ICANN85 Mumbai | CF – GNSO: CPH TechOps Work Session (1 of 2)
Sunday, March 8, 2026 – 09:00 to 10:00 IST

DEVAN REED

Hello and welcome to the CPH TechOps Work Session 1 of 2. Please note that this session is being recorded and is governed by the ICANN Expected Standards of Behavior, the ICANN Community Anti-Harassment Policy, and the ICANN Community Participant Code of Conduct Concerning Statements of Interest.

Please observe the following guidelines to participate in this session. I will also post them in the chat for your reference. Only questions posted in the Zoom chat identified as a question will be read aloud during the session, as time permits, and when directed by the chair of the session.

If you wish to speak, please raise your hand in Zoom or otherwise as directed. When speaking, please state your name for the record and speak clearly at a moderate pace. I will now hand the floor over to you. Thank you.

ROGER CARNEY

Thanks, Devan. Welcome, everyone. It's good to see everyone here and a good group online as well, so it's nice to see. This is my first meeting this week where I think there's just as many registries as there are registrars, so that's nice to see. That's good. Okay, I think that there's not a lot to cover here.

One thing we're going to do, our second session is probably a little more weighted, so if we get through our first session, we're just going to start the second session in the first hour. So, if we don't, that's fine, but we'll try to move that up if we can. I think that's all for me.

The Zoom room will be the same for both sessions, so if you want to stay on, you can. The recording will stop and restart. But anything else? I don't have anything. I think Zoe already covered. If you're in the room, please log in, and maybe I'll check with Marc here to see if he wants to add anything.

MARC ANDERSON Thanks, Roger. This is Marc Anderson, and maybe just give a general welcome to everybody. As we kick off the TechOps sessions at ICANN, we generally give a little bit of an intro to TechOps. I see on the list many names I'm familiar with, but for anybody that's not familiar with TechOps, we're a joint group made up of registries and registrars in the registry and registrar stakeholder group, respectively.

Roger, from the registrars, and myself, Marc Anderson, from the registries, are your co-chairs. The purpose of TechOps is to provide a forum for registries and registrars to talk about technical items of common interest between the two groups, to create an easy venue or forum for us to talk about issues of common importance to both registries and registrars.

While generally, TechOps membership itself is closed to just members of the Registry and Registrar Stakeholder Group, by

tradition, during ICANN meetings, we have our sessions as open, and they're available for anyone to join. If you are not from the Registry and Registrar Stakeholder Groups, welcome. Thank you for being here, and please feel free to participate and speak up.

We have today with us Peter Thomassen, who's been a guest speaker with us before, and he's been very well received, much so that we asked him to come back again today to talk to us. Thank you, Peter. We appreciate you coming, and it's been great having you with us in the past, so I look forward to hearing what you have to say today.

Maybe one last item on my checklist. It's a little bit of a reminder to myself, but I know we're in the middle of ICANN85, but coming up, we have the CPH Summit at the end of April, and we have a brief TechOps session, and so we're going to have updates from TechOps for or non-technical updates from TechOps during the CPH Summit, and so keep an eye out for that. We'll be looking for volunteers to talk about topics of interest and give general updates for that.

I don't think we'll get into actual planning during ICANN85 for that, but the CPH Summit's coming up really quick after ICANN85, and so we'll definitely try and get out in front of that and get speakers and topics shored up very quickly. Let me throw it back to you, Roger, see if there's anything else we need to cover before we give the floor to Peter. Thank you, and again, welcome, everybody.

ROGER CARNEY

Great. Thanks, Marc, and I'll just add to that. I don't know that we've finalized on a schedule post-85 yet, Zoe, for the TechOps meetings, but we'll be doing that, and as Marc mentioned, we're probably less than six weeks away from the CPH Summit once we leave here, so we'll probably have one meeting or so before the summit for the TechOps group, so hopefully we can get a lot of our planning done on list for the summit.

Again, we just have the one session. They made the decision for the summit to be single-track again. It seemed that most of the participants liked that, so we stuck with that, so we just have the one session, so it won't be a lot of planning, but if you haven't responded, I think Zoe did send out a poll for meeting time, so please respond to that so we can get it set up on a regular cadence again that's not stepping on other meetings, so. Okay, any other questions or comments before we get started? Okay, great. I think I'll turn this over to Peter. Thank you.

PETER THOMASSEN

Hello. Peter Thomassen. So, before I get started, I would like to take a moment to appreciate the women who are in the ICANN community and in this room and online. Today is International Women's Day, so thank you for that, and I'm glad everybody's here, and yeah, I think it's great to have this day. Thank you.

So, let me share my screen. Replace current share. Here we go. All right. I was invited to talk about DNSSEC or something else I would like to talk about. I thought maybe this time I'll not talk about

DNSSEC except in the end when you have a question. Instead, I'd like to talk about general aspects of security that you perhaps always wanted to know without knowing that, and so this is supposed to be not too technical and instead quite informative.

What I'll say is also mainly on the slides, so it may seem like I'm reading the slides, but it's more like I want this deck to be also a reference to you. I will fade in things slowly, so you don't have to read the whole slide or anything, just to explain the mode, and if you have a question, maybe somebody could monitor. If RANS or HASTE, we can make this more interactive if that's desired, and so anyway, much of security follows a small number of principles, and so that's why I'm trying to show that.

All right. So, what's a hash? You might have heard that certain things in the DNS or in security in general involve something that's called a hash, and perhaps you have wondered what that is or you have not wondered what that is. In any case, you might not have known what it is, and so let me quickly give you a feeling for it without doing any math or anything.

You might have heard about check digits. For example, if you look at ISBN numbers, which are identifiers for books, so the first row here is an ISBN number, and the last digit there is a check digit. It's currently not filled in because it is supposed to match all the other numbers, so to speak, and be consistent with them, and so if there is a typo, for example, in any of the digits, then you can determine by looking at the check digit that there was a problem in typing the number.

And so, the way that the check digit is computed is that you take each number from the front and multiply them with 1 and 3 in alternating fashion, so 9 times 1 and 4 times 3, 5 times 1, and so on and so forth, so that's written down in the red line at the bottom, and then you sum all this stuff up, and so you get 74 in this case, and the last number is such that when you add it, the sum is divisible by 10.

So, here if you add 6, the current sum would be 80, and that's divisible by 10, and that's why the check digit is 80, and if you make a mistake anywhere, for example, if you switch two numbers, or if you, I don't know, replace the 4 with a 3 here, in that case, the check digit would be rendered invalid, and then you would be able to detect, oh, that is not the correct thing.

So, the digit is sort of a fingerprint for the other numbers, same thing for credit card numbers and whatever. And so, a hash is essentially the same thing, except that it is more general, and it is longer. It's not just one digit, it's actually quite long, so there are different methods to compute hashes. One is called SHA-256 for whatever reason, and the output of that has 256 bits, which translates to 77 decimal digits if you write that down as a number, and so the different hash outputs, the different fingerprints are all different, ideally, for all inputs.

Now, of course, that isn't fully guaranteed, because you can have infinitely many different inputs. They are much longer inputs, and they all have to be compressed into this sort of short checksum number, similar as different ISBN numbers sometimes also have the

same check digit, but in general, the idea is make it such that any input will cause an output that seems pretty unique.

So, when you change any bit in the input or any number or something, even if it's a book and you hash the book, if you change only the slightest detail, then the hash output will change completely, okay? It's supposed to look like a random number. So, you can even use it like a random number generator if you hash something that is random, for example, noise or something. If you hash that, you can produce random numbers.

Then a few other properties are if you have the hash output, then you cannot reconstruct the hash input. That will be interesting later for applications of this, and also if you have one hash input, for example, one book or one signed PDF file, you know, with a contract, you cannot construct another PDF file that produces the same hash output.

So, that would be a collision, and the idea is you can use the hash output as a fingerprint, a shorthand fingerprint, for whatever, for example, the contract in the PDF file is, and you don't want somebody to be able to claim that a different contract would be valid, too, because it has the same hash output. So, the hash is constructed so that that cannot be done. That's a decent fingerprint.

What is this useful for? It's useful for IT security, for example, password storage. So, when you store passwords in a database, ideally you would not store the password directly, because if somebody steals the database, they know all the passwords. It's

much better to store the hash of the password, and then when somebody logs in, the web server takes the password that has been entered into the password field and hashes that again and compares that to the stored hash, and so if that matches, you know that the user who entered the password into the field knew the correct password, because you know no two passwords cause a different hash output, and also, sorry, no two passwords will cause the same hash output.

So, if somebody knows the password, the hash will match. Also, from the password hash, you cannot reconstruct the password. So, if somebody steals the password hash database, they cannot reconstruct the passwords. That's a useful property, because databases get stolen all the time, systems get hacked, hard disks get disposed of, and somebody buys them on eBay, and password databases are on them or something. It's better if it's hashed.

One problem is that you can see which users have the same passwords, because they'll have the same hash, and so if you know one user's password and you find other users in the database that have the same hash, you know they have that password too, and you can access their accounts.

One solution to that is that the password internally before hashing is extended by a few additional letters that are determined per user, for example, by adding the user's email address or some other string that is stored along with the user id, and so that causes the hash input to be slightly different for each user, even when the password is the

same, and by that approach you can avoid the same password looking the same hash. That's called salting. A little bit of spice in India, I guess, and everywhere else is appreciated. So, that's how you store passwords.

Another interesting application of hashes is indexing. How does a search engine find the pages that have a search term on it? How does that work quickly? I mean, you could, of course, store all the pages and then do a linear search, which takes forever. A much better approach is to hash each word that's on each page and make a table that maps each hash that you find from any word to the pages that contain it.

So, for example, if the ICANN website contains the word CPH, you could hash that and some number comes out of it, and then in a big database, you could look at the row that corresponds to this hash number, and next to that there would be a column linking the different pages that have the word CPH on it, and you can maintain that index anytime you access a new page and find new words on it. And then when you look for the word CPH, you just hash it. You go directly to the row in the table and you immediately will find the pages that have the word on it.

The table might not fit one machine because the internet is large, but you can even use the first digits of the hash output to number the server on which you will find the particular row from the table. So, you don't even have to check multiple sections of the database. That's called a hash table, and actually using a hash table prevents

the need for search. It just looks like search, but it's actually just a direct lookup and you know ahead of time where it is. You don't need to search. That's a very cool application, I think, and it makes all our lives much easier.

Another one is authenticator applications for two-factor authentication. You certainly all have scanned a QR code from some website and then it displays six-digit outputs that change every 30 seconds, and when you log in you will have to provide that current number. The way that works internally is that the QR code contains a long number that is stored on the server and it is stored on your smartphone. And anytime you look at that code, that secret number is taken and it is, you know, together with the current time with granularity of 30 seconds, it's hashed. So, every 30 seconds the hash input changes. It's the secret number plus, you know, the current time slice.

So, to speak, just counting from some sort of, you know, whenever the zero time was. And so, when you hash that, you get a hash output of which you take the first six digits and that's the authentication code. It all seems very complicated when you don't know how it works, but actually it's all the same things.

Indexing, password storage, and authenticator apps, they all just use hash mechanisms and they essentially work like the check digit in an ISBN number. What's really cool about authenticator apps is that they work offline. You don't need your phone to be online in order to compute the six-digit check number because it's just a hash

computation of the secret that the phone already has and the current time window.

You don't need to receive a number through text message or something. So, that's pretty secure. All right. That was hashes. Now we go on to encryption. Question?

ROGER CARNEY

Just one thing there. The interesting thing is I didn't know what Peter was presenting here, but the hash thing really fits in well to our transfer discussions as the new transfer authorization code attack will be hashed by the registries. So, it's a good explanation of what that actually means and how that happens. So, thank you, Peter.

PETER THOMASSEN

Yeah. So, the transfer codes are pretty much like authorization passwords and they should be treated the same. So, they should be hashed here and they should be long enough and random.

All right, cool. Okay, encryption. So, you've all been to Ikea probably buying some furniture. So, here's an Ikea style instruction for how encryption works. Let's first focus on the top left and then I will go through the different boxes one by one. You don't need to grasp all of this like at once.

So, on the top left, we have a safe and the safe has a lock and the lock has three positions. It can be at the top. In that case, the safe is open.

The lock is unlocked and you can also turn it either left or right, in which case it is locked and you have to turn it back to unlock it.

Now there is two types of keys. One type of key can be used to turn the lock towards the left and the other type of key can be used to turn the lock towards the right. So, in order to lock and unlock the safe, you need to use both keys. It doesn't matter which order you use, but you need both because you always have to go one direction first and then the other direction after that.

One key is called the public key and you share it with everybody else. You just publish it and say, this is my public key. And so, this is the one that has the globe on it. And the other one is called the private key, which has the little man with a hat on it or woman with a hat, in fact.

All right. So, now encrypting something is in the bottom left box. The first little guy here on the left is a person who wants to encrypt something for somebody else. They will have the public key from the person who is the recipient and they will use the public key to turn the lock on the safe to the left.

Now the safe is locked. Nobody can look inside the safe what's inside. You can transfer it somewhere else. You can send it away using TCP IP or whichever method you want. And so, once it's received by the recipient, the recipient has the private key and can turn it to the right again and can look inside the safe. This is how encryption and

decryption works pretty much. You use the public key for locking it and the private key for unlocking it.

Now the interesting thing about signatures is that it's essentially the same thing except that the order is switched. The signature is something that is not produced by the public for a particular person. Instead, it's produced by a particular person who has the public key for the public so that the public can verify this is in fact what somebody signed.

So, the way that works is that the signer will put something inside the safe and will use their private key to turn the lock towards the right and publish that locked thing somewhere so you can download it or something and everybody else who fetches a copy of that can use the publicly available key to turn the lock back to the left and see what's inside the safe.

Now that in itself is not very useful but the signature doesn't make things secret. So, what you would usually do is that you deliver the plain text of the actual contract for example together with the signature. So, the sender at the top would say this is the contract and here is the safe with the signature in it and then when somebody receives that they can turn back the signature lock with a public key, look inside the safe and the text in there must match the text that was delivered alongside with it.

And that equality of what's inside and outside the safe can only be achieved if the sender had the private key originally because otherwise, he couldn't have prepared the safe contents in a way that

they would match what's in the clear text contract. So, that's what a signature is. It is essentially the same thing as encryption except that the role of the keys is switched at least for the mathematical details.

It may be a little bit of an oversimplification but for practical purposes and for conceptualizing it, it is certainly sufficient. And now if somebody else has a rogue key so there is like a pirate at the top sorry at the bottom here which has a patch on his eye you know they'll have a different key that they guess and then you know the lock will come to a stop and it won't work out.

Anyway, so this is what's called public key cryptography. You have two keys. It's a key pair. They are mathematically related so that one works in a certain mode and the other works in the other corresponding mode. The mathematics problem behind these are for example, multiplying two large prime numbers or finding the two prime numbers from the product.

Those are two inverse operations and multiplying is very fast so that would be using the public key and finding the components is very slow the prime factorization and so, if you ever need the individual factors that would be contained in the private key. So, usually you can compute the public from the private key that's why the private key is secret if you can't compute it the other way around.

You have to make the keys unpredictable by the attacker so it involves randomness. Whatever that means and how you get the randomness is a different problem and it is computationally infeasible to construct the private key from the public key at least for

any reasonable method. I mean if you could do that then the whole thing would be useless.

You might have heard of the quantum threat and so, we'll talk about that later. Okay, if there are no questions, I will continue and focus on signatures because they are very important even for encryption. One second.

How does your web browser know your bank's public encryption key? So, for example you want to do a wire transfer, you log into whatever your bank is and you fill the form and you want to transfer some money to me for example, which would be appreciated, and so you have an interest and I and your bank have an interest that nobody along the way would replace the recipient or the amount or whatever detail of the transaction.

So, the web server of the bank will tell the web browser, hey, look this is my key, this is my public key, you can use it to send me encrypted stuff. But how does the web browser know that the key was not replaced in transmission? This works the following way. Before the whole website is set up, the bank goes to a notary and says to the notary we are example.bank and this is our public key and the notary says to the bank, oh, cool, prove it.

You can prove this by putting this long random number up on your website at example.bank slash whatever, some URL path. So, then the bank does that because they actually own example.bank and if the notary can confirm that the challenge number has been published there, then the notary uses its private key to sign the

bank's public key. So, that sounds complicated, but actually it's just like a text file, and in general a text file can be signed using a private key. So, the notary will take the bank's public key, put it into the text file and also the domain name to which it relates in which case you know here would be example. bank and then the notary signs that with its private key.

So, you have the mapping of the public key of the bank to its domain name and the corresponding signature of their notary. And then the bank gets that signature from the notary and publishes that signature on its web server or the web server will send it to the browser upon request when a connection is made. And if now the browser has the notary's public key, the browser can verify that the notary in fact has signed the claim that this particular bank key is related to example.bank.

So, the browser needs to have this notary public key ahead of time, but if that's given, then this is how it works. This is called the web PKI. The notary is the certificate authority. The signature is the certificate and the notary public keys are in the browser's trust store. This is used everywhere. This is used for EPP. This is used for HTTPS. This is used for email. This is more or less used in DNSSEC.

In DNSSEC actually, the notary is in the top-level domain context. The notary is the root zone key and the certificates are the DS records in the root zone with the signature from the IANA key and the DS records contain information about the TLDs key, the public key like the bank's one in this case, and allow a resolver to validate that this

is really the key of .com, for example. And then you can repeat the exercise on the next level. And this is what those DS records are. They are just essentially certificates from level to level and the notary in each case is the parent domain.

So, for DNSSEC, you only have one notary per level, whereas in the web case, you have, I don't know, 200 notaries that are all trusted at the same time. Also, this whole process depends on the DNS. So, in step two, the notary asks the bank to publish some challenge number on the website and somebody who can hijack the example.bank DNS by pointing it to a different web server can actually hijack that process and obtain a certificate for example.bank.

So, this again depends on the DNS. The web PKI is in no case stronger than the DNS. Sometimes opinions are voiced that we don't need DNS security because we have certificates for example, but they rely on the DNS.

All right. So, this was the web PKI. I hope this is interesting. All right. So, some other use cases. We'll get into all interesting things like blockchain and cryptocurrencies and dark web. Let's start with the dark web. So, there's the Tor network. The onion router is the abbreviation. It's also called the dark web. It's called the onion router because things are encrypted multiple times for multiple parties like an onion. It doesn't matter for the purposes of this.

The special TLD .onion is used for such websites and you cannot just register a name. You cannot have, I don't know, ICANN.onion because the domain name itself is the public key. So, you don't need

a notary or a certificate or something. When you access a domain name, the name that you access already tells you what is the key to use to encrypt your requests or your form input or whatever it is.

Now, for example, this year is the .onion URL from Facebook. Now I mentioned earlier that public keys are random more or less. Now this one here starts with Facebook. Technically, it's just Bits, of course, and you can represent them as letters. So, this is why this works like a domain name. They spent a lot of time, I guess, to get this particular key that starts with these letters. So, I don't know. To get a particular letter, I guess you have to try, I don't know, 30, 40, 50 times to the power of 8. I don't know what's 50 to the power of 8, but it's a large number. So, they had a long runtime for the random number generator to get this thing, and it looks nice.

Anyway, so this is essentially why the dark web is so secure, because you cannot hijack it by injecting a different DNS record or by injecting a different certificate authority into the trust store or something. The keys are directly embedded in the way the connection is established.

Okay, then there's cryptocurrencies. You might have heard of wallets, and the wallets have long identifiers. Actually, the wallet identifiers are just the public keys of the owners. The wallets don't have any owner names in the sense of like Peter Thomassen or something. They are anonymous, and anyone who has the private key can control and sign transactions relating to this wallet.

So, for example, if you want to send money to me using a cryptocurrency, I will have to give you my public key. You will have

to verify somehow that that's really mine, because maybe you meet me here. And then you will have your own wallet with your own private key, and you will use your private key to sign a statement that contains my public key and the amount that you want to transfer.

Then you will publish that somehow, and anyone can then anonymously verify that indeed that transaction was made. The wallet balance is just computed by following through the full ledger that has all the transactions. There is no particular database that holds the current balance. You will have to reprocess history to arrive at what the current status of each of the wallets is. Also, each transaction that is signed, I said, signs the amount, the recipient's public key, but it also signs a hash of the last record that was processed before.

This ensures that when you know the last messages or the last transactions hash, and you compare it to somebody else's last transactions hash who has a copy of the whole history database, you know you're on the same agreement, you have the same history. And so, this is how you can establish that not any particular actor has messed with the history, and then claims that they're right. You can very easily determine what is the majority view, pretty much.

This is called a blockchain. It is a chain because it is an append-only storage. You just add signed transactions and block because each record is called a block, and it relates to the previous one by signing its hash or hashing its hash. The details don't matter. This is pretty

much what it is. Anyone can have a copy. It gets very large very quickly, and you can then follow along.

If you don't want to keep a copy because it's terabytes large and it might not fit in your smartphone, of course, then you have to use some sort of provider that, you know, Binance or whatever those platforms are called, and then essentially, they act like a bank, and you trust them that they properly administer the ledger.

So, I leave it to you to decide where the trust difference would be to any other bank. Also, there's a lot of startups doing blockchain stuff, like, for example, for real estate ownership recordings and stuff like that. Knowing what you know about blockchains now, you might wonder why would you have to transfer a public register onto the blockchain? Like, what's really the security or trust advantage? Because you don't want anyone to write, like random people to write stuff on the real estate ledger.

It still has to be a certain government authority or something that went through some legal process first. So, you have a blockchain in that case that only has one right permission actor, in which case, essentially, it's just, you know, a disk with an append-only storage. So, I hope these things put some of those innovations into context.

Then, as I mentioned before, verifying wallet balances and all of that is quite expensive. You have to follow through all the history and you have to have a copy of it. It can be done, but it is only, I think, useful in limited cases. The blockchain domains follow a similar principle. Essentially, they record on the blockchain not, like, financial

transactions, but they record on the blockchain changes in, like, A records and things like that. And the way that's done differs by the particular solution. There is ETH names and whatever.

So, I'll not get into that in detail. The point is it works in the same way. And I had another thought. Maybe it'll get back to me later. Anyway, how does the first crypto coin come about? Because you need money to transfer, but what does it mean? Where do you get it from? That is by proving that you did some work. That sounds weird, but essentially there is a competition about solving a certain puzzle. And the first one who manages to solve a certain mathematical puzzle gets, like, the profit of having mined a coin.

So, the mathematical puzzle essentially is, find a hash input. And you have to try those because you cannot calculate back from the output. Find a hash input whose output starts with one zero. So, that's easy, right? Because you take a random number, you compute the hash of it. If it starts with a one, the output, then you just try again. And you need two or three attempts or something.

And then you will usually find one where the output starts with a zero. But if you need to find a hash input where the output starts with two zeros, you already have to try the square number of times, like two squared. If you need three zeros, you have to try eight times, two to the power of three. And if you need, I don't know, 20 zeros, then you have to try a million times, two to the power of 20, which is roughly a million.

So, the chance of finding such an input at any given attempt gets vanishingly small. And people are in a race of finding those inputs. And if you found it, you sign it with your private key and you attach your public key to it. You put it on the blockchain and then you have mined a coin.

Now, that is very expensive because you have to attempt a lot of times to find those hash inputs. And actually, the number of zeros required increases over time. This is why the number of bitcoins total is limited. And they get more expensive to mine, which sort of makes sense because I'm not an economy person. But it seems to me that it makes sense when the amount of coins is roughly bounded at some point.

And so, this is what this ensures. And once you have mined essentially all coins, it becomes ineffective and you can limit yourself to trading the existing ones. So, this is what Bitcoin mining is.

You might have heard that Bitcoin mining takes a lot of power, like more power than, for example, the whole electricity consumption in Belgium. This is the piece of information that is, I think, five years old. So, it might by now be a different country. I don't know. But anyway, this is really like trying random things for that financial gain. I'm not saying it's useless, but in terms of the mathematical problem that gets solved, it is quite in vain. It also is similar to how Facebook found their public key for the dark web. Okay. How much more time do I have? Five minutes?

ROGER CARNEY Yeah.

PETER THOMASSEN Okay. Some more thoughts. Hashes are very fast. So, you can easily compute a hundred million or something on a modern computer. And if you use graphic cards, which have certain optimized chips, you can reach billions per second on a certain graphic unit. So, that's why the miners buy all the GPUs. I mean, this was before AI arrived.

Then also password storage, I mentioned before, is done using hashes. In fact, it's not just one hash iteration. Instead, the hash output is hashed again so that it's harder for someone to, you know, when they want to try passwords and see whether they match a hash from a stolen database, for example. They have to try more frequently and spend more time on it.

So, that's the hash iterations. And usually about a million times are used. When you log in, your password is hashed a million times. I mean, it's hashed once and the output is hashed once and so on and so forth for a million times. And that takes roughly a hundredth of a second. But it happens every time you log in and it makes the work for any attacker harder by a factor of a hundred. Sorry, of a million. And it only costs you negligible response time.

On the other hand, operations with public and private keys are quite slow. So, that's only like 50k per second for short messages. It seems like that's very fast. But actually, if you have long messages like a

video stream to transmit over HTTPS, it's actually not very efficient. So, instead, other things are done.

So, for signing a long message, for example, what is typically done, you compute the hash of the message and then you sign the hash instead because the hash is something that is unique for the input. So, when you want to sign something, you have your long contract, for example. You keep the contract in cleartext. You also compute the hash of it and sign that and then you deliver the cleartext together with the signed hash.

And anyone who wants to verify that can also compute the hash. And then with a public key, they can turn back the signature and they can verify that the hash contained inside the signature is the same as the hash from the message that they computed themselves. And this is way more efficient because hashes are fast and public key operations are slow.

Also, you need less storage because it makes the signature much shorter. If you didn't do that, the signature would be as long as the input. All right. But how to speed up encryption? You cannot just encrypt the hash of something because the recipient could decrypt it, but then they have the hash of what you wanted to say. And it doesn't tell them what you wanted to say. So, what to do about that?

The solution to that is symmetric encryption. So, what we talked about before today is asymmetric encryption. It's called like that because it has a public and a private key and they have asymmetric roles. In symmetric encryption, there is only one key that is shared

by two parties. It's not very useful to use that approach with more than two parties because if your three parties using the same key than any party can claim, oh, it wasn't me sending this, it was this other guy. And the recipient cannot distinguish which of the two other guys it really was.

So, this only works in two-party problems. But you can establish a two-party problem using the methods we discussed earlier with signatures and encryption. And then inside such an asymmetrically encrypted connection, you can transmit a symmetric key, which is essentially just a secret shared between the two parties within this private context. And then going forward, you can use that to encrypt all the traffic that you have.

So, symmetric encryption methods are, for example, what's called the Caesar cipher. That is, you just move all letters by some number of steps. For example, you turn A into B and B into C and so on and so forth. That's very insecure because you can just look at the letter frequencies in a text, for example, and mount a statistical attack. More useful is what's called a one-time pad.

So, let's look at the term CPH in ASCII representation, which is just the bit representation of characters. So, the first thing here is C. It's 01000011. And then P and H. That's the plain text. The two parties agree on what's called a one-time pad. It can only be used once because otherwise it's insecure. It's a random number that's as long as the input.

Now they perform an XOR operation. The XOR operation is just an operation where you go column by column. And when the value is the same, the output is 1. And when the value is the same, the output is 0. And if the value is different, the output is 1.

So, here, for example, in the first three columns, the numbers are different at the top. So, the output is 1. Then there is one where they're equal. The output is 0 and so on and so forth. You do that for the whole input. And when the secret key is random, then the output is also fully random because there is a 50% chance of the secret key having a 0 or 1 independently of what the input was. And that, with a 50% chance, determines whether the output is 0 or 1. So, that's just a chance. And if you observe that as an interceptor, you have no way of extracting information from that. It is perfectly secure in the theoretical sense.

Now for decryption, you do the same thing. You take the ciphertext, and it turns out the operation is reversible by applying it again on itself. And then the recipient has the plaintext, which is CPH. Reuse is fatal. And the question is, how do you agree on a secret that's long enough to encrypt the stream? For example, you want to transmit 1 gigabyte of video call. How do you agree on a 1 gigabyte one-time pad?

The point is, you don't have to do that. You have to agree on a short secret. And then you apply a hash to it, which gives you a new random thing. And then you apply a hash to that, which gives you a new random thing. And that is called key stretching. That's why it's

so important that hash outputs look random and that you cannot work them backwards. That's another hash application I didn't mention earlier. I think it's actually one of the coolest. And again, hashes are everywhere. Indexing, databases, 2FA, password storage, even encryption.

All right. What happens inside an encrypted connection? First, so this is like when you do HTTP connection, for example. First, the browser verifies the server key. We talked about this earlier, the notary and the certificates and all of that. Then inside this asymmetric private context, a secret key is exchanged, as I just explained. You can do it, as I just said, sending a one-time pad and then just using that. But unfortunately, that is not secure if later the private key for the asymmetric setup is leaked because you can record the traffic and then later decrypt it with a private key. And then the attacker will see the secret key and then, I mean, the one-time pad, and then they can decrypt all the traffic.

So, if you don't want that, you will have to use what's called forward secrecy. And so, there is a Diffie-Hellman protocol for that. You might have heard about it or not. I'll show a diagram of this. I will skip over it for reasons of time. But if you're interested, we can talk about it upon request. And then once you've done all these things, the actual encryption is done with a symmetric secret.

So, this seems very complicated, but it's also very fast. I mean, you just connect to a website and it just works and all these magic things happen. This is the Diffie-Hellman thing with an analogy in colors

instead of numbers. But I'll skip over it. And also, Signal uses an advanced version of it. And still there is the risk from quantum computers.

This brings me to the last slide, quantum computers. Quantum computers are used to set up some physical situation inside of certain specialized processors, which lets nature work out certain calculations instead of doing the individual calculation steps. That sounds quite opaque, I would say.

So here's an example. How do you find the maximum height of a water wave? Let's say a water wave and it might have a certain height or amplitude. How do you find out what the height is? A classical approach would be you take a picture of that and you slice and dice it and digitize whatever. And then you go through all the different slices on the picture and you measure the heights and you take the maximum and store that temporarily until you are at the end and you know I found the maximum.

Another thing you might do more physically is you just let the wall run against the wall and check to what level it is wet. So, you wait and measure. That is essentially how you need to reframe your calculation problem and then let nature do its thing and then you do measurement. That works for some problems and not for all problems. It has nothing to do with clock speed. Most problems like, for example, I don't know, hashing, for example, won't work faster this way. But unfortunately, or very interestingly, reconstructing a private key from a public key can be done in this way.

Now, quantum computers are not very mature yet in the sense that they have very limited memory. Just a few hundred bytes or something. So, they are not very practical yet and they are quite slow. So, if you want to hack a lot of things, it doesn't work. But it seems to be coming. It seems like to be in the same state as classical computing was in the 40s. And then it took like, I don't know, a decade or two and then it was useful. So, that's the concern and that is why people are concerned about this.

This particularly affects this prime number multiplication factorization problem I mentioned earlier. And also, certain encryption algorithms that are called elliptic curve algorithms, but whatever. So, new crypto is needed for classical computers because not everybody wants a quantum computer to browse the web. You need to do all the things on the previous page to do your HTTPS connection. But using a method that cannot be attacked this way by a quantum computer doesn't mean that you need a quantum computer.

Your encryption needs to be quantum resilient. That's why new asymmetric cryptography is needed. There's NIST competition for that and all of these things and IETF does standardization. You might have heard post-quantum cryptography. For some reason, people call it that way. I think quantum resilient is a better term. Post-quantum, I don't know. I mean, you can even use quantum resilient stuff before quantum computers are practical because it's just another method for doing encryption.

That was it. Yeah, that was it, I guess. I'll leave this slide up. And if you have any questions, let me know. Also, I didn't say a word about DNSSEC. Well, maybe two words, I said.

ROGER CARNEY: Thanks, Peter. It's funny letting you walk through all these. And it's like you make it sound so much easier than how we had to learn it and everything. So, it's very appreciated. And hopefully people can get that and understand it a lot easier than it took me many years ago to go through that.

Anyone have any questions or comments for Peter? Okay, I don't see anything. Thank you, Peter. Okay. Yeah, Sarah does have one.

SARAH WYLD: Hi, this is Sarah. I really appreciate this, Peter. Thank you. You explained complicated concepts in ways that I'm getting closer to understanding. Every time we go through this, I get closer. I still don't get everything, but thank you.

PETER THOMASSEN: Thank you very much. You're welcome to go through it again. The slides will be up in public.

ROGER CARNEY: Great. Rick?

RICK WILHELM

Yeah, I'll jump in. I think that one of the things that it's important for us all, one, thank you, Peter. This is really good. And from having done this kind of stuff, that's a lot of work, right? Like you put those slides together and every slide there is, you know, there's a lot of hours that you put in. So, thank you for that.

I think pretty much everybody here within Earshot and anybody listening to this presentation is going to have to be asked to explain and help someone understand things like public and private keys. And so, hearing these kinds of explanations is good because hopefully you can take something from this or grab one of these slides or one of these things and help people that you have to talk to understand something about like the difference between a public key and a private key.

And how, when we talked about the difference between, Peter talked about the difference between encrypted and signed email. And what does it mean for an email to be signed and what does it mean for an email to be encrypted? And then, well, someone sent me an email and now I can't read it, right? And what does it mean that the certificate is invalid? And then why can't I read their email? And, oh, well, then you've got to go into Outlook and have them uncheck this box.

And so, I think it's important that we all sort of absorb this stuff and try and latch on to a couple of those simple explanations so that you can stand on one foot and explain it to either your peer, your subordinate, or your boss, why some of this stuff does or doesn't

work to help you. So, this is great and thank you very much, Peter. Thanks.

ROGER CARNEY

Thanks, Rick. Yeah, and my neighbor who always asked me. And it's like, I don't know that they still understand, but it's a question that gets asked a lot. So, yes, thank you, Peter. Okay, I think we can jump back to the agenda. We've got about six minutes left in this session. And I think this is probably plenty of time for this topic.

So, we just wanted to bring this up mostly because it's just starting. And there may be, obviously, today, there's no TechOps work for this. They just finally, I think, got their leadership up and running and their proposed timeline and everything set yesterday. There were two meetings by the Associated Domain Check PDP team, and there's two tomorrow, I think. And they're kicking that off.

So, this will probably be more interesting to registrars than registries. Registrars already do this today. Many registrars do quite a bit of this. So, I don't know that this is going to be earth shattering. It may be more educational for people that don't know that this occurs. But the idea here is if someone identifies a malicious domain, the idea of the registrar will have some responsibilities to not just evaluate that domain but evaluate things around that domain.

So, what that is, this group will decide. But, again, a lot of these things occur today. Most registrars do many of these things. I would guess it'll just get to a level playing field where some registrars aren't

doing these. But probably my only introduction for this, but I wanted to turn this over to Marc because I know he had a few comments. So, Marc.

MARC ANDERSON Thanks, Roger. Coming off the video here. Actually, you covered it quite well. I don't really have anything to add. I think our idea for adding this topic here to TechOps is to put it on people's radar and for us to track it to see if at some point there's a need, desire, or value in us becoming more involved or digging a little bit deeper into this topic. So, I think what you'll see is we'll continue to track this and keep an eye on it from a TechOps perspective. But I don't think there's a whole lot else for us to dive into today.

ROGER CARNEY Great. Thanks, Marc. Okay, anybody have any questions on this? Again, it was just an introduction, as Marc said, just to make people aware. So, Rick, please go ahead.

RICK WILHELM I think that it might be useful at some point, and Roger, I'm not sure how the representation is going, but it might be useful, depending on how this thing unfolds, that TechOps might get asked for an opinion on some things where TechOps can come in and be a little bit impartial about the mechanism and about the how of some of this stuff.

So consider that, if that's useful, where there's a technical assessment of something that might be proposed or weighing some alternatives or something like that, if that's useful. Because I know that when we were doing things on the transfer policy, a lot of the elements of the transfer policy were very registrar-heavy. But I know that TechOps weighed in a couple of times on mechanism when policy was considering a couple of mechanisms, and TechOps weighed in like that.

So, it's something to think about. But similar to transfer policy, this is going to be pretty registrar-heavy. But also, we don't know, the registries don't know exactly which way the wind is going to blow on this stuff. So, thanks.

ROGER CARNEY Thanks for that, Rick. And I would agree. I think it will be an advisory role for us in the PDP ongoing. And I know that this first DNS Abuse PDP is very registrar-focused. But I think the future ones will swing back and forth, and we'll see some registry-heavy focused ones as well. So, again, just as Rick mentioned, it is just an introduction so that we know that there's potential coming down the line. And if TechOps members are actually participating and want to bring something, obviously, it's welcome.

Okay. Any other questions or comments on the associated domain check PDP? Okay, I see nothing, and I'm not going to have Michael start his IDN stuff with one minute to go. So, we'll give everybody one minute extra to their break, and we'll be back here. Again, same

Zoom room, so if you want to stay in, please do. Recording will stop, and we'll get it restarted in 30 minutes. Thanks, everyone.

[END OF TRANSCRIPTION]