS and wait for propagation, then copy the resulting certificate around and restart all dependent services nightly so that the updated certificates are recognized.

That's the very definition of a kludge, and as I mentioned last week, I fully intend to do something similar – I'll have no choice – if the lifetimes for all certificates are forced to drop below one year.

Given that long certificate lifetimes appears to be an entirely made up problem, the more I've thought about this the more it seems that web browser certificates should be members of a separate elite class, if that's what they want. Let them expire every six days so long as anyone offering the ACME protocol will keep them fresh. But then leave everyone else alone. Let non web servers use good old reasonable lifetime 3-year certificates for our Internet appliances, email servers, and other things. Don't force this nonsense down everyone's throat. Or allow those of us who wish to obtain an identity asserting certificate – for which we're paying good money – to decide for ourselves how long that certificate should last.

Every time I talk about this I get myself all worked up, so let's change the subject...

Doug Curtis in Waukesha, Wisconsin

Steve, Thanks so much for your overview of the current state of the art on AI. It prompted me to use ChatGPT to get some recommendations for more sci-fi books. I've really enjoyed some of the recommendations that you have made over the years in various Security Now episodes. I received a gift (for Christmas) of several credits toward Audible, so I wanted something new to listen to. I started by asking ChatGPT about two of my favorite scifi series, the Bobiverse and the Giants series, then I asked it, based on those two series, if it could make recommendations based on my preferences. It made quite a few. I'm starting with something called the Murderbot Diaries by Martha Wells. So far, a few Chapters into the first book of the series, I'm liking it very much. The next book that I'm going to try after this one is one called Old Man's War by John Scalzi. I've read another of his books, called Starter Villain, which was very interesting. Thanks again for all that you and Leo do with the Security Now podcast and

for your SpinRite program. I have a license and have used it several times.
Regards, Doug

I've not looked at the Murderbot Diaries but I've definitely heard of them and other of our listeners have recommended them to me. And the same goes for John Scalzi's "Old Man's War" novel. It's been recommended, too.

Taking Doug's clue of asking ChatGPT for some recommendations, I logged into my own ChatGPT account, selected the full-strength o1 model, and asked the following question:

I enjoy reading science fiction novels, and my favorite author is Peter F. Hamilton. I also enjoyed the Silver Ships series and Ryk Brown's Frontier's Saga series. Can you recommend other authors whose science fiction novels may be similar?

This gave it some pause while it worked through what I was asking. The o1 model shows you the various phases of its work as it's going along. After cogitating for a full 10 seconds, it replied:

If you're a fan of large-scale space opera (like Peter F. Hamilton's Commonwealth or Night's Dawn universes) and long-running series with plenty of interstellar adventure (like S.H. Jucha's Silver Ships or Ryk Brown's Frontier's Saga), here are some authors/series that capture a similar feel:

1. Alastair Reynolds

- *Recommended starting point: Revelation Space*
- *Why you might enjoy it: Reynolds is known for sweeping, galaxy-spanning storylines that blend astrophysical plausibility with far-future imagination. If you like Hamilton's big ideas, Reynolds' Revelation Space universe offers a similar sense of scale and mystery.*

2. Neal Asher

- *Recommended starting point: Gridlinked or The Skinner (both set in the "Polity" universe)*
- *Why you might enjoy it: Asher's Polity novels deliver gritty, action-packed stories in a post-scarcity future run by powerful AIs. The cast includes enhanced humans, alien threats, and a lot of "big tech" ideas, echoing the grand technology vistas in Hamilton's work.*

3. James S.A. Corey

- *Recommended starting point: Leviathan Wakes (Book 1 of The Expanse)*
- *Why you might enjoy it: The Expanse is a multi-volume series known for its realistic(ish) physics, gritty political drama, and deep space exploration. While it starts in our solar system rather than spanning galaxies, it still provides epic storytelling and intense space conflicts.*

4. Jack Campbell

- *Recommended starting point: Dauntless (Book 1 of The Lost Fleet)*

- *Why you might enjoy it: Campbell's Lost Fleet series focuses on large-scale fleet engagements, strategy, and leadership under pressure—perfect if you like space opera with a strong military twist, intricate battles, and a big overarching plot.*

5. David Weber

- *Recommended starting point: On Basilisk Station (Book 1 of the "Honor Harrington" series)*
- *Why you might enjoy it: While the Honorverse is definitely military sci-fi, it features expansive world-building across numerous star systems, detailed technology, and a strong continuity across a large number of books—much like Hamilton's broad universe-building.*

6. Elizabeth Moon

- *Recommended starting point: Trading in Danger (Book 1 of the "Vatta's War" series)*
- *Why you might enjoy it: Moon's style blends space opera, military SF, and strong character development. The Vatta's War series follows Kylara Vatta and her family in an interstellar trading empire—so you get both epic space battles and corporate/political intrigue.*

7. Glynn Stewart

- *Recommended starting point: Starship's Mage (Book 1 of the "Starship's Mage" series) or Duchy of Terra series*
- *Why you might enjoy it: Stewart's series often combine space opera tropes—like starship crews, alien empires, and grand-scale warfare—with a smooth, action-driven narrative. They're fast-paced, entertaining reads that grow into multi-book arcs.*

8. Michael Cobley

- *Recommended starting point: Seeds of Earth (Book 1 of the "Humanity's Fire" trilogy)*
- *Why you might enjoy it: Cobley's Humanity's Fire features multiple human colonies struggling after Earth's near-destruction, alien civilizations, and galaxy-scale conflict. If you enjoy multi-threaded plots and epic storytelling, this trilogy is worth a look.*

A Few More to Consider

- *John Scalzi – Known for his blend of humor and military SF, especially Old Man's War series.*
- *Lois McMaster Bujold – The Vorkosigan Saga is character-driven, swashbuckling space opera with great political intrigue.*
- *Ian Douglas – Star Carrier series delivers military-heavy action with a cosmic scope.*
- *Joel Shepherd – The Cassandra Kresnov novels have a cybernetic twist, though they're a bit more focused on special-ops drama.*

Summary:

If you love the grand, interstellar scale of Peter F. Hamilton's books and the long, connected story arcs found in Silver Ships and The Frontier's Saga, these authors deliver similar epic space-operatic adventures. Whether you lean more toward military sci-fi or more traditional space opera, there's sure to be a new universe here to explore!

Wow. Doug, thank you. It had not occurred to me to consult with our Silicon Oracle for its recommendations, given my own penchant for long-story-arc multi-novel series and expansive space operas. Our long-time Sci-Fi interested listeners will have immediately recognized that ChatGPT recommended Jack Campbell's "Lost Fleet" series that was one of my earliest favorites which I've read twice. And right under it was David Weber's "Honor Harrington" series which was another of my favorites – which I read at least twice. And, of course, James S.A. Corey's "The Expanse" series. There was even an honorable mention of John Scalzi's "Old Man's War" series. The good news is that I'm completely unfamiliar with any of those others, so I'm anxious to dig in and see about them.

It occurs to me that it's somewhat fitting that after finishing the first novel in Peter Hamilton's newest two-book series, I plowed into the research to understand how ChatGPT and similar Large Language Models operate. And having done so, that technology has just recommended how I might best resume my previously interrupted work and return to Science Fiction. I believe that's what's known as closure.

Bob

Hi Steve: I wanted to supply some feedback to your last show regarding auto updates of hardware. I don't agree with your comment that enterprise level network security appliances, firewalls, routers and switches should be set up with automatic updates. History has shown that automatic updates can cause devastating outages for businesses. I find it doubtful that ***you*** *would turn on automatic updates on any of* ***your*** *systems.*

Uh… Okay. He's got me there. I'm not at all certain that I would take my own advice in that regard. Bob continues:

Maybe the point here should be if a person's company does not have the staff, knowledge, experience or money to have test systems that can be used to install updates and confirm that they are working as expected, then ***and only then*** *using automatic updates makes sense since at least that way they would be protected from unpatched vulnerabilities. But again, they would probably be better served with a managed service partner taking care of their systems.*

Meaning that smaller enterprises should perhaps outsource the responsibility for managing the infrastructure which, on one hand they need, but which on the other hand they don't have the staff, focus or care to maintain for themselves. I think Bob makes a good point even though we have seen the MSP route go very wrong when the MSP's network was compromised, which allowed bad guys to get into the networks of all of their clients. Bob continues:

I retired from a multi-national transaction processing company. After a security breach they implemented tightened security procedures that I am surprised more companies don't. This company has more than 50,000 employees:

- *We used network segmentation, and the office network was not able to directly connect to the transaction processing network without going through a Bastion Server which was fortified, locked down, and had separate 2 factor authentication.*
- *All new servers had to have a defined owner contact and business unit owner.*

- *All firewall rules had to be justified and these rules needed to be reviewed by the business unit owner quarterly to ensure that the rules were still needed.*

- *All hardware and software had to be supported by the manufacturer.*

- *Patches needed to be installed within 2 weeks (sooner if the issue was critical), allowing time for testing, production beta testing, and full roll out. We had redundant data centers, so we'd first install into the production data center and if the updates caused issues we'd fail over to the unpatched backup systems.*

- *All software being run on any systems needed to be white listed. Any exceptions needed to be reviewed and approved.*

- *No personal devices could be used on any networks.*

I won't get into the DDOS and WEB APP firewalling we used.

My point is: "Security is tough" and employees hate it. I know, because they kept complaining to me how much harder their jobs were once we implemented [the clearly required] security measures. My comment back to them was they were being paid very well, and if we were breached they likely wouldn't have a job because clients would drop us and move to a competitor.

Love your show. / Happy New year / Bob

Bob's note is a perfect case in point for the tradeoff of convenience vs security. And you might imagine the sour comments of an employee who relocates from a company with very little security and lax useless controls, to one with strong and useful security. Such an employee might well be grousing about how they didn't need to do this or that at their last company.

And finally, our listener **Patrick Beemer** who runs a 15 year old Managed Service Provider (MSP) shares some background on SonicWALL:

Hey Steve, I'm listening to your commentary about SonicWALL exploits, and I wanted to provide some additional thoughts about why over 10% of the installed base is still vulnerable to an exploit from August 2024.

I run a 15-year-old Managed Service Provider and have been a SonicWALL partner from the beginning. SonicWALL firewalls were mandatory for all our clients. (We're slowly moving our clients away from "big iron" at the edge for reasons that aren't relevant to SonicWALL as a company or this message).

SonicWALL is a very popular firewall for small businesses and MSPs. These aren't large companies with IT departments but are typically orgs with 10 - 75 staff that rely on an MSP or maybe a solopreneur to support them.

Worse, many companies this size choose not to maintain a relationship with a support partner. These firewalls are just sitting there, waiting to be exploited. And there's A LOT of them.

Secondarily, Leo asked why SonicWALL doesn't just push the firmware updates. Two reasons. First is concern about impact, responsibility and liability. Sitting at the edge of a business a firewall with a bad update immediately becomes a hair-on-fire emergency. As an MSP, I wouldn't want SonicWALL pushing updates at my client's firewalls. That is not their job. The risk here for SonicWALL is too great.

The other reason is that SonicWALL sells features. And the feature that enables cloud-based, scheduled firmware updates costs extra. A cost many budget conscious businesses are unwilling to invest in. (we make it mandatory).

I hope that provides a little context about why this is still a thing...

Finally, I want to take a moment to thank you and Leo for the expert guidance I've received over the years. I've been following Leo since the 90's. I started using SheildsUp almost as soon as it came out, and have been following you both ever since. Though it wasn't until I got my CISSP in 2019 and needed a reliable source of CPEs that I started listening regularly. And I'm also a member of Club TWiT. The information you provide each week keeps me informed and makes my job easier. Thank you. Cheers, Patrick Beemer

I thought Patrick's information was great. At first I thought I had spotted a contradiction, since he noted the potentially catastrophic danger that automatic updates posed. Then he later noted that automatic updates were actually available for an extra fee. So which is it? They're either a danger or they're a benefit? But the way out of this conundrum is that SonicWALL makes their customers pay for the privilege of these automatic updates, doubtless on an ongoing subscription basis. And I'm sure that part of the agreement with SonicWALL is that keeping one's firewall updated is a good thing – thus the reason for offering the service in the first place – but if something happens as a result, we did the best we could and, after all, you paid for us to do that to you because it's what you wanted.

HOTP and TOTP

As I mentioned at the top, today's topic was inspired by feedback from one of our listeners.

Max Feinleib sent two notes two weeks apart. I collected his two questions, which I initially started out answering as part of our regular listener feedback. But as my answer's length grew I realized that not only had we somehow never – in any of our 1,007 prior episodes – talked in details about something that probably every one of us is using, but I believed that the details of the technology that's going on would be something everyone would enjoy thinking about.

So, to get us started, here are the two pieces of feedback that Max provided:

> *Hi Steve, I've been noticing lately that the 6-digit codes I get for 2-factor authentication almost always seem to include one or more repeated digits. Of course, you'd expect some repeated digits. Nearly 85% of 6-digit numbers have 6 unique digits. However, my sense is that there are more repeats than there should be. I see a lot of codes that only use 3 or 4 unique digits (like, say, 906090 or 131327). It feels like the codes are being biased toward numbers with repeating patterns to make them easier to type.*
>
> *Have you (or any other listeners) observed this? If 2FA codes are truly being dumbed down in this way, how much of a concern is that? (Maybe it's not a big deal because the 30-second rotation makes brute-forcing 2FA codes difficult enough.)*
>
> *To note: I use Cisco Duo for my personal accounts and Microsoft Authenticator for my work accounts. Both apps seem to give me these dumbed-down codes.*
>
> *Thanks! Max*

And then two weeks later:

> *Hi Steve, I wanted to follow up on this question. Over the past several weeks since I sent you this, I've continued to note my 2FA codes. I've continued to get much below 15% of my codes with 6 unique digits, and it's more common to have 2 repeats or a tripled digit. My mom has been doing the same, and she's only told me about two occasions when she got a code with 6 unique digits. So I still believe that 2FA codes are being dumbed down for easy typing. Would love to hear if you can find anything on this.*

Before we examine Max's observation, his question does point out that in none of our previous 1007 podcasts have we ever taken the time to examine exactly how and where these time-varying digits are generated. Since that bears upon Max's observation, as the saying goes, there's no time like the present.

But even more, this provides the perfect setup for one of our theoretically pure deep dives into fundamental computer architecture and technology. So let's all have some fun!

TOTP, which is the abbreviation for the algorithm that all time-based authentication uses, stands for "Time-Based One-Time Password" algorithm. It was standardized and specified in RFC 6238 back in 2011. It builds upon HOTP the "HMAC-Based One-Time Password" algorithm which was standardized and specified by RFC 4226 in 2005.

We positively know that these standards are what everyone **must** be uniformly using everywhere, otherwise there would not be, and could not be, the universal agreement we obviously all have about the proper 6-digit code at any point in time. So that's established. These are the governing standards and specifications. So this allows us to dispassionately examine those two RFCs to see what they say, knowing that they must be operative.

Of the two, the only one that matters is the earlier HOTP since it's the standard that's used to generate the digit sequence with TOTP just being used to feed a new time-based value into HOTP every 30 seconds.

HMAC (H. M. A. C.) stands for Hash-based Message Authentication Code where the HOTP standard uses the long-proven well known to be cryptographically secure HMAC-SHA-1 hash algorithm. As we've discussed many times on this podcast, any cryptographic hash function, such as SHA-1 in this case, takes an input plaintext of any length and "digests" it into a fixed-length output. That's all any hash function does. So we can imagine that we are wanting to somehow hash the current time of day and date to produce and then display a random'ish result. The problem is, if that's all we did, everyone's authenticator would be producing the same random'ish result at the same time. What we need to do is introduce the idea of a secret key so that we can create a collection of these time-varying random'ish outputs.

Once again, our cryptographic toolkit provides a perfect tool, known as the HMAC. The long-established and well-proven HMAC algorithm uses any given cryptographic hash at its heart, but it also adds the provision of a key. So it essentially takes an unkeyed and unkeyable generic hash function and turns it into a family of hash functions where the particular hash function performed is determined by the HMAC's key.

So now we have what we need. A remote server randomly generates a secret key to be used for authentication for a specific user. It converts that secret key into a QR code and presents it to the user as part of their identity sign-up. The user's authentication app scans the QR code to capture and retain that key. And the remote server stores that key with their account.

Subsequently, at any point in the future, with each endpoint having the same shared secret to key their respective HMAC functions, they're each able to "HMAC" the current time of day and date which will result in an identical output. And since the output will only be identical if both HMACs are identically keyed, this allows the re-authenticating user to prove that they still have the previously shared secret key without ever divulging what it is. And since this correct output is based upon the time of day and date with 30 seconds of granularity, anyone who might arrange to intercept or capture the authenticating conversation will not have obtained anything that they can use in the future... since they won't ever have the secret key. So we have an extremely elegant solution that is working well today. I wanted to first establish this foundation for those who may not have been with us from the start so that we're not missing any critical pieces for what comes next.

At the heart of every HMAC lies a hash function and in the case of the TOTP and HOTP functions, which were standardized back in 2005, that hash function is the vulnerable SHA-1. The SHA-1 hash takes whatever is fed into it and hashes that into a fixed-size 20-byte, 160-bit hash digest.

What we know about any cryptographically secure hash is that the bits produced by this hash are all – every one of them – completely pseudo-random. The SHA-1 hash has been in use for decades and its bits have never shown any discernible pattern that would weaken it. The only reason SHA-1 has been deprecated over time is that, these days, the world has much more processing power available for hacking and cracking than it once did. So we'd prefer to have more bits of hash output just for the sake of more is better and makes us feel more secure.

Consequently, the world has moved to the newer family of SHA-2 hashes, typically using SHA-256 to give us 256 bits or 32-bytes of hashed output.

Now, I can hear some of our more informed listeners grumble that this old SHA-1 hash **was** found to have some weaknesses. That's true. But none of those ever related to the use of the hash for the generation of high-quality pseudo-random data. There were some so-called pre-imaging attacks against SHA-1 where it was being used to generate a cryptographically secure signature for a document. We never want to be able to manipulate the input of a signature's hash so that we're able to design a modified document that winds up having the same hash – and thus signature – as a target document. That would completely break the guarantee that document signing provides. Over time, SHA-1 was found to have some weaknesses there.

As junior cryptographers, the important takeaway lesson for us is that just saying "SHA-1 is broken" is a simplification that is untrue. The "brokenness" of a cryptographic function almost always depends upon how that function is being used. And SHA-1 remains a perfectly good and cryptographically strong pseudo-random number generator. For that application it needs no upgrade or replacement. This is why the entire industry has remained standardized upon it, even today in 2025.

Okay, so with 30-second granularity, the UTC time – as in the current time and date – along with a secret key, is fed into this SHA-1 HMAC which converts it into a cryptographically strong pseudo-random set of 160 bits, which is 20 bytes.

So we have what is essentially 160 pseudo-random bits. This can be viewed as a single very very large decimal number ranging from 0 to 2 raised to the power of 160, which is:

1,461,501,637,330,902,918,203,684,832,716,283,019,655,932,542,976

A 49 digit decimal number. So now let's explore the various ways that we might go about converting this humongous 160-bit, or 49-decimal-digit, or 20-byte SHA-1 based HMAC output into those six digits that we want our fancy authenticator to produce.

Thinking of this as a very large and long binary number, let's first say that we wanted to extract digits only ranging from 0 to 7, which is to say any one of eight possible values. One approach would be to shift the entire large number three bits to the right. In binary math, shifting a binary number to the right divides its value by two and the bit that's shifted off the right end is the remainder of the division by two. So if we shift the large value three bits to the right, that divides it by eight – because it's divided by 2, three times – and the three bits that would be shifted off the right end would be the remainder of the division by eight. That would give us a binary number ranging from 0 to 7 and when converted to decimal, a single digit between 0 and 7.

So, by dividing the massive number by eight we're able to "extract" a digit ranging from 0 to 7. And we could do this again and again as many times as we need to extract as many digits from the large number as we need.

But we do not live in an octal world, presumably because we do not have eight fingers and toes. We have ten fingers and toes, so we count in decimal with a ten-digit alphabet ranging from 0 through 9. And it's a ten-digit alphabet that we need our TOTP and HOTP to produce.

And here's the coolest thing: Since our fingers- and toes- friendly authenticator wants to produce one-time passcodes containing all ten digits ranging from 0 to 9, instead of dividing the

very large number by 8, we divide it by 10.
Dividing any large number by ten will give us a remainder ranging from 0 to 9. The solution is clean, simple and elegant. If it had been left to me to design the digit extraction algorithm for the HOTP algorithm, I would have done exactly that. I would have simply successively performed a very long division of that very large 160-bit number – by 10 – taking the remainder from each division, which would have resulted in an extremely uniformly distributed digit ranging from 0 to 9. And that simple long division could have been repeated as many times as needed to successively extract as many pseudo-randomly determined digits as would be needed.

And if we generalize this a bit, just for the sake of cool math and theoretical computer science, what's so cool about this approach is that it's wonderfully generic. If the size of one's alphabet happens to be exactly some power of two, then dividing any binary number by that is as simple as shifting the binary bits of that number to the right and grabbing the bits that fall off the end. They form the choice for the item extracted from the larger number.

But having a practical alphabet size that's exactly some exact power of 2 would mostly be a coincidence. The usual case is that the size of the alphabet is whatever it is. If we want to extract decimal digits we divide by 10. But if we wanted to extract evenly distributed English alphabetic characters, we could perform long division by 26 then map the resulting remainder, which would range from 0 to 25 to the letters of the alphabet 'A' through 'Z'. Or if we wanted both upper and lower case alphabetic characters, we'd divide by 52 to get a remainder that could be mapped to both the lower and upper case alphabet. And if we wanted upper and lower case plus decimal digits, we'd divide by 62, and so on.

This is exactly what I did with the design of the Perfect Paper Passwords system which we talked about during Security Now! Episode #115 which Leo and I recorded on October 25th, 2007. The Perfect Paper Passwords system successively performs long division of a very very long number by the size of the alphabet the user wishes to use. This generates successive division remainders of exactly the alphabet size which is used to enumerate successive items in the alphabet.

So in the case of something like HOTP, this clean and simple approach of the long division of the entire 160-bit SHA-1 number by 10, would allow any number of digits to be extracted from that very long value to satisfy the need for a maximum quality pseudo-random decimal number having any number of digits.

But I said that's what **I** would have done if I'd been given this task. It's what I **did** do, back in 2007. But the group who designed the HOTP algorithm didn't ask me, and that's NOT what they chose to do two years earlier in 2005. Looking at what they choose to do makes me want to scratch my head. The only rationale I can come up with for what they designed – which, being kind, would be termed "ad hoc" – was that it was good enough and that perhaps they didn't trust coders who would be implementing their standard to be able to divide a long binary number by 10. So they went out of their way to avoid that.

I wanted to first explain, as I have, the cryptographically optimal way of solving this simple problem of computer science so that everyone would have a reference point against which to judge what actually happened. So get a load of the universal HOTP algorithm that we all wound up with, for better or worse:

Once again we start with the output of the SHA-1-based HMAC. But this time, rather than viewing it as a large (and apparently intimidating) 160-bit binary number, we view it as an array of 20, 8-bit bytes. The bytes of this array would be numbered 0 through 19.

The officially standardized HOTP algorithm instructs us to take the last byte of the array, byte number 19, and mask off or ignore the upper four bits of that last byte. Thus, we'll pay attention to only the lower four bits. These four bits will thus have a binary value ranging from 0 to 15. So we use that 0 to 15 value as an offset into the entire 20-byte array, where, starting at that offset, we take four successive bytes – thus 32 bits. So, if after masking off the upper four bits of the last byte and retaining only the lower four, we wound up with a 0, we would obtain the four-byte, 32-bit value from bytes 0, 1, 2 and 3 of the array. And at the other end of the range, if the last four bits had their maximum value of 15, we would obtain the four-byte, 32-bit value using bytes 15, 16, 17 and 18.

Okay. So this kludge (which appears to be my word of the day) results in us having extracted 32-bits somewhere from within the first 19 bytes of the 20-byte SHA-1 hash value, where the lowest four bits determine where within those 19 bytes we grab 32 bits.

Next, believe it or not, the implementer is then instructed to set the most significant bit of those 32 bits to zero. This creates a 32-bit value which, if it were to be treated as a signed integer, would be guaranteed to be positive because signed integers use the high bit as their sign where that high bit set to '1' means that the number is negative.

So now we have what is, essentially, a very tame 31-bit positive number ranging between 0 and 2,147,483,648 which fits handily into a CPU's 32-bit register or the integer of pretty much any higher-level computer language. This makes division as simple as a single machine instruction. So the HOTP algorithm next instructs us to divide that 32-bit, guaranteed to be positive integer, by one million because the remainder of that division, when converted into a decimal number, will give us all possible 6-digit numbers from 000,000 to 999,999.

Does it work? Yes. And what it sacrifices in elegance – which is to say pretty much everything – it doubtless gains in ease of proper implementation using any high level language. I'm sure anyone could write it in BASIC and obtain the correct answer. It would be very difficult to screw that up. And since interoperability among all HOTP generators, all arriving at the same 6 digits, is paramount, I guess I can see why the designers chose the kindergarten design that they did.

Now you might ask "Kindergarten? Really? Isn't that being too critical?" Well, let's look at it. From a cryptographic standpoint the algorithm itself is really quite crappy because very little of the SHA-1 hash's entropy winds up being used. The last byte's top four bits are completely ignored. And the lower four bits select just four out of the remaining 19 bytes – completely ignoring all of the other 15 (which is 120 bits ignored out of the total 160). Then, adding insult to injury, of the precious 32 bits that **were** selected, one of those is discarded because whomever is implementing this might not know how to perform unsigned division. So the dividend is forced to be positive just to be sure. So we wind up using the entropy contained within just 31 bits of the HMAC function.

By comparison, my approach of successively taking the entire 160-bit hash output, dividing it by 10 and using the remainder, takes advantage of every one of the available bits of the HMAC output for the determination of each successive decimal digit.

But I will also be the first to conceded that interoperability of implementation matters here, far more than cryptographic perfection. Dividing the extracted 31 bit value by one million to obtain a value ranging from 0 to 999,999 will absolutely provide a completely useful and highly pseudo-random result. One of the features of a high-quality cryptographic hash function such as SHA-1 is that every single bit of its result has an exactly even, 50/50 chance of being a 0 or a 1. So taking any sufficiently large set of them and dividing by one million will give us a good result.

However, if our priority – as it appears to be – is to create a super-simple, easy to implement and highly interoperable solution, then why all the low 4-bit nibble nonsense to select the set of four bytes to use? As we all know, the definition of any cryptographically strong hash function, which lies at the heart of the HMAC, is that every single one of its many bits are treated equally. They each have that algorithmically guaranteed 50/50 chance of being either a 0 or a 1, so if we're going to go the route of using a 32-bit positive integer as our dividend, it absolutely and truly doesn't matter **at all** which 31 bits out of the SHA-1 hash's 160 bits we select to be the dividend for our division by one million. In fact, it CANNOT matter or we don't have a truly strong cryptographic hash function to begin with.

This means that an exactly equivalently strong HOTP algorithm could have told us to just take the first four bytes of the SHA-1 HMAC output, set the high bit to 0, divide the result by one million, convert the remainder from the division to an unsigned decimal number... and we're done. Period. So why all the additional, totally pointless, rigmarole?

All of the additional nonsense of using the last four bits to select which bytes to use is worrisome since it begs the question: Why? Did the designers of HOTP not understand how hash-based HMAC functions operate? If so, while it doesn't matter, it does mean that we're now stuck with a needlessly complex system that provides zero additional security over a simpler alternative.

The only additional observation I'll make is that it is only when the dividend is an exact even multiple of the divisor that we obtain a truly evenly distributed remainder. And the corollary to that is that the larger the dividend is than its divisor the more evenly distributed are the values of the remainder. More than anything else, this is why I prefer my approach: Because it uses the largest possible value for the dividend.

What am I talking about? Let's use a super simple example to clarify the point. Say that we want to extract a decimal digit from a 4-bit source. We know that we can do that by dividing the source dividend by 10. So now let's look at the result we obtain from all 16 of the source's possible values:

0 divided by 10 gives us a remainder of 0. 1 divided by 10 results in 0 with a remainder of 1. 2 divided by 10 is 0 with a remainder of 2. And so on upward where 9 divided by 10 is 0 with a remainder of 9. Next, 10 divided by 10 will be 1 with a remainder of 0, 11 divided by 10 will be 1 with a remainder of 1, and so on up to 15 – the maximum value of 4 binary bits. Dividing 15 by 10 gives us 1 with a remainder of 5.

So look at what happened: We were asking our 4-bit source to give us 10 possible output values. But because 4 bits has 16 values, it cannot be evenly mapped into 10 results. So, taking the remainder of the divisions by all possible source values, we wind up with two instances of remainders of 0, 1, 2, 3, 4 and 5, but only single instances of remainders 6, 7, 8 and 9. In other words, we do not wind up with a perfectly even distribution of all possible output values.

Our HOTP algorithm divides a 31-bit dividend – having 2,147,483,648 possible values – by one million. And since that total number of possible input values is not evenly divisible by one million, this means that not all possible 6 decimal digit values produced by the industry standard HOTP algorithm will occur with exactly the same frequency.

Now, in practical terms am I splitting hairs here? Definitely. It absolutely doesn't matter at all. It won't result in the final decimal output, which will change again in 30 seconds anyway, being usefully any more guessable. The case of generating 10 values from 16 was so horrible only because 16 was so very close to 10. By comparison HOTP's dividend is 2.147 billion which is much much larger than the one million. In fact, it's more than 2,147 times larger than 1 million.

But, that said, computer science is computer science, and all of this makes for intriguing questions. If nothing else, these questions must be examined if only to be able to judge their size and impact and to make certain that their effects will be negligible.

From me, you will only ever get long division of all possibly available bits of entropy, not because it necessarily matters to you, but because it's the most correct solution, which means that it matters to me.

Okay. So now let's return all the way back to Max's original question of any perceivable bias in the resulting numbers that might cause more "identical digits" than we would expect. Knowing what we now know, is that possible?

No.

It's not possible.

We've examined the algorithm. At its heart it takes a sufficiently large, entirely pseudo-random binary value from which we take one of 2,147,483,648 values and divide that number by one million. The dividend of the division, while not an even multiple of the divisor, is enough larger than the divisor that the remainder of that division will be an extremely evenly distributed value ranging between 0 and 999,999. And that, in turn means that when converted into a decimal number, the value's individual constituent digits, will also be extremely evenly distributed without any interaction or relationship to one another.

I, too, have observed the same illusion that Max and his mom have. But I'm certain that this must be classic observational bias, where we tend to notice much less all the times when the digits do **not** form any sort of pattern and tend to notice more those times when they **do**.

But that aside, it is provable – and we've just proved it – that there cannot be any non-uniform pattern and we know that all authenticators must be using the same algorithm which we've just examined, otherwise they would not be producing the expected result.

I asked ChatGPT: *"Is there a term for the tendency of we lowly humans to perceive a pattern where none actually exists?"*

And it replied: *"Yes. The general term for this is apophenia—the tendency to perceive meaningful connections or patterns between unrelated things. A more specific example of this phenomenon, limited primarily to visual or auditory stimuli (like seeing faces in clouds or hearing hidden messages in music), is called pareidolia."*

One thing I **am** quite certain of, Leo ... is that there is definitely a pattern to these podcasts. They routinely appear every Tuesday, come rain or shine. So everyone should expect another one next week!