On Storing Private Keys in the Cloud (Transcript of Discussion)

Jonathan Anderson

University of Cambridge

Hello, I'm Jonathan Anderson, a PhD student here in the security group in Cambridge, and these are some thoughts I've been having with my supervisor Frank, who couldn't be here today. Since it seems to be very much à la mode to have disclaimers at the beginning of the Protocols Workshop talks this year, well I disclaim you have no right to quiet enjoyment of the work I'm presenting. This is a work in progress, ideas, there's no implementation, and you may very well find some points we haven't thought of, so let's have some hearty discussion.

I am interested in the distributed authentication problem. In the current set-up with social networking, email, etc., Bob provides a user name and some proof of knowledge of a password to a server, and then he can send messages to Alice. You have integrity provided by this implicitly trusted centralised service: the server tells Alice, "you ought to believe that Bob sent this because somebody who knew Bob's password was talking to me". This provides us with some convenient properties, like the fact that Bob can login from a public library. What is less convenient is the fact that, as far as social networks go, these guys in the middle are often absolute scoundrels, they can't be trusted, and the people who write their applications are even worse scoundrels: they really, really can't be trusted. A little while ago I think MySpace decided they were actually going to start selling user data: not to pretend anymore that they're doing it through advertising, they're just going to say, "here's a DVD, would you like to buy it". So we don't like this, and so for lots of reasons, some of which I touched on last year[1], I want to move to a different model where we have some kind of decentralised or distributed social networking set-up.

Now, this means that we're no longer able to rely on the very convenient, and yet rather inconvenient, centralised third party, so we have to do things like public key crypto. It's very obvious that Bob can sign messages, and he can encrypt, and that's fine. But the use case that I'm interested in, that motivates the thinking behind the protocols I'll talk about today, is Bob at the library using a shared computer. People really like to be able to do social networking from other people's computers, from friends' computers, from wherever they happen to be. So how do we do this?

We could force Bob to carry around keys with him on a physical device like this online banking token, but who wants to do digital signatures by manually typing in codes on a calculator? Nobody. Well hang on, we can actually do something a little bit more sophisticated than that: lots of people are carrying smartphones, and so you could imagine a world in which in order to do your social networking

[1] See LNCS 7028, pp 343–364.

B. Christianson and J. Malcolm (Eds.): Security Protocols 2010, LNCS 7061, pp. 107–114, 2014.

you have Bluetooth tethered to your smartphone, and whenever you want to sign something it asks, "is this really what you want to say"?

Well, we've cheated here because we're not actually enabling the use case that I care about at all: if we're saying you require a smartphone in order to do your social networking from an untrusted computer, then why not just do your social networking on a smartphone? Frankly this is also a little bit boring, it doesn't provide us with the opportunity to try and think up new protocols, and do things in a new and interesting way. So why don't we try doing something a little bit different.

So here's an illustration of a user Bob in this social networking world, surrounded by a big cloud of people, some of whom are who they pretend to be, and some of whom aren't, and Bob has something that he knows for authentication purposes: a username and password. We're going to assume, of course, that the user name is public, and the password is garbage, it's useless, it's a very, very weak password, and this is probably a reasonable assumption to make in many cases. We're going to assume that no biometrics are required for Bob to log into his social network using his friend's laptop, so we really only have something Bob knows. We don't actually have a whole lot: Bob wants to somehow protect private keys using really, really weak secrets. This seems insurmountable, but maybe not... if it were insurmountable then I wouldn't be talking about new protocols today!

What Bob does have is other people. Now, I'm going to talk about authentication agents Alice, Alicia, and Alanas, and these might be real people, Bob's friends running a bit of code for him, but they might be VMs running on various cloud providers. And if Bob can convince these various authentication agents that he is Bob, then he can get access to his key again. OK, so that still sounds pretty straightforward, so let's introduce a new complexity. The problem is that these agents also can't be trusted because perhaps your friend isn't very good at keeping malware off of his computer, or perhaps your friend wears a black hat. Or they may not be malicious, but if there was a button that said, "would you like to read all of Bob's messages", they would probably click on it: a little bit curious, but not necessarily malicious.

So our goals are: we want Bob to be store this private key out there somewhere in this cloud, and we want him to protect it using nothing but a weak secret, and we want to use untrusted peers, so they can't know what the secret is. We are going to assume that there is some kind of public key directory, but not that it provides integrity properties — the directory could return an imposter's key. If that's not hard enough, then we're also going to say, we want Bob to be able to use a weak secret to assert his identity, to get a private key which gives him a stronger identity *without* revealing his identity.

Next, problems I don't care about. If you're sitting in front of a computer which has key loggers and all kinds of bad things installed, if it wants to mess with your social networking session while you're logged in, or wants to remember your password and re-use it later, there's nothing we can do to stop that. We can mitigate risk by only exposing limited-validity keys through this protocol,

but there's no getting around the fact that using a computer requires a certain amount of trust in the computer. What we can do is deprecate all of the trust that's required in these backend systems written by scoundrels and untrustworthy people in order to reduce the trust decision to, "do I trust this computer sitting here in front of me". While most people may not be able to answer that question, at least they have some kind of a hope of being able to answer it. I'm also not trying to solve the conspiracy problem. If Bob does this distributed authentication thing with his friends and his friends all conspire against him, then Bob needs new friends, there's no technical solution for that.

So, to the protocols, hurrah. We're going to run through kind of a series of these, start with something that's awfully trivial, and then introduce new little bits of complexity, so if your criticism of the first protocol is that it's trivially broken, well yes, because it's not actually *the* protocol, it's a starting point. But as we introduce new things, as you see little yellow circles appear that say, this is new, if you think, "that's a crap assumption", then please speak up.

So the first thing you can do is kind of a webmail model that people use today implicitly. This is where you take one authentication agent, and I'm going to call her Alice, and she stores Bob's identity, and that identity is used to look up his password, and the private key, and this is not altogether different from webmail-based route of trust kind of stuff, where if you get locked out of an online service they send a password to your email account, at which point GMail is the ultimate trusted authority for everything in the whole world. The protocol is very simple: Bob sends Alice a temporary key to use to talk to him for the time being — this could be a negotiated session key, just to say, there's confidentiality we assume here. Alice says, "let's get some freshness out of you", and Bob says, "here is my password", and Alice says, "here is your private key". OK, so far so good. Attacks are obviously trivial because Alice has access to Bob's password and private key, so she can do whatever she wants, so that's kind of obvious. However, other problems include impersonations: there's no way for Bob to be sure that he's actually talking to Alice, and he's sending his password out in the clear to somebody, and he hopes it's Alice, but maybe it isn't. People like VeriSign try to deal with this in the webmail trust model, but they don't always get it right either, especially if you're living in Turkey (I think). And then of course there is the dictionary attack, where an attacker goes to Alice and tries various passwords.

Now, the nice thing is, because there's no password file that this outside attacker has access to, the dictionary attack, has to be online, which means the password doesn't have to be very strong. So what we want to do is to force the attacker into an online dictionary attack, but in some kind of distributed way that takes this massive amount of trust out of the one authentication agent. So we can take a very small baby step and say, Bob's no longer going to send his password in the clear, he's going to send a hash that was composed from things including his password, and Alice no longer stores the private key directly, she stores it encrypted under a key that's derived from this weak password. This sends a social signal in a sense: it says, I would prefer you not read my password,

but there's no technical enforcement, because it would be very, very easy to brute-force these weak passwords that we're making assumptions about, and furthermore if there was an app that said, "would you like to read your friend's password", then again, many people would just click "yes".

So can we do a little bit better? Well, perhaps we can. Given that we're talking about multiple people, you might immediately think of a threshold encryption scheme: instead of giving Alice access to Bob's private key, she just has one chunk of it, and Bob has to go and convince n of k people that he is Bob, and then he can reassemble his private key. This is an obvious step, but it actually provides zero additional security, because in order for Bob to prove his identity to Alice, Alicia and Alanas, all of these agents still have to hold this hash, and the only secret that can possibly be in this hash is the weak password. So doing this thresholding scheme actually doesn't give you any additional security properties on its own because this can still be brute-forced, and once Alice has this, she can go to Alicia, she can go to Alanas, whoever, and she can say, "here's the password, please give me your chunk of the key", and reassemble Bob's key. Again, it's a slightly higher social barrier, because it's a little bit more of an active attack, but again, there's nothing to actually stop people from doing it.

So here is where we try to do something a little bit more creative, and a little more interesting, and when I say creative, I mean taking inspiration from the 15 year old paper that Bruce Christianson and Mark Lomas wrote[2], in which they were authenticating kernels moving around on a network. What if we intentionally introduce collusions into the hashes? Instead of Alice holding this hash, she holds this hash mod N, where N could be something like 256: she doesn't have a hash of Bob's password, she has several bytes of the hash of Bob's password. Now you might say, "whoa, whoa, whoa, this makes the authentication much weaker because it's easier to brute force". That's true, it does weaken the authentication with each individual agent, but also greatly weakens the attacks that can be performed by a malicious agent. This comes from the fact that now passwords become equivalent: "biscuit" and "cat", SHA-1 mod 256, both hash to the same value. An insider with several bytes of Bob's password hash can brute-force it, but the result isn't a password, it's a whole set of passwords; the insider becomes an outsider running online dictionary attacks against the other agents, just with a little extra information to start with.

If we do some sums, we expect that Alice will be able to guess Bob's password via a successful dictionary attack if this inequality is satisfied, where α is the number of guesses that you get per agent, small n is the number of agents Bob uses, x is the size of the dictionary — we'll talk about the assumptions on the dictionary in a second — and N is this value that you're hashing modulo to. And so for different parameters you can see α gives you different values of N.

Now let's talk about some of the assumptions that go into this. This only works if the the dictionary is composed of equally-likely passwords. Now you might say, "whoa, people don't choose equally likely passwords" and you'd be right, so we can't let people choose their own passwords. That's a bit of a downside. However,

[2] Reference [5] in the position paper.

look at the size of the dictionary: 10,000. Can we expect to remember a 4-digit PIN? Maybe. A dictionary of 10,000 words includes words like cat and dog, a dictionary of size 100,000 includes things like monkey7, so we're talking about pretty weak passwords, and the hope is that, even though they're chosen for you by the computer, if they're really, really easy then you might be able to remember them. And of course, if you start doing traditional stronger passwords then you get much better protection, so this dictionary explodes. This slide shows, for various assumptions of the number of authentication agents, and how many guesses they let you have. If you're allowing passwords like monkey7 or fish2, then if N is greater than 5,000, then you'd expect Alice to be able to conduct this attack successfully. So you set $\alpha = 1024$ or something, and that means that the probability of an outsider guessing correctly when they go to Alice is very, very low.

So that's the insider guessing attack. An outsider can come and do the dictionary attack, but again, in order to do so, the first thing they do is they go to one agent, or they go to all the agents, they try to find a password, or a bit of hash that works, and that's α times little n over big N probability of succeeding, and once they have that then they become an insider, and then they have to run this attack. So in fact we can still have some pretty strong properties, even though the authentication with each agent is weak. So we say, OK, that's pretty cool, at this point let's declare victory, hurrah, we've won, Bob was able to store a private key out there somewhere, he is able to protect it with nothing but a really, really weak password, he's able to recover it, he's able to prevent insider and outsider attacks, this is great.

But we still have the problem of an impostor doing an identity disclosure attack. In some cases Bob might not want anybody in the world to know that he's using this service, and he's stored his password with Alice: maybe that means Alice gets her door kicked down and that's very inconvenient for everybody. So if we apply a very similar kind of technique then we can in fact do identity hashing as well. The N in this modular operation can be truly, truly small. We could start with N_I as 2, and then move to 4, and then move to 8, asking "now can you uniquely identify me in the set of people that you are performing this service for?". If you're only acting as an authentication agent for ten of your friends, or if your computer is only running authentication agent code for tens of friends, you actually should be able to uniquely identify people with an NI that's very small. The "try again" message is because you do in fact want to uniquely identify which person you're authenticating as, because once you start saying, "this bit of hash is valid for somebody that Alice authenticates", we might start to get into trouble if she in fact authenticates many clients.

The "try again" message seems a little scary, like a chosen protocol attack, but it's not really: if Alice is in fact being a very bad person, and she's just sending lots of "try again" messages until she can get a really, really large chunk of hash out of Bob's identity, by the time she gets to the third or the fourth "try again" message, Bob's already successfully authenticated with everybody else. And of course, there could be a general or user-specific limit on this N. Exactly

how to deal with that is perhaps the weakest bit of what I'm talking about here, but I do think this is a viable approach to let Bob to get his key out with a very weak secret. I do mean really, really weak: cat, dog, these are not the kinds of passwords that we encourage people to use, but they could be just fine in this system as long as it's assigned by the machine. Using peers who are hopefully not conspiring against you but are otherwise untrustworthy, and we can even let Bob assert his identity without revealing his identity, which I think is kind of cool. So that's all. Any questions?

Malte Schwarzkopf: So you've outlined all of this for authentication and obviously that's the viable use case, but if you're saying, as you said at the beginning, all the social network providers are scoundrels who want to sell all your personal data, then you also need to protect all the message exchanges. So are you suggesting that you use this for authentication, then devise some sort of encryption scheme, and then use that to encrypt your messages, because your data is still being stored somewhere, and they still have access, and my question I guess is, do they actually care that much about your password if they don't actually need your password in order to get to your data?

Reply: So this all kind of comes out of work that I want to do for distributed social networking, where in fact there is no centralised provider who stores all of your data in the clear. And one of the barriers to that is the fear that people will have to carry their private keys around, so I thought about how we can do something a bit more interesting. So the assumption is that you might use a centralised provider for availability reasons, but you don't trust them with your confidentiality or integrity, they're a CDN that stores encrypted blobs of stuff, and the clients have to interpret what those blobs mean.

Ross Anderson: The collision for hash function stuff that a number of us played with 15 yeas ago, Mark Lomas, me, Bruce, etc., was kind of a response to Steve Bellovin's EKE paper. That was an attempt to set up a protocol providing the user password to authenticate to a website by authenticating the set-up of a different known key in such a way that if you didn't know the password you couldn't extract it from the protocol. Now there were several dozen variants on that theme produced using different mechanisms, and in the end none of them got deployed because Lucent, the successor of Bell Labs, where Steve Bellovin worked, took the dog in the manger attitude to patents: they just waited and hoped that Microsoft would put this in browsers so that they could sue Microsoft for $100m. And so Microsoft of course didn't put it in the browser. But it happened in a few years, so it's perhaps opportune to start thinking about this type of protocol again.

Reply: So one of the results of Olivetti cryptic key exchange stuff was the SRP protocol, which Joe pointed me at, which again does secure remote passwords, but the problem with protocols like that is, you can't extract the password by observing a run of it, but the person that you're authenticating to does have to have some kind of information that they could use to get your password if your password is as weak as we're assuming it is. So they have to store something, and they can brute your force your password out of the information that they

have. If they're compromised or they're wicked then they can get your password, and then they can impersonate you, and so that's the motivation for why I did the collision-rich hash stuff, again because that sort of a model doesn't really give you the property that I want, it's still basically requires you to fully trust an authentication agent.

Malte Schwarzkopf: One comment about the last bit where you said, OK, with the "try again" thing you're kind of protected against a malicious authentication if you are already authenticated with everyone else, so at some point you just don't care about that anymore. So that's obviously true, however the assumption there is that your connectivity to all of them is the same, and that your adversary cannot engineer the connectivity, which might be a perfectly reasonable assumption. But this only fails under the assumption that you don't have one malicious authentication agent who can send you messages very quickly, much quicker than everyone else, and thereby get access to your key.

Reply: So that's true, and that's why it would be nice if Bob could know my N_I, I have a maximum value that I'm going to let that go to, and I'm never going to reveal more than eight bytes of this identity hash, because nobody ever needs that, for instance. This is one of those things where earlier I said we assume that there is a directory where you can look up public keys, we make no assumptions about integrity. It would be nice if there was some kind of visual fingerprinting or something, Bob could at least say, yes, I think this is my public key, so Bob could read a self-signed message saying "my maximum N_I is this, and I will never respond to messages greater than that", so Bob can cache that information for himself. But again, that could just be in a lot of cases a software default sort of thing.

Malte Schwarzkopf: And the second question, so how would you go about bootstrapping this model? You still have to start off with a small number of authentication agents, you can't just join this social network and immediately have 20 friends that I can use as my authentication agents. Now they don't necessarily have to be a friend, there are other models as well, but if we assume that they are your friends on the social network, you are at some point going to start off with one, and you have full trust in them, you then distribute later, and I was just wondering if you have at all thought about how you would actually do this redistribution once you get more authentication agents, how it would scale, how you bootstrap it?

Reply: So part of what I want to say to that is that this stuff only applies when you are to the point where you want to be able to use other computers, so when you're first getting started, you have client software on your computer, you generate a master signing key for yourself, which probably gets stored on your computer, or if you're really paranoid, only on your Android phone, or if you're super-paranoid only on your iPhone. And so you set up your key, you do mutual identity verification, you do stuff like the MANA-III protocol, which allows you in a very lightweight way to figure out that the person you're talking to online is in fact the person that you talk to in real life, you could do all these things. Once you're ready to set this up, and this could be after you've identified friends, after

you've purchased virtual machine time ahead of time, then you set this stuff up, and you just do some kind of secret sharing scheme, so if you add new people, it's kind of a matter of just recreating this stuff. And this thing doesn't need to change, if you want to change your password you can, but adding new people to the scheme, they can all keep what they already have here, and just new people have, instead of Alice here in the hash, it's Alicia, it's whoever, and this is the thing that has to change.

Joseph Bonneau: Just an observation, the uneasiness about the second message is simply the fact that there's no authentication on it at all, so you don't know who is telling you try again.

Reply: Right, and it's at this point kind of impossible to have the authentication because we're assuming that Bob can't tell Alice from Eve.

Joseph Bonneau: So going back to the SRP thing, I buy the reason you're doing a collision-rich hashing is that you don't actually have to give any of these Alices an actual password, you map it down so they have a set of passwords. But it seems like you could layer that on top of some other protocol, SRP, or whatever, where at the point in SRP when Bob is supposed compute things based on an actual password they just compute it based on this collisionable hash of the password. But since you're running high-end software anyway you could do that.

Reply: And that, yes, might be a very reasonable way to proceed. What I've got here is an outline of what a protocol could look like, and I said things like, "maybe this is actually TLS", but I just don't want to have a protocol notation for TLS. The contribution here is doing the collision-rich hash and the association with the n of k secret sharing. And if you want to base this on some well-known standardised protocol that has a user base, that's fine.