# Byzantine Fault Tolerant Authentication

Vivek Pathak
*Department of Computer Science*
*Rutgers University,*
*Piscataway, NJ 08854*
*vpathak@paul.rutgers.edu*

Liviu Iftode
*Department of Computer Science*
*University of Maryland,*
*College Park, MD 20742*
*iftode@cs.umd.edu*

## Abstract

*We describe an autonomous Byzantine fault tolerant public key authentication architecture. It aims to satisfy the authentication requirements of large distributed systems consisting of semi-trusted peers. The distributed trust model does not demand the existence of predefined trusted parties and provides authentication if more than a threshold of the peers are honest.*

*A voting based public key authentication protocol implements distributed trust and supports dynamic membership over insecure asynchronous networks. This paper describes the authentication mechanism and discusses its applicability to peer-to-peer systems and the Internet. It also demonstrates the Byzantine fault tolerance of authentication protocol, and the eventual correctness of authentication in dynamic trusted groups. A lazy public key infection protocol is developed for efficient authentication.*

## 1 Introduction

As the usage of Internet is evolving from web browsing to electronic business transactions, the need for securing networked systems and data is becoming increasingly important. The goals include privacy of data and authenticity of its sources. Cryptographic techniques like encryption [17, 6, 14] and digital signatures [16] satisfy these requirements. Key authentication is the problem of securely mapping entities to their public keys. It is a fundamental problem because the security of encryption and digital signatures depends on the availability of authentic public keys. The prevalent model of key authentication uses trusted third parties to create digitally signed public key certificates. This architecture, called the Public key infrastructure, was recommended in the Directory authentication framework of CCITT [4]. While the idea of public key infrastructure has been around for more than a decade, its application has been limited to a small fringe of Internet traffic. In the absence of usage statistics, the continuing incidence of network security failures [5] is ample evidence of its limited impact.

Although the limited usage may partially be attributed to avoidable causes like lack of availability and high cost, the limitations of the underlying trust model are the major impediments to the ubiquitous usage of secure communication. Public key infrastructures impose a centralized hierarchical trust model on the users. The trusted third party is trusted by every party in the system. We believe that while centralized trust models are appropriate in a client-server setting, they are unsuitable for an increasingly important class of distributed systems. Trusted third parties may be unacceptable to the users of peer-to-peer systems. Similarly, in very large distributed systems or in highly heterogeneous systems, it might be impossible to find a party trusted by all the communicating parties.

The trusted third party model is not applicable to a Byzantine environment. A compromise of certificate authorities allows the creation of fake certificates that could endanger the privacy of trusters. In this manner, the dependence on a certificate authority limits the safety of secured networks. The trusted third party becomes a single point of security failure, and thus a natural target for attacks.

To safeguard against compromised private keys, public key infrastructures have to support certificate revocation [13, 8]. 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, we believe that it may not be possible to scale up public key infrastructures for ubiquitous security in large systems like the Internet.

We propose an *entity authentication* mechanism as a peer-to-peer alternative to the traditional public key infrastructure approach. It involves a distributed system of mutually authenticating semi-trusted parties. The protocols tolerate Byzantine faults and provide *eventually correct authentication* if no more than a

threshold of the parties are malicious or faulty. The protocols allow mutual authentication of participants that do not share a common trusted party. This property enables secure communication in highly heterogeneous groups and allows the light-weight creation of short lived trust bindings between peers that communicate infrequently. As a consequence, distributed authentication is well suited for mobile networks and peer-to-peer applications. In summary, this paper makes the following contributions:

- We propose a mechanism for public key authentication without trusted third parties. The authentication mechanism is autonomous to make it useful without human interaction.
- We provide proactive security and tolerate Byzantine faults.
- We design a secure anti-entropy public key infection algorithm for light weight authentication.

# 2 Model

Consider a distributed system of mutually semi-trusting parties. None of the parties trust any party, but each believes that the honest parties are in a majority.

Traditionally, the verification of a public key certificate makes a statement of the form: If the trusted third party is honest and capable, then the private key corresponding to the certified public key belongs to the given identity. This concept of identity is very general, it may span individuals and organizations. In contrast, we authenticate network end-points or peers. Our authentication is a statement of the form: If there is an honest majority, then the authenticated peer possesses the private key corresponding to the public key.

## 2.1 Assumptions

Entity authentication is discussed under the following assumptions:

**Eventual delivery**

The parties 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. Given sufficient number of retransmissions, every message is delivered eventually.

**Failure notification**

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.

**Disjoint transmission paths**

Without considering the details of network topology, we assume a sufficient number of disjoint message transmission paths exist to each peer. Therefore, the *man in the middle* attack can not be mounted for more than a small fraction $\phi$ of its peers. To unify the treatment of this attack with the byzantine kind of faults, we consider the peers that do not satisfy this assumption to be faulty. The faulty end points are not authenticated by our protocol as shown in Figure 1.

**Honest majority**

The system consists of honest and dishonest parties. An honest party always tells the truth about its identity and public key. Dishonest parties may behave in an arbitrary fashion, either because of being faulty or because of being malicious adversaries. Honest and non-faulty parties are not distinguished. We define honest parties and honest majority as follows:

**DEFINITION 1** *An honest party protects the privacy of its private key and executes the authentication protocol correctly.*

**DEFINITION 2** *A set of $n$ parties has honest majority if the number of malicious or faulty parties $t < \frac{1-6\phi}{3}n$.*

It is assumed that the system of mutually authenticating parties has honest majority.

## 2.2 Adversaries

The computational power of adversaries is polynomially bounded. We assume that public key cryptography and digital signatures are polynomially secure. Hence with very high probability, the adversary cannot forge digital signatures or invert encryption transformations.

We consider active and passive adversaries. The passive adversary has unlimited power to eavesdrop on any message. While the active adversaries have

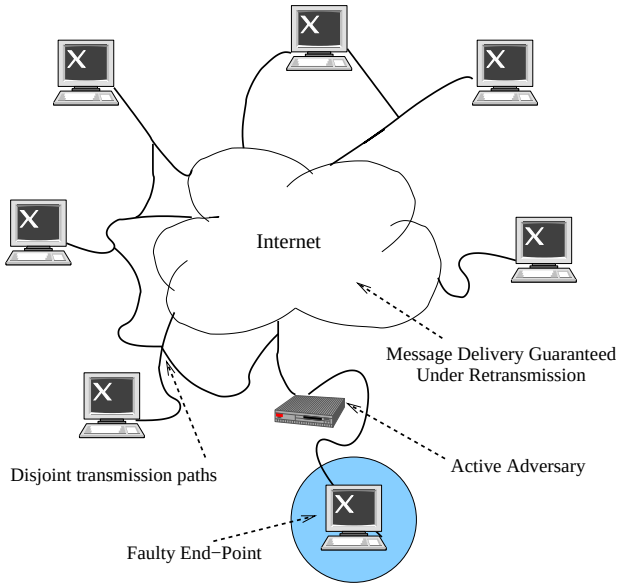


Figure 1: Adversaries and assumptions.

unlimited power[1] to inject arbitrary messages into the network, they cannot prevent message delivery for more that a small fraction $\phi$ of the honest parties.

Clearly this is weaker than the classical *"network is the adversary"* approach. The adversary network may not only inject arbitrary messages, but also prevent delivery of any message. The weakened 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 mounting a secret active attack on a large number of end-points. This is partly due to the large address space, and partly due to the protection of Internet infrastructure by a number of independent service providers. Moreover, practical experience with Internet based systems suggests that message injection or spoofing is the preferred form of attack. Prevention of message delivery is hardly ever encountered.

## 2.3 Authentication

Public keys are authenticated by a voting based protocol that involves a set of peers with honest majority. The objective is to authenticate the public key of one honest peer to another. In the absence of man in the middle attack, it is known that a challenge response protocol can authenticate public keys. 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 as shown in Figure 2.

The authentication protocol consists of three phases: Challenge response, Distributed authentication and Byzantine agreement. The challenge response phase consists of a set of peers challenging the party with an encrypted nonce. Since the nonce can be recovered only by the possessor of the corresponding private key, a correct response is a proof of possession. At the end of this phase, each peer gets a proof of possession.

In the distributed authentication phase, peers forward their proofs to the requesting 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 peers are malicious or faulty. It proceeds to the Byzantine agreement phase that discovers malicious parties by comparing the sent and received claims of different parties. Because all messages are digitally signed, malicious behavior can be discovered by this procedure.

## 2.4 Trusted Groups

Each peer has a probationary group, trusted group and untrusted group of peers as shown in Figure 3. Peers notify each other of their public keys depending on their secure communication needs. They are added to the probationary group when a new public key or identity is known. Successful authentication moves a peer from the probationary group to the trusted group. Because frequently communicating parties request admission into each others trusted groups, they are mutually trusting in the common case.

Peers that are known to be malicious are deleted from trusted or probationary groups, and added to the untrusted group. Honest peers do not communicate with the untrusted members because membership in the untrusted group implies the observation of provably malicious behavior. Continuous addition of malicious peers to the untrusted group can cause it to grow indefinitely. Therefore, peers forget malicious behavior of the distant past.

An honest peer deletes the trusted peers that fail to prove their liveness by responding to protocol messages. Since this is not a provably malicious action, they are not added to the untrusted group. Peers are also deleted from a trusted group by a periodic pruning procedure that limits the authentication cost.

[1]Since 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.

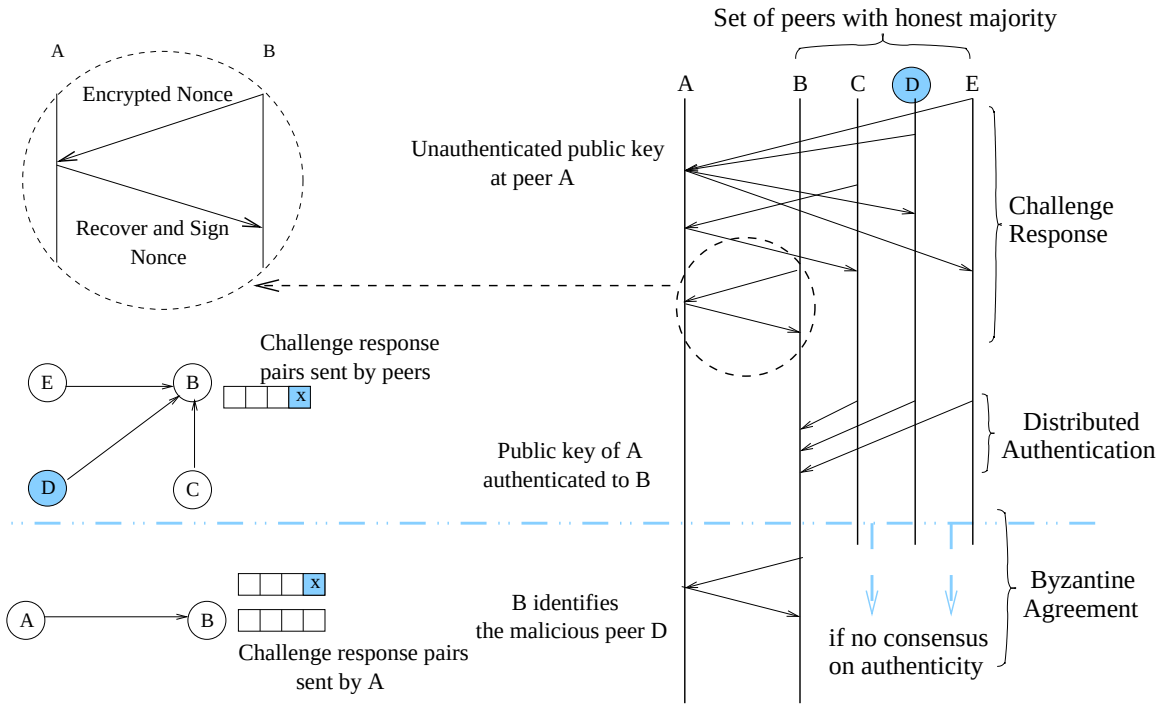


Figure 2: Authentication protocol overview. A peer $A$ is authenticated by $B$ using its trusted peers. $D$ is a malicious peer that tries to prevent authentication of $A$.

Consider a party $B$ with a trusted group $\mathcal{T}(B)$. Because any party $A$ that joins its trusted group also joins the probationary group of each peer in $\mathcal{T}(B)$, it follows that if $A$ is honest it will eventually belong to the trusted group of every peer trusted by $B$. Thus the trusted group definition is eventually consistent among a set of trusted peers. However, the asynchronous addition of parties makes the group definition inconsistent. In order to limit the inconsistency, the trusted peers frequently exchange their respective proofs of possession.

| | |
|---|---|
| $K_i$ | 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 party $i$. |
| $\{x, y, z\}$ | A message containing three strings $x$, $y$ and $z$. |
| $\{x\}_i$ | A message signed by $i$. |
| $\mathcal{T}(i)$ | Trusted group of party $i$. |

Table 1: Notation

# 3 Architecture

Peers implement the Authentication and Membership control protocols. A bootstrapping procedure is provided for system initialization. All protocol messages have timestamps, source and destination identifiers, and a digital signature as shown in Table 1. Honest recipients ignore messages with invalid signatures. They maintain a most recent received timestamp vector, and ignore stale messages to guard against replay.

## 3.1 Authentication Protocol

The authentication protocol allows authentication of peers. A peer $A$ can be authenticated by the peer $B$ with help of its trusted group $\mathcal{T}(B)$ as shown in Figure 4. The different phases of the protocol are outlined below:

- **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.
- **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. The peer $A$ can recover the nonce only if it holds the private key

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

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

   For each trusted peer $P_i$ of $B$

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

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 agreement.

   At each trusted peer $P_i$ of $B$

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

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

3. **Distributed authentication**
   Each peer returns the proof-of-posession $\{\mathcal{C}_{iA}, \mathcal{R}_{iA}\}$ 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.

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

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

   If $A$ is not proved malicious

   For each trusted peer $P_i$ of $B$

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

   For each trusted peer $P_j$ of $P_i$

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

Figure 4: Authentication Protocol

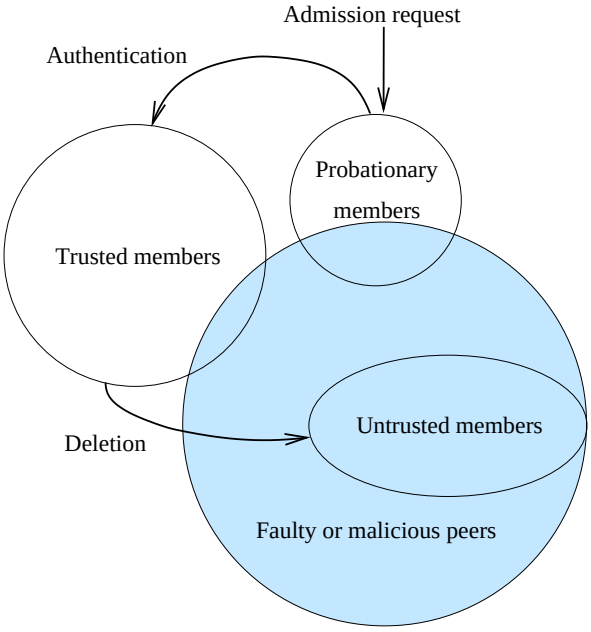


Figure 3: Group structure

$K_A^{-1}$. It returns the nonce by sending a signed response message. The challenge response 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 $K_A$. Note that since $K_A$ is not yet authenticated, digital signature is not verified on the response message. However, the challenger waits for an application specific time-out to ensure that duplicate responses and failure notifications are detected. It deletes the proof if any of them are received.

- **Distributed authentication**
  The peers respond to $B$'s authentication request by sending their proofs of possession to $B$. If all parties 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, $B$ 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 to authenticate its public key to the group members. Bootstrapping initializes the distributed authentication system by creating a trusted group consisting of the bootstrapped peers. Because correct authentication requires trusted groups to have honest majority, the initial trusted set should have at least three members to safely recover from the entry of a malicious peer.

The bootstrapping process initializes an initial trusted group $\mathcal{T}_0$ along with the respective public keys, on each peer in the group. Peers authenticate each other by requesting admission into this trusted group.

## 3.3 Membership Control Protocol

Peers control the membership of their trusted groups. While new peers are added to the trusted group on being authenticated, honest majority is preserved by deleting the malicious peers. Continuous additions and deletions cause inconsistency among the group definitions present at different peers of the trusted group. Membership control protocol limits the inconsistency in group definitions as shown in Figure 5.

### Addition to trusted groups

Each peer maintains a list of `to be sent` authentication proofs for each probationary peer. It lazily pushes these proofs so that its trusted peers may also authenticate the peer. Hence every peer gets the proofs of possession, and adds the newly authenticated peer to its trusted group.

Every peer may not be aware that its proof is expected, or a lazy timeout might delay a required authentication. Hence the peers can pull the proofs of possession. This is done by keeping a list of `to be received` authentication proofs for each probationary peer. Peers pull the proofs by sending `authentication request` messages. A correct peer responds with the required proof.

1. **Push proofs**
   A peer $D$ periodically pushes the proof-of-posession $\{\mathcal{C}_{DA}, \mathcal{R}_{DA}\}$ 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 party $A$.

   For each trusted peer $P_j$ that has not sent a proof
   $B \rightarrow P_j \quad : \quad \{B, P_j, \texttt{authentication request}, (A, K_A)\}_B$
   $P_j \rightarrow B \quad : \quad \{\mathcal{C}_{jA}, \mathcal{R}_{jA}\}_{P_j}$

Figure 5: Membership Control Protocol

**Deletion from trusted groups**

Peers are deleted from trusted groups for malicious activity or lack of liveness. If a peer sends a malicious message to any honest peer $B$, it is marked untrusted by each trusted peer of $B$. The 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 [11] on the malicious 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 group. It is possible that honest parties may be deleted from dishonest groups. This causes them to join other groups, most of which have honest majority.

A peer that fails to respond to messages in a timely manner is considered failed due to lack of liveness. Using suitable application specific values for response timeout and retry frequency, honest peers delete the non-live peers from their trusted groups.

Honest parties may voluntarily delete themselves from trusted groups. This is required because authentication performance depends on group size. The voluntary deletion is implemented by randomly selecting a trusted peer and ceasing to respond to its messages. This causes deletion by lack of liveness. The probability of deletion is chosen as a function of trusted group size to allow the creation of suitably sized trusted groups.

**Group migration**

Authentication depends on the existence of honest groups. Trusted group membership is granted on authentication which does not guarantee that the admitted peer will not act maliciously at a later time. In particular, it is possible that a number of covertly malicious parties join a trusted group so that they are in majority. Having violated honest majority, they can now mis-authenticate the honest peers. Honest peers periodically flush their trusted groups to prevent this kind of scenario.

# 4 Analysis

This section discusses the correctness of the authentication. It is shown that honest parties are authenticated correctly in honest majority groups. It is also shown that the group dynamics resulting from membership control create honest majority groups with high probability.

## 4.1 Proofs of Possession

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

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

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

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. Then it follows that:

**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.*

| | 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 |

Table 2: Effect of attacks on proofs of possession.

**PROOF:** By contradiction, let $P_i$ be honest, and let $K_A$ be inauthentic. Since $P_i$ is honest, it transmits a correct challenge containing $c_i = 3D_A(K, r_i)$ to $A$, and does not disclose its nonce $r_i$ to any party.

If the message is not delivered to $A$, the network must provide a notification causing the proof to be deleted. Otherwise $A$ must respond.

Thus if a single response is received, $A$ must be the responder. Since it computes $r_i = 3D_A^{-1}(K, c_i)$, it knows the private key, a contradiction. □

### Attacks

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 party $X'$ to impersonate $X$, and protocol attacks could be launched by a party $X$ 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 property signed; $K$-invalid if $c_i \neq 3D_A(K, r_i)$ and faulty if it is valid but $K_A$ is not owned by $A$. Because the trusted peers have each other's authenticated public keys, they can not be targeted by a man in the middle attack. This leads to the summarization of attacks on proofs of possession as given in Table 2. Further details are present in Appendix A.

## 4.2 Distributed Authentication

Distributed authentication ends with a consensus of valid or $K$-invalid proofs if every participant is honest and there are no spoofing or impersonation attacks. Given the presence of attacks and malicious peers:

**DEFINITION 3** *The authentication protocol is correct if given honest majority:*

1. *The peer $A$ is not mis-authenticated if it is not malicious or faulty.*
2. *The group membership of honest peers is unaffected.*
3. *$A$ is eventually authenticated if it is honest.*

**CLAIM 2** *Authentication protocol is correct.*

**PROOF:** Considering each of the properties:

1. 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, honest peers will receive `authentication request` messages and send challenges to $A$. Because $A$ responds by decrypting according to its private key, the proofs sent by honest peers will be $K$-invalid. If $B$ does not send the correct messages, honest peers will send no proofs. 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.

2. Consider two cases: $B$ is honest or it is malicious or faulty. If it 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 values 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. Every peer sends its proof to every other peer. Because honest peers delete those peers that gave different proofs to different peers, only malicious or faulty peers are deleted. Faulty messages do not prove maliciousness and hence do not cause deletion.

   Malicious peers could send conflicting proofs of possession to their peers. These actions are detected by byzantine agreement as follows: Consider an honest peer $P_i$ receiving the `byzantine agreement` message from other peers. Firstly, $t$ of the peers are malicious and may offer arbitrary proofs. Secondly, $\phi n$ of the peers may not be able to reach $A$, and may have faulty proofs to offer. Finally, proofs from another $\phi n$ peers may be faulty because the peer $P_i$ 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 parties 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.

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

Similarly, if $B$ maliciously sends conflicting requests to the peers, the signed `authentication request` messages will cause it to be detected by Byzantine agreement on the requests received.

3. Consider the first case. Authentication gets delayed because $B$ is malicious or $A'$ is spoofing. Since $A'$ can not spoof forever and $A$ can contact another honest party $P_i$, it is eventually authenticated. If some peers are malicious or faulty, then $A$ is not provably malicious, i.e. it recovers the valid proofs on request of some honest $B$ and the protocol moves into the second phase.

□

It can be observed that trusted groups expand in a manner that avoids man in the middle attacks. If the peer $A$ faces this attack during challenge response with $P_i \in \mathcal{T}(B)$, then it cannot join the trusted group of $B$ until $P_i$ is eventually removed from the trusted group.

### 4.3 Group Evolution

Admission requests are caused by the need for secure communication. If the parties $A$ 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).

In the improbable case that both admission requests fail, $A$ and $B$ will randomly select another party $C$ and request admission into $\mathcal{T}(C)$. The probability of $C$ being dishonest is less than $\frac{1}{3}$. Thus in a few trials, $A$ and $B$ shall find an honest party and join its trusted group. They come to know $C$'s trusted group by challenge response. By requesting each peer in $\mathcal{T}(C)$ to authenticate the other, each of them forms a new trusted group that contains both of them.

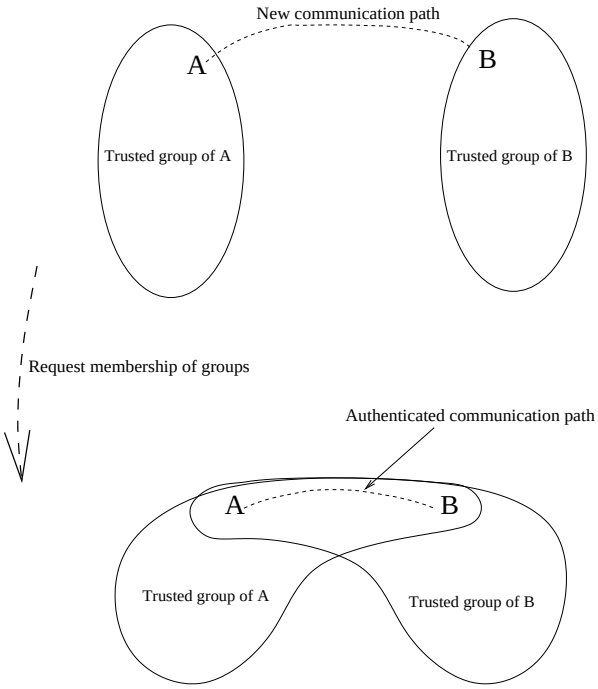


Figure 6: Dynamics of authenticated communication.

### 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 parties successfully masquerade as honest parties, then 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 $\phi$ does not change the behavior 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

$$P_h = \sum_{i=3+0}^{t} P(i)$$

of selecting an honest majority group is computed

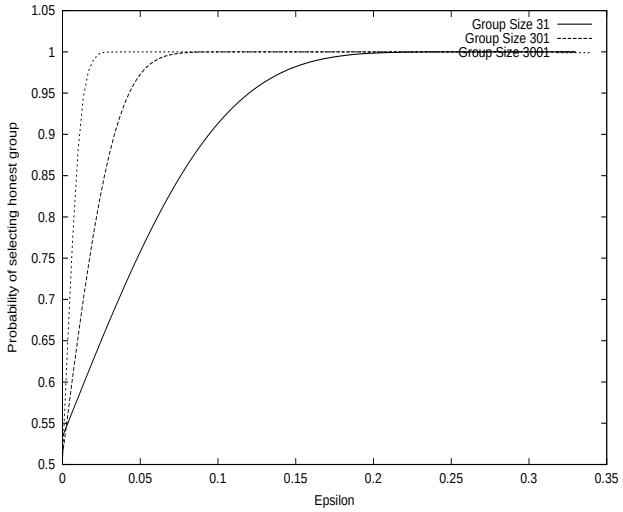


Figure 7: Probability of membership in an honest majority group.

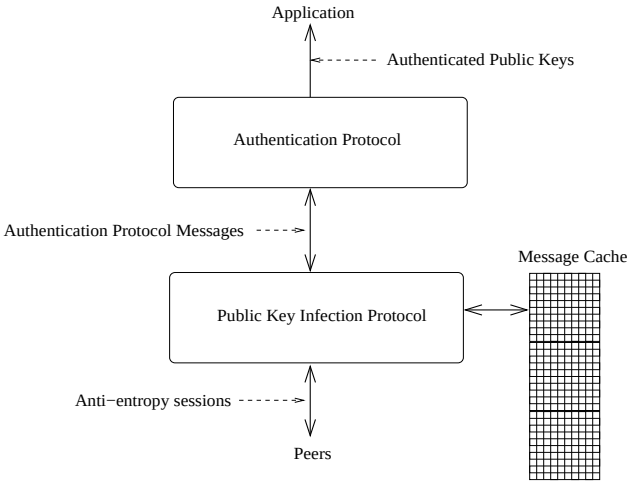


Figure 8: Overview of Public Key Infection.

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

Table 3: The cache record data structure.

and plotted in Figure 7. It shows a rapid growth in probability of randomly selecting honest majority groups from a system with an excess of honest parties.

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 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 8. 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 and correctness. The performance analysis in terms of messaging and space requirements is presented in Appendix B.

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.

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

$$\{\texttt{l}, \texttt{ctv}[s], s, d, m\}$$

### 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 piggybacking on application messages or could depend on a timeout to prevent too much divergence from the eager protocol execu-

| | |
|---|---|
| `lts` | The Lamport timestamp. |
| `ctv[i]` | The last causal timestamp known for peer $i$. |
| `ltv[i]` | The lamport time of the most recent message originating from peer $i$. |
| `srv[i]` | 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 $i$. |

Table 4: Data Structures at the Key Infection layer.

tion. 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 following holds:
   $$r.\texttt{ct} > \texttt{ctv}_d[r.\texttt{src}]$$
   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 $\texttt{ltv}[r.\texttt{src}]$, if the following condition holds $\texttt{ltv}[r.\texttt{src}] < r.\texttt{lt}$. 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 $\min_i(\texttt{ltv}[i])$. The peer $s$ assigns the value to $\texttt{srv}[s]$. 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:
   $$r.\texttt{lt} < \texttt{srv}[r.\texttt{dest}]$$
   Clearly since the value $\texttt{srv}[r.\texttt{dest}]$ is computed at $r.\texttt{dest}$ 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.

# 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 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 hierarchical notion of trust which by assumption takes it for granted that certificate authorities are honest, competent and error free. The traditional model with its all-or-none approach to authentication is a significantly stronger model of authentication with weaker degree of fault tolerance. We trade off the authentication strength (by making it eventually correct) and use stronger network assumptions to provide an autonomous and fault tolerant authentication mechanism.

Having outlined the difficulties of imposing a centralized public key infrastructure on very large systems in Section 1, we believe that the public key infrastructure approach forces the system designers to take a boolean approach to security. A large number of examples can be found both of completely untrusted and completely trusted systems. However, this boolean trust architecture has allowed the vast majority of digital traffic to remain insecure even though the computational power and software engineering needed to secure them remains abundantly available. We believe that Byzantine fault tolerant authentication is useful because it allows the cooperative formation of self authenticating systems.

# 8 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 : without a quorum of participants, it is impossible to create a digitally signed public key certificate [10]. As a consequence, unless the number of malicious parties is as large as the quorum, a false authentication is impossible. Threshold cryptography demands the existence of a dealer that initializes the system of parties with key shares. In this way, it unconditionally depends on the honesty of the dealer. On the contrary, distributed authentication does not require trusted cryptographic initialization and provides for a safe and efficient recovery after the compromise of public keys (unless the number of dishonest parties exceeds the given threshold). Threshold cryptography has been used in COCA, a fault tolerant public key authentication service [21] and as the basis of a number of other secure services [2, 15, 20].

Proactive recovery is possible through a dynamic version of threshold cryptography. To compromise such a system, the adversary is required to compromise the quorum in less than a given time out interval or lose any previous progress due to a rerandomization of key shares [3]. 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 stronger sense that it holds no long term secrets.

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 [7]. NICE allows the creation of trustworthy peer groups through a trust evaluation mechanism that allows the use of reputations [12]. Both systems aggressively eliminate (overtly) malicious parties to preserve their correctness.

The application of distributed trust to authentication in distributed systems is done by PGP through

its web of trust [22, 9]. This model demands human evaluation of trustworthiness by requiring an explicit assignment of authenticity ratings to public keys of peers. While it is suitable for small groups of highly skilled users, its utility to the vast population of simple users seems limited. Although PGP has been freely available for over a decade, it is yet to be adopted by a majority of mail users. Its limited use has been attributed to the sophistication required to evaluate the trustworthiness of peers. It is found that common users are either unwilling or unable to make these decisions [18].

# 9 Conclusion

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 the lack of single points of failure and the capability to dynamically select trusted groups based on honest or malicious behavior of the parties. Our approach allows a natural growth of trust without requiring hierarchies of delegating and recommending parties as done in other trust management systems [1, 19]. Our approach is made feasible by the adversary assumptions that are weaker than the classical *network is the adversary* model. In our model, the active attacks are limited to the creation of arbitrary messages. Silencing more than the threshold of honest parties is impossible.

The rise of peer-to-peer systems on the Internet and the popularity of wireless communication are the prime motivations behind this change of the underlying model. Although weaker than the traditional model in terms of adversary power, it is stronger in terms of fault tolerance. The system we envision is completely proactive and functions without having any long term secrets. This makes it robust in terms of recovery. We believe that the messaging cost of our protocol is reasonable even for the most weakly connected systems.

The distributed authentication mechanism is intended to be prototyped by securing electronic mail messages by having a set of mail clients that authenticate each other through an underlying distributed trust mechanism. Another candidate application is distributed authentication in ad-hoc and wireless networks.

# References


[1] Blaze, M., Ioannidis, J., and Keromytis, A. D. Trust Management and Network Layer Security Protocols. In *Cambridge Security Protocols International Workshop* (1999), pp. 103–118.

[2] Cachin, C. Distributing trust on the Internet. In *International Conference on Dependable Systems and Networks (DSN2001), Gteborg, Sweden.* (June 2001), IEEE.

[3] Canetti, R., Gennaro, R., Herzberg, A., and Naor, D. Proactive Security: Long-term protection against break-ins. *RSA CryptoBytes 3*, 1 (1997), 1–8.

[4] CCITT. The Directory Authentication Framework. Recommendation X.509, 1988.

[5] Computer Security Institute and Federal Bureau of Investigation. CSI/FBI Computer Crime and Security Survey. abridged version at http://www.gocsi.com, April 2002.

[6] Diffie, W., and Hellman, M. New Directions in Cryptography. *IEEE Trans. Info. Theory 22* (1976), 644–654.

[7] Dingledine, R., Freedman, M. J., and Molnar, D. The free haven project: Distributed anonymous storage service. *Lecture Notes in Computer Science 2009* (2001).

[8] Fox, B., and LaMacchia, B. Certificate Revocation: Mechanics and Meaning. In *FC'98: International Conference on Financial Cryptography*, R. Hirschfeld, Ed., vol. 1465 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998, pp. 158–164.

[9] Garfinkel, S. *PGP: Pretty Good Privacy.* O'Reilly & Associates, Inc., Cambridge, MA, 1995.

[10] Goldwasser, S., Micali, S., and Rivest, R. L. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput. 17*, 2 (1988), 281–308.

[11] Lamport, L., Shostak, R., and Pease, M. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS) 4*, 3 (1982), 382–401.

[12] Lee, S., Sherwood, R., and Bhattacharjee, S. Cooperative Peer Groups in NICE. In *INFOCOM* (2003).

[13] Naor, M., and Nissim, K. Certificate Revocation and Certificate Update. In *Proceedings 7th USENIX Security Symposium (San Antonio, Texas)* (Jan 1998).

[14] R. Rivest, A. S., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126.

[15] Reiter, M. K. The Rampart Toolkit for Building High-Integrity Services. In *Dagstuhl Seminar on Distributed Systems* (1994), pp. 99–110.

[16] U.S. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS), August 1991. F1PS PUB 186.

[17] U.S. National Institute of Standards and Technology (NIST). Data Encryption Standard (DES), December 1993. FIPS PUB 46-2.

[18] Whitten, A., and Tygar, J. D. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *in Proceedings of the 8th USENIX Security Symposium* (August 1999).

[19] Yahalom, R., Klein, B., and Beth, T. Trust relationships in secure systems—a distributed authentication perspective. In *In Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy* (May 1993), pp. 150–164.

[20] Zhou, L., and Haas, Z. Securing Ad Hoc Networks. *IEEE Network Magazine 13*, 6 (1999).

[21] Zhou, L., Schneider, F. B., and van Renesse, R. COCA: A Secure Distributed On-line Certification Authority. Tech. Rep. 2000-1828, Department of Computer Science, Cornell University, Ithaca, NY USA, December 2000.

[22] Zimmermann, P. *The Official PGP User's Guide*. MIT Press, Cambridge, Massachusetts, 1995.

# A Attacks

This section discusses the resistance to various attacks 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 party $A$. 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 or 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

## A.1 Malicious Outsiders

### Spoofing

A malicious party $A'$ may try to impersonate an honest party $A$ 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 $\mathcal{P} = \{A, K_{A'}\}_A$ 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 will receive two distinct responses, one valid and one invalid. It considers the incorrect message, and $A'$ is not authenticated, as required. 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 party $A'$ 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 party is to create a message of the following form:

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

This is equivalent to spoofing if $X \neq A$. If $X = A$ then the malicious party 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 $P_i$:

$$\mathcal{R}_{iA} = \{A, P_i, \texttt{response}, r_i'\}_{A'}$$

Since $r_i \neq r_i'$ is required for 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 peer is not provably malicious. Thus **G2** is not satisfied.

### A.2 Malicious Insiders

#### Spoofing

Spoofing by trusted peers is restricted to parties outside the trusted group. This is because the trusted parties 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 the peer, but these will not count in distributed authentication because of invalid signature on the challenge component of the message. Further, if the probationary peer sends the invalid proof to any trusted peer, the insider is provably malicious and will be deleted in the 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 peer $A$ into a trusted group by not sending a correct `authentication request` message. This does not satisfy either of the goals.

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. This will happen when an honest peer find a lack of consensus on authenticity of $K_A$.

If it fails to send a message, it cannot construct a bad response signed by $A$. The lack of responses only delays the entry of $A$ into the trusted group. Neither **G1** nor **G2** is satisfied.

## B Performance Analysis

The performance of public key infection is analyzed as follows.

### B.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 un-infected peers, only $f_i^2$ peers remain un-infected with the update at round $i+1$. Thus, on the average:

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

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 $\mathbf{O}(1 + \frac{1}{n})$.

### B.2 Size of the message cache

Let $\mu$ be the rate at which messages are being created at the authentication protocol layer. Thus the cache gets $\mu$ 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 $i^\star$. 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:

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

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

$$N = \sum_{i=3+1}^{\infty} \sum_{1=3+1}^{i} n\mu(1 - f^{2^i}) f^{2^{i-i^\star}} \tag{1}$$

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:

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

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 $\mathbf{O}(n \log n)$.