TRANSCRIPTS

SEDI 2026

A complete transcript of every recorded talk from the State Endorsed Digital Identity Summit (SEDI) 2026.

2	2:01:05
TALKS	RUNTIME

November 27, 2026

CONTENTS

Full transcripts of all 2 recorded sessions, in the order they appear in the archive.

01	2026-11-17-Samuel-m-Smith	1:03:55
02	2026-11-18-Samuel-m-Smith	57:10

TALK I

2026-11-17-Samuel-m-Smith

State Endorsed Digital Identity Summit (SEDI) 2026 · talks · 1:03:55

My name is Sam Smith and my parents knew that I would be identity challenged so they gave me a name I couldn't forget. I'm co-chair of the KERI Foundation. It is a organization that is promulgating the KERI and ACDC standards. There's a KERI suite of protocols includes also a standard called CESR. The Trust over IP foundation is where those are being incubated. They are official specs of the Linux Foundation as of January. And then once they pass that process, they're now starting to climb the hill of IETF standardization, which will take whatever time it takes. These are open standards. There are several open source libraries of various parts that implement the different specs. Some of the presentations that you've seen, they didn't bother to tell you that under the hood they were using KERI and ACDCs. The Veridian wallet is all based on the stuff of KERI and ACDCs. The demo for healthKERI is based on that. That's kind of more obvious. A few others. But these have been in development for quite some time and the KERI Foundation has been developing new versions of the libraries that have some of the citizen -based identity. Originally KERI ACDC's, the first big adoption was GLEIF, the Global Legal Entity Identifier Foundation, which is primarily organizational identity. A lot of the tooling was focused on organizational identity and now we're adding tooling for citizen-based identity in the state of Utah and SEDI. This will be a fairly technical session, but don't hesitate to ask questions. I'd rather get interrupted and answer questions than have it not make any sense.

I'm going to talk a little bit about using ACDCs. ACDC stands for Authentic Chain Data Container. Now, we've heard a lot today about verifiable credentials. ACDCs intentionally didn't use the term verifiable credential because they're more general than a verifiable credential. An authentic chain data container can carry any information.

And a credential is a type of permissioning system. Like I'm going to present a credential to allow me to gain access to something or a credential might be an entitlement. But an ACDC could just have evidence. It could have whatever's in there. It's a more generic approach.

From a technical point of view, an ACDC is a graph fragment that has edges which are normative property. And that's one of the superpowers of ACDCs is that early on we put edges in them as a normative property. And we'll talk about what edges allow you to do. And then there's a vertex, the node that has metadata properties and the node properties which you would call maybe claims attributes, but these are just the properties. What you use them for is dependent on the application. You can take a bunch of ACDCs and graph them together. You can link them in a graph where each ACDC is a fragment of the graph. Now what you have is a graph of containers where the source of each container can be different. This is the hardest thing to get for a lot of people get their mind around. This graph isn't sourced from one issuer. Every node in this graph could be sourced from a completely different issuer. That means you can build and construct data structures that span trust domains. Think of each one of these is coming from a different trust domain. Now you have a completely verifiable graph of information that spans trust domains. Trust domains aren't a barrier to being able to send around verifiable data. That's like a superpower of ACDCs.

You've heard the term durable. Perpetually verifiable issuances, they're durable. That is a core feature of ACDCs. That was a hard problem we had to solve. Each fragment is separably authenticatable. That means it comes from a different source. Each one could be authenticated differently. You don't put them in a big glob, authenticate the whole glob from a single issuer. You authenticate everything in every chain and in every part of the graph. The important thing is that when you do it, model it this way, we can now do high fidelity modeling of real world authority structures. That's the hard problem. If I want to model how the real world works, it doesn't work the way most people build verifiable credentials or

verifiable issuances. Most identity does not have high fidelity modeling. If you go to an organization, even an organization as small as a family. Is there a hierarchy of authority in something as small as a family unit? Now, some family units probably not. But there is.

In the human world, everything has some aspect of this delegable authority.

We went out of our way to model that. That's why ACDCs exist. It's not because we didn't like other structures. It's because the other structures made no meaningful effort to model delegable authority structures. We said, well, there's a need for that. We're going to build a standard that does that as a first order property. I have this mnemonic for authorization because one of the things you can do with this is you can build authorizations. And these authorizations meet the model of what most people want to do for agentic delegation.

Delegation, whether I'm delegating to humans or delegating to agents or delegated to humans, to humans, to humans, to agents, to agents, to agents. As far as an ACDC is concerned, it's the same structure. It's just what goes in each of the nodes that is different. But the overall structure doesn't change. The tooling doesn't really change. Agents have higher velocity. That affects something in your tooling, but that's the main difference. The talk from Sarah from NIST where she had that diagram. One of the things on there was delegation. She had a few other things. You'll see some of those show up in this ACDC support delegable, attenuable, aggregable, auditable, revocable, transferable authorizations. All of those properties or some combination can be had because we didn't leave anything out. It's a complete approach to solving the problem of modeling real-world authority structures. And that's why some people say, well, that seems really complicated. If you're a technical person and you actually tried to solve those problems, it would be complicated, but once you solve them, then it's really just up to the UI guys to make it so you don't see that.

Like the Veridian demo. The Veridian demo did a really good job of showing mutual authentication with a lot of delegable stuff going on in there and you didn't see it as the user. But it was going on under the hood. It can happen. One of the things that I think Chris Bramwell mentioned is about a public utility, a digital identity or digital public utility. Well, public utility is necessary because if I want to be able to operate in a decentralized world where I don't have a central single trust domain, I have to have a way to solve the problems of moving data and trust across trust domains. And that requires some form of infrastructure because that means you have to have something that people can look to. We want to do is build infrastructure that has the lowest possible cost floor. I spent years working in the blockchain space building identity on blockchains and it has problems. One of the problems is the cost floor is really high. You put anything on blockchain, every transaction is really expensive.

But if you look at web, the web has pushed the cost floor down to fractions, small fractions of a penny per transaction. And it's just going to keep going down. By design, we want this infrastructure to have that kind of cost floor, but we do have to have infrastructure. That means somebody has to stand up the infrastructure. It has to be monetized in some way. And there are different strategies for monetizing that. Phil Windley, he's here, but I don't see him in the audience. He wrote a couple of blog posts and a couple of presentations about one of the adoption problems of digital identity has been that no one wanted to pay for identity assurance. The people that benefit the most from identity assurance are us. Not the organizations, but everybody just wants their digital identity for free. But if the state pays for a strong identity assurance, then that cost adoption barrier goes away. Now you've sort of primed the pump, you spun up the flywheel sort of an effect. The state and Chris is here so I don't want to put words in his mouth.

The state's going to pay for identity assurance for SEDI. – We let people in the summit take one or two steps ahead of the boss. But we do know new models are needed and there is a session here at the summit on utility and how do we take this?

– We also don't want it to be sold. – Yes. – If the state kicks it off and maintains it, we don't want somebody to say, well, we'll just sell it off. – we want a vibrant market of multiple suppliers. – What I will say is identity is going to solve hundreds of billions of dollars. We just talked about fraud, but the things that going to open up. If you view it as critical public infrastructure, usually the government plays a role to maintain that because if you privatize critical public infrastructure, especially identity, essentially it's inelastic and we need trusted identity. The risk is people saying, well, let's extract as much value as possible and we don't want that. That's why governments can play the role of the utility to say, well, we don't want this extracted, but it's for the public's good. If you figure out the right government models for it, you can get the best of all worlds and you can still have a rise for five or three market where people make money, but it's not exploding.

– That's great. I don't want to go into the economics too much, but I'm just pointing out there is infrastructure. Witnesses, watchers and a watcher network are essential and wallets. You see a lot of wallets. You don't see these other ones that are under the hood. There is a demo in the demo alley of KERI Foundation engineers. They can show you a full set of infrastructure running in the cloud. If you want to see it running in the cloud, you can look at strings scrolling by on terminals, but it's running in the cloud. And it's all open source. If somebody wants to stand up KERI infrastructure, they can just go download the code and we'll help you get it set up. The other KERI infrastructure for revocable credentials is registrars and observers. And these are split so that the issuer doesn't have visibility to the points of validation.

How the credentials are being used. This gives herd privacy to solve the privacy point there. And we also have something called bulk issuance and we're using sparse Merkle trees in there as well for managing that. This is where I'm going to start to be a little technical.

If you can't read the labels on the arrows, then you know what this is. Anyway, one of the unique features of the KERI ACDC design is we're trying to manage the trade space of having issuances that have durability, which is really difficult to do, at the same time protect privacy with revocable credentials. If you take those three things and you put them together, usually you get two of the three, but you don't get all three.

We're able to get all three to a high degree because of what we've done. Normally if you have a revocation registry, you lose privacy. because everything in the revocation registry is correlatable. Vendors can now share information that they just say, well, I did this and here you go look at this. You have vendor linkability because the revocation registry. We solved that problem with something called a blindable registry. And the way it works is that the issuer issues an ACDC to an issuee.

You've seen this in the demo. They do a presentation to a verifier. The verifier goes to an observer to look up the state of the registry. The observer gets a bulk update. The registrar itself doesn't see what the observer is giving out, but the observer can get the latest state of the ACDC and this state is actually blinded. The only way the verifier knows the state is the issuee unblinds it at the presentation and then it can be re-blinded after the presentation, which means that in most revocation registries, once you unblind it, it's unblinded forever. That means that if for example, I have a credential that is my transcript. And I go to a job interview and they want my transcript. I present my transcript. No, not my transcript, my work history. Like I work for somebody. I prove that I'm an employee and then they don't hire me.

But now they know the state of my employee status because I presented it. And if it gets revoked because I leave employment, they can then go, oh, that person just left employment and we actually wanted to hire him, but the timing wasn't right. But now we know that he's looking for a job because he just lost his last employment. We'll go search him out.

Revocation registries become forward privacy problems. But by having a blinded registry that can be re-blinded after the presentation, they don't know what the current state of my ACDC is after the presentation. I have what we call a perfect forward privacy. We made that term up just because it sounded good.

But it's analogous to what's called perfect forward secrecy in encryption. The idea is that I've got this key event log. I'm going to explain how ACDCs are durable. Now you saw an example, a simplified example of a verifiable credential that has a signature attached to it. Now a signature is only valid for the crypto period of the issuer keys that issued it. Imagine that I've got this key pair and I use the private key to create a signature and I attach it to a credential and I give it to you. Then I rotate my keys. Why would I have rotated my keys? Anybody have a good reason why someone would rotate their keys? – You lost your phone. – Okay. That's one reason. – Good hygiene on the issue.

Good hygiene. – Keys were compromised. – Keys were compromised. As a verifier, do I know that it rotated because he lost his keys or because the keys were compromised? – I don't know. – Okay. As a verifier, if I want to protect myself from fraud, what do I have to assume when the keys are rotated? – Compromised. – That they've been compromised. That's the only safe policy. That means the signature is no good anymore. Which means that digital signatures, unlike wet signatures, are like signing and disappearing ink. And that ink disappears the minute you rotate your keys. Imagine that you've issued a million ACDCs, a million credentials of the world, and all you did was sign them. And then the issuer rotates their keys after 18 months. Good hygiene according to NIST special publication. 857-1.

Revision 5. I don't remember the paragraph. It's 50 pages. It's a long document. Anyway, so what happens now is that I've effectively by rotating my keys invalidated all million issuances. What happens now is I have a catastrophe because now what if those were driver's licenses and I just invalidated everybody's driver's license?

What would happen to transportation? The idea that you can use a signature as a way to provide a verifiable credential means that it's really only good for ephemeral use cases where the likelihood that you're going to rotate your keys in that timeframe is vanishingly small. You see this like in X.509 certificates. The current best practice is that we're eventually going to be forcing everybody that's a certificate authority that's using X.509s to rotate their keys every 45 days. Okay, every X.509 on the planet for web is going to get a churn every 45 days. – I think that's going down. – It's going to go below 45?

I think […] – The cyber AI threat is putting downward pressure on on crypto key periods. If you say that a year is reasonable today, I can't hazard what's going to be reasonable in 10 years with cyber AI threats. We have to have a way to issue things that are durable that survive key rotations. And that's when we say perpetual or durable issuances, we have a mechanism for that. The mechanism we use is a key event log and the state in the registry. We talked about the blindable state registry, but the main purpose of the registry, even if it's not blindable, is to make the issuance perpetually verifiable. I could either anchor the issuance directly in my key event log or I could anchor a state registry event in the key event log. It doesn't matter. The fact that it's anchored in some way means that now when I rotate my keys, the seal in my key event log is still there despite the key rotation, which means the verifier could

still find it and then go, it was validly issued when the keys were not rotated. It's still valid today. It doesn't matter what happened. The KERI protocol is all about managing this thing called a key event log. It feels like a blockchain, technically under the hood, it's a little bit of a blockchain, but it's a blockchain per identifier that each individual controls what isn't controlled on a shared distributed consensus ledger with this high cost floor. It's a really low cost floor. That's the innovation of those protocols.

One of the things when you have these registries that you can do is you can do something else that's unique to this technology. User impersonation fraud detection. If you think about it, we've been talking about the issuer having their keys compromised. And the issuer doesn't want to be impersonated. If they can detect that their keys have been compromised, they're going to rotate their keys and manage that. But what about you the user that has a credential in your wallet and you got attacked and you made a mistake and it's really hard. Private keys are much harder to phish, they're relatively phish resistant but there are still ways that could happen I mean if you talk to Steve McCown he can say pretty much every cell phone is hackable so anything that's stored on a cell phone potentially won't survive, so you have to worry about that.

That's a side channel attack. Let's say that your keys got compromised. What we have is the user themselves can have a presentation registry. It's not just the issuer has a registry for the state, but the act of presenting can be anchored in the user's KEL, not the issuer's KEL. And that means that a verifier has to see that anchor before they'll accept the credential. And what that does is it means that a fraudster who is presenting your credentials with your compromised keys is detectable by you, the user. And when you see a presentation that you didn't present, your keys have been compromised and you can rotate them and recover. You limit the timeframe over which a damage can happen. Does anybody know what the average time to detect a breach is as of the end of 2025? I haven't seen the report for 2026. – -48 hours. I was joking negative 48 hours. – It's six months. What's the average time to contain a breach? Three months. An attacker has nine months to exploit.

On average. That's the industry standard. What if we could change that detection time from six months down to six minutes or six hours? How much harm would go out of the system? That's what KERI provides. It provides the detection mechanism that is equivalent to network latency and the time that it takes for you to respond to recover from it. That's a fantastic security property. Any issuer at a state level, I forgot to ask what my audience looked like. You got too many people, but how many people here are from states? Okay, fair number. As a representative of a state, do you want you as a state to be impersonated by attackers? Of course not, right?

But if you adopt a technology that makes it so that you can't detect that you've been impersonated, is that a problem for you? Okay, what is on the priority list? Should detectability of compromise be near the top? And is six months… in my book isn't detectability. Six months is… the attacker said, well, we've exploited it all that we can. We're now going to tell the world that we did this to get cred so that we can charge more for our next attack. That's not detectability by you, the exploited. That's them just, what do you call it? They're just sort of… They're advertising at your expense. Same thing. The issuee has their own key event log and they have their own registry for presentations and all of the presentations could share one registry because it's just anchoring the presentations.

It's not managing state of the ACDC. That's in the issuer log. We have a KERI Foundation SEDI implementation guide draft. It's now about 150 pages. But in that draft, I'm going to go through what are proposed ACDCs that satisfy the SEDI requirements. One of the first things is a delegation change.

The root of trust is the state. And that is like some set of people at the state that are the Root of Trust. And I don't know who's going to be in that, but it might be the governor and the CIO and the CTO and a few other people. I don't know who's in that. Chris (Bramwell), do you know?

– […] – Oh, okay. Do you know who's going to be the signers on the Root of Trust? – Oh, it's an interesting discussion. I'm going to put this out here. This is where it gets really interesting. What's the ceremony we're going to have in government to transition, how we elect people and then the authority. Because as we move to technology, like the concept of authority is important. The people elect officials and then they have authority from us to represent us. But we haven't built technology that way. We've created tons of silos where every entity is their own root of trust. How we design this or who are the people that can sign this and maintain the keys and rotate, we need to think hard of what does this look like in a constitutional republic because we're essentially saying here's the root of authority and this should be really important like when a new governor comes in or a new whoever it is comes in there's a rotation that needs to occur and this is a new ceremony we're creating in government for this new type of digital world. I'm not going to say who's going to be in it. This is very important.

– But it's a policy discussion and it hasn't been decided. But this is modeling real world authority structures. When you say who has signing authority for the state of Utah, if you say not your keys, not your identity. If an IT person holds the keys for the state of Utah, the state of Utah is run by that IT person, not the governor. You've given sign-in authority to the IT person. You have this problem of you go, well, politicians shouldn't be managing keys.

Everybody's going: there's no way that's going to happen. The reason you use a delegation chain is so that you can push away the friction. How often did these guys ever actually have to use their keys? If you're following best practices, that might be once every two years you do a rotation. These guys here might be doing stuff with their keys like every week or daily. These guys here definitely daily hourly.

The more delegation levels you have, the more you can push the work down at the lower levels. But it's the same root of trust. There's one root of trust. Everything verifies back to that root of trust. The ultimate control rests in the root of trust. And I know the president of the United States doesn't know how to actually type in the codes on the football. There's somebody walking around that's going to tell him how to do it, but only he can do it.

You can still manage the fact that politicians by virtue of being elected are the only ones with signing authority and they can still manage private keys. – This is not just about identity. Think of the record aspect. We have created government where every entity has just been creating their own records and now they're creating digital records. And I can tell you these entities have not considered the durability concept and also the authority concept. We have so separated sometimes the processes in government from the elected authority and the legislators in the different branches and there's a unique opportunity here to get the right structure so the keys, the authority, the signatures, the trust represents what government's ability to play, what the processes and the data do as well. But every record you have in government, it should have signatures and it should be signed and that has a moved authority and it needs to be durable because some of these need to last centuries for property records and things like that. What we design here is critical for what's going to be coming and the attacks that are going to be coming that want to break trust. They want to compromise these records. They want to beat your systems.

You're going to have thought about any cities and counties that have moved their property records into a proprietary system and they're not keeping paper records anymore. I can assure you AI is going to go in there and create issues where it's like, hey, here's a fake encumbrance, here's a fake easement, here's a fake change and you're not going to be able to prove it otherwise unless you have this type of structure to say, well, that's not true. Here's the truth. But this rule of authority and this training is very important. – Digital determinism. – Yes.

– I think something especially exciting about the structure is not that you have this hierarchical digital structure because there's a lot of existing things that can do that one time. It's the succession. It's durable. – From human to human and allowing the governor to that ceremony that transfers the authority to the successor without breaking anything that his predecessor did. – Yes. When I showed the KEL and I say it survives a key rotation, the key rotation isn't just to yourself. Your key rotation is to anybody. If I'm in this office and when I walk out of the office, the new person comes in, I rotate the keys that control the office to keys that they control that I don't control. There's no sharing of keys. Never. – Without breaking the work that was done by your predecessor. – But everything that I did is still in the ledger, in the log.

It's still verifiable in perpetuity. Yes. – I was just going to put a plug out for Karla [Mckenna, red]'s session on boot three. GLEIF has basically built this architecture in reality. It's not just market texture stuff that Sam's put on the slide and talks very convincingly about. – I like how you use technical terms for non-technical stuff. – I'm glad you waited for five minutes to get into the technical stuff. If you're here day three, be here day three, but go to Karla's session because you'll be able to kind of walk through what this looks like in practice. Because it's one thing to see the diagrams. It's another to see what an organization global is going to be do with this tax credit. – Thank you for that. Any more questions? In blue, we have the AIDs. Those are the autonomic identifiers. Autonomic means self -governing. Even this person has their own AID that they control. It's their AID. It's not the government's AID. Even inside the government, everybody controls their own identity. What they do is that this person issues an ACDC that delegates to that AID whatever powers and authority come with this office that that person is sitting in as indicated by that. When they leave that office, it's still their AID. It's like in a current business place when you go to work. And you onboard and you go and they give you a set of keys to get into the building. And then when you leave, they say, oh, turn in your keys. That doesn't happen here. You're not turning your keys in.

They're just revoking your authority. They're changing the locks, not changing the keys. Yes. All right. We've got the organizational unit. That'd be like driver's license division, health and human services, the SEDI program division of GovOps. They know them better than I do.

And then issuing agent is somebody that is beneath that. It might be somebody outside the state. The issuing agent might be a third party that the state contracts with or it could be within a state agency. Now the actual identity credentials that go to the citizen are issued by this person.

And see this binds? That's the edge that goes back that says this thing is bound to that thing and if this authority is broken, then that breaks that one. And then here's the core identity which has an edge back to whoever issued it. Now this is the citizen's AID. And then these are the credentials that are issued to a citizen. And there'll be more than this. This set here is just the set that was authorized by the legislation in 2026. And one thing that I've learned is that we have to do everything by the book. And if the legislation doesn't authorize it, we don't just go do it. Which is really foreign to me as a tech person. It's like, no, we go do the tech stuff and then we try to convince the powers that be that that's their only choice.

Any questions about this? I'm now going to go one more layer down. Code. We're at the bottom layer now. We're all the way down to the bottom. This is what an ACDC looks like. This is in JSON. Or in Python dict form. But same thing. There's a version string. There's a type. And then you see a lot of these things. See all of these things. Those are all cryptographic primitives. We're a security first protocol. That means everything has to be cryptographic strength. Cryptographic strength means that the operation has a minimum of 128 bits of entropy. Now, I just used a bunch of terms I know most people don't understand. But if I have 128 bits of entropy and I'm excluding post-quantum stuff because that's a different measure, but pre-quantum. This is the standard and it means that it's computationally infeasible to break it. That means it doesn't matter how many AIs you have, how many supercomputers you have, you can't brute force break it. The only way you can attack it is through what's called a side channel attack. Side channel attacks are things like phishing. Like I can't break your signature.

I can't forge a signature. That would be a brute force attack. But if I can get your private key, then I can create a signature that I didn't have to brute force the signatures you created. That means that attacks go from trying to attack the crypto as long as you're using strong crypto. And that's not always been the case. The world of protocols that use crypto over the last 30 years are filled with examples where engineers go, that's really kind of an expensive operation. How about if we only use 64 bits of entropy? Because it'll be exponentially faster. Or four bits. Or two bits.

The number four. From the beginning, I said, no, no, the world is only going to get worse, going to get harder. Any temptation to be less than full cryptographic strength is a temptation that we just don't ever go there. We did things that people complain, why do you put that in there? And it says, no, we want zero malleability attacks. That means everything has to be protected with hashes.

We can't have any free variables floating out there. Everything has to be protected. We invented something called the SAID protocol, which is now actually an IANA protocol, which says that any document that I have that has data in it is going to have a field in it and this happens to be the D field and one of the reasons we're using really short labels here is because these are reserved fields that happen over and over again if they were really long they would actually make the labels longer than the crypto and that would be a problem. Anyway, but you can see we have some that aren't because they're not reserved labels. Anyway… – They make jokes about the 'd' 'u' 'i'. – 'd' 'u' 'i' right there. “Driving under influence”, which can't be said of me because I don't drink. But anyway, this SAID is a hash of this data that is inside the data. There's a protocol because basically you can't hash data and put it inside. We put dummy variables there, hash it, then replace it. But that means that anytime the data goes anywhere, it's already integrity protected. And it's integrity protected in an interoperable cryptographic agile way. We just have these SAIDs everywhere because they allow us to manage all of the data in a way that's protected. And if I sign just the hash because the properties of cryptographic strength hashes, it's just as good as signing the actual document. That means that I can manage things really well. I'm going to go through the field. This is the SAID.

This is unique entropy. This protects this data structure from what's called a rainbow table attack. And in a rainbow table attack, if I have a schema that defines all these fields, what these values are, then I might be able to deduce what the field values are, even though you only gave me a hash of the data. If I have enough entropy, then I can't do a rainbow table attack. That's why that's there. This is the issuer field. This is the identifier of the registry. We talked about the registry. This is the schema identifier. Notice the schema itself has a hash or a SAID, which means the schema are immutable. Nobody can play games with the schema to try to change the semantics of this. Here the attribute block.

The attribute block itself has a SAID. It has unique entropy. It has an issuee. This is where the issuee shows up. Here's the issue date in the SEDI program office. And then down here in the rule section, we don't have any rules because I can't get the lawyers to put rules in there. But this is where all the legal language goes. That's a big to do item for the... I think, is there an attorney in the room? What is Micah here? Oh, we got a couple other. Where's Micah?

But this means that you can actually embed what's called a Ricardian contract, which is a human readable machine readable contract. It's a legal contract because it's in text, but it's expressed in a way that you can also do machine operations on it. You can either embed references to the legal structure or you can put the actual legal structure.

You put liability waivers, whatever you want. You can put it inside that so that it's carried with the data. The data carries its own legal constraints, which means that in duty of data loyalty, you can define what those are with the data whenever you send the data to somebody. That's another normative feature of ACDCs that you don't find anywhere else. This is actually a pretty simple credential because all I'm doing is the designated program office role for whoever has this credential, that AID, the issuee AID. Here's the next delegation chain. It's a little bit more complicated. I've got all the other stuff is the same.

I've got a role and a name. And in this case, the name has its own block so that it can be hidden and it's Susan Park. And then here's my edge section that links back to the other one. You see that I've got SAIDs everywhere. I've got unique IDs everywhere. And then I've got the data and the edge operator tells me how to interpret the edge. DI2I is a shorthand for a delegated issuer to issuee. It says this edge represents a delegation. That's what it does. But now my tooling for the credentials can actually verify delegation change because it's a normative feature. It's not just some field that I decided meant that. It's in its own section. Now, the thing is you see all this extra crypto and that extra crypto is so that I can do something called selective disclosure or what we call nested partial disclosure so that we can present the credential and you can make a commitment to this without disclosing these other things or to this block without disclosing those other things and verify the commitment but wait until you've you come to a agreement about the terms. Here's the core identity credential. Same sort of stuff up here. But now we have more attributes and each attribute gets its own block. I've got given name, middle name, family name, name suffix. Let's just do one block: given name. Has a SAID. It's a SAID of just this block. Has a unique attribute. You can't guess the value of the given name just by giving its SAID and the schema. And each of these blocks have the same thing. But by doing this, I can make a cryptographic commitment to this whole data structure and only disclose parts of it and the cryptographic commitment still verifies. Who in here are familiar with a Merkle tree. Who knows what a Merkle tree is? Okay, most of you. A Merkle tree is a binary tree and ACDC is an n-ary Merkle tree. Each of these blocks is an edge in a tree that is hash chained up to the root. This is the root. Here's the root. These are all nodes that hash chain down. Now I have this hash chain data structure where I can selectively disclose a branch without disclosing any of the other branches.

That's the fundamental property of a Merkle tree. We made it an n-ary tree because real data isn't expressed in binary structures. It's expressed in groups that are non-binary. And I didn't mean that politically. Here's the other fields. And then here's the edge that points back and what does this edge point back to? The Utah agent. That's the structure of an ACDC. You now know as much as anybody else in the world about an ACDC.

– The thing you're hashing has 128 bits of entropy. Or the hash? – The hash itself has 120 bits of entropy. But what you're hashing also has 120 bits of entropy. And the reason is that if I have a schema that says this field, its values are, let's say, a zip code. How many zip codes are there? Not very many.

The entropy of a zip code is really small. If I have a block of data that has a zip code, a street address. A country and a county for address, the entropy of all of those fields for all of the address in the United States is way, way, way less than 120 bits. It's like maybe 40 bits. Which means that's attackable. All I need to know is the schema that says this field is a street address, this field is zip code. I can guess all the values of the field by doing a rainbow table attack. But if one of the fields is high entropy, a random number, I can't do a rainbow table attack. – I'm with you on that. – Okay. – You're making the assertion that you're just not worrying about quantum supremacy or you're […] – Okay. Hashes, cryptographic strength hashes are already post quantum secure. If you go look it up. – The crypto agility element. – I haven't talked about crypto agility. The place where quantum attacks is not the hashes. It's the signatures. The public-private key pair generation algorithm, you can invert the public key and get the private key with crypto because you can use an attack that reduces its cryptographic strength from 128 bits to something much smaller. What you need then is for the public-private key pairs, you need to have a crypto strength public-private key pair. That's all you need for this because we're not doing encryption.

We do encryption, but KERI isn't about encryption. It's about authentication with public private key pairs. Now, because which I haven't talked about KERI, this is ACDCs. We have something called pre -rotation. The pre-rotated keys are protected with a hash, which means that they're already post-quantum proof, which means that they can get my signing keys because they know the public key. But then if I have a way to detect that they've compromised me because they entered something in my key event log and I see that they entered it, then I can do a rotation at the time of attack to a post -quantum safe public-private key pair. I can wait to the last possible minute to have to migrate. I don't have to incur the cost of migration now. We're pre -migrated. I don't have to incur the size and penalty. All that stuff can wait till quantum attacks actually start happening. We'll do it before then. 2030 is the date. But we don't have to do it today. And so that's an advantage, especially if you're trying to roll it out today to millions of people in the state of Utah. And you don't have to at day one say, well, we've got to be already post-quantum when this only just barely and maybe a couple years of seasoning is a good idea.

Now we're going to do something called partial disclosure. Here's my same credential, but I've collapsed my Merkle tree to the roots of each of the nodes. So 'givenName' instead of being block is a hash. Now I can present this without disclosing any of these field values. They don't know what they are and they can't guess them because of the unique entropy in them.

That now allows me to present any subset that I choose at the time of presentation without disclosing any of the other ones. I didn't have to do anything fancy. I didn't need black-magic crypto, which is an advantage because we want to adopt this at scale. We want to use well-seasoned crypto that's everywhere.

We're using stuff that's been around for 20 years. Elliptic Curve Cryptography has been around for 20 years. It's been stable. Before ECC, crypto meant that every two or three years you had to double the size of your keys because prime factorization was subject to Moore's law attack. But ECC has maintained its strength over the last 20 years despite a thousand Moore's law doubling of computation power because it always has been computationally feasible and they haven't come up with any way to break it. Now post-quantum has. Now we'll jump to the post-quantum but then hopefully we'll have a stable period for the next 50 years or so before we have to redo everything again. But during the 80s and 90s and 2000s it was like… [sigh] because ECC didn't go off patent until like 2011. Even though it was around for 10 years before that, nobody could use it because it's patented.

I'm going to talk about guardianship. A little more sophisticated delegation chain. Here's the same one here, but now I'm delegating to a citizen. Here's their AID that they're a guardian of a different AID. This ACDC is saying that this citizen guardian AID is a guardian of this citizen ward AID and this citizen guardian now, not the state, can now issue an ACDC as an authorization to the ward or the child in this case, that ward can present to social media. What we've done is that we've inserted the parent as now the authority structure. The authority isn't going from the state to here. The authority to access the social media is going through the parent. Even though this citizen might be over 13, which the law says, if the parent says, no, no, no, I want to control the access when and how much and how often and not just let the state determine that they're over 13 so they can do whenever they want to, now they can issue this social media authorization. And we're using the same technology all the way down. All we've done is we've changed to the issuer of this. This person is the issuer of that.

It's not fundamentally different from what I showed before. I'm at the point where without going into more detail you get a sense of why delegation solves the problem of modeling authority structures? That was a bad way to ask the question. What don't you get about that? Here's a problem with delegation. Up until ACDCs as far as I could tell when people built cryptographically verifiable delegation structures, these delegations were signed.

They weren't durable. Which means that if I have a chain of any length and that chain will break whenever anybody rotates their keys. Not because they wanted it to break, but because their key management forced it to break. DNSSEC is an example of that. You read the spec and you go, oh, this is a great idea. I'm going to delegate stuff. But does anybody use DNSSEC? Most people use DNSSEC say, can't really use it because I keep DDoSing myself. And why are they DDoSing themselves? Because every time anybody in the chain rotates their keys, it breaks their credentials. And they go, well, wait, you rotated your keys. My credentials should be valid. No, the authority is based on your current key state, not based on some other mechanism that is durable. We couldn't do this if we hadn't solved the durability problem a long time ago. Once we solve the durability problem, then it's like, oh, we can do delegation. Delegation actually becomes something that's not that difficult to do once you solve the durability problem. And if you don't solve the durable problem, then delegation is fragile in every case. And when people try to do it, it sounds like a good idea. And then it goes, oh, it doesn't really work. And then they throw it away.

And there have been lots of specs that have delegation in them, but they're not used very much. The idea is that if you don't start with durability, you can't get… there's certain things you just can't do like guardianship because guardianship is durable. How long does the guardian relationship last?

18 years. Sometimes longer than that. Actually, if you're on the other end of the age scale, it could be much longer than 18 years. My kids are already telling me that I need to… They need to be in charge of what I do. Anyway, the point is...

Now, don't get me wrong. There are lots of really good valuable applications of ephemeral permissions and verifiable credentials. No doubt about it. And ephemeral things have lower friction so they can be used more easily at scale, but they don't solve the hardest problems that are essential to government function. If you can't solve the hardest problem, adopting things that can never solve it means that you're just going to have years of heartache and headache because you're going to keep trying to be saying, well, can't we just do this way? And what you end up with is ephemeral systems trying to emulate durable systems and that's a really difficult thing to do because it means you're just reissuing

all the time. – It's what the internet's built on right now with certificate orders. – yeah. – The whole mechanism is you have a DigiCert up their own. It's a billion dollar company that rents itself the fact that you have to rotate your keys and they take advantage of that.

There's… – There's a business model for churn. I don't want to criticize certificate authorities. – Can I ask a dumb business guy question? – Yes. – Absolutely. – The dumb business guy question is this, would it be correct to assume that each delegation step has to have its own key management infrastructure? – Yes. Every identifier is independent. – Each one's got its own HSM, each one's got its own.

Yep. – The business guy nerd question is, that's really expensive and do they all have those already? – When you use “Hardware Security Modules”, if you're talking about like enterprise class HSMs, which are like $10,000 a piece.

Tomorrow I'm going to do a dinner talk on the security properties of KERI. This is ACDC. Tomorrow is KERI and we'll talk about tunable parameters that allow you to use relatively weak infrastructure, low cost infrastructure to get the same security properties as expensive infrastructure.

An HSM could be a $10,000 server that has signing infrastructure or it could be a $10 smart card. And these guys have a demo. The Veridian guys have a demo of a $10 smart card. Now you're just talking about each of these people don't necessarily need an expensive HSM. They just need a phone and a couple of smart cards. It's not a lot of infrastructure. Now, this person, this agent who might be issuing hundreds of credentials a day, they might want to have a little bit better infrastructure. But hundreds is not 10,000. And because of this delegation, you could have a hundred issuers and it doesn't matter. You can horizontally scale with issuers that don't issue a lot.

and guess what happens if somebody's keys get compromised? The whole infrastructure doesn't go down. Just that one person. You've minimized the blast radius by distributing the signing authority, which is why we delegate in organizations anyway. Can anybody by themselves run an organization of 20,000 employees? No, they got to have 10 layers of management and each layer of management has at max 20 direct reports. Anybody with more than 20 direct reports is either working 80 hours a week or… I know there are people who do that. Did that answer your question? – It did actually. – Great.

Any more questions? Yes. – I'm curious how latency and timeouts on the network actually impact this especially if you get into a doom. Can you get into a doom loop with this over network? No. That's because the proof is static. What happens is that we have something called percolated discovery.

I got to go back a few diagrams. See this seal right here, this little box, that's an anchor to an issuance. This KEL up to that point and seal is immutable. It doesn't change. If I give this much to the issuee and they put it in their wallet, they can now present that to anybody and prove. It's a proof. It's a durable portable proof. Nobody has to go back to the issuer. Nobody has to go back actually even to the network to verify that proof. No phone home anywhere. There's no network latency problem there because the verification is just a static proof. It's just a verifiable set of things in this key event log that was issued. And if I go a few events past that, you can see that there wasn't a recovery or something like that that could have invalidated it.

There's sort of like, well, up to this rotation, nothing can change. That's good. You're set. You're static. Everybody just has static proofs. And the only thing that the network affects is revokability. But revokability can be batched. I can say I got a 24-hour revocation time period, which corresponds with most legal basis for revocation, which means that if I don't have network for 24 hours, somebody who

had a revoked credential, I can't detect it. But guess what? Even if they did, it wouldn't be valid for 24 hours till the next day anyway. It didn't really hurt me. There's only a few cases where they do revocation faster than that. Absent somebody upgrading the FAA's routers… But the actual issuance isn't affected. It's only the revokability. And you might get a few people that had their credentials revoked that got away with it for two or three days. And that's a pretty minimal damage. Because how many people actually get their credentials revoked? Of the total population? Drunk drivers. How many of those?

On New Year's Day, there's a few. But the rest of the year, not too many. Does that answer your question? You'd have to see but that's the idea. – Where and how are you storing your raw data? – What do you mean by raw data? – Everything. It's a small blockchain.

Blockchain stores data. Where are we storing? – You're just storing it in a key value store. – So that amount of data you're talking about millions of people, how does that get secured? – The hashes of all data in the hash chains is self-securing. If somebody tampers with it, you can detect it. – I'm just wondering, fundamentally though, is it storing it in some kind of a block? Is it storing it in some kind of a text file? – It doesn't matter. Any database will work. What we use in our code is we use LMDB. LMDB is a really fast key value store… You're familiar with it. – Thank you. – Okay. Yes.

– Which hash scheme are you using now? SHA3, Blake? – It's perfectly cryptographically agile. We have a protocol called CESR. It's called Compact Event Streaming Representation. It's a protocol that has a compact encoding scheme for all cryptographic primitives. For example, this first character is actually not part of the hash. It is the algorithm. This is Blake 3. If it was Blake 2B, it would be an F. If it was SHA 3, it would be a G or an H. And if it was SHA 3-512, it'd be a different letter.

All of the hashes are in the table. And anybody that's processing it just looks it up and go, I know how to verify that hash. – In the distributed authority world, I know how this happens when something goes bad. You're looking at those statements, transparency logs, the distributed authority tells you. There was a slide here where it's like, and you will know something has gone bad and the word you was carrying a lot of weight. – Yes. We'll go back here.

Tomorrow's talk, I will answer that. – To be continued. All right. Thank you so much.

TALK II

2026-11-18-Samuel-m-Smith

State Endorsed Digital Identity Summit (SEDI) 2026 · talks · 57:10

I'm going to be talking about… I've learned my lesson. I used to talk about KERI and go through some of the simpler things. And I'm actually going to talk at a more high level but more technically deep if that's contradictory because I found that actually explaining the how and why of KERI is more interesting to people than the what. The what is technical detail that's easy to figure out if you're technical. If you're not technical, it's a waste of your time. But the how and why I think it motivates KERI a little bit more. I'm going to talk about that today. It's a long title. Somebody said this sounds like a thesis. And since I used to be a professor, I can't help but write everything like it's an engineering paper or whatever, but that's the way it is. And for those of you who don't know anything about KERI. If you go to KERI.one, the resources page, there's a bunch of links for articles that you can learn about KERI. KERI Foundation has videos online. You go to there. There's several videos from the last two conferences. The KERI Foundation also has a GitHub for those who want to download code. You can go there. And for those of you that have been out in the hallway, there's a couple of KERI Foundation developers that are showing demos. The Web of Trust, There's a lot of code there with KERI stuff. The KERI working group of the Trust over IP Foundation. You can go there to see when we meet. And also they have repositories of the specs. And then I also have a lot of papers and presentations on my own personal GitHub page. Those are all places you can go to learn more about KERI if this doesn't provide enough information. I'm going to start with a few preliminaries. We talk about the sort of fundamental assumptions on which KERI is based. And the fundamental assumption is that if I'm using really good crypto, which means crypto that is seasoned, it's been around for a while, it's not new, it's not black magic, it has been proven to not be vulnerable to brute force attack. That's the whole point. Now, quantum computers are changing that.

Don't worry. We're cryptographically agile. We can move to that. But at this point in time, these assumptions are valid and all it does is change what is good crypto in the future. But the point is that if it's good crypto, by definition, it's not vulnerable to a brute force attack. When you see movies and they're breaking keys or breaking signatures or breaking encryption and they're just doing it with some magic key, that's not how it happens. They can't brute force it. That doesn't happen. And we can migrate, so we don't have to worry about that. What the weakness is, is not brute forcing the crypto. It's called a side channel attack. What that means is you're not attacking the crypto, you're attacking something else. And principally what you're attacking is the key management. You can't break the crypto, but you can break the key management.

Most people when they talk about cryptography, they assume that the key management is not broken. You go read a spec for some sort of crypto and you get down to the security considerations. They'll say things like, we assume that your private keys remain private. And if your private keys have been exposed, then nothing that we said above works because it's all broken. If you go through life thinking that crypto solves the hard problem of key management, you would be absolutely wrong. Crypto does not solve key management. It solves a problem that is brute force attack. You can't brute force attack. And they do a really good job of that. Key management is the principal weakness because that's where most site channel attacks. When you get phished, are they attacking the crypto? They're attacking the key management. What are they phishing? Your keys or your password, which is your secret. Vulnerabilities are almost always side channel attacks. KERI uses a signed everything no shared secret security system. What are shared secrets? Passwords, tokens, TLS uses shared secrets. Even though it has public private keys, it's exchanging a shared secret. Shared secrets are fundamentally weak because two parties know the secret.

It's shared. That means either party is a vulnerability. A key compromised impersonation attack is where the host has really, really good security and the client has really, really bad security. They don't attack the host, they attack the client. They compromise the client side of the shared secret and then they can impersonate the host. Now the thing you want to get into is not the client. You want to get into the host. You've now made it so that some other client thinks that they're getting into the host and they're not. They're getting into a compromised host and then you're asking them to reveal other keys and other permissions and now that's the first step in a recursive privilege escalation attack. By saying no shared secrets anywhere, we've eliminated the single biggest chunk of vulnerabilities in all of the internet. We've thrown them out the window and said those don't apply to us. All right.

That's one of the principal basis of KERI is saying we're going to avoid being vulnerable to things by not using those things that make us vulnerable. It's a signed everything no shared secret approach. Principal weakness, side channel attacks, management of private keys. We still have private keys. We got rid of shared secrets. Private keys are not shared secrets if they're truly private. If I share something that's not private. The definition in the dictionary of private has 20 definitions. The first three or four are about intimate information. As soon as you share something, you're using a different definition of privacy. If I share something with another party, it's not private anymore. It's private if it had never left me. The idea of a private key is that it's never shared. If it's never shared, it's hard to phish because everybody is educated to share their secrets if they're shared secrets.

How do I log in with a password? I have to share my secret. Phishing is about confusing the user when to share and when not to share. If I never share, it's much harder to confuse me. Now, you still can, but that's why I say it's phish-resistant.

The idea is we're starting with the hardest thing for an attacker to attack is the private keys. For example, FIDO and passkeys use private keys in signatures. They're fundamentally more secure than anything that doesn't use private keys. But they don't solve the hard problem of key rotation. KERI solves the hard problem in key rotation. That's what we're going to talk about. What is KERI? It's a tunable fault-tolerant decentralized key management infrastructure based on a complementary set of threshold structures that make system failure practically infeasible. Up above, I talked about computational infeasibility. That means you can't brute force the crypto. And computational infeasibility means if you had all the supercomputers in the world trying to brute force it for the next trillion years, they couldn't do it.

That's what cryptographic strength means. It is just not possible. Practical infeasibility means that the resources available to the attacker to attack you, if you make your system sufficiently fault tolerant, it will always cost them more to succeed in an attack than they get from the attack.

They won't attack. Does that make sense? Does that definition resonate with you guys? Anybody? The idea is to make an attack more costly to the attacker than what they get from a successful attack. And despite what everybody might think, most attackers, there is a cost-benefit analysis they go through because there's a rich target environment. They're going to attack the things that are going to give them the maximum return. They're not going to attack the things that don't give them return. The design ethos of KERI is to make attack practically infeasible. Let's talk about fault-tolerant systems because I use that term. The definition of fault-tolerant system and in the English dictionary fault and failure are synonyms, but in fault-tolerant systems we define them differently. A fault-tolerant system continues operating despite faults.

An example would be you're driving a car and the rear passenger window breaks, the car can still drive. But if it's raining and your windshield breaks and you can't see, then that's called a failure because you can't drive anymore. If you have a flat tire for most cars, that's a system failure. Lots of systems that are fault tolerant have typically enough redundancy that things can fail and you can still complete your mission. I use this example from World War II. Tiger tanks had thicker armor, bigger guns, were really tough to beat. The Americans had Sherman tanks, which compared to a Tiger tank were just not all that great.

But the Americans had production capacity, they could afford to produce five Sherman tanks or even 10 or 15 Sherman tanks for every German Tiger tank. Five Sherman tanks can take out a Tiger tank. Now they can't take out the Tiger tank in a frontal assault. The front armor on the Tiger tank is too thick for the Sherman tanks. Their guns won't pierce the armor. They're just not big enough. They just bounce off. But if you can get behind the Tiger tank and shoot it in the behind, the armor back there is thin enough that you can take it out. The whole strategy is to send five tanks around the Tiger tank and get it confused enough that it finally lets you get behind it and you can shoot it. And there's a movie where they demonstrate that. What's it called? – Is it Fury? – Fury, yes. They actually do that in the movie. They have a squad of five Tiger tanks. I mean Sherman tanks.

The Tiger takes a couple of them out right away and then the other ones manage to kill it. A squadron of Sherman tanks is fault tolerant. Much more so than the one tiger tank. What we call a fault is that if some part of the system malfunctions, that's the fault. That's not the same as a failure. I've already given examples. You can have faults all the time, but if you can complete your mission, you're fault tolerant. In other words, the faults occur, but it doesn't matter. You can continue.

When the system as a whole is no longer able to achieve its mission, then we call that a system failure or a failure. What we're trying to do is prevent failures. One of the things you do in a fault tolerant design is you look for single points of failure. If I got a car with four wheels and I can't drive with three, then every wheel is a single point of failure. I've got four single points of failure. That's not a really good design. It's just not going to survive. Another point is called a common mode failure. That means multiple components that share the same failure mode. The same thing takes them out. And in computer security systems the FAAs routers with copper wires that takes down every single FAA system, that's called a common mode failure. You just experienced a common mode failure. In fact, we had multiple common mode failures this last week. In multiple systems. People are not using survivability engineering to design those systems. This is an engineering design failure. This is people building systems not following textbook design principles for building those systems. And guess what? Key management historically has not followed engineering survivability engineering practices. It just doesn't because cryptographers in general don't care. About key management. They care about brute force attacks so that it's really good. And it's not a criticism of cryptographers. If you're an academic who's trying to publish papers in cryptography, you're not going to get very much acolytes for publishing a paper on key management. It's just like, that's just boring, dumb something called key management that's about user interfaces. The key is if you can repair faults as they occur so that a cascade of faults does not lead to failure, then you have a really fault -tolerant system. If you have just redundancy, let's say you have five Sherman tanks. And you go to battle and two of them get killed. Now you're down to three. You go to battle again, two more get killed. Now you're down to one and then you're done. That's a system failure. But if you have a way to replenish those Sherman tanks as fast as they get killed. Like today you fought, you lost two, and then two more get dropped from the sky and with new guys and the next day you're back to full strength. If you can replenish every day, then you

never fail. You go the whole war and you never fail. They'd have to take all five of you out in the battle for you to fail. The goal of really good fault-tolerant systems is recovery from fault, not redundancy, not resiliency, but recovery. If I want to recover from a fault, I have to be able to detect a fault. If I don't know that I had a fault, I can't recover from it. Most computer security systems, you can't detect the faults until they cascade into a failure. If you think about what a breach looks like, they're usually like a recursive privilege escalation attack. There are a set of faults that happen. And after a certain amount of time, those faults result in a true breach. But none of those faults were detected until it was too late and then you had the breach. Failure to detect describes pretty much every computer breach. There was no ability to detect systems. Survivability engineering is something that I worked on as an academic. I did a lot of work for the Navy. I started doing autonomous underwater vehicles and I went into survival shipboard automation. And then when I retired from the university, I went into critical infrastructure system design and so on. The idea though is that there's this old technology called reliability engineering. It's been around for years.

Reliability engineering is why the Japanese car industry, automobile industry was able to become high quality in a very short period of time because they, in their factory, said, we're going to do reliable engineering. Anybody heard the phrase six nines? Reliable engineering is where six nines comes from. You look at all of the faults and you figure out the likelihood of the faults and then you work to make those faults only occur once in every 10,000 times. So that you just don't have faults. You build your systems like that. The problem is that in the military, we have a hazardous environment. An automobile motor that fails because the rings on the piston wore out after 100,000 miles is reliable engineering. An automobile engine that fails because a rocket hit it and blew it up is survivability engineering. The time frame the dynamism of the fault is what makes it go from reliability to survivability. You're in a hazard environment. People are blowing things up.

Like we did autonomous underwater vehicles underwater. These vehicles are electric in saltwater. If you know what happens to electricity in saltwater, it conducts. And you have metal canisters in saltwater with electricity inside of them. That's a dangerous thing. That's corrosion waiting to happen. And in corrosion at accelerated pace means that at some point you're going to have a leak. In fact, that was… Never mind. I won't say that story. Anyway, hazardous environments. What we did is we take the reliability engineering techniques, we combine it with critical infrastructure because we're trying to make that survivable, and then we come up with risk mitigation design techniques. What critical infrastructure engineering, which is related, did is it's you look at a system and you say, where do I put my resources?

How do I allocate resources to mitigate bad things from happening? Who in here knows… What it says is you take the likelihood of failure, you multiply it by the significance of failure. And typically what you do is you just assign a dollar value. That the actuarial insurance industry has produced a cost of death. Everybody's life is worth a few thousand dollars on average. It doesn't sound really great, but it's true.

If an asteroid the size of Rhode Island were to strike the Earth, what would be the significance of that failure? Depends on where it lands. If an asteroid the size of Rhode Island hit anywhere on the Earth, what would happen?

Instinction level event. The significance of that failure is all the value of everything on the earth as we know it. It is as high a significance as you can get. Now, what's the likelihood of an asteroid the size of Rhode Island striking the earth? It's a really, really, really teeny number. The criticality is actually quite low. We don't spend resources to protect ourselves from asteroids striking the earth. Now, there is an extinction-level event that has a actually quite high likelihood. Does anybody know what it is? Nuclear war. Another one is a solar flare. We have solar flares that would put us back to the dark ages and we've

had them happen in the last 10 years. They just happened to be pointing away from the earth at the time. But if they had hit the earth, they would have taken out all of our satellites, all electronic infrastructure. We wouldn't have electricity. Everybody would starve and die of food and die of disease. And we'd be living like in the 1700s in about a year. I know you really like that.

The point is that when you're designing a system using these principles, you need to figure out what your risks are in a criticality. And I often get in these discussions with people who say, well, this could happen and this could happen and this could happen and they don't multiply the significance it by the likelihood. There's lots of things in a security system that if they happen, it would be really, really bad. But the likelihood is so small that it's not worth the resources. Could an attacker compromise grandma's cell phone? Absolutely. Grandma's cell phone is protecting $100 in grandma's bank account. What's the likelihood that that attacker is going to waste 10 times that amount of effort or 100 times that amount of effort to actually compromise grandma's cell phone? Zero. The point is that unless you weight every failure mode by the significance, it doesn't matter that there's a failure mode because it won't happen because there's no incentive. You rate and rank infrastructure threats mitigation via criticality. You then establish a risk profile based on the threat environment. And then you use survivable engineering to figure out how to design the system to protect against the risk profiles and the threats that have high criticality. That's the number. You go to the highest criticality, protect against that one, then you just go down the list until you run out of resources. And then you go, did we get practical infeasibility when we ran out of resources? If we did, great, we won because they're not going to attack us. Or if they attack us, they're going to lose money, so they'll give up after a while.

What you usually do is you build what are called reliability block diagrams, where you take all the system components and all their dependencies and you calculate all of this stuff. And I'm not going to go into the math here, but it's well established. What does survivability look like? These are some of the systems I worked on. These are the autonomous underwater vehicles. This is a ship at the Naval Academy that participated in Fleet Week. What that ship had is a self-healing communications network, a self-healing power network, and that was all automating the steering and propulsion on the system. And we built scale models of that same technology used for the next generation surface combatant.

And that was going really well until 9-11 hit. And then because that was to solve blue water Navy problems and not brown water Navy problems. The Iran-Iraq war was a brown water Navy problem. And then behind that is a picture of a electronic substation that we automated with robotics and stuff like that, but also done on that same design. And one of the ways I got into it as a consultant, we did a criticality systems analysis for all of Miami-Dade's infrastructures. It took a year. And that was actually after 9-11 because everybody after 9-11 was worried that all of their buildings would get bombed. They spent time on it.

Survivability is three factors. Susceptibility: likelihood of attack, vulnerability: likelihood of damage at a given significance given an attack and recoverability: the likelihood of timely repair of damage at a given level of significance before failure. Damage is a fault. I have a slide. I don't have it here, but you guys know what it is. You've seen this. It's a joke. If there are two people in the woods and they see a bear and one goes to the other one, there's a bear. What do we do? And the other one says, run. And he goes, we can't outrun the bear. And the second one says, I don't have to outrun the bear.

I have to outrun you. When attackers are attacking a system, they're going to attack the slow fat one before they attack the fast one. And if there are enough slow fat ones, they never get to the fast one. You only have to be a little bit more secure than your target-rich environment. Now, the problem with

that is that with AI, the resources of the attacker have been multiplied so that everybody looks slow and fat compared to the AI. That's a problem. But still low value targets are low value targets even if they're easy to eat.

You're still not going to expend resources to get them. In other words, the bear can chase you down and eat you, but if the bear has to run 50 yards, it may not because it says there's just not enough food there. You're just not worth it. You're not worth the effort. If you're not worth the effort, your susceptibility is really low, which means you won't be attacked, which means your survivability is actually quite high. Vulnerability is given that you're attacked, can they do damage? If a Sherman tank is attacked by a Tiger tank, it's done. It's vulnerable. But can the Tiger tank take out five Sherman tanks? No, because they're mobile and agile and they can move around and it might get one or two of them, but it won't get all five of them. The vulnerability significance is a function of how much damage happens and does the damage result in failure or is the damage only a fault that you can recover from? Recoverability is the likelihood of timely repair, which means you can repair the damage before failure. And if I can repair the damage before failure, then my likelihood of failure goes way down. Likelihood of failure is susceptibility times the vulnerability times one minus the recoverability, which means if I have 100% recoverability, then I can have zero failure because zero times anything is still zero. If I have one of my recoverability, then my likelihood of failure is zero. That's an important insight. We did some work, especially when we were doing the shipboard automation. We used AI back in the 90s. When nobody knew what it was to analyze all the different topologies of networks and figure out which ones had the highest survivability for cost. You do a cost survivability trade-off and guess what? Networks that could recover were much more cost-effective at survivability than networks that were highly redundant. If I had 5x redundancy, it was actually less survivable than a 1.5x redundancy, but where I could repurpose the redundant components to smartly repair failures where they occurred on the network. I needed a different topology than I would with just hot backups everywhere.

What's a threshold structure? A threshold structure multiplies the number of attack surfaces that must be simultaneously compromised in order for an attack to fully succeed. I've already given an example of that. A squadron of Sherman tanks can take the tiger out unless the tiger kills all five of them first. If one's left and is behind the tiger, he wins. That means the tiger has a… They should be afraid. They should be going, we got stronger armor. We've got a bigger gun, but we're slow and that Sherman's much faster than we are. And if you watch the movie Fury, the rate at which that tiger tank's turret could turn to track a Sherman, if the Sherman's 500 yards away, it can track it. If the Sherman's 20 yards away, it can't keep up with it, which means the Sherman can get around behind it and kill it before it can turn and bring its gun to bear. You have to be really brave. You have to get in close. You have to fight dirty, but you can win. A fault-tolerant system detects and recovers from a partially successful attack, thereby reverting the attack progress. Imagine the attacker starts to attack your system. They get a little bit of the ways in it. You detect it. You recover from it. And now they have to start all over again. And then the next day they did the same thing. Detect, recover. Same thing. Next, detect, recover. And that goes on for several months. What's the amount of resources they spent to attack you? Futily with no return. That's part of the ROI. Recoverable systems that have fault detection are really, really good against attackers because you can get a perpetually secure system. You can have a system that if it's timely enough and has enough recovery resources and in the threat environment, you've done the design right, makes it so the attacker doesn't want to attack you. In KERI, a recoverable fault is a successful side channel attack on signing keys.

KERI design assumes the worst case failure that everybody else says can't happen. Everybody else says if you compromise your signing keys, you're done. All of blockchain for the most part, it hasn't changed. Not yet. If your blockchain signing keys are compromised, they can take all of your money and there's no way for you to recover. – If they have a verifiable smart contract. – Okay. They're working on it.

But if I have a passkey that allows me to access my bank account and the signing key is compromised, I'm done. That means that in KERI, we say that the fault isn't compromising the signing key. The system failure is a different thing. It's loss of control over the identifier. Does that mean that if your signing keys are compromised, there's no danger to you? Absolutely not. In KERI, if your signing keys are compromised, you could have harm. But if you look at the types of systems that identities use to protect things, there are things we can do so that a loss of signing keys, you can mitigate that harm significantly as long as you don't lose control of your identifier. I just want you to understand that that's our definition. We could have defined different faults and come up with something else, but the side channel attack that is really hard to do, that we say we're going to allow that as a fault that we can recover from means that we're in a different regime from a security perspective. We've gotten rid of all shared secrets. Shared secrets aren't in our vulnerability profile. And even if you compromise signing keys, it's a recoverable fault for us. We're already just in a different regime. By design, KERI is in a different regime. And a lot of people don't understand that KERI isn't about cryptography. KERI is about fault tolerant key management. KERI less about the cryptography. I'm just using vanilla dumb, what do I call it? White magic cryptography. It's off the shelf stuff. It's nothing fancy. It's nothing like even now with the post - quantum crypto. One of the problems people are saying is, NIST is standard on all these post-quantum crypto stuff, but you know how much seasoning that post-quantum crypto has had? Like two or three years at most. Do you know how long it usually takes for good crypto to be seasoned before people are confident that there aren't any lurking vulnerabilities? More than a decade. – But with agentic probing, perhaps, maybe that timeframe can be compressed. – Perhaps.

The point is that if we design a system that already has vulnerabilities built into it because we've taken risky design choices, then it's hard to make guarantees. Elliptic-curve cryptography has been around for over 30 years. It was on patent for 19 years. But since 2011, it's been off patent. Since 2011, it's proven itself to be invulnerable. Despite multiple Moore's law increases in computation and stuff like that, it's still just as strong as it ever was until we get quantum computers and then quantum computers can break it. We start by designing the system so that faults are always exposed. The rule is if you're going to build a fault-tolerant system that has recoverability, the one thing that you have to build first is detection of fault. You don't hide the faults. Lots of systems are built to hide faults. A lot of distributed consensus algorithms hide faults. They make it so the system is live. It always has to operate. It can't ever stop. And that means that it has to make a decision.

And if that decision bakes in into history that a fault like your keys are compromised, it's unrecoverable. It's baked in. And people don't think of it that way. I'm not criticizing blockchain systems. I'm just saying that exposing faults means that you usually make a trade-off where you say if a fault occurs, we stop, we repair it, and then we resume. We don't just keep going and ignoring the fault. It's better the fault you know about but can't fix just yet than the fault you don't know about even if you don't need to fix it just yet. You think, wait, I don't need to fix it yet. Why do I care? Because a failure is a cascade of faults. If I don't know about faults, then they're going to compound. They have no choice but to compound because everything's faulty. Until eventually you have an unanticipated catastrophic system failure. You didn't see it coming. And that describes pretty much every security breach. The first thing we have to do

is build a detection mechanism. In KERI, the main fault that we're detecting is duplicity. The aha moment I had in designing KERI was that a compromised controller and a malicious controller (compromised means their signing keys are compromised or the signing infrastructure is compromised) and a controller is malicious can be forced to exhibit the same fault, duplicity. If I can force the fault to exhibit in a way that I can always detect, then I have the first fundamental principle of a good fault - tolerant system that has recovery. I can have a fault-tolerant system without recovery, but it's not a very good fault -tolerant system because eventually after enough faults, it'll fail.

The timeframe extends. But if I can recover, then it's perpetually never going to fail. Does that make sense? If I want it to never fail, I have to have recovery. I can't just depend on redundancy. Almost every cryptographic system that I've seen depends on redundancy, which only extends the life it doesn't prevent failure. It just means you won't fail today, but you'll eventually fail. External inconsistency looks like duplicity. And I'll define that in a little bit.

The policy is that I can trust KERI as long as all the non-reputable signatures verify and there's no evidence of duplicity. I verify everything and then I check for duplicity. If duplicity exists, then I stop trusting. I don't continue to operate. I say, I can't trust that controller.

Very first sign of duplicity, I stop. That protects me from that fault causing me any harm. And I don't resume trusting until the duplicity has been reconciled. There's two parties here. There's a controller that's been compromised and there's the relying party that's interacting with the controller. The duplicity thing means that the relying party doesn't suffer any harm when the controller is compromised. Now, the controller can suffer harm, but I've already gotten rid of half the harms. Half of the interacting parties are protected. Now, I have to worry about the controller. We force all faults to exhibit duplicity. I'm kind of saying the same thing over and over again.

But this forcing of faults to exhibit as a detectable fault makes it so that the system relative to its properties is actually very simple. And that's the thing. People say KERI is complex. I say no, KERI is actually the simplest way to get this level of security. If I tried to get the same level of security not using KERI's properties, it would be way more complicated. If you think it's complicated now, try to get this. And that's the reason why most systems don't have this level of security because it's too complicated. They don't even try. I was stupid enough to try and it actually worked. I'm like, Okay. But it took a while. I simplified fault detection. I made it as simple as possible, but no simpler, to quote a phrase. If I only have one fault I have to detect. And all the faults exhibit that way, and I can design the system so that everything that fails, fails in a duplicity evident way, then I have an endemically fault-tolerant system because I'm just detecting one thing and repairing one thing, over and over again.

And if I can make the cost of compromising my duplicity detection higher than the return gain from doing so, then I have I've achieved my goal, which is making it so the attacker always loses whenever they attack. How do we make things duplicity detectable? We sign and hashchain everything. Everything is signed in hash chains. If you saw my talk yesterday and I had hashes everywhere, that's because I have to hash everything to make it so that I can detect duplicity because a hash or a digest of something that is signed cryptographically is as strong as signing the thing that is hashed, which means now I can build hierarchical data structures of hashes and I just sign the hashes. I don't sign the other stuff and that means that I can build more complex structures and still force the duplicity. A digital signature requires the signer to know the private key. I'm just kind of restating the obvious, but I want to go back there. If the signer has to know the private key in order to sign it, then if something is verifiably signed, then there's only two possibilities.

Either the signer acted maliciously using that private key to sign something they weren't supposed to sign, or that was bad for them to sign. Or their signing infrastructure was compromised. But from the verifier's perspective, it doesn't matter whether it's malicious or compromised, it looks like duplicity because it's non-reputably signed, which means if they sign two different things that are supposed to do the same thing, then you have duplicity. This signed hashchain data structures make everything provably duplices. And provably means that I don't have to trust anybody.

Anybody that detects duplicity can publish the detection in a provable way. I don't have to trust the detector. Everybody can become a detector and I can verify that it happened. A verifier can say I don't have to have any trusted third parties. I just need to have mechanisms that prove to me duplicity. And then I'm protected. That means that successful attacks are forced to exhibit in a detectable way. I know I've gone through a lot of slides with a lot of stuff, but I'm going to do this and then go back to the actual detection mechanism so you can understand what they look like. What KERI does is that we have multiple threshold structures. We have a single fault type, a fault tolerant detection mechanism for that fault type, which I haven't explained what it is. We have layered threshold structures, not just one, but multiple. Each layer makes it exponentially harder to attack because the threshold structure multiplies the number of attack surfaces that have to be signlessly compromised. Let's say that I have a Tiger tank and a squadron of Sherman tanks. That's one threshold structure. They got to get four or five or something like that to win. And then I layer on top of that another five squadrons. Now to be successful, they have to take out five times five. And then I layer another on that another five times every one of those five. Now they have to take out 125 to be successful. That's why it's exponential. If it's low cost for me to layer threshold structure, which it is because it's just software, then I can quickly build a system that is practically infeasible to attack because I've exponentially increased the cost of attack without exponentially increasing the cost of building the systems. And then I build in recovery mechanisms. Even if the attacker attacks is partially successful, I can detect the attack in progress, recover from it, and make the attacker start over. Tunability is that I can take those layered threshold structures and tune them down to the particular risk profile. Grandma may only need single SIG with a pre-rotated key or one of two pre-rotated keys with a smart card. And that's all she needs ever will need for security. Whereas the root of trust of GLEIF might need a hierarchy of delegated AIDs with 12 signers at the root and key escrow mechanisms and you can't have all seven signers that are GLEIF employees being in the same building at the same time. You can add all kinds of stuff to make it so the attacker can't get to it. Tunability means that the hardest thing about security is friction. If there's too much friction, nobody will use it. You want to reduce the friction to the point to where people will use it while still maintaining the fault tolerance, fault resistance for the risk profile. And if the tunability is there, then you can find a solution that fits the risk profile. KERI is complicated because it's tunable.

But once you tune it, it can be as simple as it needs to be. It doesn't have to be all that complicated because a given risk profile may say, I don't really need more security than X. If a passkey is really strong compared to passwords and that's strong enough, then KERI is already stronger than a passkey. Just a Single Sig KERI is stronger than a passkey. That's all you'd ever need. The friction is really small. But if I'm trying to protect a large enterprise like the state of Utah or any state for that matter at their root of trust, I might want to have a few other tunable parameters to make my security little bit stronger. What are those threshold structures? Key Event Logs, signing keys, and pre-rotating keys.

There's two different sets of keys. That's a threshold structure. Multi-sig signing keys. I can have multiple signing keys. Multi-sig rotation keys. And understand that's multiplying. I have two and then each of those two could be five or four or three. Now I have six. I went from two to six just by

multiplying. Hierarchy of delegated AIDs. If I have delegated AIDs, each AID in the delegation chain protects all the AIDs below it. The only way I can compromise this AID and gain control of it is I have to also gain control of the delegator one. And if it has a delegator, then I have to gain control of that one. And if this one's a Single Sig, but the delegator is a multisig, then I have a threshold structure multiplied by the keys of this one. And if he has one that's also a multisig, then I multiplied five times five. I'm now at 25 keys that they have to compromise in order to gain control of this thing that's only single sig at the bottom. And it's not a lot of work. It's just software at this point. The user doesn't have to see that because the user is down at the bottom and the people in the middle don't have to use their keys very much because all they're doing is delegating. They're not actually signing anything other than a delegation every time they hire and fire somebody, which might be months?

Witness pools. I'm going to get to witness pools. Somebody wanted to know about witness pools. We'll talk about witness pools. The watcher network. Time delays relative to detection latency. This is probably the most subtle one that people don't understand. Detection takes time. It's not instantaneous.

People always say, but you've got network latency, so you won't detect the detection in time to stop the fault. What I do is I make failure a multi -step process. I designed the system so that the attack starts with an event that's detectable within a network they can see. And then the next event happens at a time delay greater than my detection recovery latency. They can't actually succeed in the attack until I've ensured that the first step of the attack has already been detected. Which means that I can stop the attack before it gets to any point. I'll give you an example. I want to borrow money from a bank. Somebody wants to steal my identity, so they're going to go to the bank with the stolen credentials and borrow money in my name. I'm the one that's going to have to pay the loan back. They're going to skip town with the cash.

It's a really bad thing. The bank doesn't want to loan money to somebody that's going to skip town with the cash because they have some liability as well. What the bank does, it says we don't loan money on just a simple presentation of credentials. That would be a really bad idea. What we do is we'll start a multi -step transaction where you come in, you sign a promissory note, you anchor the promissory note in your Key Event Log. That now means that you have a detectable event. The only way it gets anchored in the Key Event Log is if they control the signing keys. Either you're stupid and you're borrowing money and you know what you're doing. Or your keys have been compromised. Either way, it's detectable. Gets anchored in your Key Event Log. All of a sudden, you get a notification on your phone that says, hey, did you intend to borrow half a million dollars? Because we just have a promissory note anchoring in your key event log that you didn't put there. You go, no, I didn't. In fact, I didn't put any of that stuff in there. You do a key rotation to recover from it and then 72 hours after the promissory note, the bank says, come back in 72 hours and then we'll give you the money. The imposter comes back in 72 hours and goes, give me the money. And the bank says, well, that promissory note you anchored three days ago isn't there anymore. We're not loaning you the money. They can't ever steal your identity because you've extended it. – In this case, where your signing keys are compromised, usually the device that the person would get the signal that they would be inducing and have anchored this promissory note in their KEL might be the same device that also was… – It depends on how the signing keys are compromised. – Right. But let's just presume it's both like the same… – But you could have a watcher that's watching your witnesses. That's a different device. – No, I understand. But the user has to have awareness of some signal that isn't intercepted that tells them, hey, you did this. – Yes, that's called an eclipse attack. Most attacks, unless you're attacking a high-value individual, don't have

the resources to capture all of those. – I understand that, but I'm just still trying to think of what ways of recourse you could establish so that that signal could get to you. Even if your device… – If you have multiple watching devices, then it can get to you and you can have multiple ways to… And if you have enough time, like 72 hours, then there's a really good chance […] or you could just extend the time. – Or you have like a security mesh where you have a collective agreement where I'm watching my friend. – Yes, there's lots of more sophisticated ways you could do it. The principle isn't the actual mechanism. The principle is that if I have fault detection that has a latency, I can design high value transactions so that they're always detectable.

And that means that no harm comes to me. – I'm not disagreeing with you. – TELs plus ACDCs just extend fault detection to things that are not just your key state. Those are the threshold structures. I'll tell you why witnesses are a good fault detection mechanism. Witness pool as a fault detector. How does it work?

You may have seen this slide. This is a KERI infrastructure. You see in there that there's witnesses and watchers. I'm just going to take one person, entity A. Entity A controls a pool of witnesses. It's the entity's witnesses.

They control them. It's not somebody's shared infrastructure. They also control watchers. It's their watchers, not somebody else's watchers. These watchers, a local watcher, is doing something very important. It's watching the controller's own witnesses. It's watching its own witnesses. And what it's doing is it's comparing what the witnesses are saying about your key state to what it knows as your wallet's key state or the source of truth that the controller thinks of as the source of truth. Because the witness pool is there to be provide high availability and it has to be public and it's in the internet, it's going to use a fault tolerant structure itself. It's going to be like a three or four witness network.

What happens is that in order for a witness to witness something, it has to be verifiably signed. The only way a witness can witness something is if it's signed with the private keys. If something shows up that the witness says, hey, somebody gave me an event. It's signed. I witnessed it and a majority of the witnesses witness it. Then the watcher says, but the controller didn't tell me that that was an event that was valid. I can notify the controller that there's an event that's invalid in your witness pool. And that's evidence of duplicity because I've got two different versions of the Key Event Log.

One that is the source of truth and one that is what the witnesses are saying. And the attacker doesn't know about the watcher. The watcher is not public. The watchers can be a totally private network, put it in devices redundant. It has a different security policy from the witnesses. Now you can detect any events that are anchored. The only way that I can issue an issuance if I have a policy that I only issue issuances that are anchored, I forced all issuances to make any key compromise detectable for those issuances. Now I just have to detect that one thing. The witness pool is my detection mechanism and it's a highly fault tolerant mechanism. That's why it's there. It's not just for availability, it's for detection. Now I can detect the events, which means now I can recover from them because I have another threshold structure called my pre -rotated keys allows me to do recovery and they can't attack the pre-rotated keys if they're air-gapped or I do appropriate things. The attack surface for the attacker gets really difficult. And all I have to do is go to multisig and go to delegated AIDs and a few other things. And all of a sudden now the number of keys they have to compromise is really high. And as far as I know, for multisig, there have been very few successful attacks and most of them were things like a code supply chain attack. And I think there was an exchange where they had a multisig wallet and it was compromised. It had “two of three” and they sent an email to the three holders that said, there's an

upgrade of your wallet. Click on this link. And they clicked on the link and it downloaded a compromised wallet and then it explored… There are still things that can happen, but you can minimize them.

Why are witnesses in KERI? The witnesses are in KERI so we can detect key compromise and the watchers allow us to detect duplicity and we've got a watcher in here detecting duplicity which allows us to detect key compromise. Key compromise exhibits as duplicity. Did you guys get that?

Everything exhibits as duplicity and if I have a duplicity detection which is a watcher I just have that one thing that I can detect. Any questions? – This is all great. You see the duplicity. Yes. – How quick is the recovery? You have pre-rotated keys that you can go straight to? – Yes. – Because what I'm looking at is, they can't get to the keys, but they could give a denial of service attack. – Yes, absolutely. – And just keep doing that to where… – It depends on how you do it. For example, if my pre -rotated keys… I'm going to use the Veridian example because they give really good demo. Where was that? IIW, right?

They have these little smart cards that have a biometric on them. They cost dollars. Or pennies. What'd you say? Pennies now? – The ones with the biometrics are less than 10 bucks. – Less than 10 bucks. I have two of those.

They're air-gapped. They're NFC. They're not using a network. I have to actually touch the device. I can store them in my safe. I can put one in a different house. And if I lose my device, I just get a new device and I can recover to a new device. Even if they get my phone and compromise my phone.

As long as I have some notification that I get someplace. Any of my devices could notify me that A that my keys have been compromised, then the recovery is how long does it take me to go get my smart card wherever it is like in house of my safe and recover to a new device? That's my recovery latency.

And how do they do denial of service attack on an air-gapped system? It's really hard to do. Because I could recover on a device they don't know exists until I get the new device. They don't know that it's a device to attack unless they're like physically present following me around and most of these attackers are not physically present following me around. They're across the ocean someplace. It's not hard to build these systems to be really secure relative to the threat profile or the risk profile. Anyway, I passed my time. I thank you so much. I'm sorry.