Is That You? Authentication in a Network without Identities

Taehwan Choi
Department of Computer Science
The University of Texas at Austin
ctlight@cs.utexas.edu

H. B. Acharya
Department of Computer Science
The University of Texas at Austin
acharya@cs.utexas.edu

Mohamed G. Gouda
National Science Foundation
The University of Texas at Austin
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?**

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. *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.

Due to the space limitation, we do not discuss our protocol in detail, but interested readers can find the details in [1].

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:

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.

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

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

III. 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 I illustrates the registration table when it has two tuples.

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.

IV. 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, 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.
- 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.

TABLE II
Authentication Table of User x

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

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

V. Authentication Protocol

When a user x wants to communicate with another user y, user x initiates the connection protocol, presented in Section IV, 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 II shows the authentication table of user x. 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.

VI. Protocol Security

We now demonstrate that our three protocols are designed to defend against many attacks that could potentially compromise the security of the network.

1) Message Forging Attack: 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.

2) 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.

3) 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.)

4) Man-in-Middle Attack: In the man-in-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 proof against Man-in-Middle attacks.

VII. 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[2], DC-net[3], and Crowds[4]. 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 [5] attempts to provide a variant of Onion Routing [6]. Unfortunately, Tor has well-known scaling problems. In order to resolve the scalability problem of Tor, peer-to-peer (P2P) networks and Distributed Hash Tables (DHT) are proposed. APFS[7] and Tarzan[8] use P2P networks whereas Salsa [9], Cashmere[10], and AP3[11] use DHT.

Despite this extensive body of research, the problem of constructing a truly secure anonymous network is still open. Mittal and Borisov [12] 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[13], and Ristenpart et al. in cloud computing-based anonymous networks[14]. 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.

VIII. 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 VI that the network is proof against many different kinds of attacks, notably the impersonation and Man-In-Middle attacks.

References

[1] T. Choi, H. B. Acharya, and M. G. Gouda, "Is That You? Authentication in a Network without Identities," Department of Computer Science, The University of Texas at Austin, Tech. Rep. TR-11-07, March 2011.

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

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

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

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

[6] 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

[7] 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

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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

[14] 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