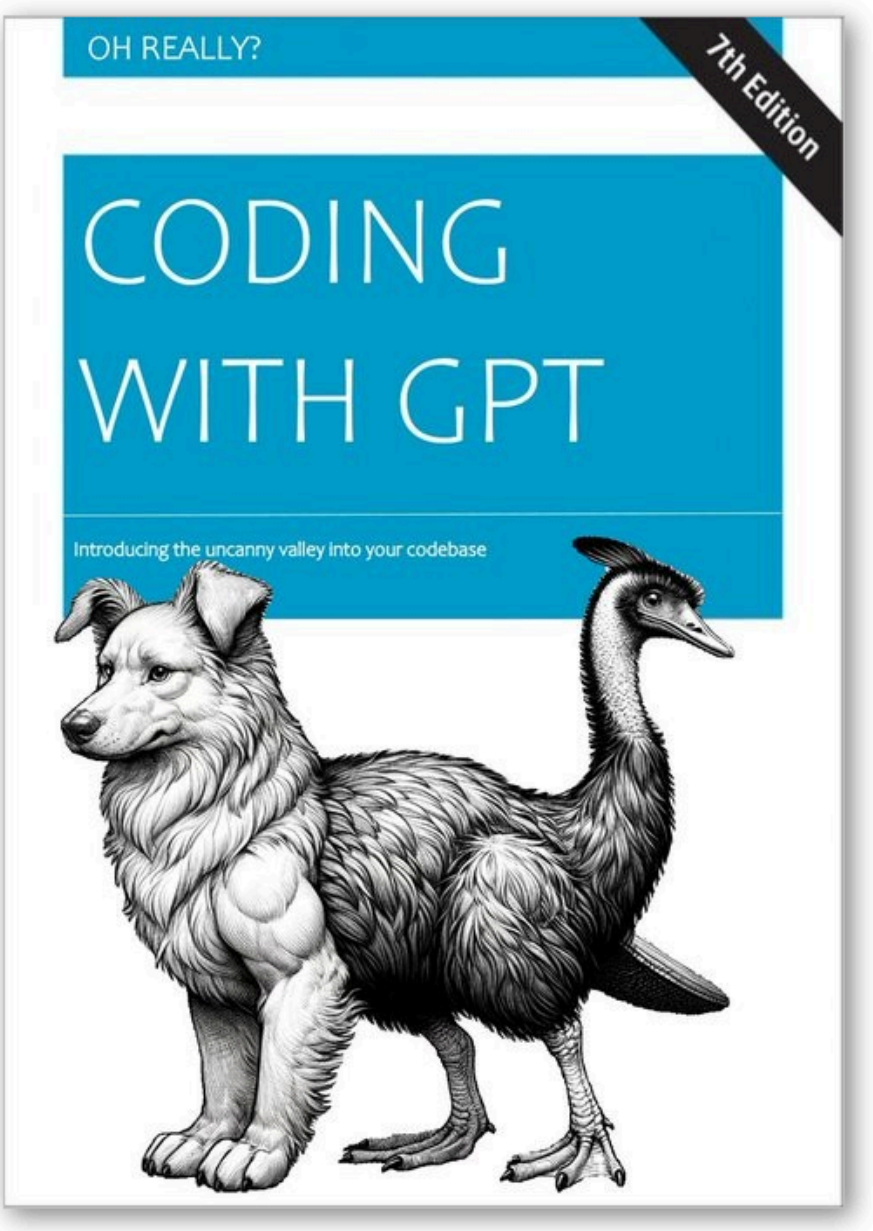
Security Now! #964 - 03-05-24

PQ3

This week on Security Now!

Last week we covered a large amount of security news; this week, not so much. There are security stories I'll be catching us up with next week, but after sharing a wonderful piece of writing about the fate of Voyager 1, news of an attractive new Humble Bundle, a tip of the week from a listener, a bit of SpinRite news and a number of interesting discussions resulting from feedback from our listeners, our promised coverage of Apple's new "PQ3" post-quantum safe iMessaging protocol consumed the entire balance of this week's podcast budget, bulging today's show notes to a corpulent 21 pages. I think everyone's going to have a good time.



Miscellany

Voyager 1

We've had some fun keeping an eye on Voyager and on the occasion that we may have finally lost control of this intrepid explorer, I found a wonderful piece of writing about this that I know our listeners will enjoy and appreciate. It's a blog post by Doug Muir titled "Death, Lonely Death."

Billions of miles away at the edge of the Solar System, Voyager 1 has gone mad and has begun to die.

*Let's start with the "billions of miles". Voyager 1 was launched in early September 1977. Jimmy Carter was a hopeful new President. Yugoslavia and the USSR were going concerns, as were American Motors, Pan Am, F.W. Woolworth, Fotomat booths, Borders bookshops, and Pier 1. Americans were watching Happy Days, M*A*S*H and Charlie's Angels on television; their British cousins were watching George and Mildred, The Goodies, and Tom Baker as the Fourth Doctor. If you turned on the radio, "Hotel California" by The Eagles was alternating with "Dancing Queen" by Abba (and, if we want to be completely honest, "Car Wash" by Rose Royce). Most cars still ran on leaded gasoline, most phones were still rotary dial, and the Internet was a wonky idea that was still a few weeks from a working prototype.*

"The Thorn Birds" was on top of everyone's bestseller list. The first Apple II home computer had just gone on sale. The Sex Pistols were in the studio wrapping up "Never Mind The Bollocks"; they would tour on it for just three months and then break up, and within another year Sid Vicious would be dead of a heroin overdose. Barack Obama was a high school junior living with his grandparents in Honolulu, Hawaii: his grades were okay, but he spent most of his time hanging with his pot-smoking friends in the "Choom Gang". Boris Johnson was tucked away at the elite Ashdown House boarding school while his parents marriage was slowly collapsing: although he was only thirteen, he had already adopted his signature hair style. Elvis had just died on the toilet a few weeks ago. It was the summer of Star Wars.

And Voyager 1 was blasting off for a tour of the Solar System.

There's no way to pack the whole story of Voyager 1 into a single blog post. Here's the TLDR: Voyager was the second spacecraft to fly past Jupiter, and the first to take close-up photos of Jupiter's moons. It flew on past Saturn, and examined Saturn's moon Titan, the only moon with an atmosphere. And then it flew onwards, on and on, for another forty years. It officially left the Solar System and entered interstellar space in 2012. It just kept going, further and further into the infinite emptiness.

(You know about the Golden Record? Come on, everybody knows about the Golden Record. It's kind of hokey and cheesy and also kind of amazing and great.)

Voyager has grown old. It was never designed for this! Its original mission was supposed to last a bit over three years. Voyager has turned out to be much tougher than anyone ever imagined, but time gets us all. Its power source is a generator full of radioactive isotopes, and those are gradually decaying into inert lead. Year by year, the energy declines, the power levels relentlessly fall. Year by year, NASA has been switching off Voyager's instruments to conserve that dwindling flicker. They turned off its internal heater a few years ago, and they thought that might be the end. But those 1970s engineers built to last, and the circuitry and the valves kept working even as the temperature dropped down, down, colder than dry ice,

colder than liquid nitrogen, falling towards absolute zero.

(Voyager stored its internal data on a digital tape recorder. Yes, a tape recorder, storing information on magnetic tape. It wasn't designed to function at a hundred degrees below zero. It wasn't designed to work for decades, winding and rewinding, endlessly re-writing data. But it did.)

Voyager kept going, and kept going, until it was over 15 billion kilometers away. At the speed of light, the Moon is one and a half seconds away. The Sun is about 8 minutes away. Voyager is twenty-two hours away. Send a radio signal to it at lunch on Monday, and you'll get a response back Wednesday morning.

I could go on at great length about Voyager — the discoveries it has made, the Deep Space Network that has maintained contact over the decades, the ever shrinking crew of aging technicians keeping it alive on a shoestring budget, how amazing it has all been.

In 1990, just before Voyager's camera shut down forever, the probe turned around and looked backwards. It zoomed in and took a picture of Earth. But by that time, it was so far away that Earth was just a single pale blue pixel.

Seeing that blue pixel, Carl Sagan wrote: "That's here. That's home. That's us. On it everyone you love, everyone you know, everyone you ever heard of, every human being who ever was, lived out their lives. The aggregate of our joy and suffering, thousands of confident religions, ideologies, and economic doctrines, every hunter and forager, every hero and coward, every creator and destroyer of civilization, every king and peasant, every young couple in love, every mother and father, hopeful child, inventor and explorer, every teacher of morals, every corrupt politician, every "superstar," every "supreme leader," every saint and sinner in the history of our species lived there – on a mote of dust suspended in a sunbeam."

Voyager kept going for another 34 years after that photo. It's still going. It has left the grip of the Sun's gravity, so it's going to fall outward forever.

Here's a bit of trivia: Voyager 1 currently holds the record for most distant active spacecraft. It's not even close. The only other contender is Voyager's little sister, Voyager 2, which had a different mission profile and so lags billions of kilometers behind their older sibling.

Here's another bit of trivia: if you're reading this in 2024? It's very unlikely that you will live to see that record broken. There are only two other spacecraft outside the Solar System — Voyager 2 and New Horizons. Both of them are going to die before they get as far as Voyager 1. And nobody — not NASA, not the Chinese, not the EU — is currently planning to launch another spacecraft to those distances. In theory we could. In practice, we have other priorities.

We thought we knew how Voyager would end. The power would gradually, inevitably, run down. The instruments would shut off, one by one. The signal would get fainter. Eventually either the last instrument would fail for lack of power, or the signal would be lost.

We didn't expect that it would go mad.

In December 2023, Voyager started sending back gibberish instead of data. A software glitch, though perhaps caused by an underlying hardware problem; a cosmic ray strike, or a side effect of the low temperatures, or just aging equipment randomly causing some bits to flip.

The problem was, the gibberish was coming from the flight direction software — something like an operating system. And no copy of that operating system remained in existence on Earth.

(This is a problem NASA long since solved. These days, every space probe that launches, leaves a perfect duplicate back on Earth. Remember in "The Martian", how they had another copy of Pathfinder sitting under a tarp in a warehouse? That's accurate. It's been standard practice for 30 years. But back in 1977, nobody had thought of that yet.)

Voyager Mission Control used to be a couple of big rooms full of busy people, computers, giant screens. Now it's a single room in a small office building in the San Gabriel Valley, in between a dog training school and a McDonalds. The Mission Control team is a handful of people, none of them young, several well past retirement age.

And they're trying to fix the problem. But right now, it doesn't look good. You cannot just download a new OS from 15 billion kilometers away. They would have to figure out the problem, figure out if a workaround is possible, and then apply it… all with a round-trip time of 45 hours for every communication with a probe that is flying away from us at a million miles a day. They're trying, but nobody likes their odds.

So at some point — not tomorrow, not next week, but at some point in the next few months — they'll probably have to admit defeat. And then they'll declare Voyager 1 officially over, dead and done, the end of a long song.

A Cory Doctorow Humble Book Bundle

https://www.humblebundle.com/books/cory-doctorow-novel-collection-tor-books-books

The description reads:

Doctorow's visions of the future: Lose yourself in the visionary fiction of Cory Doctorow, the celebrated author and digital rights activist known for his masterful explorations of the intersection of tech and society. Little Brother is a stark exploration of surveillance and authoritarianism in the backdrop of a major terrorist attack on San Francisco. Radicalized features four distinct sci-fi novellas, each telling a gripping story inspired by today's technologies and societal trends. In Red Team Blues, you'll follow along with a well-connected money laundering expert on the most dangerous and exciting gig he's ever taken on. Get all these and more—18 books in total—and help support the Electronic Frontier Foundation with your purchase.

A VERY COOL Tip of the Week!

Patrick Johnson / @PSUnconquered

Hi Steve, Long time listener (and when downloading 6.1 I discovered that I've been a SpinRite owner for 10 years as of Tuesday). I was surprised to hear in SN 963 that Firefox had let you keep the dedicated search box for so long. I forget when it was turned off by default for new installs, but I did keep turning it back on for quite a while ***until I discovered that the CTRL+K shortcut for search still worked,*** *treating the text entered in the omnibar as a strict search query, even for terms that look like a URL, no quotes needed. As a .NET developer, libraries and frameworks with the _____.NET naming convention turn up quite a bit in my searches. Chromium seems to have copied this, so that now Brave, Edge, and Chrome all behave like Firefox with CTRL-K forcing a search. Thanks for everything you do, Patrick*

Thank you, Patrick! This was news to me, so I tried it and it works perfectly. I previously shared my discovery, which I now use all the time, that CTRL-L highlights and selects the contents of the omnibar. And now I have CTRL-K to use when I want there to be no confusion over whether I want to go somewhere or look something up.

SpinRite

I have no big SpinRite news – which at this stage is what we hope for. Everything continues to go well. I'm pushing forward on several fronts. I'm spending time in GRC's forums watching as new people encounter SpinRite 6.1 for the first time. So I'm learning about their experiences which will be informing SpinRite's forthcoming FAQ. One thing I've seen is that Linux users need SpinRite in a format that's directly usable to them. Requiring non-Windows users to briefly use someone else's Windows system to have SpinRite create a bootable USB drive for them is something I always planned to work around. So I've been working on the tech to add direct bootable image downloading to GRC's servers so non-Windows users will be able to obtain a bootable drive image they can copy directly to a USB drive to boot their licensed copies of SpinRite.

I also have the beginnings of GRC's forthcoming eMail list system running, but not yet fully configured. I plan to maintain two lists, one specifically for weekly podcast announcements and another for GRC-related news of new software, updates, features and so forth. So people will be able to join either or both as they choose. And I'm also working on SpinRite's documentation, which is coming along nicely,

Having learned from Microsoft last week that certificate reputation matters, even though they are deliberately mute about how exactly that reputation is earned, I presume it's a function of the exposure of a new certificate in the signing of non-malign software. So after last week's podcast I used this relatively new certificate to co-sign GRC's top six most downloaded Windows freeware. In order of decreasing average daily download rates, the top six are: ValiDrive, the DNB Benchmark, InSpectre, Securable, InControl and Bootable. What surprised me was that for the first time ValiDrive has taken the top slot to become GRC's most downloaded freeware with nearly 2200 downloads per day. Anyway, taken together, those top six are being downloaded 5323 times per day. So I expect that new certificate to become golden before long, if it's not already.

Closing the Loop

Tom Desmond / @tadesmond

Hey, Steve - I'm just finishing listening to episode 963 and I think I'm missing something on the cookie email link login loop. The "absolutely secure" process seems to assume that no bad person has compromised the email. Think about this... Someone gains access to the email from wherever. They see an old email loop link. They go to the site and guess the username (maybe it is the same email address) - they get the link in the compromised email and they are in - Am I missing something?

I hate it when I miss something that's so obvious. Tom is, of course, completely correct. The solution I described last week, of embedding the eMail link requestor's IP address and browser cookie into the link would absolutely and strongly prevent anyone who did not request the eMail-based login link from using it, if they were to just passively observe it in flight or at rest. That's nice. But as Tom points out, that doesn't solve the problem that we're still just as dependent upon eMail security as ever. Nothing prevents an attacker, who has the ability to monitor the user's eMail account, from going to the site themselves, requesting a link from **their IP** with **their browser**, then clicking the link that arrives in the compromised user's eMail. Whoops!

This means that while there's no reason not to at least embed the requestor's browser cookie in the link, we don't really gain anything from doing that and we remain dependent upon the security of eMail for all of our login security, whether or not it's passwordless using eMail or passworded with the ubiquitous "I forgot my password" total security bypass. Thank you, Tom!

Alex Neihaus ☁️ and fediverse @alex@air11.social / @yobyot

Re: email/password links in SN! 962: If you send a link with a hash/IP address/etc, you eliminate the ability to copy the link and open it in another browser. It would also _encourage_ clicking on links in email, something enterprises are trying hard to un-train users from doing. Also, if you did get such a link that you opened in Incognito mode and the server expected a persistent session cookie, next time after restarting the browser you asked for the link in either normal mode or Incognito, it wouldn't exist. Email links instead of passwords are a good idea -- and can be made secure, as you describe. But it doesn't fit with the way many people interact with browsers, esp. those who use multiple browsers.

Alex's point is a good one. Heavy use of eMail links for common logon would have the effect of "untraining" users against clicking on links in eMail. If the world were to switch over to that, there would certainly be a lot more eMail link clicking going on. And it's also true that the tighter we make the anti-spoofing security the greater the chance for rejecting a valid user. The presumption is that the user wishes to logon now, not later. So they would request a link then immediately go to their eMail to find and execute the link, which would take them back to where they just were. But Alex is right that there are various things that could go wrong. So, as we so often encounter, everything is a tradeoff. The great benefit of eMail-link logon is that the user needs to remember and possess nothing and they are able to login from anywhere, as long as their eMail is secure and they have access to it.

But I think the best thing to come from the last few weeks of this discussion has been the

recognition that passwords really only amount to logon accelerators, and that as long as every username and password logon opportunity is accompanied by a "The dog ate my password!" bypass, nothing else we do to further increase logon security matters – not hardware dongles, not fingerprints, not one time passwords, nothing. It's all just "security theater." This, in turn, implies that the security of our eMail is far more important than is commonly appreciated.

And another listener sent this bit of fun. He wrote:

The Taco Bell app seems to have gone passwordless in favor of emailing you a link whenever you try to login in. It's not great UX for me in that it's slow and prone to spam filtering. When you're trying to put your order in from the car on your way to the Bell, it's enough to make you drive to Chipotle instead.

Joel Clermont / @jclermont

There is a security-focused reason to enforce a max password length when hashing with bcrypt. I recently discovered the recommendation and wrote up more details here https://masteringlaravel.io/daily/2024-02-05-why-does-laravel-offer-a-max-password-length-validation-rule

I followed Joel's link to his write-up at "masteringlaravel.io", which was interesting. It turns out that Laravel uses Bcrypt as its PBKDF password hashing algorithm, and for reasons that defy understanding, the Bcrypt algorithm has a hard and fixed internal password length limit of 72 bytes. Now, normally we'd think that 72 bytes was way more than anyone might need. But unfortunately, many characters in non-Latin alphabets are encoded in 3 and even 4 bytes. So it would be possible for a non-Latin password to max-out Bcrypt's fixed 72 byte limit with just 18 characters.

So Joel makes a good point about there being possible exceptions to the boundless upper password length presumption. I'll just note that a simple solution to this problem for anyone who might be stuck using Bcrypt would be to first run the user's unlimited length password through an SHA256 or even an SHA512 hash – just once. SHA512 will take a password of any length, even a long one using non-Latin characters, and reduce it to 64 bytes, which can then be sent into Bcrypt to be strengthened against brute-force guessing.

Earl Rodd / @nc360370

Regarding the discussion of Nevada's request for an injunction against META encrypting messages for minors in SN 963: As I listened, I was aware that the great missing piece is DATA. All of those quoted on both sides, privacy advocates and law enforcement advocates, have lots of opinions – but they present no data. Data, like the number of times criminals were believed to go free because of encrypted messages stopping prosecution, or the number of such crimes not prosecuted at all due to encryption; or data like times that using unencrypted messages was known to lead to harm to minors. Is this really a case of balancing privacy, i.e. being pummeled with advertising, versus throttling the ability of law enforcement to find and prosecute serious crime? I suspect it's not that simple. But without data, who knows the actual trade-off?

I think Earl makes ***such*** a good point! We've been driven so far away from actual data collection that we appear to have forgotten that actual data ***could*** be collected and made available. We've become "the anecdotal example" culture, being slammed from one extreme to the other. And here's the problem: Expressing a strong opinion is easy; it excites; it's junk food for the mind. But actually collecting, tabulating and evaluating data is difficult time-consuming work. And the biggest problem is, as Earl suggests, the result of all that actual work probably doesn't support the extreme views of either side. So it's not in either side's best interest to be presented with any of those pesky facts which would likely lead to some form of compromise policy. Better just to wave our arms around, attempting to terrify everyone and jam through regulations that support an ideology rather than reality. It seems that too much of the world is working this way these days.

We've seen countless examples of how the move to an online digital world has very likely provided law enforcement agencies with a treasure trove of readily accessible, indexable and searchable information the likes of which has never been known before. We've recently learned that these agencies are consumers of the information that commercial data brokers have been collecting about us for years. Today, everyone leaves footprints and bread crumbs wherever they go online and whatever they do. If law enforcement knew what all of our various service providers know, and what our ISPs know about where we wander on the Internet – and we should assume that they **do** know all of that when they want to – then law enforcement knows pretty much everything about us; certainly more than was never knowable before the Internet.

As a law abiding citizen of the United States, which for the most part leaves its citizens alone unless some intervention is required, I'm all for law enforcement working behind the scenes to keep a lid on criminal behavior. So, given the unparalleled tools that are now available to aid in that work, it's difficult to get too worked up over the encryption debate. Law enforcement authorities can even know who we're talking to if they wish, even if it's much more difficult to peer into the content of those conversations. Although I, also, do not have any data to back up my option, my intuition is that we're already sitting with a workable compromise for both sides.

Could the privacy absolutists wish for more privacy? Absolutely. Are they going to get much more? Probably not. Do our law enforcement agencies wish they could also listen in on anyone's conversations? I'm sure they do. Are they going to be able to do that? Let's hope not, because that would clearly be a step too far in the "Big Brother" direction. And, again, they must already have access to far more data about us than they know what to do with. It seems to me that the right balance has already been struck.

Elliot.Alderson / @ElliotAlders369

Hi Steve, the issue with passwordless login the way you described is if someone only has email on their phone. I don't have email on my computers because I don't need it. If the IP and everything is baked into the link, I'm SOL. That also makes it very difficult to login to other browsers.

Having listened to everyone's thoughts about this, it's clear that the potential automation offered by an eMail link creates more problems than it solves. So the solution for eMail-only login is to

return to a visible easy-to-transcribe one time 6-digit token. Just eMail the token to the user and request that they enter the token into the browser they're currently using. If they receive the eMail on their phone they have the advantage of seeing both the token and their browser page at the same time. This eliminates the "untraining" about never clicking on a link and it's not really much more work for the user. They still don't need to remember anything!

SecFan / @SecFan9669

Steve: In SN963 yesterday, a listener had mentioned the idea of having a standardized way of documenting a site's password requirements so that they could be automated. In writing my own algorithmic password manager, named Passify (that I messaged you about previously) I had the same thought and wanted Passify's algorithm to accommodate known requirements whenever possible. I discovered that Apple had put some thought into this and created "Password Manager Resources" on Github:
https://github.com/apple/password-manager-resources

It includes a JSON format for documenting password rules as well as a number of other helpful things for Password Managers to automate or semi-automate password management. The idea isn't completely new -- Apple's Safari browser has supported a "passwordrules" html attribute that uses the same rule markup as the JSON to describe requirements for a while:
https://developer.apple.com/password-rules/

Thanks again for all your work. Looking forward to the official release of the new SpinRite!

It's cool that Apple has already done the work of creating some of this structure and my comment last week about it being a heavy lift to get industry adoption appears pertinent since this work is now 4 years old and it has remained somewhat obscure. Of course, if any website wanted to modernize, they could simply place a relatively high lower-limit on password length and accept passwords of any greater length containing any characters. And also, on the server-side, they would prevent endless brute force password guessing.

Rob

Security Now question: After hearing your discussion of session cookies I was reminded of my own security concern I've had for years... Seems I'm always logged into my Google Account based on session cookies, and so it seems if a hacker could grab those cookies from my computer, they'd now have access to all my emails and many other Google things. This, even with Google's Advanced Protection Program. I believe I've even migrated to a new MacBook and was logged in fine on the new MacBook (though my memory could be off on that). Didn't seem there was any other verifying that my hardware was the same, etc. Just doesn't feel very secure to me. Am I missing something?

So, a bit of clarification first. When a Cookie is set in a browser the browser can be told how long to retain that cookie in the form of either an expiration date or a maximum age. If either of those are provided, then the cookie is considered to be "persistent" and the browser will retain and return that cookie's value until it is refreshed, deleted by the user or the website, or the cookie's end-of-life is reached and it self-expires. Any cookies that have not expired will be retained from one browser session to the next.

But I mentioned that the specification of these expirations was optional. If a cookie is set without any explicit expiration **then** it is considered to be a session cookie because it will be retained only for the current browser session. It is explicitly retained only in RAM and it is never written to any permanent storage. Once the browser closes the browser session ends and the cookie will be forgotten.

So to Rob's question and uneasiness: It is the case that our browser's persistent cookies being retained over the long-term is what keeps us logged into our various websites persistently from one boot up of our computers to the next. And it is definitely the case that if bad guys could arrange to obtain those persistent login cookies they could immediately impersonate us.

In fact, this is exactly what the FireSheep Firefox add-on did many years ago. And this was one of the primary motivators for the move to always-present HTTPS. FireSheep relied upon the mistake that most websites of the era were making: After authenticating with a username and password whose communication was protected by HTTPS, most popular sites would switch back to plain cleartext HTTP... figuring, wrongly, that the user's identity had been protected. They failed to take into account that the only way that user was remaining logged on from our HTTP page query to the next was that each browser query was accompanied by cookies... but without the encryption protection provided by HTTPS, those cookies were being sent in the clear so that anyone who could eavesdrop could obtain them and immediately impersonate the user, obtaining parallel access to their web session.

So yes, Rob, our browser cookies do serve as persistent long-term authentication tokens and anyone who might have some means for obtaining them could, indeed, impersonate their owner.

Sometimes when logging into a site you'll see a "Trust this browser and remain logged in?" checkbox. This is useful when you do not want to leave a persistent cookie behind in that browser, for example when logging into a shared Internet café machine. You'll also want to carefully and explicitly log out of that machine once you're finished. But that "Trust this browser and remain logged in?" question is offering to change the logged on authentication cookie from a temporary session cookie to a long-term persistent cookie.

Andrew

Hi Steve, Listening to SN 962 and wanted to provide a bit of additional information regarding your discussion on Password requirements. I work for a company in the financial services sector, specifically insurance, but I would prefer to remain somewhat anonymous. I did want to note that many companies in this sector STILL rely on mainframes, and that is likely the cause for many of these maximum password length limits and character restrictions due to IBM's fanatical backwards compatibility.

Several years ago, I interned for another company in this sector and was SHOCKED at the requirements for passwords. 7-8 characters ONLY, and the only symbols allowed were the @ and $ characters. How much of this is who's fault is a bit opaque to me even still, but I can say that it isn't entirely the mainframe's fault. I've seen that more 'modern' implementations allow up to a 40 character 'password phrase' but still restrict a few characters. All that being said, you see these kinds of restrictions OFTEN in sectors that continue to rely on the mainframe,

and other legacy platforms and products.

I appreciate the feedback that we still haven't moved very far. One of the most significant things we've seen and learned through the years of this podcast is the power and prevalence of inertia that acts to keep things from changing.

One trick that I ought to mention that works quite well, is to map a hashed value to some fixed alphabet. Say that a back-end system can only accept 8 characters of upper and lower case alpha, numbers, and Andrew's at sign (@) and dollar sign ($). So that's 26 lowercase, 26 uppercase, 10 digits, and two other characters for a total alphabet size of 64.

Now first of all, the fact that the alphabet size turned out to be 64 – representable by exactly 6 binary bits – should jump out at everyone. So let's first take this case. We want a web front end to give its users unrestricted passwords. So it hashes whatever they provide, preferably using a contemporary PBKDF function like Argon2. The output will be 256 evenly distributed bits. So they are simply taken from either end, 6 bits at a time with each of those 6 bits mapped into one of those 64 characters. After eight of them have been taken, those 8 characters are passed back to the creaky old mainframe as the user's password. The result is that the user is able to use whatever crazy long and special-character filled password they choose, and whatever they choose is first strengthened by a modern PBKDF to harden it against brute force attack and then convert into a high-entropy 8 character password which meets the needs of the back end system.

But what if the back-end system's password alphabet wasn't conveniently 64 characters?

Consider this: One way to visualize taking 6 bits at a time is dividing the large hash value by 64. In a binary representation, division by 2 is a right shift of the bits. So dividing by 64 is shifting right by six bits. We can think of the bits that are shifted off to the right as the remainder of the division. So to extract a single character from an alphabet of 64, we are actually dividing the large binary hash value by 64 and taking the remainder, which will range from 0 to 63.

So this means that we can extract characters of an alphabet of any size we wish from a large hash value by performing successive long division of the hash by the size of the alphabet we wish to extract where the remainder from each division will be the value that's mapped back into the alphabet. I've always been fond of this solution since it provides high entropy evenly distributed characters from an alphabet of any size extracted from large binary values, such as those produced by hashing.

For our second-to-last bit of feedback, we have one of our longer format pieces of listener feedback, which I would title: “From the Field”:

robertblaakmeer / @robertblaa42909

Hello Steve, I have been a long time listener (and even longer time SpinRite user) and I enjoy your weekly updates and common sense perspectives on all kinds of security topics. I want to share a (rather long) story with you and would like to hear your opinion about an interesting in-progress responsible vulnerability disclosure. Here goes:

In October 2023 I came across a sign-in page of a high profile company with 20+ billion US$ revenue and more than 100 million monthly active users on their website. Not a small business. For my work, and sometimes just for fun, I regularly visit websites with the Developers Tools tab open. This time I noticed that the sign-in page of this website returned a Response Header that mentioned ***nginx/1.12.2*** *as the webserver. In itself not a big problem, but it is better to hide this information from the public. However, if the page was really served by nginx 1.12.2, then the website would have a big problem, because this version of nginx is very old and has a number of critical and high severity vulnerabilities that are known to be actively exploited.*

So as a good citizen, I wanted to tell them about the issues, just for the sake of improving their security and of course also for improving the security of their 100M monthly active users. So I searched on the company's website for their policy for responsibly disclosing these vulnerabilities, but the best I could find was the email address for privacy related questions. I wrote an email telling them about the vulnerabilities, included some screenshots for evidence and asked some questions about their statements in their privacy policy and terms and conditions on protecting my information and keeping the site bug-free. I was not looking for any financial compensation, I just wanted to make the world a better place.

I soon received a response that a bug report had been filed and that it could take up to 120 days for bug reports to be tested and validated. I received no answers for my questions about privacy.

Fast forward to February 2024, 118 days since their message that the bug report had been filed. I politely asked them about the status of the bug report, notified them that the vulnerabilities still existed and asked them if my personal data is still safe with them. A week later I received an e-mail from them, saying that after review and risk rating by their information security team, the bug report qualifies for a bounty payment of $350. If I agree to this payout agreement, I also agree to keep my silence about the issue. My questions about privacy and trust and new questions about their responsible disclosure policies remain unanswered. In the meantime, the issues still exist, and a small tour around some of the other websites of the same company show similar problems.

Now the fact that these vulnerabilities are so easy to find, the fact that the reported webserver is so old, the fact that it is so easy to remove a server banner and the fact that it is still not fixed after more than 4 months, makes me believe that this could be a honeypot. If it is, then I think that it is a very, very risky attempt of having a honeypot, because it is easy to find, it is on pages where people enter their username and password and if the media get hold of it, then the brand damage of such a high profile company will be big.

I have asked them directly if it is a honeypot, but I did not get a reaction yet.

What do you think about this? What should I do? Accept the payout and keep my silence? Should I also report the 30+ other vulnerabilities on their websites and sign-in pages? Should I tell them that just by looking at the Webserver Response Headers, I now know that they use a mix of nginx 1.12.2, Microsoft IIS 10.0, http://ASP.NET 4.0.30319, Shopify, Wordpress, etc etc? Or should I just leave it up to them to find it all out? I mean, what could possibly go wrong?

Kind regards and again, I love your show, keep going! Robert Blaakmeer

Robert, I agree that this is a conundrum. I took a quick peek at that version of nginx and what immediately jumped out at me – as it would any researcher, nefarious or not – was CVE-2021-23017. It's a well known, remotely exploitable, "off by one" error which gives an attacker unauthenticated remote code execution capability. And what's more, a 3-year old working Python proof-of-concept exploit is publicly available.

So Robert, I agree that you're right to be worried about them, and worried in general. It appears that any miscreant wishing to target this enterprise has everything they need to do so. Unfortunately, although you've done the right thing every step of the way, their ongoing negligence in dealing with this, which you had no way of anticipating, has now compromised you, since you've acknowledged to them that you're in possession of information that could damage them – either reputationally or for use in an attack. And, as we also know, attorneys can allege that just by looking at their server headers you've obtained "non-public network hacking" information, and some judge who knows no better could be convinced. It's happened before.
I think I would have nothing further to do with them. The problem with a company that has become this large is that the people who you really should be talking to are unknown and inaccessible to you – by design. Between you and them are layer upon layer of chair-warming functionaries who have no idea what you're even talking about. At this point, my recommendation would be for you to turn everything over to CISA and let them take it from there. This completely discharges your responsibility in the matter while insulating you from any blowback, since no one – not the company in question nor any judge could fault you for confidentially and privately informing the US Government's Cybersecurity & Infrastructure Security Agency of this potential critical vulnerability after having first given the company in question all of November, December, January and February to deal with upgrading their servers.

CISA, the rare government agency that appears to actually have its act together, has a simple procedure for report filing. Go to: https://www.cisa.gov/report. I also created a GRC shortcut so that even if someone forgets the agency's name, going to https://grc.sc/report will bounce you to that page at CISA. The page has several categories of problems to report. If you scroll down you'll find "Report software vulnerabilities or ICS vulnerabilities" which appears to fit best. If you go there, you get bounced over to CERT.org, which is the CERT Coordination Center at Carnegie Mellon University: https://www.kb.cert.org/vuls/vulcoordrequest/.

So I'd fill out that form to let the US government powers that be know about this. One of the things you can do is attach a file to your submission. So I'd send them your entire eMail thread with the company so that they can see that you have acted responsibly at every step. You'll have done the right thing. And when CISA and CERT reach out to contact the company about their

negligence over a 4-year old known critical vulnerability across their web platforms you can bet that they'll wind up speaking to someone who can affect the required changes. And at that point you'll have done everything you can, while insulating yourself as well as possible from any annoyance this company might feel over being made to help themselves.

And finally, for next week, we have the first appearance of a 0-click GenAI application worm:

Ben Nassi / @ben_nassi

> Hi Steve, I am Ben Nassi (a former researcher at the Ben-Gurion University and Cornell Tech) and the author of Video-based Cryptanalysis and Lamphone which you previously covered in your podcasts. I just published a new research paper on a worm that targets GenAI applications. We named it Morris-II (guess why?). I think the audience of security now will love to hear more about it. And keep on with the great podcast that you and Leo are doing beyond episode 999. BTW, the research was also published on Bruce Scheneier's blog.

So, I suspect that next week's podcast will be titled "Morris II" – Stay tuned for a look at the "Worminization" of GenAI where I'm sure we'll be answering the question "What... could possibly go wrong?"

But now, we're going to take a close look at "What has probably been done right?" by digging into Apple's new Post-Quantum Encryption upgrade, named "PQ3."

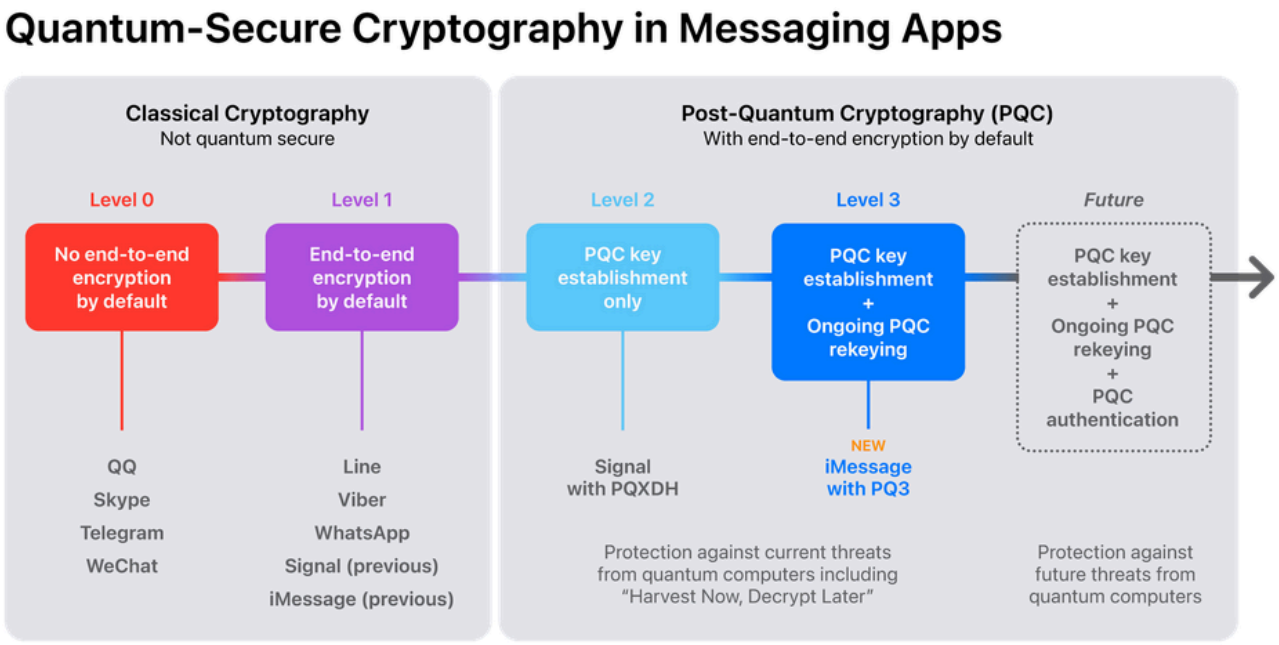
PQ3

So, what is Apple's PQ3?
We can guess that the "PQ" stands for Post Quantum, and we'd be right. So is this Apple's 3rd attempt at a post quantum protocol after one and two somehow failed or fell short? No. Apple has apparently invented "Level's" of security, so PQ3 offers what they call "Level 3 Security." Here's how the SEAR – Apple's Security Engineering and Architecture group – introduced this new protocol in their recent blog posting. They write:

> *Today we are announcing the most significant cryptographic security upgrade in iMessage history with the introduction of PQ3, a groundbreaking post-quantum cryptographic protocol that advances the state of the art of end-to-end secure messaging. With compromise-resilient encryption and extensive defenses against even highly sophisticated quantum attacks, PQ3 is the first messaging protocol to reach what we call Level 3 security — providing protocol protections that surpass those in all other widely deployed messaging apps. To our knowledge, PQ3 has the strongest security properties of any at-scale messaging protocol in the world.*

https://security.apple.com/blog/imessage-pq3/

So they're proud of their work and they've decided to say that not only will iMessage be using an explicitly quantum safe encrypted messaging technology, but that theirs is bigger than – I mean better than – anyone else's. Since, so far, this appears to be as much a marketing promotion as a technical disclosure, it's worth noting that they have outlined the four levels of messaging security which places them alone at the top of the heap.



Note: This comparison evaluates only the cryptographic aspect of messaging security, and therefore focuses on end-to-end encryption and quantum security. Such a comparison doesn't include automatic key verification, which we believe is a critical protection for modern messaging apps. As of the time of this writing, only iMessage and WhatsApp provide automatic key verification. The iMessage implementation, called Contact Key Verification, is the state of the art – it provides the broadest automatic protections and applies across all of a user's devices.

Apple has defined four levels of messaging with levels 0 and 1 being pre-quantum – offering no protection from quantum computing breakthroughs being able to break their encryption – and

Levels 2 and 3 being post-quantum. Level 0 is defined as "No end-to-end encryption by default" and they place QQ, Skype, Telegram and WeChat into this level 0 category. This causes me to be immediately skeptical of this marketing nonsense, since although I have never had any respect for Telegram's ad hoc basement cryptography, which they defend not with any sound theory, but by offering a reward to anyone who can break it, we all know that Telegram is encrypted and that everyone using it is using it because it's encrypted. So lumping it into level 0 and saying that it's not encrypted by default is, at best, very disingenuous and should be beneath Apple.

Level 1 are those pre-quantum algorithms that Apple says are encrypted by default. The messaging apps in the Level 1 group are: Line, Viber, WhatsApp, the previous Signal and the previous iMessage.

Now we move into the quantum-safe realm for levels 2 and 3. Since Signal beat Apple to the quantum safe punch, it's sitting all by its lonesome at Level 2 with the sole feature of offering Post-Quantum Crypto (PCQ) with end-to-end encryption by default. Presumably, as Signal's relatively new PQXDH protocol moves into WhatsApp and other messaging platforms based on Signal's open technologies, those other messaging apps will also inherit Level 2 status, lifting them from levels 0 or 1.

But Apple's new PQ3 iMessaging system adds an additional feature that Signal lacks, which is how Apple granted themselves sole dominion over level 3. Apple's PQ3 adds ongoing post quantum rekeying to its messaging which they make a point of noting, Signal currently lacks.

This blog posting was clearly written by the marketing people, so it's freighted with far more self aggrandizing text than would ever be found in a technical cryptographic protocol disclosure. But I need to share some of it to set the stage, since what Apple feels is important about PQ3 is its attribute of ongoing rekeying. So, to that end, Apple reminds us, quote:

> *When iMessage launched in 2011, it was the first widely available messaging app to provide end-to-end encryption by default, and we have significantly upgraded its cryptography over the years. We most recently strengthened the iMessage cryptographic protocol in 2019 by switching from RSA to Elliptic Curve cryptography (ECC), and by protecting encryption keys on device with the Secure Enclave, making them significantly harder to extract from a device even for the most sophisticated adversaries. That protocol update went even further with an additional layer of defense:* ***a periodic rekey mechanism*** *to provide cryptographic self-healing even in the extremely unlikely case that a key ever became compromised. Each of these advances were formally verified by symbolic evaluation, a best practice that provides strong assurances of the security of cryptographic protocols.*

Okay. So Apple has had on-the-fly ongoing rekeying in iMessage for some time and it's clear that they're going to be selling this as a differentiator for PQ3 to distance themselves from the competition. After telling us a whole bunch more about how wonderful they are, they get down to explaining their level system and why they believe PQ3 is an important differentiator. Here's what they say:

> *To reason through how various messaging applications mitigate attacks, it's helpful to place them along a spectrum of security properties. There's no standard comparison to employ for*

this purpose, so we lay out our own simple, coarse-grained progression of messaging security levels: we start with classical cryptography and progress towards quantum security, which addresses current and future threats from quantum computers. Most existing messaging apps fall either into Level 0 — no end-to-end encryption by default and no quantum security — or Level 1 — with end-to-end encryption by default, but with no quantum security.

A few months ago, Signal added support for the PQXDH protocol, becoming the first large-scale messaging app to introduce post-quantum security in the initial key establishment. This is a welcome and critical step that, by our scale, elevated Signal from Level 1 to Level 2 security.

At Level 2, the application of post-quantum cryptography is limited to the initial key establishment, providing quantum security only if the conversation key material is never compromised. But today's sophisticated adversaries already have incentives to compromise encryption keys, because doing so gives them the ability to decrypt messages protected by those keys for as long as the keys don't change. To best protect end-to-end encrypted messaging, the post-quantum keys need to change on an ongoing basis to place an upper bound on how much of a conversation can be exposed by any single, point-in-time key compromise — both now and with future quantum computers.

Therefore, we believe messaging protocols should go even further and attain Level 3 security, where post-quantum cryptography is used to secure ***both*** *the initial key establishment and the ongoing message exchange, with the ability to rapidly and automatically restore the cryptographic security of a conversation even if a given key becomes compromised.*

Okay. It would be very interesting to hear Signal's rebuttal to this, since it's entirely possible that this is mostly irrelevant marketing nonsense. It's not that it's not true, and that continuous key rotation is not useful. We've talked about this in the past. Key rotation gives a cryptographic protocol a highly desirable property known as Perfect Forward Secrecy. Essentially, the keys the parties are using to protect their conversation from prying eyes are ephemeral. A conversation flow is broken up and compartmentalized by the key that's in use at the moment. But the protocol never allows a single key to be used for long. The key is periodically changed. The reason I'd like to hear a rebuttal from Signal is that their protocol has always featured perfect forward secrecy. Remember "Axolotl" ?? Here's what Wikipedia says:

In cryptography, the Double Ratchet Algorithm (previously referred to as the Axolotl Ratchet) is a key management algorithm that was developed by Trevor Perrin and Moxie Marlinspike in 2013. It can be used as part of a cryptographic protocol to provide end-to-end encryption for instant messaging. After an initial key exchange it manages the ongoing renewal and maintenance of short-lived session keys. It combines a cryptographic so-called "ratchet" based on the Diffie–Hellman key exchange (DH) and a ratchet based on a key derivation function (KDF), such as a hash function, and is therefore called a double ratchet.

The algorithm provides forward secrecy for messages, and implicit renegotiation of forward keys; properties for which the protocol is named.

Right. In 2013. In other words, it would be nice to hear from Signal, since Apple appears to be suggesting that they alone are offering the property of perfect forward secrecy for quantum safe messaging when it certainly appears that Signal got there 11 years ago. This is not to say that

having this feature in iMessage is not a good thing. But it appears that Apple may not actually be alone at PQ's level 3. So when do we see it from Apple? They write:

> *Support for PQ3 will start to roll out with the public releases of iOS 17.4, iPadOS 17.4, macOS 14.4, and watchOS 10.4, and is already in the corresponding developer preview and beta releases. iMessage conversations between devices that support PQ3 are automatically ramping up to the post-quantum encryption protocol. As we gain operational experience with PQ3 at the massive global scale of iMessage, it will fully replace the existing protocol within all supported conversations this year.*

Okay. So now I want to share Apple's description of the design of PQ3. It includes a bunch of interesting details and also something that Telegram has never had, which is multiple formal proofs of correctness. Since we're now able to do this, it's a crucial step for trusting any newly created cryptographic system. Here's what Apple wrote:

> *More than simply replacing an existing [cryptographic] algorithm with a new one, we rebuilt the iMessage cryptographic protocol from the ground up to advance the state of the art in end-to-end encryption, and to deliver on the following requirements:*
>
> - *Introduce post-quantum cryptography from the start of a conversation, so that all communication is protected from current and future adversaries.*
> - *Mitigate the impact of key compromises by limiting how many past and future messages can be decrypted with a single compromised key.*
> - *Use a hybrid design to combine new post-quantum algorithms with current Elliptic Curve algorithms, ensuring that PQ3 can never be less safe than the existing classical protocol.*
> - *Amortize message size to avoid excessive additional overhead from the added security.*
> - *Use formal verification methods to provide strong security assurances for the new protocol.*
>
> *PQ3 introduces a new post-quantum encryption key in the set of public keys each device generates locally and transmits to Apple servers as part of iMessage registration. For this application, we chose to use Kyber post-quantum public keys, an algorithm that received close scrutiny from the global cryptography community, and was selected by NIST as the Module Lattice-based Key Encapsulation Mechanism standard, or ML-KEM. This enables sender devices to obtain a receiver's public keys and generate post-quantum encryption keys for the very first message, even if the receiver is offline. We refer to this as initial key establishment.*
>
> *We then include — within conversations — a periodic post-quantum rekeying mechanism that has the ability to self-heal from key compromise and protect future messages. In PQ3, the new keys sent along with the conversation are used to create fresh message encryption keys that can't be computed from past ones, thereby bringing the conversation back to a secure state even if previous keys were extracted or compromised by an adversary. PQ3 is the first large scale cryptographic messaging protocol to introduce this novel post-quantum rekeying property.*

PQ3 employs a hybrid design that combines Elliptic Curve cryptography with post-quantum encryption both during the initial key establishment and during rekeying. Thus, the new cryptography is purely additive, and defeating PQ3 security requires defeating both the existing, classical ECC cryptography and the new post-quantum primitives. It also means the protocol benefits from all the experience we accumulated from deploying the ECC protocol and its implementations.

Rekeying in PQ3 involves transmitting fresh public key material in-band with the encrypted messages that devices are exchanging. A new public key based on Elliptic Curve Diffie-Hellman (ECDH) is transmitted inline with every response. The post-quantum key used by PQ3 has a significantly larger wire size than the existing protocol, so to meet our message size requirement we designed the quantum-secure rekeying to happen periodically rather than with every message. To determine whether a new post-quantum key is transmitted, PQ3 uses a rekeying condition that aims to balance the average size of messages on the wire, preserve the user experience in limited connectivity scenarios, and keep the global volume of messages within the capacity of our server infrastructure. Should the need arise, future software updates can increase the rekeying frequency in a way that's backward-compatible with all devices that support PQ3.

I'll just interrupt here to note that it seems likely that PQ3 is rotating its quantum keys more frequently than Signal. I don't recall the details of Signal's ratchet, and it may have changed since we last looked at it. But it might also be that this is a distinction without a difference. In the case of Signal's ratchet, it was designed not only to provide useful forward secrecy, but as a means for resynchronizing offline asynchronous end points. The reason I suggest that it may be a distinction without a difference is that these key compromises are purely what-if's. No one knows of any scenario where that could actually happen. If anyone did it would be eliminated. It's a bit like saying "my password is way stronger than yours because it's 200 characters long." That's good, but does it really matter? Anyway, Apple continues...

With PQ3, iMessage continues to rely on classical cryptographic algorithms to authenticate the sender and verify the Contact Key Verification account key, because these mechanisms can't be attacked retroactively with future quantum computers. To attempt to insert themselves in the middle of an iMessage conversation, an adversary would require a quantum computer capable of breaking one of the authentication keys before or at the time the communication takes place. In other words, these attacks cannot be performed in a ***Harvest Now, Decrypt Later scenario*** *— they require the existence of a quantum computer capable of performing the attacks contemporaneously with the communication being attacked. We believe any such capability is still many years away, but as the threat of quantum computers evolves, we will continue to assess the need for post-quantum authentication to thwart such attacks.*

Our final requirement for iMessage PQ3 is formal verification — a mathematical proof of the intended security properties of the protocol.

PQ3 received extensive review from Apple's own multi-disciplinary teams in Security Engineering and Architecture (SEAR) as well as from some of the world's foremost experts in cryptography. This includes a team led by Professor David Basin, head of the Information Security Group at ETH Zürich and one of the inventors of Tamarin — a leading security protocol verification tool that was also used to evaluate PQ3 — as well as Professor Douglas

Stebila from the University of Waterloo, who has performed extensive research on post-quantum security for internet protocols. Each took a different but complementary approach, using different mathematical models to demonstrate that as long as the underlying cryptographic algorithms remain secure, so does PQ3. Finally, a leading third-party security consultancy supplemented our internal implementation review with an independent assessment of the PQ3 source code, which found no security issues.

In the first mathematical security analysis of the iMessage PQ3 protocol, Professor Douglas Stebila focused on so-called game-based proofs. This technique, also known as reduction, defines a series of "games" or logical statements to show that the protocol is at least as strong as the algorithms that underpin it. Stebila's analysis shows that PQ3 provides confidentiality even in the presence of some key compromises against both classical and quantum adversaries, in both the initial key establishment and the ongoing rekeying phase of the protocol. The analysis decomposes the many layers of key derivations down to the message keys and proves that, for an attacker, they are indistinguishable from random noise. Through an extensive demonstration that considers different attack paths for classical and quantum attackers in the proofs, Stebila shows that the keys used for PQ3 are secure as long as either the Elliptic Curve Diffie-Hellman problem remains hard or the Kyber post-quantum KEM remains secure.

Apple then inserts a quote from Professor Douglas Stebila:

The iMessage PQ3 protocol is a well-designed cryptographic protocol for secure messaging that uses state-of-the-art techniques for end-to-end encrypted communication. In my analysis using the reductionist security methodology, I confirmed that the PQ3 protocol provides post-quantum confidentiality, which can give users confidence in the privacy of their communication even in the face of potential improvements in quantum computing technology. —Professor Douglas Stebila

Apple then continues:

In the second evaluation, titled "A Formal Analysis of the iMessage PQ3 Messaging Protocol", Professor David Basin, Felix Linker, and Dr. Ralf Sasse at ETH Zürich use a method called symbolic evaluation. As highlighted in the paper's abstract, this analysis includes a detailed formal model of the iMessage PQ3 protocol, a precise specification of its fine-grained security properties, and machine-checked proofs using the state-of-the-art symbolic Tamarin prover.

The evaluation yielded a fine-grained analysis of the secrecy properties of PQ3, proving that "in the absence of the sender or recipient being compromised, all keys and messages transmitted are secret" and that "compromises can be tolerated in a well-defined sense where the effect of the compromise on the secrecy of data is limited in time and effect," which confirms that PQ3 meets our goals.

And they quote Professor Basin:

"We provide a mathematical model of PQ3 as well as prove its secrecy and authenticity properties using a verification tool for machine-checked security proofs. We prove the properties even when the protocol operates in the presence of very strong adversaries who can

corrupt parties or possess quantum computers and therefore defeat classical cryptography. PQ3 goes beyond Signal with regards to post-quantum defenses. In PQ3, a post-quantum secure algorithm is part of the ratcheting and used repeatedly, rather than only once in the initialization as in Signal. Our verification provides a very high degree of assurance that the protocol as designed functions securely, even in the post-quantum world.
—Professor David Basin

Now, okay... the dig at Signal was entirely unnecessary and gratuitous. And I don't understand why Apple apparently feels so threatened by Signal – oh wait! Yes I do! Signal is open, open design, open source, and entirely cross platform – anyone's conversations can be protected by Signal on any platform they choose, whereas iMessage, just like everything else Apple does, is all about platform lock-in.

So is PQ3 any reason to choose iMessage over Signal? No. When we're talking about cryptography we've learned that there's nothing wrong with adding a belt to those suspenders. After all, that's why Apple copied Signal in adding post-quantum crypto to existing and well-proven pre-quantum crypto. Belt and suspenders. And so, if your work model allows you to be stuck within Apple's closed ecosystem then you can be confident that iMessage will be secure against any current and future surprises. But you'll certainly be secure enough using Signal and you'll have the benefit of far more freedom.

Next, the paper drops into lots of interesting detail that I won't drag everyone through since we already have the essence of what's going on. But I wanted to share one piece of the techie bits since I think it's interesting regarding the overhead introduced by Apple's more or less continuous rekeying. Keep in mind that text messages are often no longer than old school SMS Tweets. Apple explains:

To limit the size overhead incurred by frequent rekeying while preserving a high level of security, the post-quantum KEM is instantiated with Kyber-768. Unlike the IDS-registered public keys used for the initial key establishment, ratcheting public keys are used only once to encapsulate a shared secret to the receiver, significantly limiting the impact of the compromise of a single key. However, while a 32-byte ECDH-based ratchet overhead is acceptable on every message, the post-quantum KEM ratchet increases the message size by more than 2 kilobytes. To avoid visible delays in message delivery when device connectivity is limited, this ratchet needs to be amortized over multiple messages.

We therefore implemented an adaptive post-quantum rekeying criterion that takes into account the number of outgoing messages, the time elapsed since last rekeying, and current connectivity conditions. At launch, this means the post-quantum ratchet is performed approximately every 50 messages, but the criterion is bounded such that rekeying is always guaranteed to occur at least once every 7 days. And as we mentioned earlier, as the threat of quantum computers and infrastructure capacity evolves over time, future software updates can increase the rekeying frequency while preserving full backward compatibility.

Okay. So now everyone knows as much about PQ3 as is necessary. Apple has followed Signal by adding a believed-to-be-strong post-quantum crypto algorithm to their built-in iMessaging platform. Apple has also taken the welcome step of having their largely new iMessaging protocol formally proven by highly-qualified academic algorithm

researchers. It’s only a bit sad that Apple clearly feels so threatened by Signal.