PROFESSOR: All right, guys. Let's get started. So today, we're going to talk about Tor. And we actually have one of the authors of the paper you guys read for today, Nick Mathewson. He's also one of the main developers of Tor. He's going to tell you more about it.

NICK MATHEWSON: Thank you. So at this point, I could start out by saying, please put your hands up if you didn't read the paper, but that wouldn't work. Because it's embarrassing not to have read a paper you're supposed to have read. So instead, what I will ask is, think of your birthday. Think of the date of your birth. If the last digit of the date of your birth is odd, or you didn't read the paper, please raise your hand.

OK, that's not far from half. So I'm guessing most people read the paper. Means of communicating that preserve our privacy enable us to communicate more honestly to gather better information about the world when we are less disinhibited from speaking because of possibly justified possibly unjustified social and other consequences.

So this brings us to Tor, which is a anonymity network that I've been working on for the last 10 years with some friends and colleagues and so on. [INAUDIBLE] there's a set of volunteer operating servers, about 6,000 of them. At first, it was just friends of ours that Roger Dingledine and I knew from MIT. After that, we built up more publicity. More people started running servers. Now it's run by nonprofits, private individuals, some university teams, possibly some of you here today, and no doubt some very sketchy people.

We've got about 6,000 nodes. We're serving on the order of hundreds of thousands to millions of users depending on how you count. It's kind of hard to count, because they're anonymous. So you have to use statistical techniques to estimate. And we're doing on the order of terabytes per second worth of traffic.

Lots of people need anonymity for their regular work. Not everyone who needs anonymity, though, thinks of it as anonymity. Some people say, I don't need anonymity. I'm perfectly fine identifying myself. But there's broad perceptions that the privacy is necessary or useful.

And when regular citizens use anonymity stuff, they tend to be doing it because they want privacy in search results, privacy in doing research on the internet. They want to be able to engage in local politics while not offending local politicians, and so on. Researchers frequently use anonymizing tools to avoid gathering biased data, biased by geolocation based services that might be serving them in particular different versions of things.

Companies use anonymity technologies for protection of sensitive data. For instance, if I can track all of the movements of the legal team for some major internet company, I can probably, just by tracking when they're visiting their web server from different places around the world, or where they're visiting the company [INAUDIBLE] different places around the world, learn a lot about which teams are collaborating with which. And this is information companies would like to keep private. Companies use also the anonymity technology for doing research.

So a major router manufacturer for a while-- I don't know if this is still the case-- would regularly serve different versions of its product sheets to IP addresses associated with its competitors in order to make reverse engineering trickier. And they found this out by using our software and said, hey, wait a minute, we got a different product sheet when we came in from Tor than we did coming directly. And it's also kind of normal for some companies to serve other companies versions of their websites to emphasize the employment opportunity sections.

Regular law enforcement needs anonymity technologies as well to avoid tipping off people during investigations. You do not want the local police station to appear in the web logs of somebody you're investigating. And regular folks need it, as I said, for avoiding harassment because of online activities, to research stuff that might be embarrassing. If you live in a country with uncertain health care laws, you might want to avoid creating too much public record of what diseases you think you might have and so on, or what dangerous hobbies you might have.

And also lots of criminal or bad folks use anonymity technology. It's not their only option. But if you are willing to purchase time on a bot net, you can buy some pretty good privacy that is not available to people who think that bot nets are amoral. And Tor, and anonymity stuff in general, are not the only multi-use technology out there. Let's see, the average age of a graduate is about 20. So around when you were born-- have you talked about crypto wars at all?

PROFESSOR: No.

NICK MATHEWSON: No. During the 1990s, it was sort of an up-in-the-air question in the United States about to what extent civilian use of non-backdoor cryptography should be legal, and to what extent it should be exported. That kind of came down pretty decisively on the side of cryptography should be legal and exportable during the '90s and early 2000s. And although there's some debate about anonymity technology, it's more or less the same debate. And I think it's going to end in more or less the same way.

So here's an outline of my talk. I'm going to give you that little introduction I gave you, talk a little bit about what we mean by anonymity in a technical sense, talk a little about our motivations for getting into it. Then I'm going to kind of walk you through step by step how you start with the idea of, we ought to have some anonymity, and how do you wind up with the design of Tor from that point. And I'll mention some branching off points where you might wind up with other designs.

I'll pause to answer some of the cool questions that everyone has sent in for their class assignment. I'll talk a little bit about how node discovery works, which is an important topic. And then I'll sort of by show of hands pick which of these advanced topics to cover.

I guess we're calling them advanced because they're later in the lecture. And I can't read them all, but they're all really cool. I'll mention some related systems whose designs you ought to check out if this is a topic that interests you and you'd like to know more about it.

I'll talk about future work that we want to have done at Tor and I hope that we'll have time to do some day. And if there's time for questions, then I'll take them. And I've got nowhere I need to be for the next hour or so. So I and my colleague David over there-- can you wave your hand, David-- will be hanging around somewhere and talking to anyone who wants to talk.

So right, anonymity-- what do we mean when we talk about anonymity? There are lots of informal notions that get used in informal discussions, in online, and so on. Some people use anonymous to mean, I didn't write my name on it. Some people use anonymous to mean, well, no one can actually prove it's me even if you suspect strongly.

What we mean is a number of notions expressed in terms of the ability of an observer or attacker on a network to link participants to actions. These notions come out of a terminology paper by [INAUDIBLE] that you find a link to on freehaven.net/anonbib/, the anonymity bibliography that I help maintain. It should list most of the good papers in the field. We need to bring it up to date to 2014, but it's pretty useful.

So when I say anonymity, generally what I mean is Alice is doing some activity. She's-- what should Alice be doing? Alice is buying new socks. And there's some attacker here. Let's call her Eve for now. Eve can tell that Alice is doing something.

Preventing that is not what we mean by anonymity. That's called unobservability. Eve can tell possibly that someone is buying socks. Again, that's not what we mean by anonymity. But what we hope is that Eve cannot tell that Alice in particular is buying socks.

And we mean that both on a categorical level-- Eve should not be able to conclude through rigorous mathematical proof, this is Alice, she's buying socks-- but also, Eve should not be able to conclude probabilistically it's likelier that Alice is buying socks than some randomly selected person. And also, we would like Eve not to be able to conclude after observing many Alice activities, Alice sometimes buys socks, even if I don't know some particular activity of Alice's is a socks purchase.

There are other ideas that are related. One is on unlinkability. Unlinkability is it's like a long-term profile of Alice. So for instance, Alice has been posting as-- I'm never good at picking names for my example. Alice has been posting as Bob and writing a political blog that would disrupt her career, that would offend her department head and disrupt her career as a computer security [INAUDIBLE].

So she's been writing as Bob. Unlinkability is Eve's inability to link Alice to a particular profile. Final notion-- unobservability, some systems try to make it impossible to even tell that Alice is online, that Alice is connecting to anybody at all, that Alice is doing any active.

These are rather hard to build. I'll talk a little bit more about to what extent that they are useful later. Something that is useful in that area is you might want to conceal that Alice is using an anonymity system, but not that she is on the internet. That's more achievable than concealing the fact that Alice is on the internet entirely.

So why did I start working on this in the first place? Well, partially because of the engineer's itch. It's a cool problem. It's an interesting problem. Nobody else was actually working on it. And my friend Roger got a contract to finish up a stalled research project before the grant expired. And he did it well enough that I said, hey, I'll join up. And [INAUDIBLE]. I'll join in. After a while, we formed a nonprofit and released everything as open source. So that's part of it.

But for deeper motivations, I think humanity has got a lot of problems that can only be solved through better and more dedicated communication, freer expression, and more freedom of thought. And I don't know how to solve these problems. All I think I can do is try to make sure that what I see as inhibiting discussion, thought, speech, becomes harder to do. So that's [INAUDIBLE]. Yeah.

STUDENT: So I know there are many good reasons to use Tor. Please don't see this as criticism. I'm just curious, what is your opinion as far as criminal activity?

NICK MATHEWSON: What is my opinion on criminal activity? Some laws are good. Some laws are bad. My lawyers would tell me never to advise anyone to break the law. My goal was not to enable criminal activity against most of the laws I agree with. In places where criticising the government is illegal, then I'm in favor of criminal activity of that kind. So in that case, I suppose I was supporting that kind of criminal activity.

My stance on whether it's a problem that an anonymity network gets used for criminal activity in general, to the extent that there are good laws, I would prefer that people not break them. I would, however, think that any computer security system that does not get used by criminals is probably a very bad computer security system if the criminals are making any kind of good decision making policy.

I think that if we go around banning security that works for criminals, we wind up with insecure systems. So that's more or less where I stand on it. I'm not really the philosopher of it, though. I'm more of the programmer. So I'm going to be giving really trite answers to philosophical and legal questions. Also, I'm not a lawyer and cannot offer legal advice. Do not take anything I say as legal advice.

That said, [INAUDIBLE], a lot of these research problems that I'm going to be talking about weren't even close to being solved. So whey do we start anyway instead of going straight into research? One of the reasons, we thought that a lot of them wouldn't get solved unless there was a test bed to work on. And that's kind of been borne out. Because Tor has kind of become the research platform of choice for lots of work on low latency anonymity systems. And it's helped the field a lot in that way.

But also, 10 years on, a lot of the big problems still aren't solved. So if we had waited 10 years for everything to get fixed, we would have been waiting in vain. So why do it then? Partially because we thought that having a system out there would improve long-term outcomes for the world. That is, it's really easy to argue that something that doesn't exist should be banned.

Arguments against civilian use of cryptography were much easier to make in public in 1990 than they are today. Because there was almost no civilian use of strong cryptography then. And you could argue that if anything stronger than DES is legal, then civilization will collapse. Criminals will never be caught, and organized crime will take over everything.

But you couldn't really argue that that was the inevitable consequence of cryptography in 2000. Because cryptography had already been out there, and it turned out not to end the world. Further, it was harder to argue for a cryptography ban in 2000 because there was a large constituency in favor of the use of cryptography.

That is, if someone in 1985 says, let's ban strong cryptography, well, banks are using strong cryptography. So they'll ask for an exemption. But other than that, there weren't a lot of users of strong cryptography in the civilian space.

But if someone in 2000 said, let's ban strong cryptography, that would be every internet company. Everyone running an HTTPS page would start waving their hands and shouting about it. And nowadays, strong cryptography bans are probably unfeasible, although people keep bringing back the idea. And again, I'm not the philosopher or political scientist of the movement.

So some folks ask me, what's your threat model? It's good to be thinking in terms of threat models. Unfortunately, our threat model is kind of weird. We started not with an adversary requirement. But we started with a usability requirement. The usability requirement we gave ourselves to begin is, this has to be useful for web browsing. This has to be useful for interactive protocols.

And it actually needs to see use. Subject to that, we want to maximize security. So our threat model has lots of weird corners in it if you actually write it out as, what can an attacker do, under what circumstances, and how? And that's because we've set ourselves the goal of, it has to work for the web.

And I'll return to that in a minute or two. But let's sort of talk about now how we can use forward anonymity, how we build forward anonymity. So here's Alice. She wants to buy socks.

So OK, let's say that Alice runs a computer. Let's call it R for relay. And this computer relays her traffic to-- I want to say socks.com, but I'm afraid that'll turn out to be something horrible, so zappos.com. Yeah, they sell socks, too.

All right, so Alice wants to buy some socks from zappos.com. And she's going through a relay. Well, I said Alice runs a relay. Any eavesdropper who's looking at this will say, that's Alice's computer. It's probably Alice.

All right, so let's have somebody else run a relay and have lots of other users all visit it. I'll call them A2 and A3, because there aren't enough standard cryptography person names-- buy books, tweet cat pictures. This is like 80% of what people do on the internet, right?

So now we have three people all going into this relay, three streams exiting. Someone who's watching the relay can't easily correlate-- should not be, we hope, but we return to that later-- that this Alice is buying socks, this Alice, buying books, this Alice is tweeting cat pix.

Well, except if they're watching this side of the connections, they can see Alice telling the relay, please connect me to zappos.com. All right, so we'll add some encryption. We'll maybe do TLS on all of these links. So to the extent that you can't break TLS, to the extent you can't correlate this to this, then they get some privacy.

Well, that's still not good enough, though. Because first off, we're assuming that this relay is fully trusted. I assume you know the definition of trusted and why it doesn't actually mean trusted. OK, good. This is trusted in the sense that it can break the whole system, trusted in the sense that you can't help but trust it, not trusted in the sense that it's actually trustworthy.

So all right, we can introduce multiple relays. We can have different relays run by different people. We can have-- this is not actually the topology we use. But my blackboard technique is terrible, and I don't want to redraw anything. We can imagine tumbling these connections through multiple relays, each of which removes a single layer of encryption.

So all this relay sees is Alice is doing something. All this relay sees is someone is buying socks. But this one just sees someone is buying socks. The connection came from this relay. This one just sees Alice is doing something, and it forwards onto this relay. And no single party ought to be able to correlate the whole thing.

Now we come to a major design point. Let's suppose that Eve is watching here and here. Nothing I've said so far does anything to obscure the timing and volume of Alice's packets. Oh sure, there'll be some trivial noise added from all the computation and decryption these things do from network latency and so on.

But ultimately, if Alice is sending a kilobyte in, then the design I've sketched out so far, a kilobyte is coming out. And if the socks web page is 64k long, and is served by this web server at 11:26, then Alice is going to get something about 64k long at 11:26 or 11:27 or so.

Now, with some statistics, Eve can correlate some of these streams if we don't obscure volume and timing information. There are designs that do obscure volume and timing information. The good ones usually come out of [INAUDIBLE], although there's some work on DC-nets.

You could have something where each of these nodes received a large number of requests, just [INAUDIBLE] up all the requests they got for an hour, reordered them, and transmitted them all at once. And you could also say all requests must be the same size. Requests are 1k, responses are 1 megabyte.

And with some more work on that, we get something that would let you send an email that would arrive in order of hours, or get a web page in order of to end time, assuming that you optimize it to a single round trip. These systems exist, and existed when we started doing Tor. They don't get a lot of use, though.

I actually wrote one called Mixminion that was a successor to the Mixmaster remailer. I have not gotten a remailer message in the last three years. Tor has billions of users. Remailers, it's unclear whether they've got more than on the order of hundreds.

So you might think, well, still though, it's better anonymity for the people who really need it. Except if you've only got on the order of hundreds of users, then you're not really providing them all that much anonymity against this kind of adversary anyway. Because this adversary can simply go, OK, there's 100 people. Well, the message I want to investigate was looking at a Bulgarian website. How many of them speak Bulgarian? OK, that's five.

The saying is, anonymity loves company. Unless you have a large user base, no system can actually provide anonymity. And that's why also in this design, if these Alices all belong to an organization, they ought to have a shared public system rather than a private one. If they all work for MIT legal, and they're investigating some fake MIT website that's offering fake diplomas, then if they're just using the MIT legal anonymizer, then it's not really concealing who they are. But if you have a large number of different parties all using this, then it actually can provide some privacy.

So we'll return one more time to resisting these correlation attacks. But for now let's say that we're not resisting these correlation attacks. And instead, we assume that an attacker who sees both ends wins, and we're trying to minimize the probability that that happens over time.

All right, so I've just talked about message passing. The way you would build that with something like a mix net is you give each of these relays a public key-- K3, K2, K1. And when Alice wants to send something through here, she would say, encrypt with K3, socks, and then encrypt that with K2-- I'm leaving off writing information for now-- and then encrypt with K1.

But public key, as you know, is kind of expensive enough that you don't want to use it for bulk traffic. So instead what you do is you negotiate a set of keys with each server. So Alice shares a symmetric key with this relay, a different symmetric key with this relay, and a different symmetric key with this relay associated in what we call a circuit, which is a path through the network. And after the initial public key is set up to create those keys, Alice can then use symmetric crypto to send stuff through the network.

If you stop at that point, then you have onion routing as it was designed in the 1990s by Syverson, Goldschlag, and Reed. And I hope I get the names right. Paul Syverson is still active. The other two are working on other things.

Also, once you've added circuits like that, medium term paths through the network, you can have an easy reply channel where things sent back this way get to Alice being encrypted at each step instead of decrypted at each step.

And of course you need some kind of integrity checking, either node by node or end to end. Because if you don't do integrity checking, then-- well, let's say you're using an XOR based stream cypher for your encryption. If you don't do integrity checking, then this node can XOR in Alice, Alice, Alice, Alice, Alice to the encrypted message. And then when it's finally decrypted over here, because that's a malleable crypto scheme, if the same attacker is controlling this node as well, or if the attacker is observing it here, the attacker will see Alice, Alice, Alice, Alice, Alice XORed with a reasonable plain text and be able to use that to identify, ah, this is the stream that came from Alice.

So let's do a little more about how the protocol works. Because it would be a shame to have everybody read the paper and then not talk about the stuff that the paper is focused on. Again, I apologize for my blackboard technique. Most of the time, I'm sitting at home on a desktop. This is alien tech. So here's a relay. Here's Alice. Here's another relay. Here's Bob. Now Alice wants to talk to Bob.

So first thing Alice has to do is build a circuit through these relays to Bob. Let's say she's picked these two, R1 and R2. So Alice first makes a TLS link to R1. R1, let's say, already has a TLS link to R2. First thing Alice does is she does a one-way authenticated one-way anonymous key negotiation.

The old one in Tor is called TAP. The new one is called NTor. They both have proofs. They both even have correct proofs, although the original proof in the paper had a flaw in it.

But when that's done, she sends a create cell. And she picks a circuit ID. Let's say she picks 3, and says, create 3. The relay says, created. And now R1 and Alice share a secret key, a symmetric key, which they're going to call S1.

And they both have this stored as 3 with respect to this link. Now Alice can use that key to send messages to R1. So she says, on 3-- that's the circuit ID that everything was talking about in the paper-- send a relay extend with some contents.

The extend cell basically contains the first half of the create handshake. But this time, it's not encrypted with R1's public key. It's encrypted with R2's public key. And it also says, and this one goes to R2.

So R1 knows to open a new circuit to R2, and says, create. And it passes the initial part of the handshake as it came from Alice along. And it picks its own circuit ID. Because circuit IDs identify the different circuits on this TLS connection.

And Alice doesn't know what other circuit IDs are in use on this one. Because this one is private to R1 and R2. So it might pick 95. It actually is very unlikely to pick that, because they're randomly chosen from a 4 byte space. But I don't want to write out any 32-bit numbers today.

And this says, created in response. So this one sends back an extended encrypted with S1. And now Alice and relay share S2. So now Alice can send messages encrypted first with S2, and then with S1 as relay cells. So she sends a message like that. R1 removes the S1 encryption and forwards it on.

It says, OK, it came in on circuit 3. I know that 3 goes to 95 on this one. So I send it on 95. And I say whatever I got after decrypting. OK, and this one says, ah, I came on 95. 95 corresponds to the shared key S2. So I'll decrypt with that.

Oh, that says, open a connection to Bob. And relay 2 opens a TCP connection to Bob and tells Alice that it did it through the same process. And Alice says, great. Tell Bob http 10 get/index.html, and the world goes on.

Let's see, what did I leave out? I'll skip that, skip that, skip that. So what do we actually relay? Some designs in this area say, well, you should send IP packets back and forth. This should just be a way to transmit IP packets. One of the problems with that is we want to support as many users as possible, which means we have to run on all kinds of operating systems.

And operating system TCP stacks do not act anything like each other. If you've ever used Nmap, or if you've ever used any kind of network traffic analysis tool, you can trivially tell Windows TCP from FreeBSD from Linux TCP. And you can even tell different versions apart.

And moreover, if you can send raw IP packets to a chosen host, you can provoke different responses in part based on what the host is doing. So if you're doing IP, you would actually need an IP normalization layer if IP is what you transport back and forth. And it seems that anything less than a full IP stack is not actually going to work for IP normalization. So you wouldn't want to do that.

Instead, what we just chose is-- and this is largely because this is the easiest way-- you take the contents of TCP streams. So you just assume each of these things is reliable and in order. You have the computer analysis end, the program analysis running to do all this stuff for her, accept TCP connections from Alice's applications, and then just relay their contents and don't do anything trickier on the network level.

You might be able to get better performance by trying some other means. And there are some papers examining how you would do that. But this is the one that we could actually implement. Because we paid a lot more attention in security and compilers classes than we did in networking classes. Now we have networking people. But in 2003, 2004, we did not have any networking experts.

TCP also seems like the right level. Higher level protocols-- like in some of the original

[INAUDIBLE] designs, there were separate proxies at this end for HTTP, for FTP, and so on. That seems to be mostly a bad idea. Because any interesting protocol is going to have end to end encryption from Alice all the way to Bob. That is if we're lucky, Alice is doing a TLS connection over this to Bob so that TLS properties get her integrity and secrecy.

But if that's the case, then any kind anonymizing transformations you want to apply to the encrypted data need to happen in the application Alice is using before the TLS happens entirely. So you can't really do that in a proxy.

And that's kind of why we came out to, OK, the sweet spot is TCP contents. Somebody asked me, OK, but where are your security proofs? We do have security proofs for a lot of the cryptography that we use, standard reductions. For the protocol as a whole, there are proofs in the field about certain aspects of onion routing.

But the models that they have to use in order to prove that this provides anonymity make assumptions about the universe, the network, or the attacker's abilities that are so weird as to satisfy no one but certain program committees of more theoretical conferences.

The kind of things you can prove is that an attacker who sees this, who sees a number of strings here all of equal volume and equal timing, cannot tell which one goes to which Bob simply by looking at the bytes coming out. But that's hardly a useful result.

Also, the kind of guarantee you can get from anonymity systems that we know how to build today-- OK, I should be careful here. There are some where you have very strong guarantees that we do know how to build that you would never actually want to use. Like classical [INAUDIBLE] DC-nets, for instance, provide guaranteed anonymity. Except any participant can shut down the whole network by not participating. That does not scale.

But for the things that we do want to build these days, for the most part, the anonymity properties are probabilistic rather than categorically guarantee-able. So instead of asking, does this protect Alice, the kind of questions you could ask are, under this assumption about hacker capabilities, how much traffic can Alice safely send if she wants a 99% chance of not being linked to her activities?

So will anyone actually run these things? That was an opening question when we started. We didn't know whether the system would actually take off or not. So the only [INAUDIBLE] try to see what happens.

We got a fair amount of volunteer operators. A fair number of non-profits have formed whose sole purpose is just to take donations and use it to buy bandwidth and run Tor nodes. And there are also universities. There's also private companies. For a while, [INAUDIBLE] was running a Tor server out of their security team because they thought it was fun.

The legal issues there-- again, I'm not a lawyer. I can't offer legal advice. But five different people asked about legal issues. As far as I can tell, in the US at least, there's no legal impediment to running a Tor server. And that seems to be the case throughout most of Europe as far as I'm aware. In places that generally have less internet freedom, it's a dicier proposition.

The issues to be concerned about are not, is it illegal to run a Tor server, but if somebody does something illegal or undesirable with my Tor server, will my ISP shut me down, and will law enforcement believe, oh, you're just running a Tor server, or will they seize the computer to make sure?

For those, I would suggest not running the Tor server out of your dorm room. Excuse me, don't run an exit out of your dorm room, or really out of your dorm room, assuming the network policy allows that. I have no idea. They've changed so much since I was a kid. Running an exit out of your dorm room could get you in trouble. But running a non-exit relay that doesn't deliver traffic to the internet is less likely to create those issues in particular.

But if you do it in a nice co-lo site, and you get your ISP's permission, then it's a pretty reasonable thing to do. Let's see, someone asked, well, what if users don't trust a particular node? And this brings me to my next topic. So the software the clients use, you can't tell it, don't use this one, don't use this one, only use this one.

But remember that anonymity loves company principle. If I'm only using three nodes, and you're using three different nodes, and you're using three different nodes, our traffic will not mix at all. To the extent that we partition off which parts of the network we use, we are distinguishable from one another.

Now, if I just exclude one or two nodes, and you just exclude one or two nodes, that's not a big partitioning, and that doesn't help distinguish-ability that much. But it would be good to the extent possible to have everyone using the same nodes. So all right, how do we accomplish that?

So version one, in the first version of Tor, we just chipped a list of all of the nodes. I think there were three of them, or five, or something. No, I think there were about six, of which three were all running on the same computer in a closet at LCS in Tech Square. All right, so that wasn't a good idea. Because nodes can go up and down. Nodes change. You don't want to have to put out a new release of your software every time somebody joins to release the network.

So you could just have every node keep a list of all the other nodes that are connected to it and all advertise to each other. And then when a client connects, a client just has to know one node and then says, hey, who's on the network? And actually, a lot of designs people have built work this way. A lot of early peer to peer anonymity designs work this way.

But it's a terrible idea. Because if you go to one node and say, who's on the network, and you believe them, well, if I'm that node, I can say, yes, I'm on the network, and my friend over here is on the network, and my friend over here is on the network, and no one else is on the network. And I can tell you any number of fake nodes that are all operated by me and capture all of your traffic that way with what's called a row capture attack.

OK, so maybe we just have a single directory operated by a trusted party. That's not so good as a single point of failure. So OK, let's have multiple trusted parties. And clients go to these multiple trusted parties and get a list of all of the nodes from all of them and combine those lists.

Then you're actually-- first off, you're partitioned in that case. If I choose these three, and you choose those three, and they say anything different, then we'll be using different sets of nodes. So that's still not good.

Also, there's still a [INAUDIBLE] where if I use the intersection of the sets they tell me, then any one of them can keep me from using a node they don't like by not listing it. If I use the union, anyone can flood me by making 20,000 fake servers that are all on the list.

I might compute the result of some sort of vote on them, which would solve those two problems. But I'd still be partitioned from everyone who's using different trusted parties. We could do a magical DHT. Have we done [INAUDIBLE] hash tables? All right, we could do some sort of magical distributed structure run across all of the nodes.

I say magical, because although there are designs in this area, and some better than others, none of them really seem to have a solid security evidence for it at this point to the point where

I would be comfortable in saying, yes, this is actually secure. So the solution we wound up with is have multiple hardened trusted authorities run by trusted parties that collect lists of nodes that vote hourly on which nodes are running that can vote to exclude nodes that seem to be misbehaving that are all running on the same slash 16 that are doing strange things to traffic, and have them form a consensus that's a result of their votes. And everybody signs the consensus. And clients don't use it unless it's signed by enough authorities.

This is not the final design. But it's the best we've managed to come up with so far. And this way, all you need to distribute with clients is a list of all of the authorities' public keys and some places to get the directories. You want to have all the nodes cache these directory things. Because if you don't, the bandwidth load on authorities is catastrophic.

So I'm going to skip over that. Because I would love to talk about how clients should choose which paths to build through the network. I would love to talk about issues applications and making applications not deanonymize themselves.

I'd love to talk about abuse. I'd love to talk about hidden services and how they work. I'd love to talk about censorship resistance. And I'd like to talk about attacks and defenses. But I've only got 35 minutes. And I can't possibly cover all of these.

So show of hands for how many people think the most important-- think about what you think are the two most important topics on this list. If one of your two most important topics is path selection and how you choose nodes, please raise your hand. If one of your two most important topics is application issues and how to make applications not bust your anonymity, please raise your hand.

If one of your most important issues is abuse and what kind of abuse we see, how you can prevent it, and how that works out, please raise your hand. OK, that one's popular. If one of your most important topics is how these services work and how they can be made to work better, please raise your hand. Wow, that's much more popular on this side of the room than that side of the room. What's going on? You guys in a club? Are you up to something?

Censorship, who's interested in censorship? OK, that's fairly popular. Attacks and defenses? OK, so we're not doing paths and we're not doing apps. So apps-- guard nodes, guard nodes, C guard node designs, select by bandwidth. You need to actually weight by bandwidth, but you also need a trusted way to measure bandwidth. And that's the too long, didn't lecture of what would be on path selection.

For application issues, almost no protocol is actually designed to provide anonymity. Because almost every protocol that's widely used has the assumption in it, well, you know, anyone who wants to can just see the IPs on this traffic. So there's no point in trying to conceal identity.

So in a particularly complex protocol, like the whole stack of protocols a web browser uses, there's no real way to anonymize that just by anonymizing the traffic with something like Tor. You need to hack the web browser pretty hard to make it stop doing things like leaking the list of fonts that are identified on your system, leaking your exact window size, allowing all kinds of permanent cookie-like structures, leaking what's in the cache and what's not in the cache, and so on.

So your choices there are basically isolate everything and restart from a fresh VM all the time, or reroute the browser, or both. Other things are a lot easier than web browsers, but still problematic. That's all I'm going to say about app issues. Let's see, I think I got the most hands-- did you see what I got the most hands for, any opinions?

STUDENT: Abuse and hidden services?

NICK MATHEWSON: Abuse and hidden services. All right, I'll talk about abuse and hidden services. And if I've still got time, I'll do censorship and attacks. So let's go to abuse-- abuse, abuse, abuse.

So one problem that we've fortunately not had all that much of-- so when we were working on this stuff, the problem that everybody was afraid of was this horrible stuff that would get you kicked off of any ISP, and it would create tremendous legal issues and ruin your lives. I speak of course of file sharing.

We were terrified that people would try to BitTorrent or Gnutella or whatever over this thing. Yes, it was a long time ago. And we thought about how we'd do that. Well, you'll see in the paper that we talk a lot about exit policies, about letting exit nodes say, I only allow connections to port 80 and port 443.

This doesn't actually help with abuse at all. Because you can try to spread worms over port 80. You can post abusive stuff to IRC channels over web to IRC interfaces. Everything's got a web interface these days. So you can't really say, it's only web. It's safe. If it's useful, it can be abused.

That said, there are people who are willing to run exits that deliver 80 and 443 who would not be willing to run exits delivering all ports. So it did turn out to be useful. It just didn't turn out to be a solution. Another thing that creates problems is criminal activity generally doesn't create problems for the network operators so much.

From time to time, somebody's server gets seized and returned six months later, and they have to wipe the thing. That's still an infrequent enough occurrence that it's somewhat surprising when it happens. And so yeah, don't run an exit node on a server that you need to graduate.

What else? The biggest problem that we have for abuse of stuff is that many websites around the world, and many IRC services and so one, use IP-based blocking in order to deter and mitigate abusive behavior-- people posting road kill pictures on My Little Pony sites, people flaming everybody on IRC channels, people making love, leave, join requests, people replacing entire Wikipedia pages with racial slurs.

This stuff it's real. It's problematic. It's unacceptable to the websites and services that use IP-based blocking. They need a way to keep this from happening. And IP-based blocking is a cheap way for them to do that. So it's pretty frequent that Tor users get banned completely from some sites.

There's some work on trying to say, well, why does IP-based blocking really work? Is it because IPs are people? No. Everybody in this room knows how to get a different IP if they need one. Everybody in this room knows how to get like tens of thousands of different IPs if they need one, if they need tens of thousands.

But for most people, getting more IPs is at least a little time consuming and at least a little challenging to the extent that it imposes a rate limit and a resource cost on abuse if you don't want a bot net and if they've already blocked Tor and all the other proxy services.

So for that, you need to look at different ways to provide other resource costs. You can either say, well-- have you done blind signatures? Oh, you can construct things so that you need an IP to make an account. But what account you make with an IP is not linkable to your IP. And then later on if the account gets banned, you need to create a new account from a different IP.

That's something you can build, and we're working with people to work on it, although it needs more hacking on the integration side. Something else that needs more hacking on the integration side is anonymous black listable credentials. They're a little esoteric. But the idea is that you get something that allows you to participate on an IRC server, for example. You can use this as many times as you want.

Your using it is not linkable until you are banned. Once you are banned, future attempts from the same person with the same credential don't work. But past activities do not become linkable to one another. These can be built pretty easily. The problem is convincing people who are more or less satisfied with IP blocking to actually use them and actually integrating them with services.

Someone inevitably asks me-- it's kind of neat. So I started these lecture notes based on my lecture from 2013. And there was something about the inevitable question about Silk Road 1 getting busted. There's the inevitable question about Silk Road 2 getting busted. Silk Road 2 was a hidden service operating on the Tor network where people would get together to buy and sell illegal things, mostly illegal drugs.

So as far as we know, as far as we can find out, the guy got busted through bad OPSEC. Like he made a public posting with his actual name, and then went and deleted it and put his pseudonym on it. Tor can't help people against that kind of stuff.

On the other hand, if you've been looking at the NSA leaks, you know that law enforcement has been getting information from intelligence and then sanitizing it through a process called dual construction where the intelligence agency will say to the law enforcement agency, OK, look, it's Fred over there. He did it. But that's not admissible in a court, and you can never admit that we told you. Just find some other way to find out that Fred did it, but Fred did it.

According to some of the Snowden leaks and some of the leaks from the other guy, who has still not been caught, that's done sometimes. So OK, at this point, you use your basic Bayesian reasoning skills, and you say, well OK, would I see this evidence if the guy actually got caught by because of OPSEC? Yes, I would. I would see bad OPSEC. I would see reports that he got caught because of bad OPSEC.

But what would I see if it were a dual construction case? I would also see reports that the guy got caught by bad OPSEC. Because the evidence that would be available to us is the same in either case. We can't really conclude much from any public reports of that.

That said, it does look like the guy got busted by bad OPSEC. It does look like the kind of bad OPSEC that you would be looking for if you were trying to catch somebody running something like this. Nevertheless, earlier I suggested that please do not use myself to break any laws. Also if you're life or freedom is at stake from using Tor or any security product, do not use that product in isolation.

Think of ways to use it to construct a series of redundant defenses for yourself if your life or freedom at stake, or if having the system broken is completely unacceptable to you. And I'll say that about Tor. And I'll say that about TLS. And I'll say that about PGP. Software is a work in progress.

So that's the abuse section. I've got 25 minutes-- hidden services. Where's hidden services? So responder anonymity is a much harder problem than initiator anonymity. Initiator anonymity is what you get when Alice wants to buy socks, and Alice wants to stay anonymous from the sock vendor.

Responder anonymity is when Alice wants to publish her poetry online and run a web server that has her poetry on it, but not let anyone know where that web server is because the poetry is so embarrassing. And yes there actually is a hidden service out there of mine with bad poetry on it. No, I don't think anybody's actually published it yet. No, I'm not going to tell anybody where it is. I'm waiting for it to go public.

So all right, one thing you could do is-- let's see, how much time? OK, I can do this. So now Alice wants to publish her poetry. So I'm going to put Alice on this end, because she's the responder. Alice could build a path-- this represents a lot of relays-- through the Tor network, and then just say to this relay, please accept connections. So now anyone who goes to this relay could say, hey, I want to talk to Alice. And there have been designs that work this way.

It has some challenges, though. One challenge is this relay could man in the middle all the traffic unless there is a well known TLS key. Another thing is maybe this relay is also embarrassed by the poetry and doesn't want to be a public contact point for poetry so terrible. So this relay could also be pressured by other people who hate the poetry to censor it.

This relay could also make itself an attack target. So you want some way where Alice can go to different relays over time and no single relay is touching unencrypted traffic of Alice's. All right, that's doable.

But once you have a lot of different relays, what does Alice actually tell people? It's kind of got to be a public key. Because if she just says, relay x, relay y, relay z, but x, y, and z are changing every five minutes, that's kind of challenging to know you actually got the right relay.

So let's say she tells everybody a public key, and once she gets over here, she says, hey, this is Alice. I'll prove it with my public key. So this relay knows that public key z is running a hidden service here. And so if anyone else says, hey, connect me to public key z, they can do a handshake and wind up with a shared key with Alice. And it's the same handshake as the Tor circuit extension uses.

And now Bob can read Alice's poetry by going another path through the Tor network over here. Bob has to know PKz, and Bob can say, hey, connect me with PKz. Send this thing that's sort of like a create cell-- really it's an introduce cell, but let's forget that-- over the Alice. They do the same handshake that relays do. And now they have a shared key that they can use for end to end encryption.

Well, there's something I left out, though, which is, how does Bob know how to go here? And can we do anything about the fact that this relay has to learn to this public key? Well, we can. We can add some [INAUDIBLE] directory system where Alice uploads a signed statement anonymously over Tor saying PKz is at a relay x.

And then Bob says, hey, give me a signed statement to ask the directory system, hey, give me a signed statement about PKz. And Bob finds out where to go. And we could even do one better and have Alice give a different public key here. So this could be PKw. And the statement she uploads to the directory can say, if you want to talk to the service with public key z, then go to relay x and use public key w.

And now public key z isn't published here. You could even go one farther and encrypt this with some shared secret known to Alice and Bob. And if you do that, then the directory service and people who can contact the directory service can't learn how to connect to Alice with that. Yeah.

STUDENT: Just a quick question there. If that's not encrypted, then Rx can still find out that it's running a service for Alice, right?

NICK MATHEWSON: Yep. Well, not for Alice. It can find out that it's running PKz if this is not encrypted. We have a design for that that I'm actually going to get to at the end of this. But it's not built yet. But it's pretty cool.

So OK, and you don't want to use a centralized directory for this. So we actually do use a DHT, which is, again, not perfect, and has some censorship opportunities. But we are trying to make those less and less. And I might cover more stuff, so I can't do the whole details.

So one of the problems there though is if you are running one of these directory services, you've got a complete list of these keys pretty-- over time, you run a directory service [INAUDIBLE]. You get a complete list of all these keys, and you can try connecting to all the ones that don't have encrypted stuff to find out what's there. That's called an enumeration attack. And we didn't list that in our paper, because we weren't thinking of that. We didn't. But it is something we'd like to resist.

So in the design I hope to be hacking together sometime in 2014, we're going to move towards a key blinding approach where Alice and Bob share PKz, but this statement is not signed with PKz. This statement is signed with PKz prime where PKz prime is derived from PKz and, say, the date such that if you know PKz and the date, you can derive PKz prime. If like Alice you know secret Kz, you can generate messages that are signed by PKz prime. But if you only see PKz prime, even knowing the date, you cannot re-derive PKz.

We've got a proof. And if you'd like to find out how this works, then ping me and I'll send you the paper. It's a cool trick. We weren't the first ones to invent this idea. But that is how we're going to solve enumeration attacks sometime this coming year assuming that I can actually get the time to build it. So that's hidden services.

Attacks and defenses-- so so far, the biggest category of attacks we've seen is attacks at the application level. So if you're running an application over Tor, and it's sending unencrypted traffic, like regular HTTP, then a hostile exit node, just like anyone else who touches HTTP traffic, can observe and modify the traffic.

This is the number one attack on the whole system. The solution is encrypted traffic. Fortunately, we're kind of in an encryption renaissance over the last few years. And more and more traffic is getting encrypted with the nifty free certificate authority that EFF and Mozilla and Cisco and I forget who else announced a day or two ago. There will be even less excuse for unencrypted traffic in 2015 than there was this year. So that solves that.

More interesting attacks include things like traffic tagging. So we made a mistake in our early integrity checking implementation. Our early integrity checking implementation did end to end checking between Alice's program and the exit node.

But it turns out that that's not enough. Because if the first relay messes with the traffic in a way that creates a pattern that the exit node can detect, then that's an easy way for the first relay and the last relay to learn that they are on the same path and identify Alice.

Of course, if the first relay and the last relay happen to be on the same path, happen to be collaborating anyway, then they can already identify Alice through traffic correlation, we believe. But perhaps it should not be so easy for them as that. Perhaps traffic correlation will someday be harder than we think.

It would be good to actually solve that attack for real. We've got two solutions for that. One is the expected result of this attack is that periodically circuits will fail. Because the attacker on the first hop guessed wrong about controlling the last hop. So every Tor client checks for weird failure rates.

The real long-term fix is to make it so that messing with the pattern on the first hop doesn't create more than 1 bit of information on the last hop. You can't avoid sending 1 bit of information, because the first hop can always just shut down the connection.

But you can limit it to 1 bit-- OK, 2 bits. Because then they'll have the choice to corrupt the data or shut down the connection. Oh, I had an idea of how to make that better. I'll have to think about that.

Let's see, DOS is actually pretty important. There was a paper the other year about something that the authors called the sniper attack where you see traffic coming from a Tor node that you don't control. You want to kick everybody off that Tor node.

So you connect to it. You fill up all its memory buffers, and it crashes. Then you see whether the traffic in question gets rerouted to a node you control or not, and you repeat as necessary. For that, our best options are first off, no longer have memory DOSes. I think we have all of the good memory DOSes fixed now. There are some bad ones that still needed to get addressed. But they're screamingly inefficient.

The other option for resolving this kind of thing is make sure relays are high capacity. Don't accept low capacity relays on the network. We do that, too. If you're trying to run a relay on your phone, the authorities won't list it.

And another thing is to try to pick our circuit scheduling algorithms so that it's hard to starve out circuits that you don't control. That's very hard, though, and it's as yet an unsolved problem. Let's see, should I do an interesting attack or an important attack?

STUDENT: Interesting.

NICK MATHEWSON: Interesting, OK. So show of hands, how many people might like to write a program that uses cryptography some day? Cool, here's what you must learn. Never trust your cryptography implementation. So even when it's correct, it's wrong.

So long ago-- I think this may be one of the worse security bugs that we've had. Any relay could man in the middle any circuit because we assumed that a correct Diffie-Hellman implementation would verify that it was not being passed 0 as one of the inputs.

The authors of our Diffie-Hellman implementation assumed the proper application would never pass zero to a Diffie-Hellman implementation. So Diffie-Hellman, when I say g to the x, you say g to the y. I know x. You know y. And we can both compute g to the xy now.

You tend to feel me? Good. Well, if instead the man in the middle replaces my g to the x with 0 and your g to the x with 0, and then I happily compute 0 to the x, and you compute 0 to the y, we will have the same key. We will happily talk to each other.

But this will be a key that the attacker knows, because it's 0. 1 also works. p also works. p plus 1 also works. So you basically just need to make sure that your values here are within range 2 and p minus 1 if you're doing Diffie-Hellman in z sub p.

OK, let's see, I would love to talk more about censorship. Because actually, it's one of the areas where we can do the most good. Generally, the summarized version of that was, in the earliest paper you read, and in some of the updates, we were still on the idea that we would try to make Tor look just like a web client talking to a web server over HTTPS and make that hard to block. It turns out that's fantastically difficult and probably not worth doing.

Instead, the approach we take now is using different plug-in programs that a non-listed relay called a bridge can use, and a client can use to do different traffic transformations. And we manage to keep adding new ones of those faster than the censors have been able to implement blocking for them.

And that's actually a case where none of the solutions are categorically workable. That's not a well-formed sentence. None of these plug-ins are inherently unblockable by any imaginable technique so far. But they're good enough to keep traffic unblocked for a year or two in most places, and six or seven months at a time in China.

China currently has the most competent censors in the world, largely because China doesn't outsource. Most other censoring countries outsource their censorship to dishonest European, American, and Asian companies whose incentives are not actually to sell them good censorship, but to keep them on an upgrade treadmill.

So if you were buying your censorship software from the United States-- which technically speaking US companies aren't allowed to make censorship software for nations. But they just make corporate firewall software that happens to scale to 10 million people. Yeah, I think that's unethical.

But again, I'm not the political scientist of the organization, or the philosopher. Paul Syverson, one of the original [INAUDIBLE] authors, does have a degree in philosophy, for what that's worth, which means that he can't answer these questions either. But he takes a lot longer not to answer them.

Right, where was I? 90 minutes is a long time. Censorship-- right, so what the censorware providers do is once Tor gets around their censorship, they will block the most recent version of Tor. But they do it in a way that is the weakest possible block. So if we change 1 bit in one identifier somewhere, we get around it.

We can't prove that they're doing this on purpose to ensure that Tor will evade their version so that they can sell Tor blocking and then have it not work so they can sell the upgrade, and then sell the next upgrade, and sell the next upgrade. But it sure does seem that way. So that's another reason not to work for censorship providers. They're tremendously unethical, and they don't provide very good software.

If you're interested in writing any of these plug-able transport things, that is an excellent kind of thing to do as a student project-- loads of fun, learn a little bit about crypto, learn a little bit about networking. And so long as you do it in a memory-safe language, you can't screw it up that badly. The worst thing that happens is it gets censored after a month instead of after a year. And that's what I want to-- oh, the addenda related to work.

Tor is the most popular system of its kind, but it's not the only one. Lots of others have really good ideas, and you should check them out too if you're interested in learning all of the stuff

I'm not thinking about and all the reasons I'm wrong. freehaven.net/anonbib/ lists the academic research and publications in this area.

But not all the research in this area is academic. You should also look at I2P; Gnunet; Freedom, which is currently defunct, no pun intended; Mixmaster; Mixminion; Sphynx with a Y, Sphinx with an I is something different; DC-nets, particularly the work of Brian Ford, and also of the team at Technical University Dresden, in trying to make DC-nets practical. They're very strong [INAUDIBLE], not actually deployable yet-- and many others. Why these get less use or attention than Tor is an open topic of some interest that I don't have a solid answer for.

Future work-- so one of the reasons I do these is not just because I would like everybody to know about the cool software I work on. But also because I know students have lots and lots of free time. And I'm kind of looking to recruit.

OK, you may think I'm joking. But when I was just getting started in this field, I was complaining about how I was so busy reviewing papers for one conference, writing software, fixing a bug, answering email. I was complaining to some senior faculty member. And he told me, you will never have so much free time as you do today. You actually have a lot more free time now than you will in 10 years. So this is a great time to work on crazy software projects.

So let me tell you about future work in Tor. There's this key blinding thing and a complete revamp of our hidden services system, which was the best we could design when we came up with it. But there's been a lot of research since then. Maybe some of it will turn out to be a good idea.

We're also revamping most of our crypto. We chose schemes that seemed like a good security performance trade-off in 2003, like RSA-1024. We've replaced the really important uses of RSA-1024 with stronger stuff, currently [INAUDIBLE] 25519. But there's still some cases that we want to replace in the protocol that we need some work on.

I didn't talk too much about path selection, so I can't talk too much about improvements in that selection. But our path selection algorithms were [INAUDIBLE]. And there's been some awesome research in the past five or six years on that that we need to integrate.

There's a little work that's been done on mixing high latency and low latency traffic so that the low latency traffic can provide cover for the high latency traffic in terms of providing lots of users while the high latency traffic is still very well anonymized. It's not clear whether this would work or not. It's not clear whether anyone would use this or not. And it is clear that unless something changes, or unless some major funding for that particularly shows up, I'm not going to have time to work on it in 2015. But if somebody else wants to hack on that, my god, that would be fun.

Our congestion control algorithms were chosen questionably based on what we could hack together in a week. We've improved them, but they could use a bigger revamp. There's some research on scaling to hundreds of thousands of nodes. So in the current design, we can probably get up to 10,000 or 20,000 with no problem. But because we assume that every client knows about every node, and every node may be connected to every other node, that's going to stop scaling before 100,000. And we need to do something about that.

That opens up some classes of attacks based on attackers learning which clients know which nodes and using that to distinguish clients. So most of the naive approaches are a bad idea here. But it may be that less naive approaches might work out.

Another thing you might want to do if you're increasing 100,000 nodes is get rid of those centralized directory authorities and go to some kind of peer to peer design. I don't have extremely high confidence in the peer to peer designs I know of so far. But it could be that somebody's about to advance the next good one.

Let's see, I don't know what that means. Oh, somebody asked a question about adding padding traffic or fake traffic to try to deceive end to end traffic correlation. This is an exciting research field that needs someone smarter to work on it or someone with a more practical attitude to work on it than has previously worked on it.

Too many of the results in the research literature there are only about distinguishing the traffic of two users on a number containing one relay, because that's how the math was easy to do. So because of this kind of stuff, all of the traffic analysis defenses that we know of in this area that are still compatible with broad browsing, they sound good if you read the abstract.

You'll say, hooray, this one forces the attacker to gather three times as much traffic before they can correlate users. Except when you actually read the paper, previously the attacker needed two seconds worth of traffic, and then they won. Now they need six seconds.

That's not really a defence in this model, although perhaps against a real network, the numbers would be different and it might work. So we would actually like to see some stuff done with padding and fake traffic. But we don't like to add voodoo defenses that we conjecture to maybe do some good, although we can't do that. We actually like to have evidence that any changes we're going to make are going to help something.

I think I'm out of time. And there may be a class in here after us? There is not? All right, so I'm going to hang around for a while. And thanks for coming to listen. I would take questions now. But it's 12:25, and folks may have another class. But I'll be around [INAUDIBLE]. Thank you very much for coming.

[APPLAUSE]