

Byzantine Fault Tolerant Public Key Authentication in Peer-to-Peer Systems

Vivek Pathak^{a*} † and Liviu Iftode^a

^a Department of Computer Science
Rutgers, the State University of New Jersey,
110 Frelinghuysen Road
Piscataway, NJ 08854-8019 USA

We describe Byzantine Fault Tolerant Authentication, a mechanism for public key authentication in peer-to-peer systems. Authentication is done without trusted third parties, tolerates Byzantine faults and is eventually correct if more than a threshold of the peers are honest. This paper addresses the design, correctness, and fault tolerance of authentication over insecure asynchronous networks. An anti-entropy version of the protocol is developed to provide lazy authentication with logarithmic messaging cost. The cost implications of the authentication mechanism are studied by simulation.

Key words : Public Key Authentication, Peer-to-peer Systems, Byzantine fault tolerance.

1. Introduction

Public key authentication is a fundamental problem of digital security. Authenticated public keys are required for boot-strapping shared secret encryption methods and for verifying the integrity of digital signatures [1,2]. Trusted third parties and webs of trust are the two established methods of public key authentication.

The trusted third party model uses a centralized offline public key certifying authority that is trusted by all the participants. This authentication architecture is hierarchically extended to create Public key infrastructure [3]. While trusted third parties are acceptable in a client-server computing model, they are not suitable to the users of peer-to-peer systems for a number of reasons. It may be impossible to find sufficiently trusted parties in heterogeneous systems. Trusted third parties must support certificate revocation to prevent misuse of compromised private keys [4–6]. The off-line advantage of certificate authorities is reduced by the overhead of maintaining fresh revocation information. Considering the consistency and timely propagation issues imposed by

the mechanism, and the administrative burden placed by the security policy, it may not be possible to scale up the centralized authentication mechanism for securing large systems like the Internet.

PGP [7,8] creates a web of trust that allows peers to authenticate public keys by making manual decisions on key ownership and trustworthiness. Although PGP supports peer-to-peer key authentication, its dependence on human evaluation of key ownership and peer trustworthiness prevents its large scale usage in autonomous peer-to-peer systems. It also appears that its manual usage is limited to sophisticated users [9].

The increasing usage of large peer-to-peer systems motivates the creation of a Byzantine fault tolerant public key authentication mechanism. The proposed mechanism involves a distributed system of mutually authenticating semi-trusted parties and tolerates Byzantine faults. Authentication is *eventually correct* if no more than $\lfloor \frac{n-1}{3} \rfloor$ of the n parties are malicious or faulty. Authentication does not require predefined trusted third parties and enables secure communication in heterogeneous groups. Since it allows the autonomous light-weight mutual authentication of strangers, Byzantine fault tolerant authentication

*Corresponding author.

†Email Address: vpathak@cs.rutgers.edu

is well suited for peer-to-peer systems.

The remainder of this paper is organized as follows: Section 2 presents the system model including the trust model and its impact on system security. It outlines authentication and trusted group management. Section 3 specifies the protocols implementing distributed authentication and managing group membership. Proof of correctness and protocol analysis are done in Section 4. Section 5 describes an anti-entropy algorithm for efficient authentication. Section 6 describes an application, Section 7 describes and discusses simulation results. Section 8 discusses this authentication approach and Section 9 compares it with traditional methods. Section 10 concludes the paper.

2. Model

Public key authentication by trusted third parties is traditionally done through certificate authorities that digitally sign a public key certificate. Verification of the certificate makes a statement of the following form: If the certificate authority is honest and capable, then the private key corresponding to the certified public key belongs to the given identity. Here the concept of identity is very general. It may span applications, individuals, and organizations. In contrast, our mechanism authenticates network end-points or peers. Authentication is a proof of possession of the private key under honest majority and other assumptions as described below.

2.1. Network

Consider a distributed system of mutually semi-trusting peers. They are interconnected by an asynchronous network and are identified by their network identifiers. The network does not guarantee message ordering or delivery. However, no part of the network becomes permanently disconnected. The network is assumed to return delivery failure notifications. In particular, a notification is expected if a message is sent to a non-existent end point. Similarly, if an end-point does not implement the authentication mechanism, this fact can be detected by receiving a connection failure message.

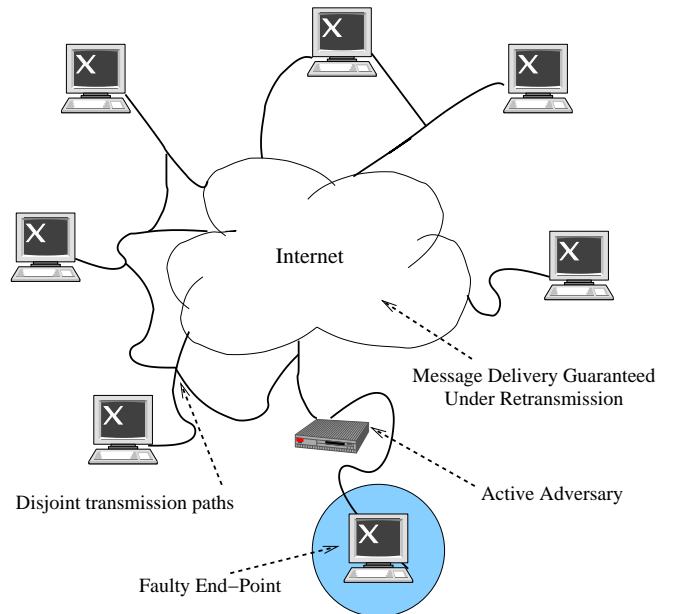


Figure 1. Adversaries and assumptions.

We assume that disjoint message transmission paths exist to each peer from some of its peers. Further, if the *man in the middle* attack can be mounted for more than a fraction ϕ of peers, then the peer is considered to be faulty and is not authenticated by this method. Faulty end points, as shown in Figure 1, are not authenticated by our protocol. The authentication mechanism is designed to detect and ignore faulty peers.

2.2. Honest majority

Correctness of authentication depends on the existence of honest peers that faithfully execute the protocol. Informally, they tell the truth about their network identity and public key. While none of the peers inherently trusts any other peer, each believes that the honest peers are in a majority. We define honest peer and honest majority as follows:

DEFINITION 1 *An honest peer protects the privacy of its private key and executes the authentication protocol correctly. A set of n peers*

has **honest majority** if the number of malicious or faulty peers $t < \frac{1-6\phi}{3}n$.

Dishonest peers may behave in an arbitrary fashion, either because of being faulty or because of being malicious adversaries. It is assumed that the system of mutually authenticating peers has honest majority.

2.3. Adversaries

The computational power of adversaries is polynomially bounded. Hence with very high probability, the adversary cannot forge digital signatures or invert the encryption transformations.

We consider active and passive adversaries. The passive adversary has unlimited power to eavesdrop on any message. While the active adversaries have unlimited power to inject arbitrary messages into the network³, they cannot prevent message delivery for more than a small fraction ϕ of the honest parties. Clearly this is weaker than the classical *network is the adversary* approach.

The weakened active adversary is appropriate in wireless networks because of physical difficulties in silencing radio transmissions. Its use in Internet applications is justified by considering the difficulty of preventing message delivery to a large number of end-points. Practical experience with Internet based systems also suggests that message injection or spoofing is the preferred form of attack.

2.4. Authentication

Challenge response protocols can authenticate public keys in the absence of man in the middle attack. Since we allow for a limited number of such attacks, a public key can be authenticated by multiple challenge response exchanges originating from different end-points.

The authentication protocol (Figure 2) consists of three phases: *Challenge response*, *Distributed authentication* and *Byzantine agreement*. During challenge response, the peer to be authenticated is challenged with encrypted nonces by a set of peers. Since the nonce can be recovered only by

³Since we do not address denial of service type of attacks, the spoofing power is not large enough to break the network or the parties processing the forged messages.

the possessor of the private key, a correct response is a proof of possession.

In the distributed authentication phase, peers forward their proofs to other peers. A peer B can authenticate a peer A after it receives a number of valid proofs from different peers. If all the participants are honest, there will be consensus on validity. In this common operating case, the protocol terminates with B becoming convinced that the public key is authentic.

If there are conflicting claims on authenticity, B can deduce that either A or some of the peers are malicious or faulty. The protocol proceeds to Byzantine agreement where the sent and received messages of different parties are validated. As all the messages are digitally signed, malicious behavior can be discovered by this procedure.

The messaging cost of authentication motivates optimization of the common case when all trusted parties are indeed honest. The public key infection protocol implements *optimistic authentication* that hides latency by proceeding before a public key is authenticated. Public keys and their authentication proofs are propagated efficiently by an anti-entropy public key infection algorithm.

2.5. Trusted Groups

Each peer has a probationary group, trusted group, and untrusted group of peers as shown in Figure 3. Peers gain knowledge of each others public keys depending on their communication patterns. Newly discovered peers are added to the probationary group. Successful authentication moves a peer in the probationary group to the trusted group. Malicious peers are moved from the trusted group to the untrusted group.⁴ Peers are also deleted from trusted groups for lack of liveness and for periodic pruning of trusted group. This is done to improve authentication performance.

3. Architecture

Byzantine fault tolerant authentication is implemented by executing Authentication protocol

⁴Continuous addition of malicious peers can cause the untrusted group to grow without limit. Therefore, peers may forget malicious behavior of the very distant past.

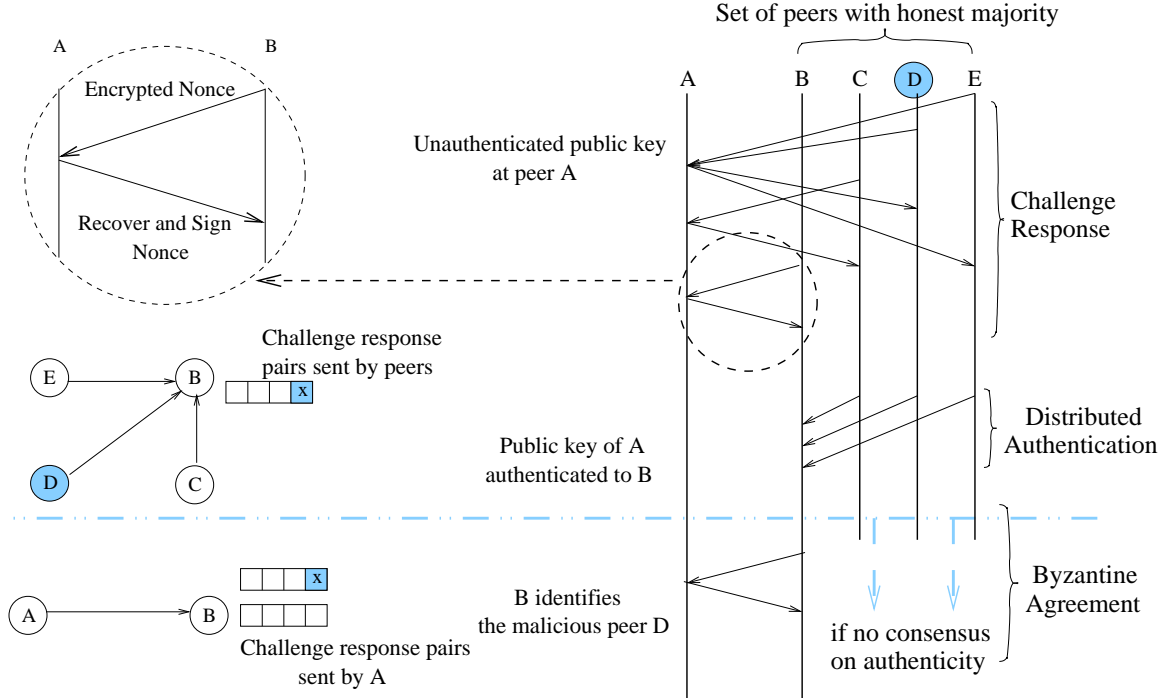


Figure 2. Authentication protocol example: A peer A is authenticated by B using its trusted peers. D is a malicious peer that tries to prevent authentication of A .

and Membership control protocol at each peer. The protocols are described as per the notation given as shown in Table 1. A bootstrapping procedure is also provided for system initialization. All protocol messages have timestamps, source and destination identifiers, and digital signatures. Peers ignore messages with invalid signatures and maintain a most recent received time-stamp vector to guard against replay.

3.1. Authentication Protocol

The authentication protocol (Figure 4) consists of the following steps:

- **Admission request**

The protocol begins when B encounters an unauthenticated public key K_A . It announces the key to its trusted group and asks them to verify its authenticity.

Table 1
Notation

K_i	Public key of the principal i .
K_i^{-1}	Private key of the principal i .
$K_i(x)$	A string x encrypted with the public key of i .
r_i	A pseudo random number generated by peer i .
$\{x, y, z\}$	A message containing three strings x , y and z .
$\{x\}_i$	A message signed by i .
$\mathcal{T}(i)$	Trusted group of peer i .

- **Challenge response**

Each peer P_i challenges A by sending a random nonce encrypted with A 's supposed public key in the signed challenge message.

1. Admission request

A peer A makes a key possession claim by notifying the peer B . If A has an expired authenticated public key K_A^* , it includes the proof of its possession $\mathcal{P} = \{A, K_A\}_{A^*}$. B announces the claim to the group.

$$A \rightarrow B \quad : \quad \{A, B, \text{admission request}, \{A, K_A[\mathcal{P}]\}_A\}_A$$

For each trusted peer P_i of B

$$B \rightarrow P_i \quad : \quad \mathcal{B}[i] = \{B, P_i, \text{authentication request}, \{A, K_A[\mathcal{P}]\}_A\}_B$$

2. Challenge response

Each peer challenges A with an encrypted nonce, and A responds with the signed response. A also stores the challenge response pair $\{\mathcal{C}_{iA}, \mathcal{R}_{iA}\}$ from its interaction with peer P_i as $\mathcal{V}_A[i]$ for use in Byzantine agreement.

At each trusted peer P_i of B

$$P_i \rightarrow A \quad : \quad \mathcal{C}_{iA} = \{P_i, A, \text{challenge}, K_A(r_i)\}_{P_i}$$

$$A \rightarrow P_i \quad : \quad \mathcal{R}_{iA} = \{A, P_i, \text{response}, r_i\}_A$$

3. Distributed authentication

Each peer returns the proof-of-possession $\{\mathcal{C}_{iA}, \mathcal{R}_{iA}\}$ to B . B saves the pair in a local variable $\mathcal{V}_B[i]$ and determines the public key to be authentic (or inauthentic) if there is consensus on validity (or invalidity) in the proofs received. If there is no consensus, B calls for Byzantine agreement.

At each trusted peer P_i of B

$$P_i \rightarrow B \quad : \quad \{\mathcal{C}_{iA}, \mathcal{R}_{iA}\}_{P_i}$$

4. Byzantine agreement

B asks the peer A for the challenges it received, and its responses to them. It then compares the proofs received from the peers and those received from A . It also notifies the peers of the received proofs so that malicious parties are eliminated from the trusted group.

$$B \rightarrow A \quad : \quad \{B, A, \text{proof request}\}_B$$

$$A \rightarrow B \quad : \quad \{A, B, \text{proof}, \mathcal{V}_A\}_A$$

If A is not proved malicious

For each trusted peer P_i of B

$$B \rightarrow P_i \quad : \quad \{B, P_i, \text{byzantine fault}, \mathcal{B}, \mathcal{V}_B\}_B$$

For each trusted peer P_j of P_i

$$P_i \rightarrow P_j \quad : \quad \{P_i, P_j, \text{byzantine agreement}, \mathcal{B}, \mathcal{V}_j\}_{P_i}$$

Figure 4. Authentication Protocol

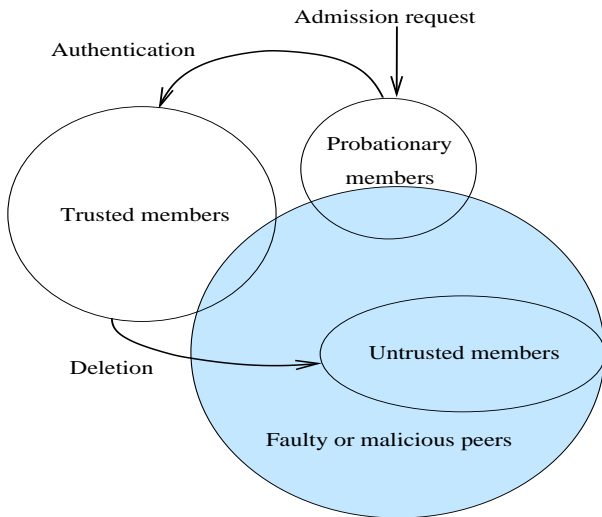


Figure 3. Group structure

A can recover the nonce only if it holds the private key K_A^{-1} . It returns the nonce in a signed response message. The challenge response message pair is a proof of possession for the public key. At end of the challenge response phase, each peer gets a proof of possession for K_A ⁵. Each challenger waits for an application specific time-out. It deletes the proof if duplicate responses are received.

- **Distributed authentication**

The peers respond to B 's authentication request by sending their proofs of possession to B . If all peers are honest, then there will be consensus on the validity of proofs. In this case, B gets the authentication result and the protocol terminates.

- **Byzantine agreement**

If there are differing authentication votes, then either A or some of the peers are malicious or faulty. To detect if A is malicious,

⁵Note that since K_A is not yet authenticated, digital signature is not verified on the response message.

B sends the proof request message to A . The response consists of all challenge messages received, and the responses sent by A . If A is honest it can prove that it received the messages because they were signed by the sending peer. It can also show a correct response. If A is not provably malicious, then some of the peers must be malicious or faulty. This leads B to announce a byzantine fault to the group. Now each group member will send the byzantine agreement message to others. At end of this phase, the honest peers will be able to recognize malicious peers causing the split in authentication votes.

3.2. Bootstrapping

The bootstrapping procedure is provided to cold-start the system. This is in contrast with the situation when trusted groups already exist and a peer joins some of them. Bootstrapping initializes the authentication system by creating a trusted group consisting of the bootstrapped peers. The peers authenticate each other by requesting admission into this trusted group. It should have honest majority to function correctly.

3.3. Membership Control Protocol

Membership control (Figure 5) serves three purposes. It preserves honest majority of trusted groups, maintains consistency of trusted group definition among sets of frequently communicating peers, and prevents excessive growth of trusted group size to limit the cost of authentication. The group operations of the protocol are described below:

Addition to trusted groups

Each peer maintains a list of to be sent authentication proofs for each probationary peer. It lazily pushes these proofs to its trusted peers. Thus the probationary peer becomes trusted at each trusted peer. A peer may pull proofs because lazy push may delay a required authentication. Peers pull the proofs by sending authentication request messages.

1. Push proofs

A peer D periodically pushes the proof-of-possession $\{\mathcal{C}_{DA}, \mathcal{R}_{DA}\}$ to peers that have not yet received its proof.

For each trusted peer P_j that has not been sent the proof

$$D \rightarrow P_j \quad : \quad \{\mathcal{C}_{DA}, \mathcal{R}_{DA}\}_D$$

2. Pull proofs

A peer B has some, but not all proofs of authenticity. It can ask any peer P_j for the proof to arrive at the authenticity, and hence trusted group membership decision for a probationary peer A .

For each trusted peer P_j that has not sent a proof

$$\begin{aligned} B \rightarrow P_j & \quad : \quad \{B, P_j, \text{authentication request}, (A, K_A)\}_B \\ P_j \rightarrow B & \quad : \quad \{\mathcal{C}_{jA}, \mathcal{R}_{jA}\}_{P_j} \end{aligned}$$

Figure 5. Membership Control Protocol

Deletion from trusted groups

Peers are deleted from trusted groups for malicious activity or lack of liveness. A malicious message causes execution of the Byzantine agreement phase that captures malicious behavior except for the lack of liveness. Deletion occurs as a result of byzantine agreement [10] on the maliciousness of the proof. If the group has honest majority, all the honest trusted peers delete the malicious peer from their trusted groups and add it to the untrusted group⁶.

A peer that fails to respond to messages in a timely manner is considered failed due to lack of liveness and is deleted from the trusted group. Performance of authentication is maintained by preserving a suitable group size. Thus honest peers voluntarily delete themselves from trusted groups by randomly selecting a trusted peer and ceasing to respond to its messages. The probability of deletion is chosen as a function of trusted group size in order to create suitably sized trusted groups.

⁶It is possible that honest peers may be deleted from dishonest groups. This causes them to join other groups, most of which have honest majority.

Group migration

Authentication depends on honest majority of trusted group. However, trusted group membership is granted on key authentication and does not guarantee that the authenticated peer will not act maliciously in future. In particular, it is possible that a number of covertly malicious peers join a trusted group so that they are in majority. Trusted groups are periodically flushed to provide proactive security against unobservable loss of honest majority. This process also guards against the Sybil attack [11] that relies on a malicious peer creating multiple fake identities. The progress made by multiple fake identities is lost by group migration.

4. Analysis

This section analyzes the correctness of authentication in honest majority groups. It is also shown that the group dynamics resulting from membership control creates honest majority groups with high probability. Previous studies [12] have focused on proving boolean correctness assertions on two and three party authentication protocols. The authentication protocol described here is open ended in number of peers and

incremental in its approach. Therefore, a direct case by case analysis of the protocol is developed below.

4.1. Challenge Response

Consider a peer B with a trusted group of peers $\{P_1, \dots, P_i, \dots, P_n\}$. Let A request admission into the trusted group of B . Each peer P_i sends a proof of possession $\{\mathcal{C}_{iA}, \mathcal{R}_{iA}\}_{P_i}$ to B , where

$$\mathcal{C}_{iA} = \{P_i, A, \text{challenge}, c_i\}_{P_i}$$

$$\mathcal{R}_{iA} = \{A, P_i, \text{response}, r_i\}_A$$

Let the proof of possession be valid if $c_i = K_A(r_i)$ and both \mathcal{C}_{iA} and \mathcal{R}_{iA} are properly signed.

CLAIM 1 *If P_i and A are honest, the proof of possession valid, and the communication path P_iA does not lose messages, then K_A is authentic with very high probability.*

PROOF: By contradiction, let K_A be inauthentic. Since P_i is honest, it transmits a correct challenge containing $c_i = K_A(r_i)$ to A , and does not disclose its nonce r_i .

Since the network path does not lose messages, the challenge will be delivered to A and the response delivered to P_i . Thus, if a single response is received, A must be the responder⁷. Since it computes $r_i = K_A^{-1}(c_i)$, it knows the private key, a contradiction. \square

4.1.1. Attacks on Challenge Response

The challenge response protocol can be attacked in a number of ways. Messages may be spoofed and originating from sources other than their apparent origin X . Man in the middle attacks may cause a peer X' to impersonate X and protocol attacks could be launched by a peer X not following the prescribed protocol.

Let a proof of possession be P -invalid if the challenge is not properly signed, A -invalid if the response is not properly signed, K -invalid if $c_i \neq K_A(r_i)$ and faulty if it is valid but K_A is not owned by A . Messages exchanged between the trusted peers are safe from spoofing and man in

the middle attacks since they are signed by authenticated public keys. Considering the various possibilities of attacks on the protocol, the effect on correctness of challenge response is analyzed below. A summary is provided in Table 2.

We consider Spoofing, Impersonation and Protocol attacks on the authentication architecture. Spoofing is defined as the attack where an adversary A' assumes the identity of a peer A . This attack is detected by the challenge response mechanism. Impersonation is a man in the middle type of attack where an adversary M impersonates A while communicating with B , and B while communicating with A . In accordance with the mechanics of the attack, A and B cannot communicate directly without passing through M . We define protocol attacks as the set of attacks that are mounted by providing incorrect responses (or lack of responses) to various protocol messages. A number of other protocol attacks like replay, type flaws and encapsulation are rendered ineffective by the use of timestamps, message identifiers and digital signatures respectively. In general, source and destination identifiers are part of message definition when the identity of communicating parties matters.

The adversary mounts a successful attack if at least one of its following goals are satisfied:

G1 Violate authentication

The adversary convinces an honest peer that the public key of A is $K_{A'}$ when it is not.

G2 Violate honest majority

The adversary creates an adverse selection of group members that lack honest majority

Consider the case of malicious peers that are not trusted by honest parties. They can attack challenge response in one of the following ways:

- **Spoofing**

A malicious peer A' may try to impersonate an honest peer A by sending the **admission request** message. If A is already part of the trusted group, then each trusted peer has its correct authenticated public key K_A . Since A' cannot produce the required proof

⁷If multiple responses are received, they are marked invalid by the protocol.

Table 2

Effect of attacks during challenge response. A is authenticated by B and its peers P_i .

Sender under attack	A	B	P_i
Spoofing	delay	K -invalid	K -invalid
Man in the middle	faulty	faulty	faulty
Incorrect response	P -invalid, A -invalid or K -invalid	delay or K -invalid	delay or A -invalid
No response	delay	delay	delay

$\mathcal{P} = \{A', K_{A'}\}_A$ without the knowledge of K_A^{-1} , each honest peer will ignore the invalid request.

Each peer will challenge A if it does not belong to the trusted group. The peer A responds to the challenges because it is honest. As it does not have $K_{A'}^{-1}$, the response will be invalid. If the adversary also sends a spoofed response, then the honest peer H will receive two distinct responses, one valid and one invalid. H considers the incorrect message and A' is not authenticated. Therefore, none of the goals are satisfied.

- **Impersonation**

By the limited power of active adversary, some of the challenges must reach A . Thus some peers will see valid proofs for $K_{A'}$ while the others will get invalid proofs. The authentication will fail because the majority of peers contact A . Thus **G1** is not satisfied.

Because each peer can prove that it issued the proper challenge, and received the corresponding response, none of them is proved malicious. Thus **G2** is not satisfied.

- **Protocol attack**

The malicious peer A' can only choose the responses for the messages it sends. This limits it to the **admission request** message and the response message. Because a receiving peer B shall verify the validity of the digital signature, the message format and the correct recipient, the only choice for the peer is to create a message of the following form:

$$\{X, B, \text{admission request}, \{X, K_X[\mathcal{P}]\}_X\}_X$$

This is equivalent to spoofing if $X \neq A'$. If $X = A'$, then the malicious peer will receive challenge messages from the peers. If it responds with incorrect source or destination identifier, incorrect signature or incorrect format, then the message will be discarded as ill-formed. Thus it can at best send a message of the following form to the peer P_i :

$$\mathcal{R}_{iA} = \{A', P_i, \text{response}, r'_i\}_{A'}$$

Since $r_i \neq r'_i$ causes invalidity, the peer P_i sends the received response to its peer B that does not find a consensus. Thus **G1** is not satisfied.

The protocol now goes into the Byzantine agreement phase. B asks A' to send its copies of the challenge response proofs. A cannot produce a valid challenge since it does not know K_{P_i} and cannot create a new challenge. Its only option is either to send nothing or to send the available copy. Since it cannot prove that it received a correct challenge and recovered its response, the trusted peer is not proved to be malicious. Thus, **G2** is not satisfied.

If the malicious peer already belongs to a trusted group, then the following attacks are possible:

- **Spoofing**

Spoofing by trusted peers is restricted to parties outside the trusted group. This is because trusted peers have an authenticated public key of the peer being spoofed, and can detect an incorrect signature on the spoofed message. A malicious trusted peer

may send spoofed challenge messages to a probationary peer. These messages will not count in distributed authentication because of invalid signature on the challenge. Further, if the probationary peer sends this invalid proof to any trusted peer, the insider is provably malicious and will be deleted in byzantine agreement phase. Neither **G1** nor **G2** is satisfied.

- **Impersonation**

Consider a trusted peer P_i being impersonated by P'_i due to a man in the middle attack. P_i belongs to the trusted group because of successful authentication. The adversary P'_i cannot convince the trusted group that it is P_i because it does not know $K_{P_i}^{-1}$. Since badly signed messages are ignored, P_i appears to be non-live to some of the peers and **G1** fails. Also P_i is not proved to be malicious and **G2** fails.

- **Protocol attack**

A malicious insider can delay the entry of a peer A into the trusted group by not sending a correct **authentication request** message. This does not satisfy either of the goals because A can find other honest peers.

The challenge response phase can be affected by the malicious insider in the following way: It can fail to send the challenge or send a number of challenges. However, A cannot be proved malicious by any such strategy because it can remember the challenges and produce them to other trusted peers. These challenges will be sent in the Byzantine fault phase when an honest peer observes lack of consensus on authenticity of K_A .

A malicious insider can delay sending the proof of possession. However, it cannot construct a bad response signed by A . As the malicious peer may only delay the authentication of A , neither **G1** and **G2** are not satisfied.

Therefore, the challenge response protocol provides correct authentication and preserves honest

majority of trusted groups.

4.2. Distributed Authentication

Distributed authentication ends with a consensus of valid if every participant is honest and there are no attacks. Given the presence of attacks and malicious peers, distributed authentication of A by B is correct as follows.

CLAIM 2 *A peer A is not mis-authenticated if it is honest.*

PROOF: Mis-authentication requires consensus on faulty proofs. Because A is not malicious or faulty, consider two possibilities: B is malicious or A' spoofs messages.

If B is malicious it can either inform its peers of an incorrect $K_{A'}$ or fail to send correct messages to its peers. If B sends incorrect key(s), the honest peers will receive **authentication request** messages and send challenges to A . Because A responds by decrypting according to its correct private key, the proofs will be K -invalid. If B does not send the correct messages, honest peers will send no challenges and some proofs will be missing.

If A' spoofs responses for A , then the honest peers will delete their proofs because A will respond too. Since there must be missing or K -invalid proofs, there can not be a consensus on faulty proofs. \square

CLAIM 3 *Honest majority is preserved at every honest peer.*

PROOF: If B is honest, lack of consensus leads to byzantine agreement. It compares proofs sent by peers with the proofs sent by A . If A is provably malicious because of sending a K -invalid proof to some peer and valid to another, or because of sending P -invalid proofs on request of B , the protocol ends with A being marked malicious. No honest peer is deleted.

If A is not provably malicious, the protocol moves into the second phase. This implies either some trusted peers are malicious or there is a man in the middle attack.

Each peer sends the proofs it has (for authenticity of K_A) to its trusted peers. Malicious peers could send conflicting proofs of possession to their

peers. These actions are detected by byzantine agreement as follows: Consider an honest peer P_j receiving the **byzantine agreement** message from other peers. Let t of the peers be malicious and may offer arbitrary proofs. Secondly, ϕn of the peers may not be able to reach A and may have faulty proofs to offer. Finally, proofs from another ϕn peers may be faulty because the peer P_j has a compromised path to them. In the worst case these three sets of peers are disjoint, and $2\phi n + t$ of the proofs can be missing. Thus, every peer eventually gets at least $n - 2\phi n - t$ proofs. However, the malicious peers could respond eagerly, causing $2\phi n + t$ of the $n - 2\phi n - t$ received proofs to be faulty. Therefore, a majority of the proofs are identical and correct at every honest peer if

$$n - 4\phi n - 2t > 2\phi n + t$$

i.e.

$$t < \frac{1 - 6\phi}{3}n$$

Therefore, using a majority vote after Byzantine agreement allows the peers to form trusted groups that contain only the honest peers that are not in the path of a man in the middle attack. This preserves the honest majority.

If B is malicious and sends conflicting requests to the peers, its signed **authentication request** messages will cause it to be detected by Byzantine agreement on the requests received. Again, by deletion of the malicious peer B , honest majority is preserved. \square

4.3. Group Evolution

Admission requests are caused by the need for secure communication. If the peers A and B intend to communicate securely, they will check if $A \in \mathcal{T}(B)$ and $B \in \mathcal{T}(A)$. In this case, the problem is trivially solved.

Otherwise, A will request admission to $\mathcal{T}(B)$ and B will request admission to $\mathcal{T}(A)$. If both A and B are honest, the admission requests will succeed in the common operating case when their groups are also honest as shown in Figure 6. If one of the requests fails, then either Byzantine agreement will correct the groups as described

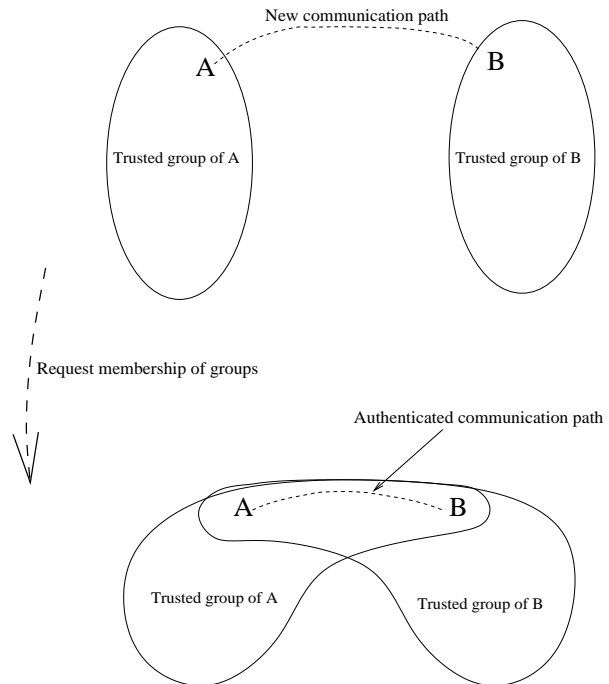


Figure 6. Dynamics of authenticated communication.

earlier, or periodic pruning of trusted groups will ensure honest majority as described in the following section. In either case the honest peers can eventually authenticate each other.

4.4. Formation of Honest Majority Groups

Since honest members form trusted groups by following the membership control protocol, any provably malicious peers are deleted from trusted groups. On the other hand, if malicious peers can successfully masquerade as honest peers, then the continuous group migrations cause the distribution of covertly malicious parties to be same as a random selection. Therefore, honest majority groups are formed with a probability greater than that of random selection.

A trusted group with $\frac{3t}{1-6\phi} + 1$ peers has honest majority if t peers are malicious or faulty. Because the value of ϕ does not change the behavior

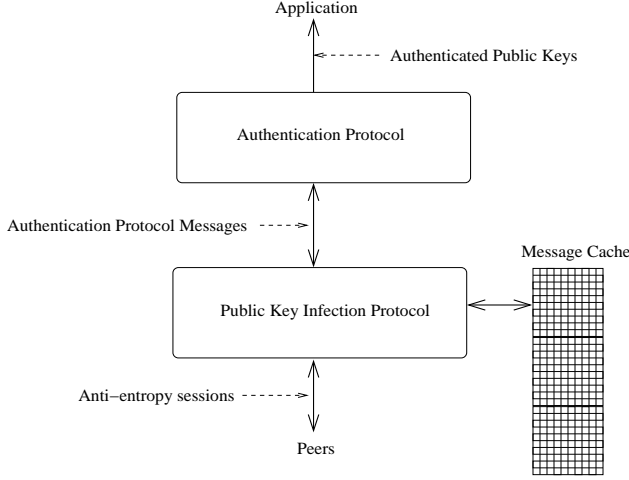


Figure 7. Overview of Public Key Infection.

of random selection, consider the honest majority group with $3t + 1$ peers. Let the fraction of dishonest parties be $\frac{1}{3} - \epsilon$, where $0 < \epsilon \leq \frac{1}{3}$. Under the assumption of independent random selection of members, the probability $P(i)$ of choosing \mathcal{T} with i dishonest members is given by the following binomial probability:

$$P(i) = \binom{3t+1}{i} \left(\frac{1}{3} - \epsilon\right)^i \left(\frac{2}{3} + \epsilon\right)^{3t+1-i}$$

The probability $P_h = \sum_{i=0}^t P(i)$ of selecting an honest majority group is computed numerically and rapidly converges to 1 as ϵ approaches $\frac{1}{3}$.

In practice the untrusted sets preserve information about past malicious activities and impede the free assimilation of dishonest parties into trusted sets. Thus, two honest parties A and B can have an expectation $(1 - P_h)^k < \frac{1}{2^k}$ of being incorrectly authenticated if they continue communication through k group migrations.

5. Public Key Infection

The protocols implementing distributed authentication are expensive in messaging cost. Trusted groups execute protocols that include

Table 3

The cache record data structure.

lt	The Lamport timestamp
ct	The causal timestamp at source
src	Source
dest	Destination
mesg	The protocol message

broadcast messages leading to an $\mathbf{O}(n^2)$ cost for mutual authentication in trusted groups. In order to reduce the messaging cost, we use an epidemic algorithm called *Public key infection* for lazy propagation of protocol messages as shown in Figure 7. Because each protocol message is protected by an unforgeable digital signature, it is possible to store and forward messages through intermediate peers. This section discusses the protocol design, correctness, and the performance analysis in terms of messaging and space requirements.

Consider a lazy messaging layer underlying the authentication protocol discussed earlier. This layer maintains a cache of the undelivered messages and provides eventual consistency in the following sense: The outcome of the lazy protocol will approximate the outcome of the eager protocols described earlier. Thus, before the two protocols achieve the same assignment of public keys to peers, we shall be in the state of *optimistically trusting* the authenticity of public keys. If the optimistic trust is broken, we mark the offending peer untrusted as required by the authentication protocol.

Public key infection does not require knowledge of physical time. However, each peer maintains a numeric logical timestamp, **lts**, called the Lamport timestamp. It is maintained by setting it to the maximum of the local timestamp and incoming message timestamp [13]. This timestamp provides a partial order on all events in the distributed system. Each peer also maintains a causal timestamp, **cts**, which is simply a local event counter incremented on send and receive events. This timestamp is sent with each outgoing message allowing peers to maintain a timestamp vector **ctv** of causal timestamps.

Table 4
Data Structures at the Key Infection layer.

<code>lts</code>	The Lamport timestamp.
<code>ctv[i]</code>	The last causal timestamp known for peer i .
<code>ltv[i]</code>	The Lamport time of the most recent message originating from peer i .
<code>srv[i]</code>	The stable read timestamp. Its value is the largest (known) Lamport time such that all messages with a smaller timestamp have been received at peer i .

Key infection works by deferring the transmission of messages sent by the authentication protocol. The following steps are taken when a protocol message m with source s and destination d is pushed to the key infection layer: Firstly the causal timestamp `ctv[s]`, and the Lamport timestamp `lts`, are incremented to record the change of state (Table 4). Secondly, the record $\{1, \text{ctv}[s], s, d, m\}$ is inserted into the message cache (Table 3). It holds the outgoing message and timestamps to achieve eventual delivery.

Anti-entropy sessions

Anti-entropy sessions between peers transfer the protocol messages in an epidemic manner. The timing of anti-entropy sessions can either be application dependent to take advantage of piggy-backing on application messages or could depend on a timeout to prevent too much divergence from the eager protocol execution. The following steps define a session between the peers s and d .

1. Exchange time stamps
The Lamport timestamps are exchanged and used to update the Lamport timestamp vector `ltv`. The local Lamport time is also maintained using the usual algorithm. The causal timestamp vectors are compared to decide which messages should be sent to the other peer.
2. Send cached messages
For any locally cached record r , if the fol-

lowing holds:

$$r.\text{ct} > \text{ctv}_d[r.\text{src}]$$

It can be inferred that d has not seen the message stored in r . The causal timestamp is later than the last causal timestamp known to d . Thus, all records satisfying the *message exchange condition* have to be transmitted to the peer d to enable eventual delivery at the destination.

3. Receive cached messages
The peer d computes a set of records to be transmitted by the same logic. On receiving a record r , the receiver inserts r in the cache and updates the component `ltv[r.src]`, if the following condition holds $\text{ltv}[r.\text{src}] < r.\text{lt}$. Thus the Lamport time vector reflects the latest known Lamport time for each peer.
4. Exchange stable read time stamp
The stable read time stamp is computed as $\min_i(\text{ltv}[i])$. The peer s assigns the value to `srv[s]`. The updated vector `srv` is now transmitted to d .
5. Delete received messages
The exchange of stable read timestamp advances its components in the usual manner. Consider a cached message r such that:

$$r.\text{lt} < \text{srv}[r.\text{dest}]$$

Clearly since the value `srv[r.dest]` is computed at $r.\text{dest}$ as minimum of the Lamport timestamps received from the peers, this implies a message with greater Lamport time was received from the message source as well. By assumption of ordered message delivery, the older timestamped message has already been received. Thus, the peer can delete the records that satisfy the *deletion condition*.

Encrypted timestamps

The epidemic algorithm operates in a semi-trusted environment. Thus, it is necessary to ensure the integrity of timestamps. We create

encrypted timestamps by requiring the peer i to generate the pair $\{t, K_i^{-1}(t)\}$ instead of the timestamp component t . Thus, the protocol processing outlined above would always pass timestamp values as pairs. The receivers would be required to verify the correctness before acting on a timestamp value. By the assumption of non-invertibility, the secure timestamp can be generated only by the peer i . Thus, for an honest peer i , it is impossible to forge the timestamp component representing the state at i . Since the authentication protocol requires correct operation only from the honest peers, secure timestamps are sufficient to preserve the correctness of authentication protocol.

5.1. Complexity and coverage

Let the peers do an anti-entropy session with a randomly chosen peer every unit time. Consider a message transmitted at the first round of exchanges. If $|\mathcal{T}| = n$, then we know the fraction f of initially uninfected peers is $\frac{n-1}{n}$. Now at round i , if f_i is the fraction of uninfected peers, only f_i^2 peers remain uninfected with the update at round $i + 1$. Thus, on the average:

$$\begin{aligned} f_{i+1} &= f_i^2 \\ &= f^{2^i} \end{aligned}$$

Thus the number of uninfected peers drops doubly exponentially with time. Since the number of exchanges initiated by a peer is one per unit time, the number of messages sent and received by a peer is in $\mathbf{O}(1 + \frac{1}{n})$.

5.2. Size of the message cache

Let μ be the rate at which messages are being created at the authentication protocol layer. Thus the cache gets μ new messages from the higher protocol every unit time. Now let us consider the situation at round i with respect to the messages created during the first round. Since the fraction of uninfected peers is same as the probability P_d of the destination being uninfected, we have:

$$P_d = f^{2^i}$$

Now assume that the destination gets infected at round i^* . Again by anti-entropy propagation of its stable read timestamp, we have the probability P that a peer is infected with the message but not with the stable read timestamp of the destination:

$$P = (1 - f^{2^i})f^{2^{i-i^*}}$$

Since infection with the update but not with the timestamp ensures that the message is not deleted, the expected fraction of cached messages is $(1 - f^{2^i})f^{2^{i-i^*}}$. Again the cached messages can be created at any previous exchange round. Thus we have the summation for log size N :

$$N = \sum_{i=1}^{\infty} \sum_{i^*=1}^i n\mu(1 - f^{2^i})f^{2^{i-i^*}} \quad (1)$$

Relating the series to the integral $\int e^{e^x} dx$, which is evaluated using integration by parts, we have the following relation on log size:

$$N < \mu \frac{e \log e}{2} n \log\left(\frac{n+1}{n}\right)$$

We know that $\log(\frac{n+1}{n})$ is $\mathbf{O}(\frac{\log n}{n})$. Also, the rate of message insertion is $\mathbf{O}(n)$ because messages are sent to all members of the trusted group. Hence the number of cached messages is in $\mathbf{O}(n \log n)$.

6. Application

We have implemented the Byzantine fault tolerant authentication protocol as a standalone library to make it available to a variety of applications. Our first application target is an electronic mail authentication system implemented through a self authenticating mail (SAM) client.

Electronic mail is one of the most popular applications on the Internet. Although it has gained wide acceptance both for business and personal use, its usage is limited by the lack of security in the mail transport protocol. Unlike conventional mail that can be signed by hand, electronic mail does not come with any inbuilt authentication mechanism. Thus, it is possible to forge sender identity and modify contents en-route. To be at par with paper documents, email must provide these minimal authentication capabilities in

a convenient and unobtrusive manner. As the email client is in first hand contact with mail users, we need to address manual overrides over protocol decisions. Thus, the system will allow a manual override for all trustworthiness decisions. It can be observed that in absence of the autonomous authentication protocol, such a system would be equivalent to PGP in terms of the trust model.

Compatibility with existing email infrastructure is part of the design goals. We use the SMTP extensions to enforce backward compatibility. The extension fields will carry protocol messages as part of the mail header. Since the clients that cannot interpret the extended field will ignore it, the protocol data is an overhead for clients that cannot use it. We eliminate the overhead of sending the extension fields to non-SAM clients in the following manner. Identification of SAM end-points is enabled by requiring each application client to send messages with the extension field. If there are no outstanding protocol messages, then the sender sends an empty field for the extension. Thus all SAM enabled addresses are identifiable on receipt of an email message.

A cache of recent deductions like the authenticated public key and the authentication protocol capability of a peer are kept in the SAM client. The act of receiving an email message makes a SAM client aware of the capabilities at the sender. As the end result of protocol execution is the authentication of the public keys at mail addresses, the data associated with an end point includes the known public key and the number of group migrations since it was first authenticated. This information helps to assign a *confidence* value to the authentication. We expect to gain real life experience on the utility of Byzantine fault tolerant authentication by implementing self authenticating mail.

7. Simulation

We devised a simulation system to investigate authentication cost in various scenarios. The simulation system is a stripped down version of the authentication module implemented in the li-

brary. All the message transfers are replaced by function calls that update counters to simulate system activity. The simulation is designed to replicate the authentication protocol in a universe of peers. The trust relationships between the peers can be preset during bootstrapping. Application level message exchange can be simulated to trigger authentication of unknown peers. The simulation consists of about 1500 lines of C++ code and uses the libcrypto pseudo random number generator to make application level choices.

The scenarios of interest are bootstrapping and the authentication of new peers on an ongoing basis. The flow of time is uniformly measured in epochs with each epoch consisting of the time needed for one cryptographic operation and one message transfer. This way of measuring time allows extrapolation of the simulation results to systems having various tradeoffs of processing power and network latency.⁸ It also allows us to state all simulation results in terms of number of messages exchanged by a peer.

7.1. Bootstrapping Cost

The bootstrapping procedure requires the trusted peers to authenticate each other as if they were mutually probationary. This process was simulated with respect to the bootstrapping set size, the topology of the trust graph connecting the bootstrapping peers, and the proportion of malicious peers in the universe. We simulated the bootstrapping process on a universe of 1000 peers with trusted group sizes from 6 to 56.

We chose different types of trusted groups as shown in Figure 8. Bidirectional trust was used to create clusters of mutually trusting peers. This is realistic in a geographically local or in a small worlds type of trusted environments. These clusters of bidirectional trust were perturbed by randomly selecting some of the members from the universe. This makes the trust unidirectional and imposes additional cost on the peers. The cost increases because it becomes less likely that the trusted peer can return a cached proofs of possession.

⁸The extrapolation should assign zero message transfer time and the measured cryptographic operation time to calculate the cost of Public key infection.

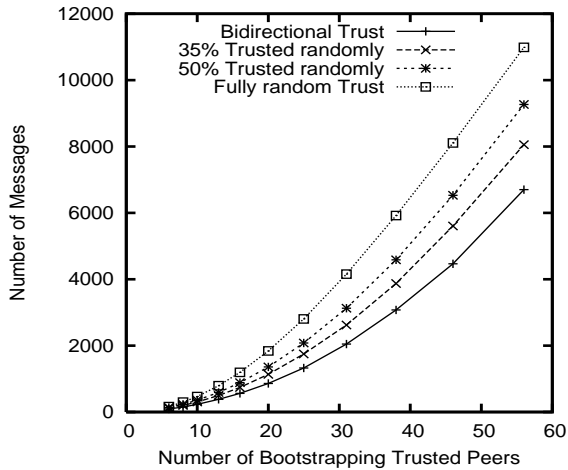


Figure 8. Average Bootstrapping cost per Peer.

The effect of malicious peers was studied for various group sizes. The malicious peers were randomly distributed in the universe. Their actions were not provably malicious on challenge response leading to the execution of Byzantine agreement. The trusted groups were created with 15% random selection. The results shown in Figure 9 indicate that the incurred cost increases rapidly with increasing group size and increasing proportion of malicious peers. This is expected because each malicious peer forces an overhead of $O(|\mathcal{T}|^2)$ message transfers on each of its peers.

7.2. Authentication Cost

We investigated the incremental cost incurred by group members and the probationary peers for various group sizes. This simulation was conducted on a universe of 2000 peers with statistics collected in 1000 runs of 1000 application messages each. The application messages were targeted to an unauthenticated peer with probability 0.1. As shown in Figure 10, the cost of authenticating a new peer is independent of group size for the trusted peers. The cost increases linearly with group size for probationary members. It is slightly cheaper to authenticate peers in groups

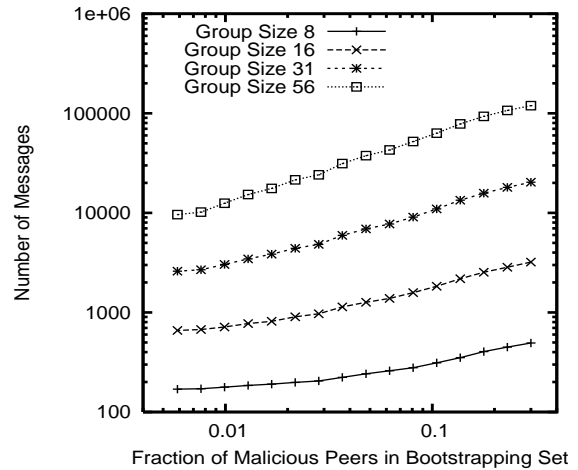


Figure 9. Effect of Malicious Peers on the Average Bootstrapping Cost.

with randomized selection because some of the challenge response steps are avoided if the probationary peer is already trusted by some group members. The role of malicious peers is investigated in Figure 11. It shows a rapid increase in the authentication cost in trusted groups with increasing proportion of malicious peers. The effect of malicious peers is greater on larger groups as expected by the quadratic messaging cost of byzantine agreement phase of the authentication protocol.

8. Discussion

Our model supports an *incremental growth of trust*. Optimistic authentication allows the trust to increase by the successful authentication of a public key through many group migrations. Since most of the peers are in honest groups, it becomes increasingly unlikely that a long sequence of dishonest groups is selected. Thus, our protocols provide *soft authentication*, which contrasts them from the traditional notion of authentication.

The traditional model with its all-or-none approach provides stronger authentication with

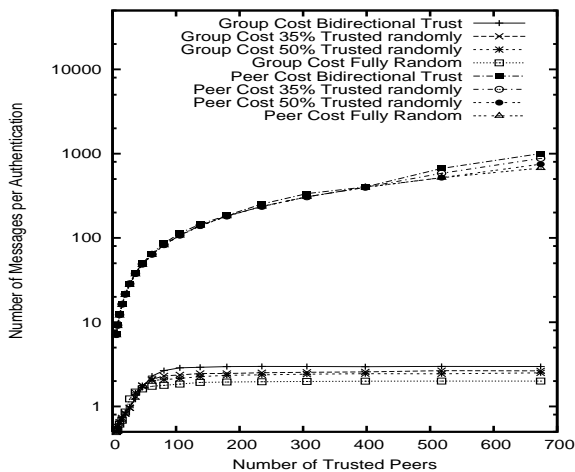


Figure 10. Cost of Authentication.

weaker fault tolerance. We trade off the authentication strength and use stronger network assumptions to provide an autonomous and fault tolerant authentication mechanism. It can be argued that the public key infrastructure approach forces the system designers to take a boolean approach to security. This has allowed most of the Internet traffic to remain insecure even though the computational power and software engineering needed to secure it are available. Byzantine fault tolerant authentication is useful because it allows the co-operative formation of self authenticating systems.

The authentication mechanism has not addressed denial of service type of attacks. A possible solution for this problem would require keeping track of the cost incurred on behalf of other peers and making peers untrusted if they launch such attacks.

9. Related Work

Alternative approaches to provide fault tolerant authentication are known. There is a class of protocols relying on threshold cryptography that uses key shares with the following property: with-

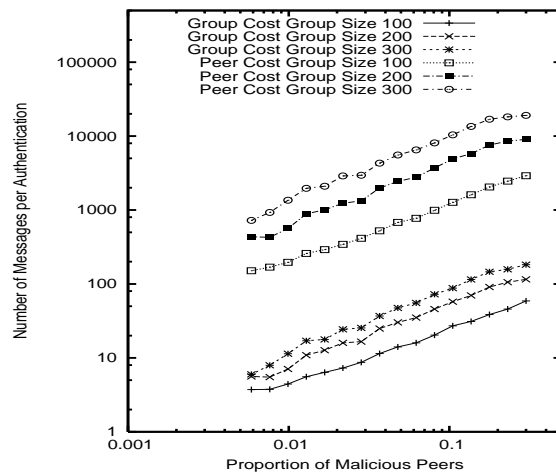


Figure 11. Authentication with Malicious Peers.

out a quorum of participants, it is impossible to create a digitally signed public key certificate [14]. As a consequence, unless the number of malicious parties is as large as the quorum, false authentication is impossible. Threshold cryptography requires the existence of a trusted dealer that initializes the key shares. In this way, it depends on the honesty of the dealer. Threshold cryptography has been used in COCA, a fault tolerant public key authentication service [15] and as the basis of a number of other secure services [16–18].

Threshold cryptography is also used to implement proactive recovery. To compromise such a system, the adversary is required to compromise the quorum within its vulnerability window or lose any previous progress due to a re-randomization of key shares [19]. Although better than static key shares, the scheme cannot recover from the compromise of a quorum because the same long term shared secret is recycled among the trusted parties. In contrast, distributed authentication is proactively secure in the sense that it holds no long term secrets.

Role based access control in a distributed system has been studied earlier [20]. It uses roles instead of identities for granting access and

therefore avoids the issue of identity authentication [21]. Reputation based distributed trust has been investigated by a number of previous works. The Free Haven project uses a proactive mechanism based on recommendations to protect anonymity of the users [22]. NICE allows the creation of trustworthy peer groups through a trust evaluation mechanism based on reputation [23]. Both systems aggressively eliminate (overtly) malicious parties to preserve their correctness. Byzantine fault tolerant authentication builds upon this idea of peer reputation. It does not require external identity authentication and works on provable observations rather than recommendations.

PGP has applied decentralized trust to authentication in distributed systems [7,8]. However it requires human evaluation of trustworthiness that limits its applicability for unsophisticated users and autonomous systems [9]. Trusted ambient communities [24] is an approach that incrementally builds trusted groups by observing the behavior of securely initialized peers. It is more permissive of malicious peers than our authentication mechanism that needs verifiable challenge response proofs. Cryptographic identifiers [25] provide authenticated identities by selecting network identity related to the public key. This mechanism does not handle the man in the middle attack. This approach is simple but constrained by the need to acquire specific network identifiers. A randomized approach to setting up weakly secure peer-to-peer networks is used in Smart Dust [26]. Although it does not focus on authentication, it shares distributed trust and a weakened adversary model with Byzantine fault tolerant authentication.

10. Conclusion and Future Work

Byzantine fault tolerant distributed authentication provides a new approach to tackle the authentication problems of distributed and peer to peer systems. The salient features are lack of total trust and single points of failure. Our approach allows a natural growth of trust without requiring trustworthy hierarchies of delegating and recommending parties as done in other

trust management systems [27,28]. Our approach is made feasible by the weakening the *network is the adversary* model.

The rise of peer-to-peer systems on the Internet, and the popularity of wireless communication are the prime motivations behind the change of the underlying model. Although weaker than the traditional model in terms of adversary power, it is stronger in terms of fault tolerance.

Byzantine fault tolerant authentication was implemented as a library which will be used to build a secure e-mail system. The system will support a set of e-mail clients that will authenticate each other through an underlying distributed trust mechanism.

11. Acknowledgments

This work is supported in part by the National Science Foundation under CCR-0133366 and ANI-0121416. The authors thank Dr. Marios D. Dikaiakos for his constructive feedback and suggestions. The authors are thankful to the anonymous reviewers for their thoughtful comments that helped improve the quality of this paper.

REFERENCES

1. W. Diffie, M. Hellman, New Directions in Cryptography, IEEE Trans. Info. Theory 22 (1976) 644–654.
2. A. S. R. Rivest, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.
3. CCITT, The Directory Authentication Framework, Recommendation X.509 (1988).
4. M. Naor, K. Nissim, Certificate Revocation and Certificate Update, in: Proceedings 7th USENIX Security Symposium (San Antonio, Texas), 1998.
5. B. Fox, B. LaMacchia, Certificate Revocation: Mechanics and Meaning, in: R. Hirschfeld (Ed.), FC'98: International Conference on Financial Cryptography, Vol. 1465 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 158–164.
6. W. Aiello, S. Lodha, R. Ostrovsky, Fast digital identity revocation, in: Advances in Cryptology - CRYPTO '98, 18th Annual International Crypt-

- tology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings, Vol. 1462 of Lecture Notes in Computer Science, Springer, 1998, pp. 137–152.
7. P. Zimmermann, *The Official PGP User's Guide*, MIT Press, Cambridge, Massachusetts, 1995.
 8. S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, Inc., Cambridge, MA, 1995.
 9. A. Whitten, J. D. Tygar, Why Johnny can't encrypt: A usability evaluation of PGP 5.0, in: Proceedings of the 8th USENIX Security Symposium, 1999.
 10. L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4 (3) (1982) 382–401.
 11. J. Douceur, The sybil attack, in: Proc. of the IPTPS02 Workshop, Cambridge, 2002.
 12. P. Syverson, I. Cervesato, The Logic of Authentication Protocols, *Lecture Notes in Computer Science* 2171.
 13. L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Commun. ACM* 21 (7) (1978) 558–565.
 14. S. Goldwasser, S. Micali, R. L. Rivest, A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks, *SIAM J. Comput.* 17 (2) (1988) 281–308.
 15. L. Zhou, F. B. Schneider, R. van Renesse, COCA: A Secure Distributed On-line Certification Authority, Tech. Rep. 2000-1828, Department of Computer Science, Cornell University, Ithaca, NY USA (December 2000).
 16. C. Cachin, Distributing trust on the Internet, in: International Conference on Dependable Systems and Networks (DSN2001), Gteborg, Sweden., IEEE, 2001.
 17. M. K. Reiter, The Rampart Toolkit for Building High-Integrity Services, in: Dagstuhl Seminar on Distributed Systems, 1994, pp. 99–110.
 18. L. Zhou, Z. Haas, Securing Ad Hoc Networks, *IEEE Network Magazine* 13 (6).
 19. R. Canetti, R. Gennaro, A. Herzberg, D. Naor, Proactive Security: Long-term protection against break-ins, *RSA CryptoBytes* 3 (1) (1997) 1–8.
 20. J. Bacon, K. Moody, W. Yao, Access Control and Trust in the Use of Widely Distributed Services, *Lecture Notes in Computer Science* 2218 (2001) 295+.
 21. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, Role-based access control models, *IEEE Computer* 29 (2) (1996) 38–47.
 22. R. Dingledine, M. J. Freedman, D. Molnar, The free haven project: Distributed anonymous storage service, *Lecture Notes in Computer Science* 2009.
 23. S. Lee, R. Sherwood, S. Bhattacharjee, Cooperative Peer Groups in NICE, in: INFOCOM, 2003.
 24. S. U. V. Legrand, D. Hooshmand, Trusted Ambient community for self-securing hybrid networks, *INRIA Research Report*, 5027 (2003).
 25. G. Montenegro, C. Castelluccia, Crypto-based identifiers (cbids): Concepts and applications, *ACM Trans. Inf. Syst. Secur.* 7 (1) (2004) 97–127.
 26. R. Anderson, H. Chan, A. Perrig, Key infection: Smart trust for smart dust, in: Proceedings of IEEE International Conference on Network Protocols (ICNP 2004), 2004.
 27. M. Blaze, J. Ioannidis, A. D. Keromytis, Trust Management and Network Layer Security Protocols, in: Cambridge Security Protocols International Workshop, 1999, pp. 103–118.
 28. R. Yahalom, B. Klein, T. Beth, Trust relationships in secure systems—a distributed authentication perspective, in: In Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy, 1993, pp. 150–164.