Is That You? Authentication in a Network without Identities

Taehwan Choi*

Department of Computer Science,
University of Texas at Austin,
Austin, Texas
E-mail: ctlight@cs.utexas.edu
*Corresponding author

H. B. Acharya

Department of Computer Science,
University of Texas at Austin,
Austin, Texas
E-mail: acharya@cs.utexas.edu

Mohamed G. Gouda

National Science Foundation and
University of Texas at Austin,
E-mail: mgouda@nsf.gov

Abstract: Most networks require that their users have "identities", i.e. have names that are fixed for a relatively long time, unique, 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 trustworthy. 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 can one design a secure authentication protocol in a network without identities?

Keywords: identity, anonymous communication, authentication

Reference to this paper should be made as follows: Choi, T., Acharya, H. B. and Gouda, M. G. (2011) 'Is That You? Authentication in a Network without Identities', *Int. J. of Security and Networks*, Vol. 0, Nos. 0/0, pp.000–000.

Biographical notes: Taewhan Choi is a PhD candidate in the Department of Computer Science at the University of Texas at Austin. He received a BS degree in Math from Sogang University in Korea in 1998 and received a MS degree in Computer Science from Yonsei University in Korea in 2002. His research interests are security in wired and wireless networking.

H. B. Acharya is a PhD candidate in the Department of Computer Science at the Univeristy of Texas at Austin. He received a B.Tech (with Honors) in Computer Science and Engineering in 2006, from IIT Kharagpur in India. His formal research interests include network security, tracing, monitoring, and stabilization; he also has interests in distributed and parallel computing, and in causality.

Mohamed G. Gouda received his first B.Sc. in Aeronautical Engineering and his second in Mathematics; both from Cairo University. Later, he obtained M.A. in Mathematics from York University and Masters and Ph.D. in Computer Science from the University of Waterloo. He worked for the Honeywell Corporate Technology Center in Minneapolis 1977-1980. In 1980, he joined the University of Texas at Austin where he currently holds the Mike A. Myers Centennial Professorship in Computer Sciences. He spent one summer at Bell labs in Murray Hill, one summer at MCC in Austin, and one winter at the Eindhoven Technical University in the Netherlands.

Since 2009, he serves as Program Director of the Computer Systems Research Program in the National Science Founbdation.
His research areas are distributed and concurrent computing and network protocols. In these areas, he has been working on abstraction, formality, correctness, nondeterminism, atomicity, reliability, security, convergence, and stabilization. He has published over twenty book chapters, over one hundred journal papers, and over one hundred and fifty conference papers.
Prof. Gouda is the author of the textbook "Elements of Network Protocol Design", published by John-Wiley & Sons in 1998. This is the first and only textbook where network protocols are presented in an abstract and formal setting. He coauthored, with Tommy M. McGuire, the monograph "The Austin Protocol Compiler", published by Springer in 2005. He also coauthored, with Chin-Tser Huang, the monograph "Hop Integrity in the Internet", published by Springer in 2006.
He supervised Twenty Three Ph.D. Dissertations. Five of his former Ph.D. students are now Full Professors at the University of Colorado at Colorado Springs, the University of California at Santa Barbara, the University of North Carolina at Chapel Hill, Ohio-State University, and the University of Iowa. Another four of his former students are now Associate Professors, and two of his former students are Assistant Professors in the U. S.
Prof. Gouda was the founding Editor-in-Chief of the Springer journal Distributed Computing 1985-1989, the first ever journal in the area of Distributed Computing. He also served on the editorial board of Information Sciences 1996-1999.
He was the program committee chairman of ACM SIGCOMM Symposium in 1989. He was the first program committee chairman of the IEEE International Conference on Network Protocols (ICNP) in 1993. He was also the program committee chairman of the IEEE International Conference on Network protocols (ICNP) in 1998 and 2005. He was the first program committee chairman of IEEE Symposium on Advances in Computers and Communications, which was held in Egypt in 1995. He was the program committee chairman of the IEEE International Conference on Distributed Computing Systems (ICDCS) in 1999.
He is on the steering committee of the IEEE International Conference on Network Protocols (ICNP) and on the steering committee of the International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS). He was an original member of the Austin Tuesday Afternoon Club from 1984 till 2001.
Prof. Gouda is the 1993 winner of the Kuwait Prize in Basic Sciences. He was the recipient of an IBM Faculty Partnership Award for the academic year 2000 - 2001 and again for the academic year 2001 - 2002 and became a Fellow of the IBM Center for Advanced Studies in Austin in 2002. He won the 2001 IEEE Communication Society William R. Bennet Best Paper Award for his paper "Secure Group Communications Using Key Graphs", published in the February 2000 issue of the IEEE/ACM Transactions on Networking (Volume 8, Number 1, Pages 16-30). In 2004, his paper "Diverse Firewall Design", published in the proceedings of the International Conference on Dependable Systems and Networks (DSN), won the first William C. Carter Award. In 2005, he won a teaching excellence award from the College of Natural Sciences at the University of Texas at Austin.

1 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. *Fixed:* Once an identity is assigned to a network user, then this identity remains assigned to this 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.

ii. *Unique:* The identity assigned to a network user is distinct from that assigned to any other network user.

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 first condition, *fixed identity*, needs some explanation. Assume that a user x in a network is assigned an identity id_x. Thus, when each other user in the network needs to send a message to user x, this other user needs to name id_x. Assume also that user x quits the network and its now available identity id_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 id_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 trustworthy.

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.

2 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:

i. *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.

ii. *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.

iii. *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 first condition, that ad_x and NM_x are required to satisfy, is the opposite of the first 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.

With ad_x and NM_x, we design the three protocols: (1) registration protocol (2) connection protocol (3) 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. We discuss these three protocols in detail in the following three sections.

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.

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 connects x to 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.

3 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 any one 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:

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

ii. *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. Each message, that is sent by the adversary, is composed 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 that (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, and 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.

Table 1 Registration Table

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

4 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, and NM_x is the current pseudonym set of the user at ad_x, and RK_x is the current registration key of the user at ad_x, and t_x is the real time, or timestamp, of the user at ad_x when this user sends the registration message, and $sign_x$ 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 1 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:

> **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).

If user x fails or leaves the network, then its tuple in the registration table remains unchanged for more than $2T$ seconds. This causes the network to remove this tuple from the registration table.

If user x changes its address from ad_x to ad'_x, then the first registration message after the change will cause a new tuple (that has the new address ad'_x) to be added to the registration table, leaving the old tuple (that has the old address ad_x) unchanged. The old tuple remains unchanged in the registration table for more than $2T$ seconds, causing the network to remove this old tuple from the registration table.

If user x changes its registration key from RK_x to RK'_x, then the registration messages after this change will be discarded, leaving the tuple of user x in the registration table unchanged. This tuple of user x (with the old registration key RK_x) remains unchanged in the registration table for more than $2T$ seconds, causing the network to remove this tuple from the registration table. Once this tuple is removed, the next registration message from user x will cause a new tuple of user x (with a new registration key RK'_x) to be added to the registration table.

5 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

i. each of the two users knows the current address of the other user (and so the two users can now exchange messages), and

ii. 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, and nm_x is a currently registered pseudonym of user x, and nm_y is a currently registered pseudonym of user y, and t_x is the real time, or timestamp, of user x when it sent the request message, and $sign_x$ 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 $nm_y \in NM'_y$
>
> 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.

Table 2 Authentication Table of User x

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

Table 3 Authentication Table of User y

my-pseudonym	my-token	other-pseudonym	other-token
nm_y	tk_y	nm_x	tk_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.

Otherwise, the network discards the message and terminates the protocol.

6 Authentication Protocol

When a user x wants to communicate with another user y, user x initiates the connection protocol, presented in Section 5, 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 on four items in their last established connection:

1. nm_x: is a new pseudonym for user x.
2. nm_y: is a new pseudonym for user y.
3. tk_x: is a new token for user x.
4. tk_y: 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 2 and Table 3 show the authentication table of user x and user y, respectively.

Note that each authentication table has four attributes named: my pseudonym, my token, other pseudonym, 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 users x and y are being 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.

7 Protocol Security

In the previous sections, we have described the working of a network without identities. Our network has three protocols – the registration protocol, the connection protocol, and the authentication protocol. Our description shows that, even if there are no permanent identities, a user x can find another user (say y, though x does not know the identity of y) by searching for his pseudonym (say nm_y). In this section, we focus on another important aspect of the network, its security.

In this section, we assume two users x and y are communicating using our network. The adversary is another user z. The powers of the adversary are as listed in Section 3.

We now demonstrate that our three protocols are designed to defend against many attacks that could potentially compromise the security of the network. We provide seven example attacks that can be launched by adversary z, and show how our protocols defeat these attacks.

Pseudonym Theft Attack

In this attack, adversary z seeks to acquire pseudonym nm_y, which is currently held by user y. The network, which maintains the registration table, does not allow two users to simultaneously register the same pseudonym. Hence, as long as nm_y is registered to y, z will fail to acquire nm_y.

We note that the registration protocol is soft-state: if a time period of $2T$ passes, and y does not send a message to update its tuple in the registration table, then the entire tuple is discarded. z can now acquire the pseudonym nm_y. But z cannot cause this to happen (unless y voluntarily leaves the network), because z cannot delete registration messages sent by y to the network.

Address Theft Attack

This attack is similar to the one above. z seeks to acquire address ad_y, currently held by user y. We can show the network is secure against this attack by adapting our argument for the case of pseudonym theft. The network only allows one user to be present at an address, so z cannot capture an address that is already registered. z can send registration messages to claim the address – but as z does not have the private key corresponding to RK_y, these messages have at least one of the following two characteristics:

1. Wrong registration key. The message does not contain RK_y.
2. Not properly signed. The message contains key RK_y, but is not signed by the private key corresponding to RK_y.

In both these cases, the registration protocol simply discards the messages. Also, z cannot prevent y from updating its registration tuple. Hence z cannot steal an address from y, while y is in the network.

Wrong Address Attack

In this attack, z tries to acquire an address ad'_z that is not its true address ad_z. Given that ad'_z is not the address of any other user, z succeeds in obtaining this address. But this attack is of no practical value – z successfully acquires a new address, but cannot send messages from it or receive messages sent to it (because it is not physically present at this address). Hence, even though this attack succeeds, it is of no importance.

Message Forging Attack

In this attack, z sends messages, either to the network or to a user (say x), which purport to be from user y. We have demonstrated, in our discussion of the address theft attack above, that z cannot successfully forge registration protocol messages.

In the case of connection protocol messages, note that every message is signed by the sender's private key. Consider a message that claims to be from a user with pseudonym nm_y, where nm_y is in fact a pseudonym of user y (and not user z). The network can verify the

signature in the message (because, by inspecting the registration table, the network can find the public key to check the signature: it is the registration key of the user with pseudonym nm_y). z does not have the private key of y, and therefore cannot forge connection protocol messages to look like they are from y. Similarly, z does not have the private key of the network, and cannot forge connection protocol messages to look like they were sent by the network.

All the messages in the authentication protocol are encrypted by a shared key CK, which is not known by z: CK is randomly chosen by the network, and is sent to the users x and y, encrypted with their respective registration keys. Consequently, z cannot even read, much less write messages encrypted by CK. Hence, we conclude that z cannot successfully forge messages in any of our three protocols.

Replay Attack

In the replay attack, the adversary makes additional copies of a legitimate message, which was originally sent by another user (or by the network), and sends the copies to the destination of the original. In other words, z causes multiple copies (rather than a single copy) of a message to arrive at its destination.

In order to solve the problem of replay attacks, we design the messages in our protocols to bear timestamps. If the recipient receives a message which is delayed, it discards the message as being stale.

Impersonation Attack

In the impersonation attack, z tries to convince a user x that he (z) is in fact another user y, known to x as nm_y. Our connection protocol ensures that, when a user x believes that he is connected to a user with pseudonym nm_y, this is in fact true. It is therefore essential for z to register pseudonym nm_y. As long as y is present in the system, and has pseudonym nm_y, this is also impossible: the network does not allow multiple users to share the same pseudonym. But it is possible for y to leave the system, in which case z can acquire the pseudonym nm_y.

Suppose z does acquire nm_y, and obtains a connection to nm_x. z still fails to impersonate y, because, in addition to the use of pseudonyms, x and y also use a pair of tokens to authenticate themselves to each other. z does not know tk_y, the token used by y (the chance of guessing tk_y correctly is vanishingly small). Hence, z cannot supply tk_y, as required by the authentication protocol, and x can identify z as being distinct from y.

Is it possible for z to know tk_y? Note that, if instead of initiating the connection, z simply waits for x to connect to nm_y, then in accordance with the authentication protocol x supplies its own token tk_x. It is natural to question whether z can leverage this feature to obtain tk_y.

Unfortunately for adversary z, this is not possible. The only user who can send tk_y, is y, who has already left the network. To counter this argument, suppose we consider a different scenario, where z obtains tk_y before y leaves the network. Even in this case, this attack is not possible. y will only send tk_y to nm_x. The only way z can obtain tk_y is by waiting for x to leave the network, then acquiring nm_x and waiting for y to contact nm_x. But in this case, x has already left the network, so obtaining tk_y is useless: there is no longer a party who can be deceived using token tk_y. (Note that, if x does rejoin the network later, he starts with a new pseudonym and a fresh authentication table; token tk_y no longer authenticates y to x.)

Man-In-The-Middle Attack

In the Man-in-the-middle or Janus attack, adversary z simultaneously impersonates x to y and y to x. x and y believe that their conversation is private, but in fact all messages are seen (in the clear) by z.

Our analysis of this attack follows directly from that of the impersonation attack. In order to impersonate x to y, z must have the pseudonym nm_x; to impersonate y to x, he must have nm_y. If in fact, he manages to acquire both pseudonyms, nm_x and nm_y, then this shows that x and y have left the network. There is no user in the network who can be affected by the attack. Hence, our network is robust against Man-in-the-middle attacks.

8 Related Work

The problem of anonymous communication over a network is an old and respected problem, and has inspired a considerable amount of research. The original papers proposing architectures for anonymous networks were MIX-net(1), DC-net(2), and Crowds(3). MIX-net, the original anonymous communication system, provided untraceable email by making use of public key cryptography to hide the participants and contents of communications. Email only requires periodic deliveries, and DC-net was proposed for applications requiring continual delivery over unconditionally secure channels. Crowds, an application-level protocol, uses probabilistic random forwarding. Unfortunately, it has scalability problems, as it depends on a centralized admission control server. Most subsequent solutions have implemented variants of the same basic idea: to have nodes relaying traffic, so it is hard to determine where a message originated. Perhaps the best known, Tor (4) attempts to provide a variant of Onion Routing (5). Tor uses directory servers that maintain router information. A user can send (encrypted) traffic on a long path through many proxy servers, before a Tor node finally sends the (unencrypted) message to its final destination. Unfortunately, Tor has well-known scaling problems.

In order to resolve the scalability problem of Tor, some authors propose peer-to-peer networks to relay messages. One such solution, APFS(6), provides two variants: 1) unicast communication with a central

coordinator, and 2) multicast routing. Tarzan (7), a P2P anonymous IP network overlay, extends MIX-net using ideas very similar to Tor: layered encryption and multihop routing. Further, Tarzan uses gossip-based protocols for peer discovery, rather than a central directory. Despite all these scalability improvements, Tarzan still has problems scaling to large networks.

Distributed hash tables (DHTs) are often used to try and overcome the scalability and security problems of the P2P-based approach. For instance, Salsa (8) is a structured approach to organizing highly distributed anonymous communications systems. Salsa uses its own secure lookup operation over a custom DHT to locate forwarder nodes. Cashmere(9) focuses on the fragility of paths through a network when there is anonymous communication. Its solution is to use regions in the overlay name space as mixes, rather than single nodes; this reduces the probability of a mix failure. AP3(10), a cooperative decentralized anonymous communication service, is similar to Crowds: it selects random relays, and implements routing dynamically.

Despite this extensive body of research, the problem of constructing a truly secure anonymous network is still open. Mittal and Borisov (11) demonstrate how to compromise Salsa and AP3 with information leaks. More recently, Tran et al. have discovered information leaks in Salsa and Cashmere, Hintz in Safeweb using SSL(12), and Ristenpart et al. in cloud computing-based anonymous networks(13). There continues to be a great deal of research into the problem of making it hard to trace the IP address of the initiator who originates traffic.

However, a characteristic feature of all the above authors is that they universally assume that IP address is identity. The aim of these networks is to obfuscate the link between the user of a network, or more precisely his role on the network, and his IP address, i.e. his identity. The above authors do not consider exactly what constitutes an identity, and what it means to protect the identity of a user. This is a glaring shortcoming, considering the critical importance of the concept of identity: not only is the word itself ubiquitous in security (for example, consider the Department of Homeland Security's recent draft, "The National Strategy for Trusted Identities in Cyberspace"(14)), it is the basis of many concepts such as access policy, information integrity protocols, and reputation systems.

In this paper, we attempt to define the concept of 'identity', and suggest that an identity is a persistent, unique, and undisputed (hence distributed by a central authority) name. Having provided this definition, we then ask the very important question, "Is it possible to have a network without identities?" The question seems absurd at first glance - how is it possible to trust some users, and give them exclusive rights and privileges (not enjoyed by other users), without identities? But we show that, in fact, it is indeed possible to build a network where users authenticate themselves to each other using only their pseudonyms - "temporary names" adopted for use on the network - without the use of fixed identities. Further, we demonstrate that this network is highly resistant to a wide variety of attacks. We suggest that this may in fact be an example of a novel class of networks.

9 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 an identity 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 attacks which seek to steal a user's identity.

In this paper, we 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. In this network, as there is no identity, there is no identity theft. Further, we show in Section 7 that the network is robust against many different kinds of attacks, notably the impersonation and Man-in-the-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 client-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] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, February 1981.

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

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

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

[5] 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`

[6] 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`

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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`

[13] 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`

[14] "National strategy for trusted identities in cyberspace," Department of Homeland Security, June 2010.