Security Now! #1034 - 07-15-25

Introduction to Zero-Knowledge Proofs

This week on Security Now!

• A glorious takedown of quantum factorization. • Notepad++ signs its own code signing certificate. • Dennis Taylor has Bobiverse Book 6 on his lap. • Crypto/ATM machines flat out outlawed. • Signal vs WhatsApp: Encryption in flight and at rest. • A close look at browser fingerprinting metrics. • Rewriting interpreters in memory-safe languages. • An introduction to zero-knowledge proofs.

What's that about the weakest link?

Security News

A takedown of Quantum Factorization

One of our listeners (thanks, by the way) sent a link to a paper that was co-authored by Peter Gutmann. I'm not sure that our listener knew exactly what to make of this paper since its title is *"Replication of Quantum Factorisation Records with an 8-bit Home Computer, an Abacus, and a Dog"*. I'm sure that the title would have hooked me with a "Huh?!?" even if Peter's name wasn't immediately familiar to me. Wikipedia opens its lengthy description of Peter by writing:

> *Peter Claus Gutmann is a computer scientist in the Department of Computer Science at the University of Auckland, Auckland, New Zealand. He has a Ph.D. in computer science from the University of Auckland. His Ph.D. thesis and a book based on the thesis were about a cryptographic security architecture. He is interested in computer security issues, including security architecture, security usability (or more usually the lack thereof), and hardware security; he has discovered several flaws in publicly released cryptosystems and protocols. He is the developer of the cryptlib open source software security library and contributed to PGP version 2.*

So Peter very clearly knows his way around computer security and cryptography and I felt compelled to venture into anything he might write on the subject of replicating various quantum computer factorization records using an 8-bit Home Computer (which turned out to be a 6502-based Commodore VIC-20), an Abacus and his dog named "Scribble". Before I go any further, I do not want to forget to leave a link to this work in the show notes. So it's there now for anyone who might wonder what "Scribble" looks like: https://eprint.iacr.org/2025/1237.pdf

Since I hardly consider myself to be any kind of expert in the field of quantum computing – I certainly am not – I have felt somewhat self conscious through the years as I have continually poking fun at what appear to be the extremely meager quantum computing number-factoring accomplishments that seem to be celebrated by the popular press, especially when I see very little evidence of what we would call scalability. The fact that some super-cooled chip was able to factor the 6-bit number 35 into its two products 5 and 7, doesn't clearly suggest that more bits will be forthcoming, nor when.

So I felt somewhat heartened to learn at least that my intuition here, and my understanding of physics where it matters, may be intact. Peter and his co-author, in this recently releases March 2025 paper, are quite clearly similarly unimpressed. Their paper's abstract is short and to the point, saying:

> *This paper presents implementations that match and, where possible, exceed current quantum factorisation records using a VIC-20 8-bit home computer from 1981, an abacus, and a dog. We hope that this work will inspire future efforts to match any further quantum factorisation records, should they arise.*

Peter is having some fun with this although his sincere intent appears to be to roundly debunk the *"accomplishments"* that have been achieved so far. Peter understood that his use of the term *"factorise"* might strike some as awkward. It did me. So the first footnote of his paper tackles this head-on, writing: *"We use the UK form "factorise" here in place of the US variants "factorize" or "factor" in order to avoid the 40% tariff on the US term."* It occurred to me that he might qualify for a quantity discount here, since the term is used in nearly every sentence.

Nevertheless, I need to share this with everyone, since I'm sure what Peter has to say will be of

great interest to our listeners. We all know that Quantum Computing presents what has been described as a clear and present danger to the world's current quantum-naive cryptography, and specifically to the threat that the prime factorization problem, upon which much of today's security technology critically depends, might be at risk of being solved by sufficiently powerful quantum computers. Peter is here to disabuse us of any such concern. They write:

In 1994, mathematician Peter Shor proposed his quantum factorisation algorithm, now known as Shor's Algorithm. In 2001, a group at IBM used it to factorise the number 15. Eleven years later this was extended to factorise the number 21. Another seven years later a factorisation of 35 was attempted but failed. Since then no new records have been set, although a number of announcements of such feats have cropped up from time to time alongside the more publicly-visible announcements of quantum supremacy every few months. These announcements are accompanied by ongoing debates over whether a factorisation actually took place and if so what it was that was factorised, with the issue covered in more detail in this paper's section 3. Of particular note was the claim in 2024 by researchers to have factorised an RSA-2048 number ("the D-Wave paper"). In this paper we focus on the factorisations of 15, 21, and 35, as well as the claimed RSA-2048 factorisation.

New technologies, when introduced, are typically given names that overstate their capabilities, usually by equating them with existing familiar systems or technological artefacts. For example the first computers in the 1940s and 1950s, often little more than glorified electric adding machines, were nevertheless described as "electronic brains". More recently, large language models (LLMs) have been touted as "artificial intelligence", and complex physics experiments have been touted as "quantum computers".

In order to avoid any confusion with actual computers like the VIC-20 with which they have nothing in common, we refer to them here as "physics experiments". Similarly, we refer to an abacus as "an abacus" rather than a digital computer, despite the fact that it relies on digital manipulation to effect its computations. Finally, we refer to a dog as "a dog" because even the most strenuous mental gymnastics can't really make it sound like it's a computer.

When stage magicians perform sleight-of-hand tricks, traditionally card tricks, they use specially constructed decks called force decks with which they can force the participants in the trick to pick a card of the magician's choosing. An example of such a force deck is a Svengali deck which when shown to the participant or the audience appears to contain a standard mix of cards but which only contains a single repeated and therefore entirely predictable card.

*Similarly, **quantum factorisation** is performed using sleight-of-hand numbers that have been selected to make them very easy to factorise using a physics experiment and, by extension, a VIC-20, an abacus, or a dog. A standard technique is to ensure that the factors differ by only a few bits that can then be found using a simple search-based approach that has nothing to do with factorisation. For example the lenghty RSA-2048 number (which he then enumerates in this paper) is the product of two factors (which he also lists) that differ by only one or two [least significant] bits. This makes it possible to perform the "factorisation" through a simple integer square root calculation. Note that such a value would never be encountered in the real world since the RSA key generation process typically requires that |p-q| > 100 or more bits.*

As one analysis puts it, "Instead of waiting for the hardware to improve by yet further orders of magnitude, researchers began inventing better and better tricks for factoring numbers by exploiting their hidden structure". [I should mention that all of Peter's assertions contain references to their source. He's not just making any of this stuff up.]

A second technique used in quantum factorisation is to use preprocessing on a computer to transform the value being factorised into an entirely different form or even a different problem to solve which is then amenable to being solved via a physics experiment. For example the 2019 quantum factorisation of 1,099,551,473,989, of which more is below, relied on processing with a computer to transform the problem into one that was solvable with a three-qubit circuit.

Other quantum factorisations also rely on computers to reduce the problem to a form in which it can be "solved" through a physics experiment. Even the factorisations of 15 and 21 used the so-called compiled form of Shor's algorithm which uses prior knowledge of the answer to merely verify the (known-in-advance) factors rather than performing any actual factorisation. In the case of 15 and 21 this reduced the number of qubits required from 8 and 10 to 2 in the compiled form. The paper that discusses this result comments that "it is not legitimate for a compiler to know the answer to the problem being solved. To even call such a procedure compilation is an abuse of language".

The paper then presents the factorisation of a 768-bit (231-digit) number and a 20,000-bit number, both of which can also be factorised using 2 qubits in the compiled form. As the paper points out, "our technique can factor all products of p, q such that p, q are unequal primes greater than two, runs in constant time, and requires only two coherent qubits". Still other quantum factorisations go even further. For example one claimed factorisation involved working backwards from the known answer to design a physics experiment that produced the known-in-advance solution. There is no equivalent computation for such a sleight-of-hand operation so we have no means to show an equivalent using a VIC-20 or an abacus. The trick in all of these cases is to figure out how to construct a value such that it can then be transformed into a vastly simpler form in which it can be "factorised" via a physics experiment.

These types of factorisations have also been referred to as "stunt factorisations". For example the main effort in the 2012 factorisation of 143 into 11 × 13 consisted of finding a value with the special properties required that allowed it to be "factorised" by a physics experiment. This feat was then extended in 2014 to 56,153, in 2018 to 4,088,459 and later that year to the impressive-looking 383,123,885,216,472,214,589,586,724,601,136,274, 484,797,633,168,671,371.

Many further types and techniques for stunt factorisations exist, far too many to catalogue here, with the practice typically being to manufacture a small value that's easily "factorised" via a physics experiment and then later figuring out how to stretch the value to add more and more digits (383,123,885,216,472,214,589,586,724,601,136,274,484,797,633,168,671, 371) while still allowing it to be "factorised" by the same physics experiment. For example the compiled Shor's algorithm can factorise any composite number p × q on a very small physics experiment, a "factorisation" mechanism that has been given the tongue-in-cheek name Smolin-Smith-Vargo Algorithm after the authors of the paper that pointed out the technique. It should be noted here that all of these sleight-of-hand and stunt values are trivially factorised by Fermat's method on a Raspberry Pi or similar.

Similar to stage magic, the exercise when responding to a new quantum factorisation announcement is not only to marvel at the trick but to try and figure out where the sleight-of-hand occurred. One simple technique to catch the use of sleight-of-hand numbers is to view them in binary form. If they consist almost entirely of zero bits, as did the 2019 factorisation of 1,099,551,473,989, which begins with 100000000000000… when expressed in binary, then it's a sleight-of-hand number. Similarly, numbers with repeating bit patterns 10101010… or similar are sleight-of-hand numbers. Section 7 presents a technique for selecting non-

sleight-of-hand numbers for future quantum factorisation work.

A second technique is to check whether the value submitted to the physics experiment was the one being factorised or whether it has been first transformed on a computer into an entirely different form that's solvable with a physics experiment. A standard trick here is to transform the factorisation into a combinatorial minimization problem which is readily solved using Grover's algorithm, completely impractical for factorisation but perfectly suitable for publication credit.

Many other sleight-of-hand tricks exist for creating apparent quantum factorisations. One example is what we are calling the Callas Normal Form for Sleight-of-Hand Quantum Factorisation or "Callas Normal Form" for short after cryptographer Jon Callas who first described it. In the Callas Normal Form, the factors are integers p = 2n-1 and q = 2m+1, where n ≤ m, and p and q are ideally prime, but don't have to be. The binary representation of the product N = pq then starts with 'n' one bits followed by m − n zero bits and ends in another 'n' one bits. Needless to say, this is easily detected, even on a 6502, and easily factorised (no real-world RSA toolkit would ever generate such prime factors). For example, a recent preprint uses this form to claim in its title success in factorising 4096-bit integers with Shor's algorithm "under certain conditions", where the "conditions" for the 12 examples used turn out to be equivalent to the Callas Normal Form.

In other words, everyone is cheating and cheating badly. And it's no wonder that Peter finally got fed up with this and decided to author this takedown. No quantum computer that's known or has been published about has ever yet done anything actually useful, not even to factor the 6-bit number 35. Peter finishes, writing:

So far as we have been able to determine, no quantum factorisation has ever factorised a value that wasn't either a carefully-constructed sleight-of-hand number or for which most of the work wasn't done beforehand with a computer in order to transform the problem into a different one that could then be readily solved by a physics experiment. We attempt to address this deficiency by providing criteria for evaluating quantum factorisation attempts in section 7. The pervasive use of sleight-of-hand numbers and techniques and stunt factorisations throughout the field of quantum factorisation makes it difficult to select targets for our factorisation replication attempts. Since it's possible, with a bit of thought, to construct arbitrarily impressive-looking values for factorisation, an example being the 20,000-bit artificial value that was factorised with a 2-qubit physics experiment, we have to select targets that are at least within shouting distance of an actual application of something like Shor's algorithm for quantum factorisation. The three instances of this that we have been able to identify in the literature, even though they also use sleight-of-hand by using the compiled form of Shor's algorithm mentioned earlier, are the 2001 factorisation of 15, the 2012 factorisation of 21, and the (attempted) 2019 factorisation of 35, constituting not actual quantum factorisations but at least the least sleight-of-handy attempts at quantum factorisation.

At this point we're into page 5 of the 16-page paper, although, as I mentioned, the end of the paper contains nearly 3 full pages of references to support and substantiate everything these authors have alleged. They proceed to demonstrate how a Commodore VIC-20 from 1981 can match any feat that any quantum computer has performed so far. They use an abacus to do the same, and when they get to their chosen dog, "Scribble", they write:

As has been previously pointed out, the 2001 and 2012 quantum factorisation records may be easily matched by a dog trained to bark three times. We verified this by taking a recently-calibrated reference dog, Scribble, depicted here, and having him bark three times, thus simultaneously factorising both 15 and 21. This process was not as simple as it first appeared because Scribble is very well behaved and almost never barks. Having him perform the quantum factorisation required having his owner play with him with a ball in order to encourage him to bark. It was a special performance just for this publication, because he understands the importance of evidence-based science.

The process was then repeated to have him bark five times, factorising the number 35 and thereby exceeding the capabilities of the quantum factorisation physics experiments mentioned earlier.

Unfortunately this process fails for the RSA-2048 values since the size of the factors exceeds even the most enthusiastic dog's barking ability. However there is another process that allows us to factorise even these huge numbers with a dog. Recall from section 4 that the prime factors p and q were either 2 or 6 apart. This led to an analysis where it was discovered that p = x – d and q = x + d, where x is the integer in the middle between p and q and d is either 1 or 3. It can thus be argued that d is the real secret. So teaching a dog to bark three times already gives us all of the actual factorisations with Shor's algorithm, plus 50% of the moduli in the D-Wave paper, in the same way that factorising 143 also factorises 56,153, 4,088,459, and 383,123,885,216,472,214,589,586,724, 601,136,274,484,797,633,168,671,371.

Having set up the situation quite well, the paper then gets down to its serious purpose of establishing some guidelines and standards for choosing the numbers that will be factored by wannabe quantum computing experiments. This was really the entire point of the paper, though without the lead-up preamble the serious need for the guidelines might not be fully appreciated. They conclude by writing:

In this paper we showed how to replicate current quantum factorisation records using first a VIC-20 8-bit home computer from 1981, then an abacus, and finally a dog. In terms of comparative demonstrated factorisation power, we rank a VIC-20 above an abacus, an abacus above a dog, and a dog above a quantum factorisation physics experiment.

Finally, we provided standard evaluation criteria for future claimed quantum factorisations.

Listener Feedback

Benjamin Lindner

Hi Steve, Long time listener (and Club Twit member). Yours is the only podcast I listen to with my full attention, and will pause if I can't pay attention at the moment. The only podcast I make sure to hear and understand everything. I'll often re-listen several times to things I didn't fully understand. In short, high praise 😊.

I came across something this week that I thought didn't seem right. But I'm not sure. I am however sure that the above is true of many people.

Notepad++ updated (again) to v8.8.3. (we know how you love those frequent updates). In the information about the update, the developer says that this version ships self-signed, with a CA certificate, and gives instructions for installing it, as a root CA, in users' machines. He explains that he's been having difficulties getting a code signing cert, so the unsigned binaries triggered AV false positives.

See https://notepad-plus-plus.org/news/v883-self-signed-certificate

This seems to me much too dangerous. I myself am not having the problem described, but I was just struck by the danger of such a thing. Even if Notepad++ is ok, and the developer has no ill intention with a very powerful cert, seems to me like a terrible suggestion. It's just too powerful, and thus dangerous. There's also the bad habit forming of installing certs willy-nilly to solve problems.

I can see a developer being frustrated and coming to this solution. But it's irresponsible to put this out to the general public. Am I understanding this correctly? Also a SpinRite owner. (v7 when?).

I was apprised of this issue with Notepad++ some time ago by another of our listeners who wrote to ask whether I had any ideas for solving this problem for our Notepad++ author.

Before I go any further, let's see what Notepad++'s author wrote. He announced this Notepad++ v8.8.3 release under the title: *"Notepad++ v8.8.3 release: self-signed certificate"* writing:

There were - and still are - many false-positives reported in the previous version v8.8.2, by the antivirus software due to the absence of Windows code signing certificate. To prevent this issue from recurring in future releases, from this version on, the Notepad++ release is signed with a certificate issued by a self-signed Certificate Authority (CA). The root certificate is published on the Notepad++ website, GitHub repository & Notepad++ User Manual, allowing antivirus vendors, IT teams and users to verify the authenticity of each release. How to install the root certificate:

- *Double-click the certificate, it may tell you it's invalid, ignore that and click: "Install Certificate..".*
- *In the Certificate Import Wizard, select "Local Machine", then click Next.*
- *If prompted by UAC (optional, depending on admin Previleges), click Yes.*
- *Choose "Place all certificates in the following store", then browse and select "Trusted Root Certification Authorities". Click Next.*
- *On the final page of the wizard, click Finish to complete the installation.*

For detailed instructions, see Notepad++ User Manual.

We're still trying to obtain a certificate issued by conventional Certificate Authorities, for a better user experience. But let's be honest: it's probably not happening. Notepad++ isn't a business - it's certainly not an enterprise - and apparently, that makes a popular open-source project invisible to their gatekeeping standards.

If the "gatekeepers" won't issue a certificate under the name we deserve - so be it. At least it spares us from wasting time and energy on a frustrating process that demands we beg for a new certificate every 3 years. The Notepad++ Root Certificate may not carry their approval, but it leads us to freedom.

What we have here is another effectively intractable problem caused by the ever-escalating war between malware and goodware. Just like this Notepad++ guy, I write goodware. And if I fail to sign any of that goodware the A/V industry, Microsoft and Windows Defender all freak out. They sometimes freak out even IF I sign goodware with a certificate that has been continually signing goodware for years, and has never once been found to sign malware. That's just where the bar has been set. That bar has been pushed as high as it can go because it's safer for the end user for their protectors to say "no" to goodware than "yes" to malware.

And this has been an ongoing drama that, as a software developer, I've been sharing with this audience as it's been unfolding. For a while, Microsoft was giving extra Brownie points to any code that was signed with the extra-expensive EV code signing certificates. So that's what I was using. And EV certs could only be received, stored in and used by an HSM – a Hardware Security Module. This effectively raised the bar even higher and much more robustly prevented their theft and abuse. Because ***that's*** the real danger here; that bad guys will get their hands on a trusted and trustworthy code signing certificate and use it to sign some nasty malware that will stand a much greater chance of slipping right past A/V detections, specifically because it's been signed by a signer whose trust has been earned.

For reasons that remain a mystery, Microsoft then decided to deprecate the special treatment of EV certificates and treat all code signing certs alike. Naturally, this wasn't until after I had jumped through numerous high hoops to allow GRC's server to individually sign every copy of SpinRite v6.1 on-the-fly with a Hardware Security Module as it's being downloaded. Now that this is in place I expect I'll leave it there, though I'm not sure, since all maintenance of certificates in HSMs is annoying. But in any event, we're now told that the "EV'ness" of code signing certs is irrelevant. And I'm fine with that since proving my identity for EV qualification was bizarre. Remember what I was required to do? A had a one-way video call, with me on camera, holding my government issued ID up next to my face with my left hand, and then moving my right hand around in the space around it, following the real-time instructions of the person at the other side of the video call? Talk about raising the bar.

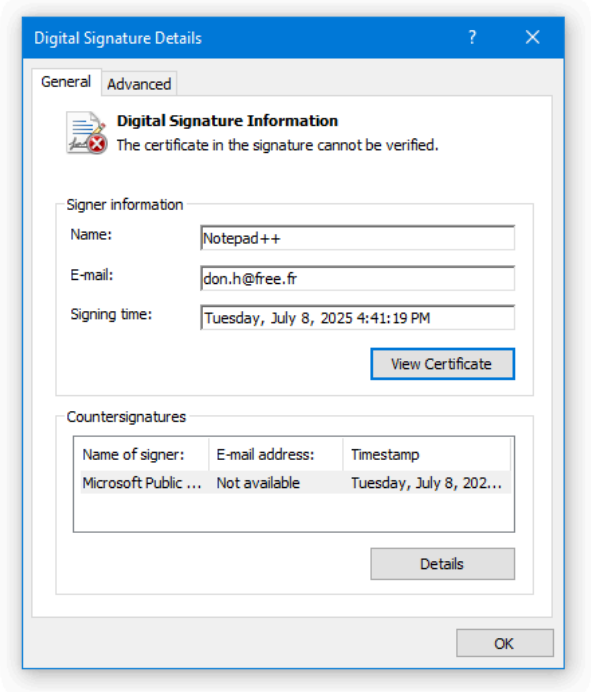
But whether my code signing certificate is EV or not, what matters most is that the private key that was used to sign the certificate was, itself, signed by DigiCert's private key and that private key's matching public key is already present in everyone's Windows PC certificate root store. DigiCert's public signing key is also present in all of the A/V testing systems so that they're able to verify that the certificate that was used to sign the code was issued by and signed by a reputable Certificate Authority.

Which brings us to the problem that Notepad++'s author, Don Ho, has. Non-EV, generic Organization Validation (OV) code signing certificates cost between $250 and $400 per year. And Notepad++ is Windows freeware. Who's going to pay that cost? GRC can afford that, since I've

been fortunate enough to develop an amazing following of terrific customers. So all of GRC's freeware gets the benefit of being signed by the same certificate as its paid-for commercial software, which is, as we know, at this time still only SpinRite. But Don is offering Notepad++ to the world for free, and he's understandably irked by the idea of needing to pay for the privilege of not being flagged by the world's overactive A/V scanning as malicious. Notepad++ is not, and has never been, malicious. But code is code and today's A/V takes a "better safe than sorry" approach. Code that is not signed is automatically looked down upon with extreme suspicion.

And the problem is not unique to Don Ho, of course. This is creating a big problem for the wider open source community... and it does not have a solution. No reputable certificate authority would be willing to put their name and reputation on the line by signing some freely distributed and trusted freeware certificate for use by the open source freeware community because malware would be signed with that certificate before you could blink.

So what Don is attempting to do is to be his own certificate authority. He created a pair of certificates, one private and the other public, and the public key certificate is self-signed. So it's serving as its own trust anchor. In that way it's just like all of the other self-signed CA root certificates that all our PCs contain. The difference is that none of our PCs contain Don Ho's CA root certificate until and unless we deliberately install it into our machines.



V8.8.3 of Notepad++ signed by its own (untrusted) certificate.

Our listener asked: *"Even if Notepad++ is ok, and the developer has no ill intention with a very powerful cert, it seems to me like a terrible suggestion. It's just too powerful, and thus dangerous. There's also the bad habit forming of installing certs willy-nilly to solve problems."*

At first we might think that the biggest danger is that Don might not be good at keeping secrets and that someone might break into this development environment to steal the private certificate he uses to sign his freeware. I don't think that's a big concern because the only place anything signed by that certificate, even if it were to be stolen, would be within the microcosm of Don's Notepad++ users into whose PCs he had convinced to install his proprietary certificate. That's not going to be of much use to someone who wishes to sneak their malware past the world's A/V. And I'm not at all convinced that Don's solution will help at all with the world's A/V tools. A/V tools don't know that they can trust code signed by Don's own certificate. And why would

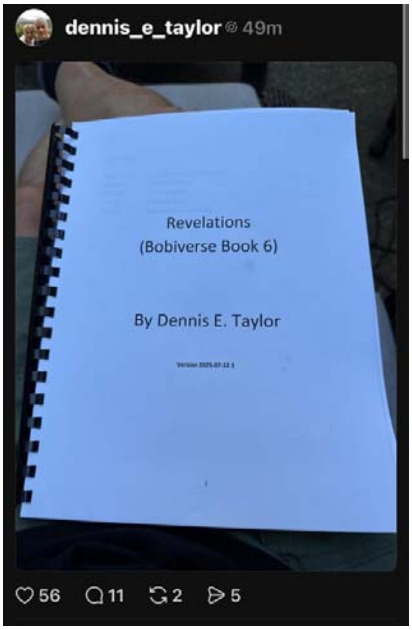
they? We know that Don's a good guy and that he would never deliberately produce malware. But nothing prevents any malware author from doing the same thing Don did, creating their own certificate pair and signing their code with an untrusted certificate having no pedigree. The fact that some of Don's users may elect to place his certificate into their own machine's CA root store certainly won't sway the opinion of A/V tools. And it's unclear to me that it would sway the opinion of Windows Defender.

So it doesn't seem to me that Don creating his own certificate is a big problem from the standpoint of it representing any great risk of abuse. And it's also not at all clear to me that it will achieve what Don hopes, since A/V tools are going to be checking code signing certs against their **own** CA root stores and not whatever random users may have installed in their own PCs. Don's site says this will <quote> *"allow antivirus vendors, IT teams and users to verify the authenticity of each release."* I would fall over in stunned shock if any antivirus vendor cared the least little bit about Don's own root CA certificate... but I suppose hope springs eternal.

What I do very much worry about, is what our listener, Benjamin, referred to as *"the bad habit forming, of installing certs willy-nilly to solve problems."* With that I really agree wholeheartedly. Just about the last thing I want is for my own machines' CA root stores to be filling up with random certificates from the authors of the freeware I wish to use. On the one hand, over time that would create a truly unmanageable mess. And as I noted, it's unclear what it would accomplish anyway, since any malware author is just as able to create their own root CA certificate as Don did. So there's no way any antivirus system or Microsoft's own Defender endpoint protection is going to care. So, hopefully, that means it's not going to happen.

I am 100% sympathetic with Don's plight. And as I noted, this is a real mess without any clear solution. A/V has been forced to raise the bar to the point that anything new that appears is regarded with suspicion and only after some code has settled down and earned a reputation will it become trusted.

The real lesson Don may learn is that any new release is going to be met with pushback from today's A/V and pain for his users. So he should work to get the code a lot more correct before he pushes out another initially untrusted release.



Kevin Zollinger shared an image and wrote:

> *Morning!*
>
> *I suspect I am #3231 to pass this along, but it looks like Book 6 is on its way. I grabbed the image off Reddit. Sometime next year we'll be back hearing about the many adventures of the Bobs.*

I don't recognize the app this screenshot was taken from. Is that Instagram? But in any event, what we see appears to be posted by Dennis E Taylor. It's a photo of a manuscript titled "Revelations" (Bobiverse Book 6) by Dennis E. Taylor. And it's sitting on (presumably) his lap. A pair of legs in shorts can be seen underneath the draft manuscript as well. And in the fine print we see "Version 2025-7-12.1 … which would make that last week.

Christopher Lawson

Mr. Gibson, While listening to podcast 1033 you were asking about the difference in WhatsApp and other messaging apps with regard to WhatsApps not using encryption for message storage. Messaging applications like Signal encrypts the SQLite database on the mobile device in addition to the built in device storage encryption and stores the encryption key on the keychain [1]. This prevents backups to iTunes and iCloud from containing the unencrypted messages along with any spyware/mercenary-ware from exfiltrating unencrypted messages from the shared application container locations. WhatsApp has chosen to keep the messages decrypted so that they can be recovered from backup or transferred to a new phone (or even phone brand) [2]. Single has chosen privacy over portability and usability in many areas, while WhatsApp is focused on usability and portability over the additional layers of security controls.

Keep up the great work with teaching/engaging critical thinking skills with your podcasts.

-Chris

Chris provided references to back up his statements. And this tracks with what we were saying last week. Both apps Signal and WhatsApp, being based upon the Signal protocol, are doing a terrific job of encrypting and protecting their users' communications on the wire. But whereas Signal further deliberately encrypts its user's communications history stored on the device, WhatsApp has chosen not to. Thanks for the verification, Chris.

Matt Oliver

Hi Steve, It's Thursday evening here in New Zealand and I am continuing with my weekly regime of watching / listening to Security Now. I was super surprised tonight with your mention of crypto ATMs and their fraudulent use, as the New Zealand Government just yesterday announced they are introducing a bill to outright ban them in New Zealand as part of an anti money laundering bill update.

They are also heavily limiting cash transfers out of New Zealand, but that's another story. It will stop these criminals from being able to extort cash from unsuspecting victims, but if someone in NZ wants to invest more than $5,000 overseas it seems they'll be limited to breaking it up into multiple transactions. I'm not too sure where the line should be.

It seems like only a few weeks since you shared episode 1000, how can you be at 1033 already!

It does appear that the Crypto/ATM business may be in for some rocky times. We know that the creation of cryptocurrency really opened the floodgates for extortion and ransomware. Crypto-currency is so handy but also so inherently prone to abuse. So I suppose it should come as no surprise that Crypto/ATM's would be no less abuse prone.

Jeff

Hello Steve, On Security Now podcast #1033, you discussed the Apple and WhatsApp responses to the Israeli spyware vendor attacks, and in particular, you mentioned that WhatsApp was able to remediate the PDF FreeType font rendering vulnerability issue on the

server-side without changes to the client.

Since WhatsApp is supposedly end-to-end encrypted, how could this have been implemented? Meta should have no visibility into chat contents, although perhaps non-text content is sent in the clear, which would be an interesting admission on its own, especially since CSAM and other illegal non-text based content is used as a justification to add backdoors to encrypted communications. Unless I'm missing something obvious, this seems like a Very Big Deal, but would want to confirm the reporting before jumping to any conclusions. /Jeff

Jeff's note was similar to that from many of our astute listeners who pointed out the same issue and asked the same question, which amounted to: Meta said that they were able to deal with the Apple PDF 0-day on the server side without requiring their clients to be updated. But if all of WhatsApp's communications are end-to-end encrypted, with the bad guys sending maliciously formatted PDFs to targeted victims, how would that have been possible?

And that is what's known in the industry as a "damn good" question. Last week I glibly and incorrectly assumed they could just scan any PDFs for the deliberately malformed exploit and refuse to deliver it. But they obviously cannot do that. Meta should have absolutely ZERO visibility into any of the content being transacted.

So to Jeff's and many of our other listener's points, there's really nothing at all that Meta should have been able to do at the server side. The only thing I can imagine is that perhaps WhatsApp doesn't attempt to contain complete rendering code internally. It might pull PDF rendering code from the server on-the-fly under some circumstances. If that's the case, they could have separately patched that PDF rendering module and all subsequent instances of WhatsApp would have essentially fixed themselves without their users needing to do anything.

A definite tip of the hat to all of our listeners who are really paying close attention.

Casey

Steve, Just wanted to share today a neat tool from the EFF called Cover Your Tracks (https://coveryourtracks.eff.org/)*. This is a useful tool for anyone who wants to test and better understand browser fingerprinting. A coworker had shared news of a breach that didn't seem completely straightforward from Cybernews (cybernews.com) that I was analyzing when I had found this link to the EFF tool. Many articles from the CyberNews site seemed to promote their own tools and to engage users most likely for advertising purposes, but one stood out since it correlated with the Browser Fingerprinting episode you recently shared.*

I would be interested to know your take on this site, and tools like their password leak checker. I'm skeptical of their interpretation that 16 billion credentials were leaked earlier this year, and skeptical of their recommendations which seem biased.

This dive gave me more reasons why I am happy to be able to hear from you each and every week! I've been following your work since before the days you published the experiences with the once-named "Wicked" script kiddie and the adventures you had infiltrating the IRC site that was being used to DoS your site.

I won't tell you how old I was at that time… but boy are those classic! Can't count the number of times I've used and recommended SpinRite, Security Now, Vitamin D, the healthy sleep formula, the picture of the week, and the many more things you share. Let's just say that even while on vacation I couldn't help but think about you and Security Now while visiting Irvine and the beautiful surrounding area…😊 Thanks for everything, Casey C.

Casey, thanks so much for your note. I'm so tickled that so many of my wanderings have been useful to you and your colleagues.

I'm aware of the EFF's CoverYourTracks page, and in past years we've talked several times about their earlier effort "Panopticlick", which they released at v3.0 back in 2017. But the https://CoverYourTracks.eff.org site and work is their latest and it serves as a perfect follow-on to our recent discussion of browser fingerprinting.

I just went there with my Firefox browser which is running uBlock Origin and Privacy Badger. The EFF's Cover Your Tracks site performs proactive fingerprinting of the same sort that's performed by tracking sites and shows what THEY are able to see. After auditing my browser, the site replied:

Here are your Cover Your Tracks results. They include an overview of how visible you are to trackers, with an index (and glossary) of all the metrics we measure below.
Our tests indicate that you have strong protection against Web tracking.

Is your browser:
Blocking tracking ads? Yes
Blocking invisible trackers? Yes
Protecting you from fingerprinting? Your browser has a unique fingerprint

Note: because tracking techniques are complex, subtle, and constantly evolving, Cover Your Tracks does not measure all forms of tracking and protection.

Your browser fingerprint appears to be unique among the 244,246 tested in the past 45 days.

Okay. Ouch. A bit shy of one quarter of a million visitors have used CoverYourTracks over the past 45 days and not one of those visitors' browsers had the same fingerprint as mine. Hmmmm.

We estimate that your browser has a fingerprint that conveys at least 17.9 bits of identifying information and the measurements we used to obtain this result are listed below.

Web Headers

Whenever you connect to a website (in our case, "https://coveryourtracks.eff.org"), your device sends a request that includes HTTP headers. These headers contain information like your device's timezone, language, privacy settings, and cookies. Web headers are transmitted by your browser with every site visit.

Some details about your browser can be discovered by using JavaScript code. This includes

fonts, certain details about your hardware configuration, and your canvas fingerprint.
User Agent:
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:140.0) Gecko/20100101 Firefox/140.0

A web header that relays information to the web server about your browser and its version. How is this used in your fingerprint?

This information can be very specific. If customized, it can single-handedly identify a specific user's browser.

Bits of identifying information: 4.41. One in 21.21 browsers have this value.

HTTP_ACCEPT Headers
*text/html, */*; q=0.01 gzip, deflate, br, zstd en-US,en;q=0.5*

A web header that is used to let the server know what types of content the browser is able to handle. How is this used in your fingerprint?

This information can be fairly unique, and varies from browser to browser. However, this string doesn't tend to change much over time, and can remain constant through many versions of the same browser.

Bits of identifying information: 1.91. One in 3.76 browsers have this value.

Plugin 0: *Chrome PDF Viewer; Portable Document Format; internal-pdf-viewer; (Portable Document Format; application/pdf; pdf) (Portable Document Format; text/pdf; pdf).*
Plugin 1: *Chromium PDF Viewer; Portable Document Format; internal-pdf-viewer; (Portable Document Format; application/pdf; pdf) (Portable Document Format; text/pdf; pdf).*
Plugin 2: *Microsoft Edge PDF Viewer; Portable Document Format; internal-pdf-viewer; (Portable Document Format; application/pdf; pdf) (Portable Document Format; text/pdf; pdf).*
Plugin 3: *PDF Viewer; Portable Document Format; internal-pdf-viewer; (Portable Document Format; application/pdf; pdf) (Portable Document Format; text/pdf; pdf).*
Plugin 4: *WebKit built-in PDF; Portable Document Format; internal-pdf-viewer; (Portable Document Format; application/pdf; pdf) (Portable Document Format; text/pdf; pdf).*

A plugin is a small piece of software that helps a browser manage content it is unable to process on its own. Browser plugins have been phased out over the past few years. Instead, browsers favor more regulated add-ons and extensions. But plugins persist in older browsers.

Bits of identifying information: 0.64. One in 1.56 browsers have this value.

Time Zone Offset: *420*

This metric is a string which indicates your time zone, like 'America/Los_Angeles'. This metric can be used to figure out your general location, especially if you live in a time zone without many other users.

Bits of identifying information: 4.11. One in 17.31 browsers have this value.

Time Zone: *America/Los_Angeles*

This metric is a string which indicates your time zone, like 'America/Los_Angeles'. This metric

can be used to figure out your general location, especially if you live in a time zone without many other users.

Bits of identifying information: 4.4. One in 21.05 browsers have this value.

Screen Size and Color Depth: 3840x1600x24

The dimensions of your current browser window, and its color depth. While this metric can supplement other information, it's often too 'brittle' to be usable by trackers because users can easily change their browser window dimensions.

Bits of identifying information: 11.65. One in 3213.76 browsers have this value.

System Fonts:
Arial, Arial Black, Arial Narrow, Calibri, Cambria, Cambria Math, Comic Sans MS, Consolas, Courier, Courier New, Georgia, Helvetica, Impact, Lucida Console, Lucida Sans Unicode, Microsoft Sans Serif, MS Gothic, MS PGothic, MS Sans Serif, MS Serif, Palatino Linotype, Segoe Print, Segoe Script, Segoe UI, Segoe UI Light, Segoe UI Semibold, Segoe UI Symbol, Tahoma, Times, Times New Roman, Trebuchet MS, Verdana, Wingdings (via javascript)

To determine your system fonts, tracking sites commonly display some text in an HTML tag. Trackers then rapidly change the style for that span, rendering it in hundreds or thousands of known fonts. For each of these fonts, the site determines whether the width of the span has changed from the default width when rendered in that particular font. If it has, the tracker knows that font is installed.

The list of fonts you have installed on your machine is generally consistent and linked to a particular operating system. If you install just one font which is unusual for your particular browser, this can be a highly identifying metric.

Bits of identifying information: 3.2. One in 9.17 browsers have this value.

Are Cookies Enabled? *Yes*

As a metric, "cookies enabled" is either 'True' or 'False', and means your browser allows cookies, rather than blocking them. Whether cookies are enabled can be determined with or without the use of JavaScript. Whether cookies are enabled or not provides a single bit of information: either 'true' or 'false.' However, this feature can be far more identifying when combined with other details.

Bits of identifying information: 0.07. One in 1.05 browsers have this value: 1.05

Limited supercookie test: *DOM localStorage: Yes, DOM sessionStorage: Yes, IE userData: No, openDatabase: false, indexed db: true*

Despite the name, "super cookies" are not technically cookies. While they also store and retrieve unique identifiers, they are much harder to detect and delete in comparison.

Super cookies can monitor what websites you visit and how long you spend on them. Super cookies can also access data collected by traditional tracking cookies, like login information. After the traditional cookie has been deleted, the super cookie will still be able to reference it. How is this used in your fingerprint?

The list of super cookies available to a tracker can be very revealing. Most browsers no longer support Flash animations, but if yours does this can be used as an extra fingerprinting metric.

Bits of identifying information: 0.18. One in 1.13 browsers have this value.

Hash of canvas fingerprint.

A tracking site can perform a specific test on the HTML5 <canvas> element in your browser. This metric is the unique identification the tracker assigns to your browser after it performs this test. Canvas fingerprinting is invisible to the user. A tracker can create a "canvas" in your browser, and generate a complicated collage of shapes, colors, and text using JavaScript. Then, with the resulting collage, the tracker extracts data about exactly how each pixel on the canvas is rendered. Many variables will affect the final result. These include your operating system, graphics card, firmware version, graphics driver version, and installed fonts.

This is a complex and very reliable fingerprinting metric for trackers.

Slightly different images will be rendered due to small differences in:

video card hardware,
video drivers,
operating system, and
installed fonts.

These settings are different from one computer to the next. But they tend to be consistent enough on a single machine to clearly identify a user.

Bits of identifying information: 1.57. One in 2.97 browsers have this value.

Hash of WebGL fingerprint.

WebGL is a JavaScript API for rendering interactive 2D and 3D graphics. The method for generating a "hash of WebGL fingerprint" is very similar to generating a "hash of canvas fingerprint." Its method is to use your browser to generate graphics, extracting data from how each pixel is rendered, serialize the result, and hash it.

The WebGL and canvas fingerprinting results are closely linked. They both examine browser-rendered graphics for tiny differences between users.

Bits of identifying information: 1.72. One in 3.29 browsers have this value.

WebGL Vendor & Renderer: *Google Inc. (Intel)~ANGLE (Intel, Intel(R) HD Graphics 400 Direct3D11 vs_5_0 ps_5_0), or similar*

WebGL is a library that allows browsers to render 3D graphics. As with other graphics-based tracking methods, trackers look for any tiny differences between how your device displays 3D on the web compared to other users.

This metric provides some level of granularity, depending on how unique your video card is. The WebGL Vendor and renderer is directly searchable using JavaScript, so trackers can access it without issue.

Bits of identifying information: 6.32. One in 79.85 browsers have this value.

DNT Header Enabled? *True*

A web header that is used to let the server know if you prefer not to be tracked. This is usually either not delivered at all, or set to '1'. A setting of '1' indicates that your browser would prefer not to be tracked. Unfortunately, most sites ignore this request.

Browsers which set the DNT header to '1' are fairly rare, and this can be an identifying metric. However, this should be left as the default for your browser.

Bits of identifying information: 1.22. One in 2.34 browsers have this value.

Language: *en-US*

This metric notes languages you prefer site content to be delivered in. This can add a fair amount of information to your browser fingerprint. This is especially true if the language is uncommon for your timezone. While some other fingerprinting metrics can be protected by the browser or add-ons, this is not possible for language. Spoofing the language header would greatly impede usability.

Bits of identifying information: 0.87. One in 1.83 browsers have this value.

Hardware Specs *Platform / Win32*

This metric includes your operating system and CPU (central processing unit) architecture and is directly searchable by trackers using JavaScript. This can either be very unique or very commonplace, depending on your particular machine.

Bits of identifying information: 1.29. One in 2.44 browsers have this value.

Touch Support: *Max touchpoints: 0; TouchEvent supported: false; onTouchStart supported: false*

This metric refers to the number of touch points on a device, such a tablet or phone. If you are using a mobile device, this may be very identifying depending on the hardware particularities. Your result will be 0 if your device has no touch points.

Bits of identifying information: 0.86. One in 1.81 browsers have this value.

AudioContext fingerprint: *35.749972093850374*

This metric is like canvas fingerprinting, but for audio rather than graphics. An audio sample is generated. That audio sample is then serialized and measured to provide this fingerprint. Like canvas fingerprinting, this can be unique depending on your audio card and drivers, and usually will not change over time. In modern handheld devices and laptops, graphics cards and audio cards will vary depending on the model. But they will not change between devices of the same model. For desktop computers, especially ones with customized hardware, the audio card will provide new information. This information is useful for fingerprinting.

Bits of identifying information: 2.27. One in 4.84 browsers have this value.

Hardware Concurrency: *8*

This metric notes the number of CPU cores in your current machine. This can provide some additional information when combined with other fingerprinting metrics, but is not identifying on its own.

Bits of identifying information: 2.27. One in 4.82 browsers have this value.

So we have all of these many parameters, none of which taken alone is useful for identifying an individual browser. But each parameter contributes a few unique bits or fractions of a bit, and JavaScript and browser headers are able to provide so many orthogonally unrelated bits that when they are all taken together, as we saw at the top, my browser was utterly unique among more than 244,000 that had visited that site in the past 45 days.

Since they are only considering browser fingerprints, they didn't consider my IP to be another block of relatively static bits.

So the takeaway here is that anyone who is doing nothing more than occasionally clearing their browser cookies is likely not accomplishing nearly as much as they hope. Today's browser tracking technology has become serious.

Thank you, Casey, for reminding me to talk about this fantastic EFF "CoverYourTracks" facility!

Paul

Steve, Listening to yesterday's podcast, and the type memory overflow Paragon used for WhatsApp PDF attacks made me wonder something. If all interpreters were rewritten in memory safe languages, would they be problem free? Or just have a way smaller attack surface? I'm not sure if the problems with interpreters have historically been memory safety related, or other problems like input sanitation not being checked. If it is input related, from my understanding that wouldn't be fixed by memory safe languages. Thanks, Paul

That's a terrific question, Paul. The fairest way to evaluate this, I think, would be to observe that the problems with interpreters have historically been the same as the problems with any other class of code, but that interpreters tend to be problem magnets due to the particularly large attack surface they present.

It is an interpreter's inherent nature to be heavily directed by the **content** of the data they're interpreting. That's the unique characteristic that makes interpreters inherently more prone to abuse. There's much greater opportunity for their abuse by way of abusing the formatting of that data. By comparison, for example, code whose functioning is entirely independent of the data it is processing is inherently far less vulnerable.

So, yes, the benefit derived from writing interpreters in memory-safe languages would likely be significant, not because interpreters are necessarily more prone to the problems that are often resolved through the use of memory-safe languages, but because interpreters are just generally

so much more prone to all problems of coding, and memory-safe languages have been shown over and over to be terrific for preventing one large class of memory-related coding mistakes.

Another way of expressing what Paul suggests is that if one wanted to begin the work of rewriting a large body of code in a memory-safe language, a great place to begin that work to quickly realize the largest increase in security and operational integrity would be anywhere the functioning of the code is subject to the specific contents of its data. That's the nature of interpretation and the location of an historically oversized percentage of coding mistakes.

Introduction to Zero-Knowledge Proofs

The recent issues surrounding the growing pressure to create some means of providing online age verification, with its accompanying worrisome implications for privacy, has brought a somewhat obscure but quite interesting bit of academic technology into the foreground. This area of study and recent development for computers is known generically as "Zero Knowledge Proofs." This is sometimes abbreviated ZK Proofs or ZKPs.

When discussing cryptographic protocols, as we have often done here and as is typically done in the literature, the various participants are given representative names. "Alice" is typically the initiator of a communication. "Bob" is the recipient of that communication. "Eve" is the name given to someone attempting to eavesdrop on that communication between Alice and Bob. "Mallory" is a malicious attacker or man-in-the-middle. And if additional participants are needed for multi-party communications, the presence of "Carol" and "Dave" may be required.

For today's discussion we're going to introduce two new characters: "Peggy" is someone who wishes to **prove** something and "Victor" is the **verifier** of what Peggy wishes to prove.

A formal description of any zero-knowledge proof will read something like this:

> A zero-knowledge proof is a protocol by which one party (the prover) can convince another party (the verifier) that some given statement is true, without conveying to the verifier **any** information beyond the mere fact of that statement's truth.

In other words, the verifier learns **nothing** beyond the truth of an assertion that the prover wishes to make. By comparison, instances of what we might term a "Knowledge-based proof" surround us. For example, anyone can prove their possession of some information simply by revealing it. The tricky "zero-knowledge proof" part is to prove this possession without revealing this information or any aspect of it whatsoever.

The formal requirements for true zero-knowledge proofs are even higher because if we want true zero-knowledge, there should be no knowledge received that can be passed on to a third party. In other words, the verifier in this setting, even after they have become convinced of the truth of the prover's statement, should nevertheless be unable to prove the statement to any other third party.

Some zero-knowledge proofs can be interactive, meaning that the prover and verifier exchange messages following some protocol, or noninteractive, meaning that the verifier is convinced by a single prover message and no other communication is needed.

I titled today's podcast "Introduction to Zero-Knowledge Proofs" because I wanted to start everyone thinking about this truly fascinating realm without getting bogged down and mired in cryptographic specifics. And it turns out that it's not necessary, since there are some very cool physical examples that we're going to look at today which should serve to get everyone thinking about this topic broadly.

Also, although computer scientists are excitedly talking about zero-knowledge proofs being useful for the online age verification problem, at best, ZKPs will only be one useful part of the solution. To me, the problem mostly involves somehow linking a person's verifiable date of birth to an unspoofable biometric feature. Once you have that, a zero-knowledge proof could be used to prove an assertion of age without revealing anything more. But we don't have any of the

infrastructure in place today to first link a date of birth to an unspoofable biometric. But just so everyone understands that this is not all pie in the sky, on July 3rd, Help Net Security ran a short new piece titled *"Google open-sources privacy tech for age verification"*. They wrote:

> *Age verification is becoming more common across websites and online services. But many current methods require users to share personal data, like a full ID or birthdate, which raises privacy and security concerns. In response, Google has open-sourced a cryptographic solution that uses zero-knowledge proofs (ZKPs) to let people verify their age without giving up sensitive information. The newly released ZKP codebase allows users to prove they are over or under a certain age without revealing their actual birthdate or identity. For example, someone could confirm they're over 18 without showing an ID or even disclosing their exact age.*
>
> *The technology could support a range of use cases, such as restricting minors' access to adult content or verifying eligibility for age-specific services. It also opens the door to meeting legal requirements without building databases of user information. Google says the open-source release builds on years of research into cryptographic protocols and privacy-preserving technologies. The goal is to provide building blocks for more secure, user-respecting identity systems.*
>
> *The release also comes as regulators around the world are pushing for stronger online age checks, particularly for content related to social media, gambling, or adult material. Google notes that these requirements must be balanced with protecting individual privacy. By offering this technology openly, the company hopes to support wider adoption of private, verifiable age assurance methods across the internet. It also highlights that the same approach could be used for other use cases beyond age, such as proving location or income level without revealing exact details.*

Now, again, I want to make absolutely certain that everyone appreciates that this is not some sort of magic age-verification technology. I'm quite certain that no magic technology exists. That article described it as a building block for more secure user privacy-respecting systems, and that's all it is – a building block. An analogy might be the Signal protocol and messaging system. Signal requires the use of the AES-256 cipher to perform the encryption step. But AES-256 by itself is certainly not a messaging application. It's only an algorithm that's able to scramble and unscramble bits based upon a key. It's a necessary thing to have, but it's just a tiny piece of any full secure messaging app. That said, what we see is that workable and working zero-knowledge proof (ZKP) systems do exist and will be ready for us to use once all of the other pieces are in place.

But our goal for today is to develop some sense for what it means to have a zero-knowledge proof system. Let's work to give its definition some context.

A perfect place to start is a classic ZKP demo, known as "Where's Wally". Imagine that there's a large sheet of paper, say two feet by two feet. Printed on this sheet is a solid mass of tiny line-drawn cartoon characters of various colors doing random things. And somewhere hidden in plain sight is Wally. But nothing obviously distinguishes Wally from any of the other characters. And there is so much going on, so much visual noise printed on this four square foot sheet, that even knowing what Wally looks like, even being able to recognize him if you saw him, there's just no way to find him hiding in plain sight among everything else that's going on.

Okay. But Peggy the prover in our zero-knowledge proof example knows where Wally is. And she wants to prove to Victor the verifier that she knows. Before this, Victor has been visually scouring the four square foot sheet over and over. He's looked everywhere, and multiple times,

until he's become convinced that Peggy is full of it and that there's no Wally printed anywhere on the sheet. Peggy claims otherwise. She's rather proud of her discovery and has become somewhat annoyed with Victor. So she's decided that she's not going to show him where Wally is. She wants to keep Wally's location a secret from Victor even after proving not only that Wally does exist on the sheet but also that she knows where he is. If she can pull this off it's going to further drive Victor nuts because he'll know that there's a Wally there somewhere, but still have no idea where.

So how does Peggy create a zero-knowledge proof of Wally's existence on the sheet while keeping his location secret? She gets another much larger sheet of paper, double the length in each dimension, so it's 4 feet by 4 feet. And in the center of that larger sheet she cuts a small hole that's the size of Wally on the printed sheet. She positions the top cover sheet over the printed sheet with Wally's image visible through the hole. ***There's Wally!***

Looking at what Peggy has done, Victor cannot deny that Wally is there. She's proven that. And the fact that Peggy was able to place the large cover sheet over the printed sheet so that Wally's image appears though the little hole in the center of the cover sheet further proves that Peggy knows where Wally is on the printed sheet. But thanks to the fact that the cover sheet is twice as long in each dimension as the sheet it's covering, the printed sheet would be covered up no matter where on that sheet Wally was printed.

Peggy has accomplished her task. She has definitively proven to Victor that Wally exists on the original printed sheet while giving him absolutely no information about where Wally might be. Peggy the prover constructed a zero-knowledge proof of Wally's existence and that she knows his location.

There's something else worth noting here which is part of the definition of zero-knowledge proofs and which can be important in some applications: Although Victor is now utterly certain that Wally exists and that Peggy knows where Wally is, Victor is unable to now prove that to anyone else. He may assert that Peggy has proven her knowledge of Wally's location to him, but Victor has no way of proving to any third party what was proven to him.

Let's take another example to demonstrate some other properties of this interesting realm.

Two competitors have, so far, both been allowed to purchase some number of a rare and precious item from a common supplier. The supplier claims that both parties were allowed to purchase the same number of these items, but as part of the purchase agreement the supplier made them each sign an NDA – a non-disclosure agreement – which has bound them to keep the number of items they were each sold secret, especially from one another. And if they violate that NDA, they will lose any opportunity to purchase any more of these precious items.

The problem is, they're competitors, they need these things, and they don't trust the supplier's claim to have, so far, sold them the same number of them. So they want to verify – or in this case we would say "prove" – that they have both purchased the same number of items without breaching their NDA and revealing the number of items each has purchased.

The items are only sold in lots of 100, so they may have each purchased 100, 200, 300 or 400 of the items. They know how many **they** have purchased but they don't know how many their competitor has purchased. And neither of them is allowed to reveal their purchase quantity to the other. What they want to know is whether the seller has told them the truth about having sold them each the same number of units – yes or no?

What they need is a zero-knowledge proof, and over drinks one night they devise a way to accomplish this.

They get four identical small lockable boxes each having a differently keyed lock. Into the top of each box they cut a small slit through which a piece of paper can be slipped – like a little ballot box. The four boxes are labeled 100, 200, 300 and 400, each box is locked with its respective key, which is left in the lock, and all four are placed alone on a table in a room with a door. They also prepare four slips of paper, one having a big green checkmark and the other three having a big red "X".

The two competitors gather outside the room containing the four boxes and flip a coin to decide who goes first. The winner of the coin toss enters the room and closes the door behind him. He goes to the box which represents the quantity of items he has purchased – 100, 200, 300 or 400 – removes its key and places it in this pocket. He also removes the other three keys from the other three boxes and brings them out of the room with him. Together they destroy the three keys for the other three boxes that will never be opened.

Next, the other competitor takes the four slips of paper into the room and closes the door behind him. He drops the slip of paper having the big green checkmark into the top slot of the box which corresponds to the quantity of items he has been allowed to purchase from their common supplier – 100, 200, 300 or 400 – and drops the three big red "X" slips into the other three boxes. He then exits the room.

Finally, the first person who removed and retained the key, which only opens the box corresponding to the quantity of items he has purchased, enters the room and closes the door. He returns to the box for which he has the key, uses the key to open the box and withdraws the slip of paper that box contains. He relocks that box and exits the room with the slip of paper and shows it to his competitor.

Only if the box he had the key for was the same box as his competitor dropped the green checkmark paper into will he have been able to successfully withdraw a piece of paper containing a green checkmark – and in that case they will have confirmed that they have each been able to purchase the same quantity of items from their shared supplier.

But if the first competitor withdrew a slip of paper with a red "X", they will both know that they have been lied to by their seller, but that is all they will know. Neither of them will have learned how many of the items the other has been able to purchase so they will not have disclosed that to the other and they will not have breached their purchase agreement.

The zero-knowledge proof upshot of this is that the only thing they are able to prove – and the only knowledge that they have been able to obtain – is verification of their seller's assertion that he has sold each of them the same number of units. If that had not been proven, they would know that, too, but still nothing more.

Let's now look at another style of zero-knowledge proof, involving statistical proof. This is another famous thought experiment often referred to as Ali Baba's Cave. For this we return to Peggy and Victor.

A cave is discovered which has an odd shape. It has a cave tunnel shaped in a ring, with an entrance in the side of the mountain on one side of the ring-shaped tunnel, and a locked door which completely blocks the tunnel at the opposite, far side of the ring. So the locked door, which is far away from the tunnel's opening to the outside, cannot be seen since it's deepest in the back of the ring tunnel.

Peggy, who is again our prover for this, claims to have discovered the magic word that can unlock the door from either side. But once again, our Victor is skeptical. Before this, Victor had tried every word he could think of and no matter what he says to the door, it remains stubbornly locked. He doubts Peggy's claim to have discovered the magic word – he thinks he know all the words she knows – and especially after that business with Wally he's more than a bit annoyed with her.

For her part, Peggy is willing to work to convince Victor that she knows the magic word, but she insists upon doing that in a way that cannot also be used to prove it to anyone else. This is crucial to her because Peggy is a stickler for details. Remember that I said earlier:

> *The formal requirements for true zero-knowledge proofs are even higher because if we want true zero-knowledge, there should be no knowledge received that can be passed on to a third party. In other words, the verifier in this setting, even after they have become convinced of the truth of the prover's statement, should nevertheless be unable to prove the statement to any other third party.*

So, for example, if Victor and Peggy were to simply stand at the cave's opening with the two tunnels heading off in opposite directions, clockwise and counter-clockwise around the ring, and if Peggy were to go down one path and a few minutes later emerged from the other, Victor would obviously be immediately convinced that Peggy was able to open the door with the magic word because the only way she could have completely circumnavigated the ring tunnel would be if she were able to open and pass through the magic word door.

But if Victor were to record Peggy's accomplishment with his phone's video camera, or if someone other than Victor happened to also be standing there too, watching Peggy do this would constitute incontrovertible proof of her grasp of magical cave door operation, and that would be unacceptable to her. So she refuses to do that.

As we've already seen with her use of a large sheet of paper to frustrate Victor, Peggy is pretty clever. She's come up with a way to prove to Victor – and Victor alone – that she can pass through that door at the far backside of the ring tunnel cave while, at the same time, preventing a video recording from creating solid evidence, or even evidence for someone standing by and silently observing. Here's Peggy's solution:

Once again, Peggy and Victor stand at the mouth of the cave with the two tunnels diverging. Peggy has Victor turn around so that she is unobserved and has him start counting down from 10. She keeps her eye on him while she disappears from sight down the tunnel of her choosing. Once Victor's count reaches zero, he turns around and shouts the tunnel he wants her to emerge from. If Peggy happened to go down that side of the ring she simply retraces her path. But if she went down the other side, she must use her magic word to open the door and emerge from the path Victor has requested.

So what do we and Victor know at this point? We know that there's a 50/50 chance that Peggy may have initially gone down the same path that Victor asked her to return from, so she would not have needed to use her magic word. Victor knows that she got it correct once, but that might have just been beginner's luck. So they do it again.

As before, Victor turns his back, counts down from 10, Peggy herself chooses a direction, gets to the door and waits for Victor to shout out which path he wants her to return from. Since Peggy does know the magic word, she is always able to succeed.

But if she did not know the magic word, and if they kept playing this game, the chances get greater and greater that Victor will ask her to return on the path opposite the one she went down, and Victor will have his – uh, Victory – of proving that she never did know the magic word.

We know how the statistics go: one test is 50/50. Two tests where both must be correct, assuming an equal probability of outcome, is 1 in 4. Three tests is 1 in 8. Four tests 1 in 16, then 1 in 32, 64, 128, 256, 512, 1024, 2048, 4096 and so on. To get to 4096 only requires 12 runs. So if Peggy did not know the secret door-opening word, there would only be one chance in 4,096 of her being able to get the path correct all 12 times. Before long, annoyed as he may be, Victor will give up and admit that Peggy must be able to cross that door's threshold. Either that, or she's incredibly lucky and needs to go to Las Vegas.

So how has this statistical variation solved Peggy's concern about keeping her magical locksmith abilities unproven to anyone else? Assuming that a camera or an observer were to turn around and be unable to see the path she initially took – which is the obvious requirement – there would be no way for a video recording or an on-the-spot observer to know that she and Victor were not conspiring by having pre-arranged the sequence of tunnels Victor would call out and Peggy would therefore choose to go down and then return from. They could have some system, like a famous phrase "Ask not what you can do for your country" where a vowel means take the left tunnel and a consonant means go to the right. This would allow Victor and Peggy to stage the entire event for an audience... and it gives Peggy the plausible deniability that she requires.

If confronted by someone who stood there or watched the video, she could ask: "How gullible are you? Victor and I simply prearranged the sequence of tunnels. Why else do you imagine I didn't just let people directly observe which path I initially took? It was so that we could stage the whole thing."

There are many other physical world constructions involving colored balls and decks of cards, but everyone should by now have a good idea of the goal. There are clearly ways to prove some statement or assertion involving other conditions without disclosing any other information about those conditions. The proof can be made while leaking zero knowledge.

As an academic pursuit, a great deal of time and attention has been devoted to the formalization of Zero-Knowledge Proofs. A zero-knowledge proof of some statement must satisfy three properties:

1. **Completeness:** if the statement is true, then an honest verifier who is following the protocol properly will be convinced of this fact by an honest prover.

2. **Soundness:** if the statement is false, then no cheating prover can convince an honest verifier that it is true, except with some acceptably small probability.

3. **Zero-knowledge:** if the statement is true, then no verifier learns **anything** other than the fact that the statement is true.
    a. Victor **only** ever learned that Wally was, indeed, printed somewhere on the sheet, but he never learned anything about where.
    b. Our pair of competitors **only** learned whether or not they had both purchased the same quantity of goods.
    c. And later, Victor was **only** ever able to convince himself that, despite himself, Peggy must indeed know the magic cave door opening word. And she was willing to prove it as many times as he needed her to until he was convinced.

Those first two properties – Completeness and Soundness – are generally true of many interactive proof systems. They're nothing special. If the assertion is true, the prover will be able to convince an honest verifier of this, and if the assertion is false, it's not possible for a prover to fool the verifier into believing it's true.

But it's the third property of Zero-Knowledge – that the verifier is unable to learn **anything** other than the truth of the assertion that's where the magic happens and turns the proof into a ZKP.

Zero-Knowledge Proof technology is around and it is currently in use for many privacy-centric purposes. It's used where it's necessary to prove an assertion – the like knowledge of a username and password – but without revealing anything about the username or password. It may be that it will eventually lie at the heart of some future age verification system suitable for use by the Internet and elsewhere. But, as I started off saying at the top, I'm not holding my breath because a great deal of supporting infrastructure will be needed before there's anything for any slick ZKP technology to do.

But the next time you're at a cocktail party – are those even a thing any longer? – and you're being annoyed by someone asking what you do, you can have some fun talking about Wally and magic caves and probably be left alone pretty quickly.

We'll see everyone next week for episode #1035.