Is That You? Authentication in a Network without Identities

Taehwan Choi*, H.B. Acharya*, Mohamed G. Gouda*†
*Department of Computer Science
The University of Texas at Austin
†National Science Foundation

Abstract

Most networks require that their users have "identities", i.e. have names that are unique and fixed for a relatively long time, and have been approved by a central authority (in order to guarantee their uniqueness). Unfortunately, this requirement, which was introduced to simplify the design of networks, has its own drawbacks. First, this requirement can lead to the loss of anonymity of communicating users. Second, it can allow the possibility of identity theft. Third, it can lead some users to trust other users who may not be trust-worthy. In this paper, we argue that networks can be designed without user identities and their drawbacks. Our argument consists of providing answers to the following three questions. (1) How can one design a practical network where users do not have identities? (2) What does it mean for a user to authenticate another user in a network without identities? (3) How one can design a secure authentication protocol in a network without identities?

I. INTRODUCTION

Almost every network is designed under the assumption that each network user is assigned an *identity*, which is a name that satisfies three conditions:

i. *Unique:* The identity assigned to a network user is distinct from the identity assigned to any other network user.
ii. *Fixed:* Once an identity is assigned to a network user, then this identity remains assigned to this user (and not to any other user) for a relatively long time, measured in months, years, or decades, even if this user decides to quit the network and no longer communicate with other users.
iii. *Approved by a Central Authority:* The network has a central authority that generates or at least approves the identities assigned to all network users. This authority guarantees that (among other things) the identities assigned to distinct network users are distinct.

The second condition, *fixed identity*, needs some explanation. Assume that a user x in a network is assigned an identity i_x. Thus, when each other user in the network needs to send a message to user x, this other user needs to name i_x. Assume also that user x quits the network and its now available identity i_x is assigned to another user y in the network. Now if some user z, who is not yet aware that user x has left the network, decides to send a message to user x and names i_x, then the network delivers the sent message to the wrong user y (instead of discarding the message after recognizing that the intended message receiver has left the network). We conclude from this scenario that when a user leaves the network, its identity should be retired and not assigned to another user, at least for a relatively long time.

Some examples of user identities are as follows. The identity of a user phone in a phone network is the phone number that is assigned to this phone. The identity of a user computer in an IP network (in the Internet) is the IP address of this computer. Also, the identity of a user website in the World Wide Web is the URL assigned to this site.

The identities assigned to the users of a network play an important role in the execution of the network:

1. *User Identification:* When a user x wants to communicate with another user y, user x needs to supply the network with the identities of x and y so that the network can compute the best route for routing the exchanged messages between users x and y.
2. *User Authentication:* Any user x can be provided with a certificate that x can later use to prove to any other user that it is indeed user x. The certificate, provided to user x, has several items including the identity of user x and the public key of user x.
3. *User Reputation:* An identity is assigned to a network user for a relatively long time, and during this time, the reputation of this user, good or bad, can develop and take hold among other network users. Thus, each network can have ”reputation systems” for recording and querying the reputations of network users. Note that these reputation systems cannot be developed unless the network users have unique and fixed identities.

Unfortunately, the adoption of user identities in a network does create some security holes in that network:

a. *Anonymity Loss:* Each message that is exchanged between users x and y needs to carry the identities of x and y in the clear in order to facilitate the routing of the message between x and y. Thus any user, that can observe this message while the message is in transit between x and y, can conclude correctly that users x and y are currently in communication (even if the message contents are encrypted).

b. *Identity Theft:* In communications that do not require strong user authentication, any user x, who happens to know the identity of another user y, can pretend to be user y while it communicates with a third user z.

c. *Misplaced Trust:* As mentioned above, the existence of user identities can facilitate the development of reputation systems. However, the data stored in some reputation systems can be corrupted, for example to indicate that some user x can be trusted whereas in fact user x is not trust-worthy.

There are two approaches to address the security holes that are created by adopting user identities in a network. In the first approach, one develops techniques to defend against each one of these holes. For example, to defend against identity theft, one may require that each communication between any two users in the network should be preceded by strong mutual authentication.

In the second approach, one decides to design their network without (unique and fixed) user identities. In this case, the designed network will not have any of security holes that may be created by adopting user identifiers.

In this paper, we follow this second approach (simply because we believe that no one has attempted to follow this approach before), and attempt to answer the following three challenging questions:

i. How can one design a network without user identities?
ii. What does it mean for a user to authenticate another in such a network?
iii. How can one facilitate one user to authenticate another in such a network?

In the next section, we answer the first question by outlining the architecture of a network that does not have user identifiers.

II. A NETWORK WITHOUT IDENTITIES

In this section, we describe the architecture of a network where users do not have identities. In this network, instead of an identity, each user x has an *address*, denoted ad_x, and a nonempty set of *pseudonyms*, denoted NM_x. The value of the address ad_x and the contents of the set NM_x satisfy the following three conditions:

1. *Unique:* The value of ad_x for a user x is not equal to the value of ad_y for any other user y. Also, the contents of set NM_x for user x are disjoint from the contents of set NM_y for user y.
2. *Not Necessarily Fixed:* At any instant, each user x can change the value of its address ad_x or the contents of its pseudonym set NM_x.
3. *Approved by a Central Authority:* Only user x can request that the value of its address ad_x and the contents of its pseudonym set NM_x be changed. However, the network acts as a central authority, and declines any part of the request that violates the above *uniqueness* condition. For example, if user x requests that the value of its address ad_x be changed to a value that is currently being claimed for address ad_y for another user y, then the network will decline the request of user x.

Notice that the second condition, that ad_x and NM_x are required to satisfy, is the opposite of the second condition that an identity of user x is required to satisfy. This shows that ad_x and NM_x do not constitute an identity of user x.

The value of address ad_x indicates a physical location where user x can receive messages. Thus when some user y wants to send a message to user x, user y sends the message to ad_x.

Each pseudonym nm_x in the pseudonym set NM_x of user x is meant to identify user x in one connection with another user in the network. However, nm_x may not identify user x uniquely since other users in the network may claim the same pseudonym nm_x as their own. Therefore, when a user y requests from the network to be connected with a user with the pseudonym nm_x, and the network recognizes that there are more than one user with the same pseudonym nm_x, then the network uses a uniform distribution to select at random one of these users and connects this user with user y.

Because at any time each user x can update the value of its address ad_x and the contents of its pseudonym set NM_x, user x needs to register in the network, every T seconds, the current value of ad_x and the current contents of NM_x. Thus, every T seconds, user x sends to the network a registration message that contains the current value of ad_x and the current contents of NM_x. The network maintains a registration table where it stores the latest registered address ad_x and the latest registered pseudonym set NM_x for each user x in the network.

When a user x with pseudonym nm_x wants to communicate with another user y with a pseudonym nm_y, user x sends a request message, that contains ad_x, nm_x, and nm_y to the network. Then the network searches in its registration table for a user y with pseudonym nm_y. If the network finds no such user y in the registration table, the network rejects the request. If the network finds (exactly) one such user in the registration table, the network makes this user y.

Next, the network computes a symmetric connection key CK and sends reply messages to both users x and y. The reply message to user x contains ad_x, nm_x, ad_y, nm_y, and the connection key CK. The reply message to user y contains ad_x, nm_x, ad_y, nm_y, and CK. When users x and y receive their respective reply messages from the network, they can start exchanging messages, that are encrypted using the connection key CK.

Once users x and y receive their respective reply messages from the network and recognize that they are connected, they proceed to execute an authentication protocol in order that each of them authenticates the other. But what does it mean for a user to authenticate another in this network (where users have no identities)? We answer this question in the next section.

III. USER AUTHENTICATION IN THE NETWORK

Consider the case where a user x with a pseudonym nm_x was connected to (and communicated with) another user y with a pseudonym nm_y as many as k times, where k is at least one. Later, user x with its pseudonym nm_x requests from the network to be connected, for the $(k+1)$-th time, to a user with the pseudonym nm_y and the network grants user x its request. Now how can either user (x or y, respectively) be sure that it is connected to the same user (y or x, respectively) to whom it was connected k times in the past?

This is not an easy question to answer. For example, it is possible that after users x and y were connected k times in the past, user y gave up its pseudonym nm_y and a third user z later claimed nm_y as one of its pseudonyms. Now, when user x requests to be connected to a user with the pseudonym nm_y, user x is connected to user z instead of user y.

The answer to the above question is a new authentication protocol that we designed for our network. When two users x and y are connected by the network, if anyone of these two users, say user x, thinks that it had been connected to the other user, user y, several times in the past, then executing the authentication protocol by the two users x and y, can lead user x to know for sure whether the other user, user y, is the same user to whom user x was connected several times in the past.

Our design of the authentication protocol is intended to defend against an adversary that can perform two dangerous operations:

1. *Eavesdropping:* The adversary can read every message that is sent between any user and the network or sent between any two connected users in the network. In particular, the adversary can read every registration

message that is sent from a user to the network. Thus, the adversary can compute and maintain an accurate copy of the registration table that is stored in the network.

Note that the exchanged messages between (connected) users are encrypted using connection keys and so the adversary cannot *understand* them, even if it does read them.

2. *Impersonation:* The adversary can pretend to be a user in the network and send a message to the network or to any other user in the network. The adversary can also pretend to be the network and send a message to any user in the network. Each message, that is sent by the adversary, is composed by the adversary using the knowledge that the adversary has gained from reading all the sent messages in the network. For example, the adversary can ”replay” a message that has been sent earlier in the network.

   Note that the adversary cannot impersonate the network, because we assume that (1) the network has a private key whose corresponding public key is known to all users in the network, and (2) the network uses its private key to sign every (reply) message that the network sends to a user.

The objective of the adversary is to be connected to a user x in the network and then to use the authentication protocol to convince user x that it (the adversary) is the same user y to whom user x was connected several times in the past.

The designed authentication protocol is simple enough. When two users x and y are connected, each of the two users selects a new pseudonym and a new “token”. Then the two users exchange their new pseudonyms and new tokens encrypted using the connection key CK. Let nm_x and tk_x be the new pseudonym and new token selected by user x and let nm_y and tk_y be the new pseudonym and new token selected by user y. After exchanging their new pseudonyms and tokens, the two users x and y end up with the following tuple which defines their next authenticated connection:

$$[nm_x, tk_x, nm_y, tk_y]$$

Now assume that user x wants to establish the next connection to user y, then x initiates the connection protocol indicating that its pseudonym is nm_x and that it wants to be connected to a user with the pseudonym nm_y. There are two cases that need to be considered in this scenario.

In the first case, the network connects user x with the correct user y where both x and y have the same two tokens. In this case, the authentication protocol proceeds as follows: User x sends tk_x to user y which checks that the received token is the expected one and sends in turn tk_y to user x which checks that the received token is the expected one.

In the second case, the network connects user x with a user z, different from the correct user y. (This could have happened as follows. First, user y decided to give up its pseudonym nm_y, then later user z decided to claim nm_y as one of its pseudonyms. Thus when user x requested to be connected with the pseudonym nm_y, the network connects user x to user z which does not know either of the two tokens tk_x and tk_y.)

In this second case, the authentication protocol proceeds as follows. User x sends tk_x to user z which sends back an arbitrary value (different from tk_y) to user x which recognizes that it is communicating with a different user than user y. Thus, each of the two users concludes that it is communicating with the other user for the first time.

In either case, at the end of the authentication protocol, user x selects a new pseudonym nm'_x and a new token tk'_x and sends them to the other user, whether y or z. Also, the other user, whether y or z, selects a new pseudonym nm'_y and a new token tk'_y to user x. Thus both user x and the other user, whether y or z, end up with the following tuple which defines their next authenticated connection:

$[nm'_x, tk'_x, nm'_y, tk'_y]$

So far, we outlined broadly the three protocols in our network (where users have no identities): the registration protocol, the connection protocol, and the authentication protocol. In the registration protocol, each user sends a registration message to the network every T seconds. In the connection protocol, a user x sends a request message to the network requesting to be connected to another user y, and the network replies by sending reply messages to both x and y informing them that they have been connected. In the authentication protocol, two connected users exchange and verify their tokens in order to check whether they had communicated earlier.

In the next three sections, we discuss these three protocols in greater detail.

IV. REGISTRATION PROTOCOL

The function of the registration protocol is to allow each user x in the network to periodically register its current address ad_x and its current pseudonym set NM_x. This protocol also allows each user to periodically register its current registration key RK_x, which is a public key, selected at random by user x, whose corresponding private key is known only to user x.

The registration protocol requires that, every T seconds, each user x sends to the network a *registration message* of the following form:

$(ad_x, NM_x, RK_x, t_x, sign_x)$

where

ad_x: is the current address of some user.

NM_x: is the current pseudonym set of the user at address ad_x.

RK_x: is the current registration key of the user at address ad_x.

t_x: is the real time, or timestamp, of the user at address ad_x when this user sends the registration message.

$sign_x$: is the message signature signed by the private key that corresponds to RK_x.

The network stores the data, that are contained in the received registration messages into a table called the *registration table*. The registration table has four columns, also called attributes, named address, pseudonym set, registration key, and timestamp. The index attribute of this table is the address. Table I illustrates the registration table when it has two tuples.

When the network receives a registration message $(ad_x, NM_x, RK_x, t_x, sign_x)$ from a user at address ad_x, the network updates the registration table by executing the following protocol:

TABLE I

REGISTRATION TABLE

address	pseudonym set	registration key	timestamp
ad_x	NM_x	RK_x	t_x
ad_y	NM_y	RK_y	t_y

Step 1:

If the timestamp t_x in the message is not "close" to the real time of the network or if the message signature $sign_x$ is not correct, then the network discards the message and terminates the protocol.

Step 2:

If the network finds no tuple in the registration table whose address is ad_x, then the network adds the tuple $[ad_x, NN_x, RK_x, t_x]$ to the registration table, where NN_x is the same set as NM_x after removing from it every pseudonym that already occurs in the registration table, and terminates the protocol.

Step 3:

If the network finds a tuple $[ad'_x, N'_x, RK'_x, t'_x]$ in the registration table where $ad_x = ad'_x$ and $RK_x = RK'_x$, then the network replaces this tuple by the tuple $[ad_x, NN_x, RK_x, t_x]$ in the registration table, where NN_x is the same set as NM_x after removing from it every pseudonym that already occurs in the registration table, and terminates the protocol.

Periodically, the network checks the registration table and discards every tuple that has not been updated for more than $2T$ seconds. Note that there are three causes for a tuple $[ad_x, NM_x, RK_x, t_x]$ in the registration table not to be updated for more than $2T$ seconds:

i. User x has failed or has quit the network.

ii. User x has changed its address from ad_x to ad'_x (possibly because user x has "moved" from one location to another).

iii. User x has changed its registration key from RK_x to RK'_x (possibly to prevent its fixed registration key RK_x from becoming a fixed identity of user x).

V. CONNECTION PROTOCOL

The function of the connection protocol is to allow two users in the network to become *connected* to one another. This means that (1) each of the two users knows the current address of the other user (and so the two users can now exchange messages), and (2) the two users share a symmetric key, called their connection key CK, that they can use to encrypt and decrypt their exchanged messages.

The connection protocol consists of three messages: a *request message* from any user x to the network requesting that user x be connected to another user y followed by two *reply messages* from the network to the two users x and y informing them that they have been connected.

When a user x with a pseudonym nm_x wants to establish a connection with another user y with a pseudonym nm_y, user x sends to the network a request message of the form:

$(ad_x, nm_x, nm_y, t_x, sign_x)$

where

ad_x: is the currently registered address of user x.

nm_x: is a currently registered pseudonym of user x.

nm_y: is a currently registered pseudonym of user y.

t_x: is the real time, or timestamp, of user x when it sent the request message.

$sign_x$: is the message signature signed by the private key that corresponds to the current registration key RK_x of user x.

When the network receives a connection request message $(ad_x, nm_x, nm_y, t_x, sign_x)$, it executes the following protocol:

Step 1:

If the timestamp t_x in the request message is not "close" to the real time of the network, or if the network finds no tuple in the registration table whose address is ad_x, then the network discards the message and terminates the protocol.

Step 2:

If the network finds in the registration table two distinct tuples

$[ad'_x, NM'_x, RK'_x, t'_x]$ and
$[ad'_y, NM'_y, RK'_y, t'_y]$

where

$ad_x = ad'_x$, and
$nm_x \in NM'_x$, and
$nm_y \in NM'_y$, and
$sign_N$ is signed by the private key that corresponds to RK'_x.

then the network does the following:

- it selects at random a symmetric connection key CK.
- it sends a reply message of the form $(ad_x, nm_x, ad'_y, nm_y, t_x, \{CK\}_{RK'_x}, sign_N)$ to ad'_x.
- it sends a reply message of the form $(ad_x, nm_x, ad'_y, nm_y, t_x, \{CK\}_{RK'_y}, sign_N)$ to ad'_y.

where

$sign_N$: is the message signature signed by the private key of the network, whose corresponding public key is known to all users in network.

else the network discards the message and terminates the protocol.

TABLE II

AUTHENTICATION TABLE OF USER X

my-pseudonym	my-token	other-pseudonym	other-token
nm_x	tk_x	nm_y	tk_y

TABLE III

AUTHENTICATION TABLE OF USER Y

my-pseudonym	my-token	other-pseudonym	other-token
nm_y	tk_y	nm_x	tk_x

VI. AUTHENTICATION PROTOCOL

When a user x wants to communicate with another user y, user x initiates the connection protocol, presented in Section V, in order to achieve two goals:

i. Each of the two users obtains the current address of the other user and so the two users can start to exchange messages.
ii. Each of the two users obtains a copy of the symmetric connection key CK, and so the two users can encrypt and decrypt all their exchanged messages.

After the connection between users x and y is established, and before x and y can start exchanging data messages over the established connection, users x and y need to execute the authentication protocol in order that each of them can determine whether or not the established connection is the "first" connection between x and y.

Consider the case where this established connection is not the first one between users x and y. In this case, users x and y have agreed (as discussed in Section III) on four items in their last established connection:

nm_x: is a new pseudonym for user x.
nm_y: is a new pseudonym for user y.
$(u_1, \dots, u_m)$: is a new token for user x.
$(v_1, \dots, v_m)$: is a new token for user y.

Moreover, each of the two users has stored these agreed-on four items in a local table, called the *authentication table*, of the user. Table II and III show the authentication tables of user x and y. Note that each authentication table has four attributes named: my pseudonym, other pseudonym, my token, and other token.

Thus, in this case, before user x sent a request message to initiate the current connection to user y, user x needed to consult its authentication table in order to determine its own pseudonym and the pseudonym of user y that needed to be included in the request message.

The authentication protocol between user x, with pseudonym nm_x, and user y, with pseudonym nm_y, proceeds in seven steps as follows:

Step 1:

If user x finds in its authentication table a tuple of the form: $[nm_x, tk_x, nm_y, tk_y]$, then user x assigns to its boolean flag $conn_x$ the value true. Otherwise, user x assigns to its flag $conn_x$ the value false.

Step 2:

User x sends the message $\{u\}_{CK}$ to ad_y where the value of u depends on the value of $conn_x$ as follows. If $conn_x$ is true, then u is the token tk_x in the above tuple. Otherwise, u is selected at random by user x.

Step 3:

When user y receives $\{u\}_{CK}$ from ad_x, then user y sends $\{v\}_{CK}$ to ad_x, where v is computed as follows. If user y finds in its authentication table a tuple of the form $[nm_y, tk_y, nm_x, u]$, then v is the token tk_y in the tuple, and user y assigns its flag $conn_y$ the value true. Otherwise, v is selected at random by user y, and user y assigns its flag $conn_y$ the value false.

Step 4:

When user x receives $\{v\}_{CK}$ from ad_y, then user x computes the value of its flag $conn_x$ as follows.

If user x finds in its authentication table a tuple of the form: $[nm_x, tk_x, nm_y, v]$, then user x assigns its flag $conn_x$ the value true. Otherwise, user x assigns $conn_x$ the value false.

Step 5: User x sends $\{nm'_x, tk'_x\}_{CK}$ to ad_y, where nm'_x and tk'_x are a new pseudonym and token selected at random by user x. Then, user y sends $\{nm'_y, tk'_y\}_{CK}$ to ad_x, where nm'_y and tk'_y are a new pseudonym and token selected at random by user y.

Step 6:

If flag $conn_x$ is true, then user x removes the tuple $[nm_x, tk_x, nm_y, tk_y]$ from its authentication table. And, in any case, user x adds the tuple $[nm'_x, tk'_x, nm'_y, tk'_y]$ to its authentication table.

Also, if flag $conn_y$ is true, then user y removes the tuple $[nm_y, tk_y, nm_x, tk_x]$ from its authentication table. And, in any case, user y adds the tuple $[nm'_y, tk'_y, nm'_x, tk'_x]$ to its authentication table.

Step 7:

If the two flags $conn_x$ and $conn_y$ are both true, then each of the two connected users x and y is sure that the other user is the same one to which it was connected in the past.

Otherwise, the two flags $conn_x$ and $conn_y$ are both false and each of the two users x and y is sure that the other user is a new one to which it was not connected in the past.

After executing the above authentication protocol, the two users x and y can now start to exchange data messages encrypted using the connection key CK.

At the end of the authentication protocol, user x has a new tuple $[nm'_x, tk'_x, nm'_y, tk'_y]$ in its authentication table, and user y has a corresponding tuple $[nm'_y, tk'_y, nm'_x, tk'_x]$ in its authentication table. As long as these two tuples remain in their respective authentication tables, the two users x and y can authenticate one another correctly in the next time they become connected in the future. However, it is possible that one of these two users, say user x, may decide to discard the tuple $[nm'_x, tk'_x, nm'_y, tk'_y]$ from its authentication table. In this case, the next time users x and y become connected, their execution of the authentication protocol will indicate (incorrectly) that they (users x and y) are connected for the first time.

Note that whenever a user x decides to drop one of its pseudonyms nm_x from its pseudonym set NM_x, then user x should also drop from its authentication table any tuple where the my-pseudonym attribute has the value nm_x.

VII. DEFENDING AGAINST IMPERSONATION

In the previous sections, we have described the working of a network without identities, which we name the ITY (Is-That-You) network. In order to use this network, the user executes three protocols: the registration, connection, and authentication protocols. We now demonstrate how the ITY network resists an adversary. In this section, we show the network resisting a standard attack, *impersonation*.

In an impersonation attack, the adversary z communicates with a user x, and causes the user x to believe, wrongly, that the user it is communicating with is not z but some other user y. As in the ITY system, no user knows the identity of any other user, the attack is somewhat modified. The adversary, as a user of the system, is allowed to take the pseudonym nm_y and contact a user nm_x as nm_y – this is normal behavior, not an attack. In order to launch an impersonation attack, the following conditions must hold.

i. The user nm_x has a relationship with a user nm_y.
ii. The adversary establishes a connection with nm_x, claiming to be not only some nm_y, but the particular nm_y with which nm_x has a relationship.

For the sake of clarity, we will refer to the particular users with pseudonyms nm_x and nm_y, who are the targets of the attack, as the "correct" nm_x and nm_y. The adversary z has all the powers of a user of the network, and only those powers; he cannot, for example, cause messages sent by another user to be lost.

In order to make the correct nm_x authenticate him as the correct nm_y, z has to come into possession of the token $(v_1, v_2, \ldots, v_m)$.

z can simply attempt to guess the token; this is easily rendered impractical, by choosing a token of reasonable length – if the token has a total of l bits, the probability of guessing the token is $(\frac{1}{2})^l$, which is around 10^{-308} for a 1024-bit token.

z can launch a more sophisticated impersonation attack, making use of the fact that, just as the correct nm_y must authenticate itself to nm_x, the correct nm_x must authenticate itself to nm_y. This attack proceeds as follows.

- z takes the pseudonym nm_x, and waits for nm_y to try to contact the correct nm_x. By random chance, eventually, nm_y gets the address of z instead of the correct nm_x. In accordance with the authentication protocol, nm_y sends the token chunk v_1 to z, to authenticate that it is indeed nm_y.
- z makes its pseudonym nm_y and requests a connection to nm_x. When it gets a connection, it sends v_1 to nm_x. In accordance with the authentication protocol, nm_x responds with token chunk u_1.

- z again takes the pseudonym nm_x, and waits for nm_y to try to contact the correct nm_x. When nm_y gets the address of z instead of the correct nm_x, it sends v_1 to z. z knows u_1, so it responds with u_1. nm_y sees the correct response, and goes to the next step – it sends z the next chunk, v_2.

  $\ldots$

- In step $2m-1$, the adversary has already obtained $(u_1, u_2, \ldots u_{m-1})$ and $(v_1, v_2, \ldots v_{m-1})$. The adversary takes the pseudonym nm_x and waits for nm_y to contact it. Eventually, nm_y makes contact and begins to execute the authentication protocol: it sends v_1 and receives u_1, then sends v_2 and receives u_2, and so on. Finally, seeing the response u_{m-1}, it sends v_m. Now z knows $(u_1, u_2, \ldots u_{m-1})$ and $(v_1, v_2, \ldots v_m)$. Note that with knowledge of $(v_1, v_2, \ldots v_m)$, z can impersonate nm_y to nm_x.
- In step $2m$, the adversary contacts the correct nm_x, and using the token $(v_1, \ldots v_m)$, starts a successful impersonation of the correct nm_y.

In order to show that the protocol is in fact secure, we now show that the probability of this attack being executed successfully is exponentially small.

Theorem 1: The probability that an adversary can successfully conduct an impersonation attack is $\frac{1}{2^{3m-2}}$, when there is one user nm_x, one user nm_y, and one adversary z.

Proof: The basis for this result is the fact that, every time the correct nm_x and the correct nm_y connect, their pseudonyms and tokens change - undoing all the work of the adversary so far. The attack only succeeds if the adversary gets all the token chunks $(u_1, u_2, \ldots u_{m-1})$ and $(v_1, v_2, \ldots v_m)$ before the correct nm_x and nm_y make a single connection.

To calculate the chance of this happening, we divide the steps of the attack above into two groups. In the even-numbered steps, (step 2, 4,...) z uses the pseudonym nm_x and connects to nm_y. In the odd-numbered steps, nm_x connects to z, who is using the pseudonym nm_y.

In an even-numbered step, there are three equally likely possibilities.

1. The correct nm_x gets a connection first (to the correct nm_y).
2. The adversary gets a connection first (to the correct nm_y).
3. The correct nm_y gets a connection first. This has two equally likely sub-cases:
    a) The connection is to the correct nm_x.
    b) The connection is to the adversary.

Note that in case 1 or case $3(a)$, the attack fails: the correct nm_x and nm_y achieve a connection. The probability of success of any even-numbered step of the attack is thus $1 - (\frac{1}{3} + \frac{1}{3} \times \frac{1}{2}) = \frac{1}{2}$.

Among the odd-numbered steps, the first step is unique. The attack only begins when the first step succeeds, so it can be considered to have a probability of 1. For any other odd-numbered step, there are two equally likely probabilities.

1. The correct nm_y gets a connection first (to the correct nm_x).
2. The correct nm_x gets a connection first. This has two equally likely sub-cases:

TABLE IV

CONNECTION TABLE

sender address	sender pseudonym	receiver address	receiver pseudonym	timestamp
ad_x	nm_x	ad_y	nm_y	$t_{current}$

a) The connection is to the correct nm_y.

b) The connection is to the adversary.

For any case except $2(b)$, the attack fails. Hence the probability of success of an odd-numbered step is $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$. Note that the attack consists of m even-numbered steps and $m - 1$ odd-numbered steps, excluding the first step, and that all of these steps must succeed (independently) for the attack to succeed.

The probability of a successful attack is thus

$$1 \times (\frac{1}{2})^m \times (\frac{1}{4})^{m-1} = \frac{1}{2^{3m-2}}$$

■

The above proof also serves to highlight a design decision in the ITY protocol - the reason why we require that mutual authentication of users require multiple m rounds, instead of sending the whole token as one chunk. Our design is highly robust against impersonation attacks - for example, with a value of $m = 10$, the probability of a successful impersonation attack is less than 3×10^{-8}.

In this section, we have assumed that the adversary alternately assumes the pseudonyms nm_x and nm_y. In the following section, we deal with the case when the adversary assumes both pseudonyms at the same time; this is known as the man-in-middle attack.

VIII. DEFENDING AGAINST MAN-IN-MIDDLE

In the previous section, we demonstrated that the ITY network is resistant to impersonation attacks. In this section, we demonstrate the robustness of the network against another standard attack, the man-in-middle (MIM) attack.

In the classical man-in-middle attack, the adversary z pretends to be y when talking to x, and x when talking to y. x and y communicate, under the impression they have a secure channel; in fact, their entire communication is visible to z. In ITY, z executes a man-in-middle attack by impersonating nm_x to nm_y, and nm_y to nm_x.

Intuitively, this seems like a harder attack than impersonation; it involves two impersonations simultaneously, so when ITY is resistant to impersonation it should resist MIM attacks, too. Unfortunately, this attractive hypothesis is not correct.

Consider the following attack:

- Adversary z takes both pseudonyms nm_x and nm_y simultaneously, and waits. Eventually, the correct nm_x, trying to contact the correct nm_y, gets connected to z by chance. In accordance with the authentication protocol, nm_x sends z the token chunk u_1.

- z immediately executes the connection protocol, as nm_x, to get a connection to nm_y. On getting nm_y, he sends u_1. (If he gets another nm_y, the connection fails; he immediately tries again. This continues until, by chance, he gets the correct nm_y.)
- The correct nm_y sees some nm_x sending it u_1, and responds with v_1. z responds to the correct nm_x with this v_1.
- In accordance with the authentication protocol, nm_x responds with u_2. z forwards u_2 to the correct nm_y.

  $\ldots$
- The correct nm_x responds to v_{m-1} with u_m. z forwards u_m to the correct nm_y.
- The correct nm_y responds with v_m, which z sends to the correct nm_x.

Note that, by the end of the attack, z has been authenticated (by nm_y) as nm_x, and (by nm_x) as nm_y. Thus, even though ITY is secure against impersonation, it is not yet secure against MIM attacks.

In order to make the protocol secure against MIM, we note that the attack requires two simultaneous connections over which authentication is going on at the same time.

Clearly, we can stop MIM attacks by requiring that the system allow at most one connection at any point in time. However, this is not a practical idea: obviously, in a large system, it is not acceptable to disallow multiple concurrent conversations.

We improve on the above solution, while still preventing MIM attacks, by requiring the following conditions:

- In the authentication protocol, if nm_x sends a token chunk u_i it must receive the corresponding v_i within time t, else it drops the connection.
- In the connection protocol, the system queues all requests for a connection. No two separate connections are allowed to be made within a time period of $2mt$.

Note that, by the above conditions, when a connection is made and the authentication protocol begins to execute, all previous connections are already authenticated or failed. This makes it impossible for z to get a connection to the correct nm_y in time to respond to the correct nm_x with v_1. In other words, it is impossible to execute a MIM attack against the ITY network.

In the last two sections, we demonstrated that ITY is proof against some important attacks. In the next section, we look at some of the other approaches made to ensure anonymity in secure systems, and other related work.

IX. RELATED WORK

Identities in cyberspace are becoming more important than any other times. Department of Homeland Security has posted the draft, The National Strategy for Trusted Identities in Cyberspace[1]. The draft indicates the importance of digital identities in online transactions and provides a vision to improve online privacy with the use of trusted digital identities and creates an identity Ecosystem. This approach is an effort to defend against security holes by user identities. On the other hand, our approach is a completely different effort to protect identities by desigining a system without identities.

Security in cyberspace is based on trust whether it is related with access policy[2], or information integrity protocols[3], or reputation systems[4]. Clearly, identities facilitate trust because trusts can be developed by identities. Thus, identities are fundamental for security in cyberspace. Though identities are convenient, they cause anonymity loss, identity theft, and misplaced trust. In order to rectify these security holes, anonymous communications are proposed.

The goal of anonymous communications is to obscure the association between IP addresses and the initiator who originated the traffic. An initiator creates a path among the pool of nodes between the entry node and the exit node and the exit node contacts the responder. The three anonymous communications are notable in their originalities: MIX-net[5], DC-net[6], and Crowds[7]. The MIX-net is the original anonymous communication system for untraceable anonymous email, which uses public key cryptography to hide participants and contents of communication. DC-net is proposed for applications requiring continual deliveries with unconditional secrecy channels while MIX-net is suitable for applications requiring only periodic deliveries like an email system. Crowds is an application-level anonymity and uses probabilistic random forwarding and is limited in scalability due to its sole dependence on centralized admission control server. Compared to the approaches of obscuring the association between IP addresses and the initiator, we propose anonymous communications using pseudonyms with the three protocols: the registration, connection, and authentication protocol.

Based on the three original anonymous communications, a number of variants are implemented in diverse environments. First, a small and fixed nodes are used to relay traffic. For example, Tor[8] attempts to address some of the drawbacks in Onion Routing[9] so that it provides directory servers to maintain router information for the set of onion routers. However, this approach is not scalable due to the use of a small set of nodes. Second, peer-to-peer nodes are used to solve the scalability problems of Tor and provides anonymity for rapidly changing networks. APFS[10] leverages the peer-to-peer environment and provides two variants to eliminate the problems in responder anonymity: 1) unicast communication and a central coordinator 2) multicast routing. Tarzan[11] is also a peer-to-peer anonymous IP network overlay by extending the MIX-net with layered encryption and multihop routing. Tarzan uses a gossip-based protocol for peer discovery for the fully connected network of nodes instead of the centralized directory authority. This p2p-based approach is more scalable than Tor, but is still limited in scalability. Third, DHT is used to solve scalability and security problems of the p2p-based approach. For instance, Salsa[12] is a structured approach to organizing highly distributed anonymous communications systems for scalability and security. Salsa is similar to Tor and selects nodes randomly from the full set of nodes with the knowledge of only a small subset of the network by using a DHT to construct a virtual tree structure. Cashmere[13] focuses on the problems of being fragile and short-lived in anonymous communications and selects regions in the overlay name space as mixes and reduces the probability of a mix failure. AP3[14] is cooperative and decentralized anonymous communication service. AP3 is similar to Crowds and uses probabilistic random forwarding and implements routing dynamically.

Information leaks are prevalent. Information leaks over web sites using TLS are well-known[16]. One of the major challenges of security in cloud computing is information leaks[17]. Even with the above efforts for anonymous communications, fundamental problems seems to exist. Information leaks are unstoppable with anonymous communications. Mittal and Borisov[15] show information leaks over Salsa and AP3. More recently, Tran et al. discover information leaks over Salsa and Cashmere.

Chaum[18] foresees the problems of computerization and proposes transaction systems using digital pseudonyms to prevent privacy from big brother. Chaum proposes to use separate pseudonyms for separate transactions. But he still had a certifying authority to say who is communicating with whom.

X. CONCLUDING REMARKS

The usual structure of networks, where users are assigned unique identities, makes communication between users – as well as development of relationships between them – simple. But this simple structure comes at a price. Clearly, associating a name with a user leads to loss of anonymity; there may be concerns about reputation – other users can judge one's actions; and the network itself may show biased behavior. Most importantly, there exist specific attacks such as phishing and pharming, which seek to steal a user's identity.

In this paper, we present a highly novel idea: could it be possible to design a network without identities? We believe this is in fact, possible to do, and present the outline of a network in which users do not have identities. Users are contacted by searching for their "pseudonyms", which they change frequently. Authentication is done by users themselves, not by the certification of a central authority. This system is not only completely proof against attacks like phishing – there is no identity, hence no identity theft! – we demonstrate that it is also highly robust against impersonation and man-in-middle attacks.

Besides the theoretical novelty of the idea, we are pleased to report that it shows considerable promise for future research. The entire concept of a network without identities is very interesting, as it opens up the question of inter-user relationships without external reputations; indeed, we venture to suggest that this may be a whole new kind of network, distinct from both traditional clent-server and reputation-based peer-to-peer networks. In our own immediate future work, we are attempting to develop our network in more detail, so that it becomes robust against other attacks such as denial-of-service and message loss.

REFERENCES

[1] W. House, "National strategy for trusted identities in cyberspace," June 2010.

[2] K. B. Frikken, J. Li, and M. J. Atallah, "Trust negotiation with hidden credentials, hidden policies, and policy cycles," in *NDSS*, 2006.

[3] G. J. Simmons and C. Meadows, "The role of trust in information integrity protocols," *Journal of Computer Security*, vol. 3, pp. 199–9, 1995.

[4] D. DeFigueiredo, E. Barr, and S. F. Wu, "Trust is in the eye of the beholder," in *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 03*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 100–108. [Online]. Available: http://portal.acm.org/citation.cfm?id=1632709.1633496

[5] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, February 1981.

[6] ——, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, pp. 65–75, 1988.

[7] M. Reiter and A. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security*, vol. 1, no. 1, June 1998.

[8] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[9] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, "Anonymous connections and onion routing," in *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, ser. SP '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 44–. [Online]. Available: http://portal.acm.org/citation.cfm?id=882493.884368

[10] C. Shields, "Responder anonymity and anonymous peer-to-peer file sharing," in *Proceedings of the Ninth International Conference on Network Protocols*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 272–. [Online]. Available: http://portal.acm.org/citation.cfm?id=876907.881583

[11] M. J. Freedman and R. Morris, "Tarzan: A peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[12] A. Nambiar and M. Wright, "Salsa: A structured approach to large-scale anonymity," in *Proceedings of CCS 2006*, October 2006.

[13] L. Zhuang, F. Zhou, B. Y. Zhao, and A. I. T. Rowstron, "Cashmere: Resilient anonymous routing," in *NSDI*, 2005.

[14] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach, "Ap3: cooperative, decentralized anonymous communication," in *EW 11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*. New York, NY, USA: ACM, 2004, p. 30.

[15] P. Mittal and N. Borisov, "Information leaks in structured peer-to-peer anonymous communication systems," in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2008, pp. 267–278.

[16] A. Hintz, "Fingerprinting websites using traffic analysis," in *Proceedings of the 2nd international conference on Privacy enhancing technologies*, ser. PET'02. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 171–178. [Online]. Available: http://portal.acm.org/citation.cfm?id=1765299.1765312

[17] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 199–212. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653687

[18] D. Chaum, "Security without identification: transaction systems to make big brother obsolete," *Commun. ACM*, vol. 28, pp. 1030–1044, October 1985. [Online]. Available: http://doi.acm.org/10.1145/4372.4373