xxDK Webinar and React.js/Golang Demo

For the Hackhazards Event in Delhi, India

Mar 16-17, 2024

SPEAKERS
FlowerTree, Rick Carback

FlowerTree 00:00
So Richard is one of the cofounders of the XX Network, and has been working on the development for this since the start. He's also the chief cryptographer, and has been doing this work for a very long time. If you look into his previous experiences on LinkedIn, you will find that he's worked as a software engineer in the late 90s for government services. And the reason why we have got this XX Network is because a lot of people who are looking forward to promote privacy, we are looking forward to getting online. And when we get online, we need a lot of tools that help us in keeping our identity, our online identity, private, that is one of the most important things that we do. So XX Network has got some amazing technology. The XX Network ecosystem has been founded by Dr. David Chaum, I'm sure Rick could tell more about him in the session.

FlowerTree 00:45
It's just that we are very excited. This is the first event that we are sponsoring officially in India. And we would really love to introduce the developers here to XX Network and its technology. India being one of the biggest producers of software, and technologies could contribute a lot in what you're trying to achieve. So without talking more, I would hand over the presentation to my colleague, Rick. Over to you, Rick.

Rick Carback 01:07
Thank you so much for the kind introduction. And this session is going to be more coding than history lesson, but I'll start out with that. And as you said, I've been working with David Chaum since I graduated college. I met him when I was at a presentation in my senior year, where he was talking about a voting system. And he made this comment. He said I haven't found anyone who knows how to build this yet. And it became very apparent to me that it was very obvious how to do it. And I think about six months later, we were demoing it for people. So it's been it's been a long journey. I think it'll be 20 years, next year. So I've been doing this a long time.

Rick Carback 01:39
Without further ado, let's get into it. So we're going to cover the XX Network's developer kit, there are a couple of public examples that you can go and use - these links here on the right https://proxxy.xx.network, you can go to https://alpha.speakeasy.tech, and you can go to https://echoexx.tech. This is a brand new system that uses the Ethereum public/private key to create an identity on the XX Network and allows you to send payments to others and be sure that the person you're sending to is

that person. And if you want to learn more generally, from a developer's perspective, right now, the best place to go is https://wiki.xx.network.

Rick Carback 02:09
So a little bit of history before we get started. Back when the internet was first developed, they decided how your digital life would work. And that happened in the early 80s. Before I think most of us on the call were born. And David produced this mixnet proposal. And the idea behind that was how do we ensure that what we have in the present day (in the 80s), where if you talk to someone one on one, that conversation is private. Can you provide that level of security when you go on to the internet? And that is how your digital life should work. And since then, they've only taken that first path, largely for optimization reasons, right? It's faster, right? But we are finally getting around to can we, (you know, 40 years later), can we can we make this work? Can we provide an option for people to really have high security, high metadata protected environments so that you can go and operate online and not become a target for malicious actors, malware, bad guys.

Rick Carback 02:56
Let's talk a little bit about what a mixnet is and why you would use it. So the basic system is you pass your message to a node, and then multiple nodes mix that message. And generally speaking, what they do is they are adding encryption and then removing a little bit of encryption at each hop. And what that allows you to do is, under the set of all messages sent, be sure that the sender and receiver are unlinkable so that you don't know who necessarily sent the message to the recipient, and the recipient has no linkage back to the sender, per se unless that linkage is included in the message that was sent.

Rick Carback 03:26
It was introduced, like I said back in the 80s. And it was at the time impossible to build these systems because we did not have the computational power. And that's changed recently in two major ways. One is... David and a bunch of colleagues, my former adviser Alan Sherman, and I can't remember the names but they created a... they realised that you could pre-compute the message computation, the public key part of the messaging. And they published a paper about eight years ago in 2016, 2017. And the second thing is that we now have these GPUs, they're doing AI, they also happen to be very good for these types of public key operations at scale. So when you have that pre-computation step, you can reduce the real time latency down to a very small amount. And we have we have real time latency in the network on the order of under a second at this time.

Rick Carback 04:10
And what that means is we can we can produce and push significant amounts of data through the network, not direct connection level amounts of data. But it's competitive with a lot of messaging platforms out there today. So the messages will appear, you can bring up, you know, iMessage, or telegram or something like that. And you're compatible with the delay on systems like that. And that's kind of how we built our platform. So we take advantage of this pre-computation, step, the P here in the diagram on the bottom right and the real time - we line up a bunch of real time operations so that you can get a continuous flow of real time sends and real time receipts. And the delay on that is definitely on the order of about a second. And if you break that down, there's five nodes in a team, and there is roughly 50 to 150 millisecond delay between the nodes and you're gonna pass through those nodes. So add that up, you're well within a second to one and a half seconds if one of the nodes is slow.

Rick Carback 04:59
And why would you use this? You use it for messaging applications, you're not necessarily going to post a video or audio through it, but you're going to, you know, establish the link and establish sort of an anonymous key, so that you can use a separate server to send and receive audio and video, there's all kinds of possibilities. The big one for me has always been election systems, you literally can't have a secure election system without a platform like this. And it's very exciting. We've now got 370 nodes that are active. And this this has been running continuously since 2021, in November.

Rick Carback 05:24
So that's enough of the high level academic discussion, let's talk about how you use it. And all the examples I'm gonna show you today are at this Git addresses - it's https://git.xx.network/xx_network/xxdk-examples, you can download them, you can run them, they've got READMEs that explain how to run them and how they were built. So that you can go and build applications with this technology using this technology. So like I said, the beginning we've got JavaScript, we've got iOS, we've got Android support. The main clients are on the left here on the bottom side under the brand name elixxir, that is the US base entity that built a lot of the mixnet client software. And on the right, you've got Speakeasy, which is a public open source application, you can download that and compile it. There's an example Android client and an iOS client if you're interested as well. And those are a full featured applications.

Rick Carback 06:04
Let's dig in a little bit and give you a very simple JavaScript example. That example is the React.js folder inside of the xxDK examples repository. The first thing you're going to have to do is download Node.js. I hope everyone on this call knows how to do that. But if you don't, go to https://nodejs.org/en/download/, and it will present you two or three installers depending on your operating system. Once you install it, you can open up the terminal window, you can type "node -v" to get the version. And "npm -v", these are the versions we're using for the example.

Rick Carback 06:32
Once you've done that, you can create a brand new app. "npx create-next-app@latest". I literally use the defaults here. And that will create a basic react app. Once you've got that to use the xxDK, you install a package called "xxdk -wasm". I'll give people a minute to digest that. If you go and look at the example, you'll see that this is the main page. And I'm gonna show you a video of this at the end. So what you see here is literally what I'm gonna be showing now. And you'll see that you have an XXNetwork directive. And you have a XXDirectMessages directive, we're using the XXDirectMessages sub module of XXNetwork for this demo example. We have a box for receiving messages and a widget for displaying those receive messages and returning them, then we have a box for sending those messages. And that displays like a little text box for you to type and then a send button for you to send.

Rick Carback 07:20
And the important things to note here are that these directives here, these tags, they both have context that you need to build out your application. So let's look at the first one XXNetwork. So XXNetwork - there's two contexts that uses... one of those is XXDKUtils. This is all built on top of something called wasm, which is WebAssembly. And this is your link into the functions that are exposed under WebAssembly. And the cMix one (the nodes that do the message processing), we call that protocol the cMix protocol and essentially treat that like a network connection. So I call it XXNet in many of the examples. So we're just gonna create a standard react system, where you have children that are passed in that are just plain vanilla react nodes. And then we're going to set up state variables that this

context provider is going to use, in order to set and update those state variables. They're gonna be linked up to the XXContext and XXNet variables.

Rick Carback 08:13
So when you get into your useEffect function, which is over here, on the right hand side, you'll see that we have to tell it where to find the wasm variables. What we do in the example is we locally host them. And you can do that just by linking the xxdk-wasm. We also have an s3 bucket where the stuff is hosted by default, so you can go and just use the default s3 bucket and that'll work fine for you, as well and it'll be downloading from our CDN. But I think the preference here is probably to host it locally, because it'll be a bit faster, because you want to download a 30 megabyte binary blob.

Rick Carback 08:42
Once you've got that path set, you can initialise the XX Network. This will set that context variable, the local state variable, which will get linked to that context variable. And then it will go and initialise the network. And at the end of that - let's walk through that. So we have in order to connect to XX Network, you need what's called a Network Definition File. We have that Network Definition File just inside of the app, it's called ndf.json So it just loads it. And the next variable is the statePath variable. So this is a prefix added to local storage. Generally speaking, we can leave this empty, I tend to put some small prefix. If you put too big of a prefix, you can get past the key state size. So generally don't make it more than four characters long. The secret is the password used to encrypt all the data in the state... Your identity and the secrets used to connect to the network, those are generated randomly - you can export and store those, that's part of the API, that's something that you're gonna have to look for, if you wanna learn how to use it. But the the actual data that's stored on the browser is in storage, which is sort of a separate password, which is what this is.

Rick Carback 09:43
So it's not a seed phrase or anything like that just an encryption password. For now we just use the dummy string hello. The params is an option. In general, I recommend that you use the default parameters, you can do that just by using an empty string. And all of the data that we use tends to be byte arrays, and to generate those byte arrays here. You're gonna be seeing a lot of Buffer.from. You're also gonna be see Buffer.from parsed in Base64. And creating Base64 and so on and so forth. Once you have that, you also need a variable to check to see if you've initialised cMix already. You initialise it with a new call, and then you mark that you've done it (if it returns and doesn't crash). Once you've done that, you can load cMix and then you can set the state variable. So the two state variables are at the top here, setXXDKUtils and setXXCMix. Once you've got those state variables, you can look them up on the left hand side here. That XXContext.Provider is set to the XXDKUtils. And that XXNet.Provider is set to XXCMix, you can see XXDKUtils and XXCMix.

Rick Carback 10:35
That is the XXNetwork provider. Let's look at direct messaging. Direct messaging uses the context variables from the provider. It also creates two more state variables that provides its context, and it creates a reference variable for a database lookup. And by default, when you're on the web3 version, it forces you to use IndexDB, because it's the only option. And that's what that dmDB is using. Dexie is just a common library. You don't need to use Dexie, but I recommend it. It's the same idea, you've got context variables at the top - the receiver and the client. The receiver receives the messages, the client provides some state and it lets you send messages. And you're going to do the same pattern as before, where we - first we verify that the context is passed down, it's not null. And you need this guard at the

top. And you also need - I'm going to skip ahead to the bottom of the useEffect function - you also need the xx and xxNet listed on the list down below. Because they are going to update... they're first going to be set to null.

Rick Carback 11:27
This useEffect function will run, it will hit that first guard at the top here and it will return. To get the useEffect function to run when those variables are set, you need to mark them as being listened to in this useEffects function for the direct message receiver. Once we've done that, we're going to create a random identity if one's not set, then we're going to export that we did it and then we're gonna store it. Notifications is not a an option inside of the web, so just a dummy variable there. But this database cipher is required. And this is a separate encryption password for entries in the database. So the actual text of the message is encrypted in that database. We also define a reception handler, a callback function - that's what onDmEvent is. And you'll see that this onDmEvent is literally going to take in an eventType and data which is a JSON byte array. And it's gonna parse that data, and then it's going to do this database lookup on the right hand side.

Rick Carback 12:17
This Promise.all is calling two functions. The first one is a lookup on the message ID, which is that e.uuid. And the second one is a lookup on the conversation, which will have information like the user's nickname, and things like that. If it doesn't find those things, it prints an error. If it does find those things, it'll print a little message and then add that to the dmReceiver listener. So that's what setDMReceiver is doing. So that's what the callback function does. The rest of the use state finishes initialising the client, so it's going to grab the the dmIndexedDbWorkerPath - is a web worker. Again, that's something I have to look at that's kind of outside the scope here. It will essentially initialise a separate thread inside of the web app that will handle database reads and writes for you. And then it will initialise the client once that web worker is initialised.

Rick Carback 12:59
Oh sorry, once that web worker path is known it will then initialise the client - that client will start the web worker. We're printing out the direct message ID and we're printing out the direct message token. So a token is just a number. It's a random number. And it acts as an additional password. So you can have apps that broadcasts the a public ID, a public key in many contexts. And then you can disable direct messages by just not listening on that token. And it's a very nifty little feature that enables you to have different groups of direct messages that you can control and protect your identity on the system in general - oh, protects your inbox not your identity.

Rick Carback 13:34
This database, we use a really basic name, which is the public key without the... when you do Base64 in general, it adds these equal signs at the end to make it a multiple of four. And uses the raw encoding, so it drops the equal signs and then adds a postfix of "_speakeasy_dm", and that sets up the database. And once you've set up that database, you are good to go. And this dmDB.current is actually setting the reference for the database. And then we start the networking. And then we check to see if the direct messaging client or.. sorry, we don't check... we set the direct messaging client to the result.

Rick Carback 14:08
Once that's set, the DMClient is set, then all the context becomes available to the people listening or the objects listening under the DOM. And you can see on the right hand side, you've got

XXDMClient.Provider and XXDMReceiverProvider. And they're just set to that dmClient and dmReceiver. So at this point, all the context is loaded. Let's look at how messages get received. It's super simple. We're looking at the context with dmReceiver, which is calling that callback and sending us messages. If we don't have any messages, we just print "Nothing yet..." If we do have messages, we print them. And if you've ever done React programming, this looks incredibly familiar to you.

Rick Carback 14:44
And let's look at message sending. So message sendng is slightly more complicated because it needs to look at the direct message object and actually send that message. So we've got handlers for when you're typing. That's what handleInputChange is doing. And we've got a handler for actually submitting the message. And what that does is just clear the message after calling the (XX)DMSend function, and that (XX)DMSend function is pretty simple, we're just going to send to ourselves. So we get the current token and the current public key. And we send that whatever was in the text field to ourselves. And you've got onChange and onClick. These are pretty standard events inside of React.

Rick Carback 15:19
So that's not the only example. There's an Android example. There's an iOS example. And then there's Golang is what the app is written in. So there's also a Golang example. And if you need it, I have a C# example too. But it's not in this app. It's not in this repository just yet. Let's take a look at some demos. This is the Android example running. And on the top here is just the console log for the XXNetwork Library, once you load it.

Rick Carback 15:38
And what you'll see in the bottom is this "Received Messages" window. And what this is doing is it's processing events, you can have up to three events. So if you're sending a message, the first event is the message being sent and accepted by the network. The second event is the message being processed by the network, which means you've got an alert that the so-called 'round' is done. And then once the 'round' is done, your client can go pick that message up. And that's the 'message was actually retrievable' event, so you typically get three events per message. Same thing is gonna happen in this demo for iOS. Slightly different, but same idea, you're gonna have log messages on top, and you're gonna have the events happening on the bottom. You get that second event and it's receiving that message.

Rick Carback 16:36
Let's take a look at a more complicated example, where we're going to send messages to each other. But before we do that, let's take a look at the Golang example. So, for the purposes of the competition, I think this is probably the most important slide for you. I think the most likely thing you're gonna do is you're gonna run a bot and you're gonna send something from your app to the bot, and you want the bot to respond. Easiest path is probably going to be to learn a really basic amount of Golang, and do it here inside of the Golang example inside of the "cmd/root.go" file. Around line 205, you will see the message reception loop. So you can look at message content, parse it - say, oh, blah, blah, blah, it did whatever. And then actually send a message in reply, that message send - how you do that is on line 172 right above it. And so you're gonna put something like on line 172 on the right hand side into line 200 or so here, 205 or so to create any kind of bot you want. And just make this for loop run forever. So get rid of line 186, 187 and you'll run forever.

Rick Carback 17:29
So if you're gonna create a bot that is the shortest path, you can do it in probably about five to ten lines of edits. So let's take a look at the React app running against the Golang. app. And what I'm gonna do is run the React app - oh sorry, run the Golang app, pull its public key and dmToken. And then set that public key and dmToken in the React app and send a message to the Golang app. Oh, and this.... doesn't run in Internet Explorer but will run in Safari, Chrome Brave and the browser I'm using here is Carbon, it's a web3 browser. You can see they link our echoexx tool on the bottom.

Rick Carback 18:02
So, on the top left here, you see that I highlighted the reception keys. And I'm going into the React.js example now and I'm installing the dependencies. I'm going to link the 'public' folder appropriately to the xxdk-wasm. And that's just gonna link that xxdk-wasm folder to xxdk-wasm in the 'public' folder. And 'public' is where you put files that get hosted on a local web server. So it's just enabling you to download the wasm blobs. And now we're going to edit the xxdk.tsx file, the (XXDM)Send function. I'm going to modify that token and public key, instead of using the local one that was generated, I'm going to use the one I copied and pasted from into above.

Rick Carback 18:36
And just so you can see, it prints it out pretty frequently. Alright, so "RECVDMPUBKEY", "RECVDMTOKEN", those are the same value, I'm just gonna set those values. And because it's JavaScript I've got to use Buffer.from, and then parse it using Base64. And this is Emacs. I probably should have used Visual Studio Code. I would say that's probably the best editor for this kind of stuff. But whatever you want to use is fine. I tend to use Emacs. But I'm old. And I'm stuck in my ways. But I recognise that Visual Studio Code is just as good.

Rick Carback 19:07
It started up a server on 3001. That's what the React app looks like. If you open up the console (that's Command, opt and 'I') it shows you that it's connecting to all 370 nodes. So it's not connecting to some kind of intermediate API or anything like that. There's no man in the middle. This is straight up running directly against the XX Network. And you can see that it sees the message. I did not send it emojis because it'd be problematic to display that in the terminal window I think.

Rick Carback 19:38
All right. And with that, I think we can take some questions. Neovim is what the old, cool kids are using! That is true. So Matt is another member of our team. He is there to answer any questions you might have on the on the chat. And now's a really good time to ask them, we're getting to the end. So in terms of Hackhazards, let's talk about how things are gonna happen. Matt and I and Sidhant Sharma will be on the Discord. And you'll notice that my Discord name is 'cryptomoose', but I renamed myself to 'rick@xx' on this moose symbol from an old cartoon. We are going to give away 2500 in XX prizes, XX Coins, that's gonna be spread out across the top three teams, I'm not sure what the division is. And we are also going to give out physical goodie bags to the top teams and the runners up. And on the right here you see this QR code. If you have a QR code scanner, I would scan it. If not, I will post it to discord so that you can see it later, just the just the image. And what you're gonna want to do is... <edit in video reading "only for participants">...

Rick Carback 20:38
So what's our selection criteria going to look like? Anybody who has a section in their app description that talks about privacy and XX Network is eligible. In particular, we're looking for folks where XX Network solves your critical privacy problem. And also, if you're proposing to solve it in the right way. As I said, during the training presentation, we're really geared towards messaging apps. So you're not sending video, you're not sending large images, you're establishing a key or doing something else, where the privacy and the metadata around that action is very important. And then I'd like to see some discussion of how much does it help improve the privacy of your app? And what that means is, without it, what are the risks of privacy? And with it, how much does that buy you? How does the situation change without versus with?

Rick Carback 21:20
After that, I want to see some kind of approach on how you would implement it in a final system - it doesn't have to be how you've actually built it, I understand you're doing a hackathon, you're gonna be working on this for a maximum few hours, and you're you're gonna get working on you can't and you're gonna move on. Totally understood. I would really appreciate what you struggled with and get your feedback. And if you ask a really good question on Discord, I'm going to note that and what project you were on. And that's going to go into our internal scoring criteria. Again, Matt, Sidhant and I are the judges here. And... the final least important thing is, does the implementation work? It is better if your implementation works, for sure. But if you had a really good approach, and you just couldn't get it working for one reason or another, it's not the end of the world. So definitely have a section on your submission, or in your readme of your submission, that talks about how you apply privacy to your to your project. And that is really what we're looking for.

Rick Carback 22:06
And now we're gonna go through and give you like a bunch of architectural diagrams of like, how to integrate into your app, I'm just gonna jump through these very quickly. And we're gonna do 1520 minutes of questions, and then that'll be the end of the presentation. So you could have some kind of bot running local to the app that communicates with cMixx and does something with many others. So one of the things that cMixx has, is it has a broadcast mechanism built in.

Rick Carback 22:30
So you can have one ID that is listened to by many receivers. So it's one potential architectural decision. You can have direct messaging and that happening, you can have direct messaging with multiple bots, which also have like particular functions that are outside of the system. This is where you have shared state behind each bot instance. So they're all updating maybe the same database or something like that. You could do some kind of telegram bot integration, any kind of integration with any kind of messaging system is a great idea. So just I'll just point that out. If you have messaging or the need for some kind of messaging in your app, this kind of integration is a good idea. This is just pointing out, you could have a single bot serving a telegram channel, you could do the same thing for Twitter.

Rick Carback 23:16
Well, this one's got botched. So because it has a broadcast mechanism, you can have one person producing content, and many clients receiving it. I'm sorry that for whatever reason, the slide did not copy and paste very good. This is just showing that there's two different ways to do it. You can also do a push to talk app using that mechanism, you can store roughly 720 bytes, 725 bytes or so in each message, when you send the messages over the XX Network. Unlike TCP/IP, it uses a mechanism by

which you don't have to wait and see if the other person saw it. It's sort of guaranteed delivery. And that's because the messages last on the network for about three weeks, you can just send it to the address. And assuming you've you know that it was the right address to send it to, you don't have to check that the recipient actually saw it, they will see it eventually.

Rick Carback 24:01
So it's a guaranteed message delivery with some caveats on the word guarantee. But you can just assume that any message will be received if the 'round' succeeded. So essentially, if you get that second update of the three (remember I talked about the three states) sent and accepted, sent and delivered and then sent and actually read on the output. So those are the three states. If you get to state 2, you can assume that you're successful. You can do it with fax machines. You could do it with whistleblowers. You could have some kind of calendar synchronisation app. You could do some kind of secret sharing. So secret sharing is the key word to lookup is Shamir - S.H.A.M.I.R. Shamir secret sharing, you could do some Shamir's secret sharing by having the user that's doing the secret sharing send messages to each of the people they're trying to secret share with. And this allows you to do that without having any association between the devices, which is really cool.

Rick Carback 24:51
You could do.. this is for video games... or not video games, but board games or some kind of implementation like that, where you share your position on the board with those individuals. You could do some kind of public key registration where you don't want to reveal who was registering, but that the public keys are legitimate. If you look at... you know, apps that need this are apps like Worldcoin and other KYC apps, where they're doing something that is extremely dangerous. In the case of Worldcoin, they're taking people's eye scans. And they're creating a database of... or an API that lets you check to see if the person behind this private key is a real person or not right. And because they have all that eye scanning data, it's extremely, extremely fraught with all kinds of privacy problems. But if you can store that eye scan data in a way that it can never be decrypted, but still checked, and you never have any association between any devices, you can effectively protect privacy in that mechanism. So similar idea, if you have a database where it's generated with very sensitive personal information, it definitely plays a role here.

Rick Carback 25:47
And you could use it to establish keys with a VPN for various devices. Potentially, you could use XX Network as a VPN, but it would be very low latency. So this one I'm less sure about being a great idea. But it's something you can do. You could have some kind of betting app where you all submit and then using the information that everyone posted, you essentially create a random... not random, but it is a cryptographically, secure, unpredictable number at the end. And then whoever is closest to the guess, or whoever, you know, is the result of some algorithm is the winner of that computation.

Rick Carback 26:26
You could use this to potentially write commitments to a blockchain. And it's really interesting because you can have multiple blockchain servers, all listening for these transactions, and then the first one to write it gets the miner extractor value. So it's a very interesting one for this group. You could play poker with it, you can have a bookie use it. You can do multiple bot instances where the bots get some reward if they service the client. You can vote with it. And this is, I believe, trying to show that you could establish WiFi registration with it. Again, not all of the ideas are the best idea but that's kinda what we're doing.

Rick Carback 27:00
So with that, I really thank everyone for your time. Again, I am going to be on Discord, rick@xx. Please reach out, we'll be we'll be very active on the discord and I'll be staying up very late to help support you guys, everybody over the weekend. And I'm looking forward to seeing what you come up with.