On “identities”, “names”, “NAMES”, “ROLES” and Security: A Manifesto

Charles Rackoff[*]

There is a great deal of confusion in the cryptology literature relating to various identity related issues. By “names” (lower case), we are referring to informal, personal ways that we indicate others; by “NAMES” (upper case) we are referring to official ways that we use to indicate others. Both of these concepts are often confused with “identity”, which is something else altogether, and with “ROLES”. These confusions can lead to insecurities in key exchange as well as in other internet activities that relate to identity. We discuss why we should not use names in protocols and why we *cannot* use identities. We discuss why, in a public-key infrastructure, we need to use NAMES in key-exchange protocols, and how they should be chosen and why they have to be unique, and why we should *not* use NAMES in session protocols. We also argue for the importance of secure ROLEs in key-exchange protocols.

1 Introduction: Alice and Bob

The RSA 2011 Conference used “The Adventures of Alice and Bob” as its advertising theme, since “Alice” and “Bob” are used throughout the protocol literature. But what are Alice and Bob? There are at least four different meanings to Alice and Bob, and we believe that the confusions between these different meanings cause many confusions in the literature.

One important meaning is that Alice and Bob refer to particular individuals – what we refer to below as “identities”. An example of this use would be, “Alice and Bob are two people who wish to have a secure conversation over the internet”. The intent, obviously, is that Alice and Bob refer somehow to unique individuals.

A second important meaning is that Alice and Bob are unique NAMES in, for example, a public-key infrastructure. In this case, Alice and Bob will be used as strings in some protocol, for example a session-key exchange protocol.

A third important meaning is that Alice and Bob are ROLEs in some protocol. For example, Alice may be a cute word for “Server” and Bob a cute word for “Client”. Or one of them might be “Sender” and the other “Receiver” in a one-way session.

A fourth, but less important, meaning is that Alice and Bob are informal strings – what we refer to below as “names” – by which two people conveniently refer to each other. For example, since I only know one person named Alice, I have an entry in my address book under the name “Alice”. Or perhaps I know a number of people named Alice with confusingly similar last names, so I have an entry in my address book under the name “Alice-Toronto”. In any case, (the identity) Alice neither knows nor cares what my name for her is.

[*]University of Toronto, `rackoff@cs.toronto.edu`

The goal of this paper is to better explain these distinctions. As a case study we will describe how they relate to uses in session-key exchange protocols with respect to a public-key infrastructure, and to session protocols that use a shared key.

2 names

A name is something we use to identify someone for personal reasons. For example, I have an email "address book" which links names with email addresses. The name I use for someone is a string that I find convenient to identify that person. For example, someone might have their entry for me as "Charles Rackoff", whereas someone else might have the entry "Rackoff" (clearly they know no other Rackoffs) or "Charlie" (because I am the only Charlie they know).[1] Another use of names is when two people talk about a third. Clearly, they must use an agreed upon, informal name for that person.

3 identities

A discussion of identity actually should precede a discussion of names, since names are personal ways of identifying someone. If you want to specify my identity to someone, what should you do? You could of course use your personal name for me, but that only works if you and the other person already have a shared name for my identity. You could give my full first and last name and say where I work. But that is only sufficient if that information uniquely specifies my identity, which begs the question of what is meant by my identity in the first place.

I believe that that the only way for you to correctly specify my identity is to point to me! As an extreme case, imagine that there are two identical twins that no one can really tell apart, and that use the same name and that randomly substitute for each other in marriage, at work, etc. How to specify the identity of one of them? Obviously, the only way to do this is to point, and to follow this person around without losing track of him. We will deal with simpler cases below, but ultimately it all comes down to pointing.

This notion of identity is very subtle and largely empirical. For example, it is easy to imagine other planets with intelligent life but without the concept of identity. Perhaps two "people" on this planet can get together, merge, and then separate in such a way that various aspects of their identities are mixed and then divided up amongst the two of them. We have the notion of identity on *our* planet only because it empirically makes sense.

Even on this planet, however, problems emerge with the notion of identity. The most obvious issue is death. When we identify a corpse as being a specific identity, we might mean it, or we might mean that the body in question used to house the identity in question. This issue becomes much more important when we consider someone with advanced Alzheimer's disease: has the identity been retained, or has it been replaced by a totally new identity, or with none at all? What about someone who has become schizophrenic? Often we describe such a person as having acquired a completely new identity; often we think of the person as having two identities, depending on whether or not he is taking his medication. What about the Sybils of the world (let us assume that multiple personality disorders exist) who have many different personalities? Is it possible to send

[1]Unfortunately, these personal name indicators are usually sent along with the email, so the recipient learns your personal name for him. A similar problem is that when you attach a file to an email, the recipient learns your name for that file. This issue will not be dealt with further here.

one of Sybil's personalities an email? Is it possible for each personality to have her own private-key? Lastly, often a person is described as "suffering" from Down's syndrome, as if there is an identity buried deep inside that we have never met and never will.

Is it possible for a person who does not have schizophrenia or multiple personality disorder to nonetheless choose to have more than one identity – for example, an email address for his relatives to use and another for his coworkers? What about corporate identities? We will address these issues in Section 9.

4 NAMEs and public-key infrastructures

Now consider how a public-key infrastructure (PKI) should work. A trusted authority maintains a database that can be reliably accessed in some way. Each entry in the database corresponds to an identity and should have three fields:

- a description of the identity in question
- a public-key for the identity
- a NAME for the identity

The first field exists so that the user of the database can look up a chosen individual. This field will contain a description of the identity in question. Since it cannot actually be the identity, it attempts to describe it as well as possible. It will contain, for example, names, and possibly other stuff such as age, a city of residence, occupation, workplace, photos, distinguishing scars, awards received etc. Someone using this database has in mind an identity (that is, a particular person) and may search the database by searching this field for a name, and will hopefully find exactly one entry that well describes the desired person. Alternatively, he might find – or someone (perhaps the person in question) might send him – the signature by the authority of the three fields of an entry, and he will be convinced that the description describes the identity he is interested in. Alternatively he might get this information through a chain of authorities. Alternatively, the person (i.e., identity) in question might give him the NAME and public-key in person.

The second field contains the public-key for that identity. We are assuming the setting where the person who the entry denotes chooses the public-key himself, and then presents it to the authority, together with the desired description, together with sufficient proof that he matches that description; the authority will verify (as well as he can) that the person is consistent with the description. An honest person will choose his public-key using a specified key-generation algorithm. A dishonest person, however, can choose his public-key however he likes; he can even choose it to be related to or even identical to someone else's public-key. The person will also choose what and how much information to provide about himself (perhaps trading off uniqueness of identification for privacy) in the description, and he might have to pay extra money to have extra information verified.

The NAME field will contain a string of a predetermined length, and **the only "meaning" of this NAME string is that it is unique and unrelated to the public/private-keys**. By unique, we mean that the NAME field of every entry must be different from the NAME field of every other entry. There are any number of ways the authority can ensure this uniqueness, such as keeping a counter, or choosing each NAME randomly, or hashing (using a collision resistant hash function) the identity description.[2] Alternatively, the person denoted can choose the NAME himself in any way unrelated to the public/private-keys, and the authority can check that it is unique. We don't attempt to stop a dishonest party from choosing a NAME that relates to his keys, but his NAME must be unique.

The reason we need the NAME to be unique and (for an honest party) unrelated to his keys will become clear in the next section.

5 NAMEs and key-exchange protocols

NAMEs play a crucial role in key-exchange protocols because public-keys are not guaranteed to be unique. If public-keys were guaranteed to be unique then we wouldn't need NAMEs at all in these protocols. However, if an adversary is free to choose his public-key to be the same as someone else's, and if we didn't have NAMEs to force uniqueness, then an adversary can force a situation that most people would probably view as an insecurity. Here's what can happen:

- Alice and Bob are two honest identities. (I am using names to refer to them, since I cannot point.) Evil is a dishonest party who chooses his public-key to be the same as Bob's. Consider a process P_1 that Alice launches in order to talk with Evil, and a process P_2 that Bob launches in order to talk with Alice. An adversary can cause these two processes to communicate with each other; since Bob and Evil have the same public-key, neither process will detect anything amiss and the two (non-matching) processes will compute the same session-key. Most people consider this to be an insecurity. For instance, Bob will see messages that he thinks Alice intended for him, whereas Alice was actually intending those messages to go to Evil.[3]

The reason we insist that honest people have NAMEs that are independent of their public/private-keys is more subtle. It is because in most definitions of security for key-exchange, the adversary is forced to choose an honest party's NAME independently of its public/private-keys. If the honest party was not similarly forced, we could have the following bad scenario: Imagine that public-keys are chosen so that the first n bits are random, and that a person's NAME is then chosen to be the first n bits of his public-key; what could go wrong? The problem is that there are weird key-exchange protocols that are secure according to most definitions, but that would be insecure in practice in the scenario just described. An example is the following.

- Alice and Bob are two honest identities, and Alice launches a process P to talk with Bob. P begins by checking if her NAME is the same as the first n bits of her public-key (which we assume is part of her private-key, which she sees); if not, then she continues to perform a secure key-exchange; if so, however, she does something stupid, such as choosing her session-key to be all 0's.

So NAMEs serve a technical purpose in key-exchange protocols in a PKI; they exist at a logically lower level of protocol than that representing the intent of the user, who rather thinks in terms of names and identities.

[2]There is a problem with the last suggestion if two descriptions are identical, but there is a problem in any case if two descriptions are identical.

[3]More rigorously, according to an "indistinguishability-based" definition of security, the adversary will "open" P_1 to see the session-key, and then "challenge" P_2. The challenge string will be either the session-key or a random string, and the adversary will be able to easily tell which. This issue is sometimes called an "unknown key share" attack.

6 Why and how key-exchange protocols are performed, and how their results are used

In a PKI, how should a key-exchange come about? It should happen as follows. An identity who (since I cannot point) we will call Alice, informally and insecurely talks over the internet with someone she thinks is Bob (the name she uses for a particular identity) and they decide (or so she thinks) to have a conversation (or session). She then looks up Bob's NAME and public-key as discussed above in Section 4. Next, she launches a process P whose inputs are:

- Alice's NAME, her private-key, Bob's NAME, and Bob's public-key

P then executes the key-exchange protocol.

What does Alice get returned from the key-exchange protocol? The protocol might FAIL, but if it doesn't, then **she receives the session-key and nothing else** and she is now ready to engage in a session using that key. To summarize: Alice decided to have a session with Bob (her informal name for an identity), and now Alice has a session-key to use for that session. At this point she does *not* know Bob's NAME or public-key, or her own NAME or private-key; she does not know the IP-address she thinks she is communicating with; she does not know any of the data that formed part of the key-exchange protocol. All of this information only exists at lower logical levels, and Alice only knows the session-key. This is discussed more in Section 7. (Actually, **there is one extra bit that Alice must know**, and we will return to this issue shortly.)

7 ROLEs and sessions

Assume that identity Alice has securely shared a session-key with identity Bob. How should Alice use the session-key to have a secure session with Bob?

They should use a "secure session protocol". The session protocol will use the session-key, but **the designer of the session protocol should not have to know how the session-key was exchanged**. What should Alice and Bob have as inputs? Certainly they should have the shared key. What about their NAMEs? Since the the designer of the session protocol should not have to know how the session-key was exchanged, **the session protocol no longer has access to these NAMEs**; they were rightly discarded as existing on a lower logical level from the intent of a person who has shared a session-key with an identity and now wishes to engage in a session. In fact, a secure session can take place in a setting where no NAMEs exist. For example, the two parties might have gotten together and flipped some coins to create a session-key, or they might have used a long-term shared private-key to exchange a session-key, or they might have used yet another method such as Kerberos (which involves a third party).

Should names, or some other proxy for identity, be inputs? No. But the reader may worry, don't I need to know who I am talking to? Obviously I need to know who I am talking to, but this doesn't mean that I calculate what I say as a mathematical function of someone's name. In fact, a session protocol doesn't model at all how the conversation is chosen[4] – it merely comes from outside as a stream of inputs.

Should there be some sort of "address" or "session identifier" as input? No. Again, this is a confusion of levels. Of course, there has to be some way for people to communicate that works well in the absence of an adversary. The technical term for how this is done in practice is "The magic

[4]When we define security, of course, we model the conversation as chosen by the adversary.

Of The Internet". That is, somehow bits get routed properly if there is no adversary and not much noise (noise is really just a type of adversary). I haven't got a clue about how this works, but I'm told that it involves stuff called "IP addresses" and other stuff called "port numbers", together with some liberally sprinkled pixie dust. The point is that it somehow works, and tomorrow it might work differently, and it really should not be the concern of session protocol designers to worry about this. (Some theoreticians have claimed that a key exchange should somehow incorporate addresses or session identifiers, presumably to later serve as pixie dust, but this is just magical thinking.[5]) The goal of a secure session protocol is not to effect the actual communication path, but rather to ensure privacy and integrity, no matter what an adversary might do to eavesdrop upon or interfere with the internet magic.[6]

Let us assume Alice and Bob take turns talking. Does Alice talk first or does she wait for Bob to talk first?[7] Because of a serious flaw in the above development, there is no good answer to this question! Certainly the question is of great importance. It would be bad if neither party starts. But it could be much worse if *both* parties start. For example, it could happen that each party exclusive-or's some message bits with the identical piece of a string that was pseudo-randomly generated from the session-key; this would be a terrible insecurity.

So how can Alice and Bob decide who starts? **The flaw in the development in Section 6 is that the key exchange did not create a secure asymmetry,** so Alice and Bob must securely create that asymmetry now. How to do this? Even if we designed a session protocol that was intended to follow a PKI (public-key infrastructure) based key-exchange as above, one could not use the names or the IP-addresses (for example) to create an asymmetry since these played no role (let alone a secure role) in the key exchange. We *could* base an asymmetry on the NAME of Alice and the NAME of Bob. One problem with this idea is that it is possible that Alice and Bob are really the same identity with the same NAME. (For example, maybe you launched a process on your laptop to talk with a process on your home computer.) A worse problem is that, as we said above, it is not reasonable to view NAMEs as existing on the same logical level as a session protocol.

An example of one way that *does* work to securely create an asymmetry is as follows (omitting some details): Alice and Bob use their session-key K to each send an authenticated random string to the other; if the strings are identical, then they FAIL; otherwise, the one with the smaller string gets ROLE bit 0 and the one with the larger string gets ROLE bit 1, and they then use a key derived from K as a new session-key.

We believe that the best solution, however, is to fix up our notion of key-exchange to involve ROLEs, where **the two parties must understand themselves – in a secure way – to have two different ROLES,** where a ROLE is simply a bit. Why is this better than the previous solution? For one thing, the previous solution involves extra flows, whereas a key-exchange protocol can almost always be modified to use ROLEs in a secure way without adding any extra flows. Much more importantly, it is natural for people designing a session protocol that uses a shared session-key to *assume* that the parties start with an asymmetry. For example, often session protocol designers assume that the party that "initiated" the key exchange will begin the session; but this only makes sense if the parties agree in a secure way on who initiated the exchange, and this is exactly the purpose in having a secure ROLE bit. For another example, often session protocol designers think

[5]Sometimes, when defining security for session-key exchange, it is useful to have a notion of session identifier. This is useful for the definition, but there is no way it can or should play a role as an actual internet address.

[6] We are making no effort here to deal with defense against "denial of service" attacks.

[7]An alternative way to have a session is that the two parties pseudo-randomly derive two new keys K_1 and K_2 from the session-key, and one of them uses K_1 to talk to the other, and the other uses K_2 to talk. But which one should use which of these keys? This is essentially the same problem.

of one party as a "server" and the other as a "client"; they would be horrified if it were possible for two servers or for two clients to share a key.

So a process running a session protocol not only has the session-key as input, it also has a ROLE bit as input.

8 ROLEs and session-key exchange

So let us return to the flawed discussion in Section 6 where Alice informally and insecurely talks over the internet with someone she thinks is Bob and they decide (or so she thinks) to have a session. We now in addition specify that they (informally and insecurely) also agree on ROLEs for the two of them, where one of them supposedly gets ROLE 0 and the other gets ROLE 1. This agreement on ROLEs can come about in a very natural way. For example, we can say that the (supposed) initiator gets ROLE 1, or we can say that in a client/server setting the (supposed) client (for example) gets ROLE 1.

The inputs to process P launched by Alice now become:

- Alice's NAME, her private-key, Bob's NAME, Bob's public-key, and a ROLE bit

The crucial point is that the ROLE bit must play a security role in this protocol. In particular, **an adversary must not be able to cause two processes with the same role bit to create the same session-key.**[8]

We will not present a complete definition of secure key-exchange here. Our goal has only been to clarify the notions of identity, name, NAME, and ROLE, and especially to discuss their importance and uses in key-exchange. We could link to a paper containing a suitable definition, but there have been no references up to this point, and it would be a shame to start now.

9 corporate identities and sub-identities

The notion of pointing doesn't work when we consider corporate identities. For example, consider the well-known McDonald's corporation. Or should that be MacDonald's, or maybe McDonalds? Clearly security should not depend on my skill as a speller, and in any case, there are many people and entities with similar names. Of course, I know which Mc-something I have in mind. Or do I? This is really an empirical issue again. For example, it may turn out that the institutions I have in mind are mostly unrelated, and McDonald's doesn't really exist as an entity. Hoping that it does, I will look it up in the authority's database (or check a signed entry I received) for a description that seems to well-describe the institution I think I'm thinking about. After doing this, I discover that it is actually spelled Macdonald, and I know it is the right entry because its description matches what I was looking for: a chain of dry cleaners in Vancouver.

Sometimes a particular identity will choose to have two or more versions of himself, or *sub-identities*. For example, he may wish to have a sub-identity for his relatives to use and another for his coworkers? The easiest way to deal with this is to have two (or more) entries in the database,

[8] Often people acknowledge that processes should have ROLE bits (for example in the case of asymmetric protocols), but don't view this as a security issue. Just as we should be wary when someone claims that a software bug has no security implications, we should also be wary when someone claims that there are no security implications when an adversary can cause a condition to be violated that we normally believe to be true: namely, in this case, the condition that if two processes share a session-key then they should have opposite ROLEs.

where each will describe the identity, and then in addition state something like "This is a sub-identity of Joe for use by his family."

Can the multiple identities use different NAMES but the same public/private-keys, or should each one use an independently chosen public/private-key? **It is best to use independent keys.** Although it appears harmless to use the same keys for each sub-identity, there is again a technical reason not to do this. The problem is that there are weird key-exchange protocols that are secure according to most definitions, but that would be insecure in practice if we use the same keys for each sub-identity. An example is the following.

- Robert and Bob are two honest sub-identities, and Robert launches a process P to talk with Bob. P begins by checking if his public-key is the same as Bob's (even though his NAME is different from Bob's); if not, then she continues to perform a secure key-exchange; if so, however, she does something stupid, such as choosing her session-key to be all 0's.

The reader might complain that this example is not too interesting since it involves sub-identities talking to each other. It is possible, however, to construct a complicated but more interesting example: a secure (according to most definitions) protocol where an adversary pretending to be Alice can talk to Bob and Robert and cause them to do bad things – for example, reveal their private-key. We omit the details.

10 "identity-based" public-key authorities

In all of the above, we were assuming a traditional public-key authority. The conceit (if viewed in a naive way) of "identity-based" encryption is that a person's "identity" can be used as his public-key. Of course, this makes no literal sense since an identity isn't even a string. It also wouldn't be possible to use my name for someone as his public-key, since that name is a personal and arbitrary way of denoting someone. Clearly in an identity-based system, it must be the NAME that can function as the public-key.

So an identity-based authority database would have one entry for each identity; the fields would be the same as before, except that there would be no public-key field – only the description field and the NAME field. To encrypt a string to that identity, I would apply the appropriate algorithm to: the string, the NAME, and the master public-key. To decrypt that encryption, the identity would have to go to the authority and convince the authority that he matches the description; if successful, he would be given the appropriate private-key.

What is the advantage of such a system? It allows for a message to be encrypted to an identity *before* the person gets his private-key. For example, the description may have a second part that specifies circumstances under which the private-key should be released; for example, perhaps a certain date should pass, or perhaps the person should acquire a security clearance. The person therefore makes two trips to the authority: the first trip is used to convince the authority about the first part of the description so that the entry can be placed in the database, and the second trip is used to convince the authority about the second part of the description so as to receive the private-key.

Of course there is an alternative way to achieve this effect; all the authority need do is generate a private-key/public-key pair for each identity, and then let the NAME be the public-key; the authority remembers the private-key, so he can release it at the proper time. However, this forces the authority to remember a (possibly) huge number of private-keys.

Lastly, we note that there is a more sophisticated version of identity-based encryption called "functional encryption", but we will not discuss this further here.