Improving Robustness of PGP Keyrings by Conflict Detection

(author names omitted)

June 20, 2003

Abstract

Secure authentication frequently depends on the correct recognition of a user's public key. When there is no certificate authority, this key is obtained from other users. If such users can be malicious, trusting the key information they provide is risky.

Previous work has suggested the use of redundancy to improve the trustworthiness of user-provided key information. In this paper, we build on this work and address two issues not previously considered. First, we explain and solve the problems presented by users who falsely claim multiple identities, or who (possibly legitimately) possess multiple keys. Secondly, we show that conflicting *certificate information can be exploited to increase the reliability of the key information provided by users.*

Our methods are demonstrated on actual and synthetically-generated webs of trust (PGP keyrings) and their performance is discussed.

Keywords: authentication, security, certificates, public keys, trust

1 Introduction

Authentication is one of the most important objectives in information security. Until the mid-1970s, secrecy was generally recognized to be a necessary part of authentication. With the invention of several cryptographic techniques, such as hash functions and digital signatures, it has been shown that secrecy and authentication are not necessarily connected. Applying these cryptographic techniques, public key infrastructures have been established, with authentication one of the provided functions. Some examples are X.509 [28] and PGP [29]. In the public key infrastructure, each user is associated with a public key, which is publicly available, and with a private key, which is kept secret by the user. A user signs something with her private key, and this signature can be authenticated using the user's public key.

The ability to exchange public keys securely is a very important requirement for the authentication scheme to work. Various ways may be used to exchange the public keys. For example, certificates are considered to be a good way to deliver public keys, and are popularly used in today's public key infrastructures. Intuitively, a certificate is an authority telling about a user's public key. In a hierarchical system, such as X.509 [28], we usually assume that the certificates contain true information because the authority is secured and trusted. The problem appears in a non-hierarchical system, such as PGP [29], which is referred to as a "web of trust". In such a system, each user becomes an authority. It is risky to believe all certificates contain true information, because not all users are fully secured and trusted.

Accepting a false public key as true undermines the foundation of authentication. For example, a user Alice may incorrectly recognize a false public key k created by a *malicious* (i.e., untrustworthy, unreliable) user John as Bob's public key. Any confidential information for Bob encrypted with k would be readable by John, which is undesirable. Bob's ability to authenticate is also compromised because John could generate digital signatures which would make another user, Alice, believe information comes from Bob. For this reason, a method that can be used to securely verify a user's public key is very much needed.

Our main goal in this paper is to develop a robust scheme to determine if a certificate is trustworthy. Our method uses redundant information to confirm the reliability of a certificate. Previous work [25] has shown that redundancy in the form of multiple, independent certificate chains can be used to enhance reliability. We build on this work, but show that in some circumstances multiple certificate chains do not provide sufficient redundancy. This is because a single malicious user may possess multiple public keys, or (falsely) claim multiple identities, and therefore can create multiple certificate chains which seem to be independent. Our solution to this problem is to also consider identities when making use of multiple certificate chains.

In addition, it has previously been observed that conflicting certificates may occur, but the consequences of this possibility have not been explained. Conflicts are easy to detect. We show that this conflicting information can be used to narrow the list of *suspects*, or possibly malicious users. Based on that information, the number of certificates which can be proved to be true is increased, improving the performance of our method.

The organization of the paper is as follows. In section 2, we discuss related work. In section 3, the notation used in this paper and the assumptions of our method are described precisely. The research problem is defined, and the criteria to evaluate possible solutions are also discussed. Section 4 presents our solutions, and the corresponding algorithms. In section 6, we present the results of experiments on both synthetic data and PGP keyrings, and discuss the performance of our solutions. The last section summarizes our results and suggests future work.

2 Related work

Current public key infrastructures, such as X.509 [28] and PGP [29] are used to provide authentication and other security functions. In X.509, certificates can only be issued by a certificate authority (CA). This structure is hierarchical, and the method of distributing certificates is called "hierarchical trust". PGP works in a different way, in that each *user* can issue certificates. The structure is horizontal and is called a "web of trust". In the web of trust model, a means of validating the user-issued certificates is needed. Other public key infrastructures, such as SPKI/SDSI [11] and PolicyMaker [4], mainly focus on access control issues. They differ from X.509 and PGP in that they bind access control policies directly to public keys, instead of to identities.[1]

Existing methods of improving the reliability of webs of trust can be classified into two categories. In the first category, *partial trust* is used to evaluate the confidence in the target public key. In [27] and [21], the confidence is computed based on the trust value in a single certificate chain. Multiple certificate chains are used in [3] and [19] to boost assurance. [15] tries to reach a consensus by combining different beliefs and uncertainties. In [24], insurance, which can be seen as another form of reducing risk, is used to calculate the reliability of a target public key.

In the second category of methods, there is no partial trust; a key is either fully trustworthy, or else it is untrustworthy. In this category, an upper bound is assumed on the number of participants that could be malicious or compromised. Some examples from this category are [6] and [5], which require a bound on the minimum network connectivity in order to reach a consensus. This work is related to the classic Byzantine generals problem [23, 17, 9, 8]. It has been shown in [9, 8] that if the number of faulty (malicious) entities is bounded by u, then consensus can only be achieved if the network is at least $(2u + 1)$-connected.

Another important method in the second category is [25]. This method suggests using multiple public key-independent certificate chains to certify the same key. The paper showed that if n public keys are compromised, a public key must be true if there are at least $n + 1$ public key-independent certificate chains certifying it. This observation is the motivation for computing the network flow in certificate graphs. For the case where only cer-

[1]In SPKI, "local names" are akin to roles or group names, rather than individual identities.

tificate chains with bounded length are considered, network flow becomes NP-complete; heuristics for solving this problem sub-optimally were presented.

Methods in the first category (partial trust) are based on probabilistic models. Because of this, they are appropriate for computing reliability of (unintentionally) faulty systems. In the presence of malicious users, however, they may not be as appropriate. One limitation is that they require proper estimation of partial trust on each user. If the trust profile is not set correctly, or a trusted user is compromised, then the output produced by these methods could be misleading. We believe methods in the second category are more suitable for the case of (intentionally) malicious users, because their assumptions better match this scenario. That is, it is much easier to bound the number of users who may be malicious, than to specify how trustworthy each user is.

A limitation with methods in the second category is that the importance of identities, as well as public keys, has not been fully considered. That is, these methods have not considered the possibility that each user may claim multiple identities, or possess multiple public keys. Disregarding such a possibility gives the malicious user the ability to create enough redundant information to make a false certificate seem trustworthy.

All methods of distributed trust computation assume there is some initial trust between selected users. Without such initial trust, there is no basis for any users to develop trust in each other. In [10], it is shown that forging multiple identities is always possible in a decentralized system. We assume that the initial trust must be negotiated in an "out-of-band" way (such as direct connection, or communication with a trusted third party) from the distribution of trust, and that proof of identity is available during this initial phase. For the initial secure key exchange, [12] and [2] discussed several methods. PGP keysigning "parties" are another mechanism for secure initial trust distribution. We do not address this issue further in this paper, except to reiterate that this initial trust information is assumed to be correct.

Our work extends and improves upon the method of [25]. Note our work mainly deals with but is not restricted to authentication using public keys. It can also be extended to a broader sense of authentication, as defined in [18]. A public key "speaks for" an identity, in the same way that a *channel* speaks for a *principal* in that work.

The next section presents the definitions and assumptions on which our methods are based. We also define precisely the problem to be solved, and the criteria for assessing the proposed solution.

3 Notations and assumptions

A *user* is an entity in our system represented by an *identity*, such as the names "Bob" and "Alice". An identity must be established when the initial trust information is negotiated between users. We assume in this work that each user legitimately has exactly one, unique *true identity*. An identity which does not belong to a real user is a *false identity*. We consider the possibility in the following that a user may forge multiple, false identities, but there are not multiple true identities for each user.

We further assume each user can have, or be associated with, one or more public keys. In the case where a user has more than one public key, we assume the user further specifies each of her keys by a *key index number*. The combination of a user identity x and a key index number j is denoted x/j, and uniquely identifies a public key. If the user with identity x only has a single public key, j will be omitted, for the sake of convenience.

A *public key certificate* and a *certificate chain* are defined as follows, based on [22]:

Definition 1 *A public key certificate is a triple $\langle x/j, k, s_{k'} \rangle$, where x is an identity, j is a key index number, k is a public key, and $s_{k'}$ is the digital signature over the combination of x/j and k.*

Note $s_{k'}$ is associated with the certificate but k', the public key that could be used to verify $s_{k'}$, is not necessary in the certificate.

Given a certificate $C = \langle x/j, k, s_{k'} \rangle$, if (i) the identity x is a true identity, and (ii) the user with identity x requested a certificate be issued to it for public key k, then C is a *true certificate* and

k is a *true public key* for x. Otherwise, C is a *false certificate* and k is a *false public key* for x. If $s_{k'}$ is generated by y, we say the certificate is *issued* by y. If all certificates issued by y are true certificates, then y is a *good user*. If there exists at least one false certificate issued by y, then y is a *malicious user*.[2]

Two certificates are said to *agree* with each other if the identities, key index numbers and public keys are the same. Two certificates are called *conflicting certificates* if the identities and key index numbers are the same but the public keys are different. Note that the two conflicting certificates may both be true by our definition (they both agree with information negotiated initially with at least one other user). This may happen when a user x intentionally has two conflicting certificates issued to herself, by two separate parties. In this case, x may be regarded as a malicious user, but the other parties are not, and the certificates are defined to be true.

For a user w to decide a certificate C which contains identity x is false, C must conflict with another, true, certificate, and w must believe that x wouldn't ask for conflicting certificates to be issued to herself.

Each user x may accumulate a set R^x of certificates about other users. Obtaining these certificates may be done in a variety of ways, and may be volunteered (i.e., the *push* model), or may be provided on request (the *pull* model). For example, certificates can be obtained by retrieving from a (trusted) key server, by transmission directly from user to another in a network, etc. We do not discuss further in this paper how certificates are distributed.

In each user's set of certificates, some of them are assumed by x to be true and others are not. Denote T_0^x the set of certificates assumed by x to be true initially (i.e., they are provided by means of the initial trust distribution). Because this initial trust information is assumed to be true, the signatures on the certificates in T_0^x do not have to be further verified. We now define a *certificate chain*:

Definition 2 *A certificate chain is a sequence of certificates where:*

1. *the starting certificate, which is called the "tail" certificate, is assumed or determined to be true;*
2. *each certificate contains a public key that can be used to verify the digital signature associated with the next certificate in the sequence; and,*
3. *the ending certificate, which is called the "head" certificate, contains a desired name-to-key binding, which is called the* target.

Each user x's set of certificates R^x may be represented by a directed *certificate graph* $G^x(V^x, E^x)$[3]. V^x and E^x denote the set of vertexes and the set of edges in the certificate graph G^x, respectively. A vertex in V^x represents a public key and an edge in E^x represents a certificate. There is a directed edge labeled with y/j from vertex k' to vertex k in G^x if and only if there is a certificate $\langle y/j, k, s_{k'}\rangle$ in R^x.

A certificate chain is represented by a directed path in the certificate graph. Two conflicting certificates are represented by two edges with the same label, but different head vertexes (i.e., different keys for the same identity/index number). In this case, the two different head vertexes are called *conflicting vertexes*.

For computation purposes, we add to the certificate graph an "oracle" vertex k_0. There is a directed edge from k_0 to every key which is assumed to be true (i.e., by way of the initial trust distribution), labeled with the identity/index number bound to that key.

To depict true and false public keys in the certificate graph, we will paint vertexes with different colors. A vertex painted white represents a public key that is either assumed or determined to be true for the identity on the edges incoming to that vertex. A vertex painted a medium gray represents

[2]We ignore the fact that a false certificate may be mistakenly issued with no malicious intent, for purposes of this paper. The method of detecting mistakes is the same as the method of detecting deliberately false information, so no distinction is required.

[3]Certificate graphs are also referred as *trust graphs* in some other research papers.

a public key which is known to be false for the claimed identity. If a public key is not assumed to be true and hasn't been determined to be true for an identity, we paint it with a light gray color.

Figure 1 is a sample certificate graph, k_0 is the oracle vertex. k_1, k_2 and k_3 are three public keys that are assumed to be true by user x. k_1 is y's public key, with key index 1. k_2 is y's public key with key index 2. k_4 and k_5 are are undetermined, and they are conflicting vertexes because the labels on their incoming edges are the same. k_6 is a false public key.

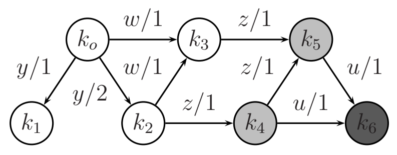

Figure 1: User x's certificate graph

When there are more than one malicious users, there may be a relationship between them. Two malicious users who cooperate with each other to falsify information are said to be *colluding.* In the case where it is unknown if two users have such a a relationship, a conservative assumption is that they are colluding. We say that two users x and y are colluding if either of the following is true:

1. There exist two false certificates, one issued by x and one by y, and they agree with each other[4] ; or,
2. x issues a false certificate upon y's request, or vice versa.

3.1 Problem Description

Now we give a formal problem description, and our criteria to evaluate the performance of any solutions. The problem statements are introduced gradually. First we consider the case that there's only one malicious user. When there are multiple malicious users, we consider two cases, one in which the users are assumed not to collude, and one in which they do. The goals for these three problems are the same, that is, to determine the maximum number of true certificates.

PS 1 *Given a set R^x of certificates and a set $T_0^x \in R^x$ of true certificates. Assuming there is* at most one malicious user*, maximize the number of certificates which can be proved to be true.*

PS 2 *Given a set R^x of certificates and a set $T_0^x \in R^x$ of true certificates. Assuming there are* at most n malicious users*, and these users do* not collude*, maximize the number of certificates which can be proved to be true.*

PS 3 *Given a set R^x of certificates and a set $T_0^x \in R^x$ of true certificates. Assuming there are* at most n malicious users *and these users* may collude*, maximize the number of certificates which can be proved to be true.*

It is necessary for us to develop a criteria to evaluate the performance of proposed solutions for the above problems. In that way we can compare different approaches. Let U be the set of all users. Denote by K_a^x the set of true public keys that can be reached by at least one path from the oracle vertex k_0 in the certificate graph. K_a^x is the maximum set of true public keys that any method could determine by means of certificate chains. Let K_0^x be set of public keys that are assumed to be true initially, and K_M^x the the set of public keys that are determined to be true by a method M. Ideally, K_a^x would be equal to $K_0^x \bigcup K_M^x$, but in practice this may be difficult to achieve. The fraction $\frac{|K_0^x|+|K_M^x|}{|K_a^x|}$ expresses the performance of a given method:

Definition 3 *Given a solution s to a problem in 1, 2 and 3 generated by a method M. The performance $q_x(s)$, i.e. s's performance for user x, is*

$$q_x(s) \;=\; \frac{|K_0^x| + |K_M^x|}{|K_a^x|}$$

To capture the performance for a set of users, rather than just one, we propose using the weighted average of each user's performance:

[4]We ignore here the very small possibility that two non-colluding malicious users could by chance create identical false certificates. Since keys are generated randomly and are quite long, the probability of independently generating the same key to put in two false certificates is extremely low.

Definition 4 *Given each user x's performance $q_x(s)$ for solution s generated by method M, the performance $q(s)$ of solution s for the set of users in U is the weighted average of the performance of s for each user. The weight of a user x is $|K_a^x|$, i.e., the number of public keys in x's certificate graph which are true.*

$$q(s) \; = \; \frac{\sum\limits_{x \in U} \{q_x(s) \cdot |K_a^x|\}}{\sum\limits_{x \in U} |K_a^x|}$$

Definition 3 expresses the fraction of true public keys that are assumed or can be determined to be true for each user. Definition 4 expresses the fraction of true public keys that are assumed or can be determined to be true for all the users.

We now present methods for solving problems 1, 2, and 3.

4 Maximizing the number of true certificates when there are no conflicts

In this section, we present methods for solving problems 1, 2, and 3 under two assumptions (single or multiple keys per identity). We use redundancy, as have others, to confirm certificates that must be true. As mentioned, we assume there is a known upper bound on the number of users who may be malicious. We ignore in this section the case in which there are conflicting certificates, which is considered in section 5. We begin with some basic definitions and insights.

Two certificate chains are *public key independent* if

1. their head certificates agree; and,
2. their remaining certificates have no public keys in common.

Two certificate chains are *identity-independent* if

1. their head certificates agree; and,
2. their remaining certificates have no identities in common.

Theorem 1 *Given two* identity-independent *certificate chains, if there is at most one malicious user, the head certificates of the two chains must be true.*

Proof: We prove Theorem 1 by contradiction. The head certificates of the two chains, by definition, agree with each other. Suppose the head certificates were false. Then in each of the certificate chains, the malicious user's true identity must appear in the chain (i.e., the malicious user has participate in creation of the the chain in order to insert a false certificate). In this case, however, the same identity would appear in each chain, violating our assumption that they are identity-independent. □

For multiple non-colluding malicious users, an analagous result holds:

Theorem 2 *Given two identity-independent certificate chains and any number of* non-colluding *malicious users, then the head certificates must be true.*

Proof: The head certificates of the two chains, by definition, agree with each other. If the head certificates were false, each chain (for the same reason as above) must contain the true identity of a malicious user. Since the chains are identity-independent, and we assume one user has only one true identity, these malicious users must be different. But if the head certificates are false and they agree, these two malicious users must have been colluding, which violates the assumption. □

In the case that there are multiple colluding malicious users, a greater degree of redundancy is needed to verify a certificate is true:

Theorem 3 *Given* $n+1$ *identity-independent certificate chains, if there are at most n colluding malicious users, then the head certificates must be true.*

Proof: We prove Theorem 3 by contradiction. Suppose the head certificates were false. Then in each of the $n+1$ certificate chains, there must be a malicious user's true identity. But if there are at most n malicious users, there cannot be $n+1$ identity-independent certificate chains. □

Based on these results, we now present methods for maximizing the number of true certificates (when there are no conflicting certificates).

4.1 One public key allowed per identity

Reiter and Stubblebine [25] were the first to consider this problem. We summarize their results.

When each identity corresponds to only one public key, public key-independent certificate chains are also identity-independent. In the certificate graph, public key-independent certificate chains corresponds to paths with the same head vertex, but which are otherwise vertex disjoint. For each vertex k_t, it is possible to use standard algorithms for solving maximum network flow [1] in a unit capacity network to find the number of vertex-disjoint paths to k_t from k_0.

It can be shown that:

1. if there is at most 1 malicious user, and the maximum flow from k_0 to k_t is greater than or equal to 2; or,
2. if there are any number of non-colluding malicious nodes, and the maximum flow from k_0 to k_t is greater than or equal to 2; or,
3. if the number of (possibly colluding) malicious users is no greater than n and the maximum flow from k_0 to k_t is greater than or equal to $n + 1$,

then all certificates (edges) ending at k_t must be true. This procedure is run once for each vertex k_t in G^x to maximize the number of certificates known to be true.

The algorithm for solving maximum flow in unit capacity networks runs in $O(|V||E|)$.

In this paper, we do not consider the case in which certificate chains are restricted to be no greater than a specified bound, which is solved heurisitically by [25]. We are not convinced this is a necessary requirement at this time.

4.2 Multiple public keys allowed per identity

We now address the case in which an identity may be associated with multiple public keys in x's certificate graph. The method of [25] does not apply here. For example, in figure 1 there are two vertex-disjoint paths from k_0 to k_6. If it is assumed there is at most one malicious user and no legitimate user asks for more than one public key, the method of [25] concludes that k_6 is u's public key. However, consider if user z has two public keys k_4 and k_5, and this is allowed. In this case, it is not safe to conclude that k_6 is u's public key.

We still wish to use the notion of redundant, identity-independent paths to overcome the influence of (single or multiple, colluding or non-colluding) malicious users. To ensure that two paths are *identity-independent* under the new assumption, it is necessary that the two paths have no label in common on their edges. In this case the paths are said to be *label-disjoint*. In this context, when we refer to the label on an edge, we use only the identity of the certificate issuer, and ignore the key index number.

Suppose there exists a solution to the problem of determining the maximum label-disjoint network flow in the graph G^x with unit capacity edges, from k_0 to a vertex k_t. We conclude (by the reasoning previously given) that k_t is a true public key for the identity on the edges in the maximum flow ending at k_t if:

1. the maximum flow is 2 or greater, and there is at most 1 malicious node, or any number of non-colluding malicious users; or,
2. the maximum flow is $n + 1$ or greater, and there are at most n colluding malicious users.

We now state a theorem about the complexity of finding the maximum label-disjoint network flow in a graph:

Theorem 4 *Given a certificate graph G^x and a vertex k_t. If one identity may legitimately correspond to multiple public keys, the problem of finding the maximum number of label disjoint paths in G^x from k_0 to k_t is NP-Complete.*

The proof may be found in the appendix.

Therefore, to solve this problem exactly will be very expensive computationally, in the worst case. We present two possible approaches.

We first consider an approach following an idea from [25]. The problem of finding the maximum number of label-disjoint paths between k_0 and k_t can be transformed to the maximum independent set (*MIS*)[13] problem. The *MIS* problem is defined as follows: Given a graph G, find the maximum subset of vertexes of G such that no two vertexes are joined by an edge. The transformation from our problem is trivial. From the certificate graph G^x, let each path from k_0 to k_t be transformed to a vertex in a new graph G^{MIS}. If two paths in G^x have a label in common, then the two corresponding vertexes in G^{MIS} are joined by an edge. Otherwise there are no edges in G^{MIS}. The size of the maximum independent set in G^{MIS} is the maximum label-disjoint network flow from k_0 to k_t in G^x.

Although *MIS* is also a NP-complete problem, there exist several well-known approximation algorithms for it. See, for instance, [16] and [14].

Alternatively, we may consider solving the problem optimally if the required number of label-disjoint paths is small. Suppose we wished to solve the problem of whether there exist at least b label-disjoint paths in a graph from a source to a sink. The maximum number of label-disjoint paths from k_0 to k_t equals the size of the minimum label-cut for k_0, k_t. A label-cut is a set of labels on edges whose removal would disconnect k_t from k_0. The following enumeration algorithm 1 will determine the minimum label cut in a graph, up to a size of b. The algorithm runs in $O(|E||V|^b)$ (proof omitted due to space limitations). The algorithm takes the certificate graph G^x as input, and outputs i, the size of the minimum label-cut.

In this section we considered how to prove certificates are true without reference to the possibility of conflicting certificates. We now turn to this problem.

algorithm 1 *label disjoint*

```
1: Delete all vertexes from $G^x$ that are not reachable from $k_0$, and that can't reach $k_t$.
2: for $i = 1...b$ do
3:   for each subset $L_i$ of $i$ labels in $L$ do
4:     set $L$=the set of labels on edges in $G^x$, and $G$=$G^x$
5:     delete each label in $L$ that is in $L_i$, and delete the edges with these labels from $G$
6:     if $k_t$ is not reachable from $k_0$ in $G$ then
7:       return $i$
8:     end if
9:   end for
10: end for
```

5 Maximizing the number of true certificates when there are conflicts

We assume that conflicting certificates occur because of malicious intent, and not by accident. A malicious user may create conflicting certificates for several reasons. For example, one use is to attempt to fool user x into believing a false public key is true, by creating multiple public key-independent certificate chains to the false public key. In this case, the method of section 4.2 can first be applied to determine the set of true certificates.

However, we can exploit the existence of conflicting certificates to prove an even larger number of certificates must be true. Conflicting certificates have been mentioned in some research papers, such as [25], for instance. Although it has been pointed out that conflicting certificates represent important information, it has not been suggested how they can be used. We propose below a method of doing so, based on the notion of the *suspect set*:

Definition 5 *A suspect set is a set of identities that contains at least one malicious user. An member of the suspect set is called a suspect.*

We now describe how the suspect set can be constructed, and how it helps to determine more true certificates.

5.1 Constructing suspect sets (single or multiple non-colluding malicious users)

Suppose we have determined a certificate is false by some means. If there is only one malicious user, or multiple non-colluding malicious users, the true identity of the malicious user creating this false certificate must appear in a certificate of every chain ending with a certificate that agrees with this false certificate.

Using this insight, we propose constructing suspect sets using algorithm 2. The algorithm takes a certificate graph G^x as input.[5]

Algorithm 2 *suspect set one*

```
1:  delete all vertexes that can't be reached from the oracle vertex $k_0$
2:  for each label $y/j$ in the certificate graph do
3:    if there exists a medium gray vertex whose corresponding edge has a label $y/j$ then
4:      for each medium gray vertex $k_i$ whose corresponding edge has a label $y/j$ do
5:        construct a new suspect set consisting of the set of labels in which each single one is a label-cut for $k_0, k_i$.
6:      end for
7:    else {there exists a white vertex whose corresponding edge has a label $y/j$}
8:      for each light gray vertex $k_i$ whose corresponding edge has a label $y/j$ and $k_i$ is in conflict with a white vertex do
9:        construct a new suspect set consisting of {the set of labels in which each single one is a label-cut for $k_0, k_i$} $\cup\ y$
10:     end for
11:   else {there exists two light gray vertexes whose corresponding edges both have a label $y/j$}
12:     for each pair $k_i, k_h$ of light gray vertexes whose corresponding edges both have a label $y/j$ do
13:       construct a new suspect set consisting of {the set of labels in which each single one is a label-cut for either $k_0, k_i$ or $k_0, k_h$} $\cup y$
14:     end for
15:   end if
16: end for
```

The intuition behind algorithm 2 is as follows.

Lines 2-6 For each false certificate, the true identity of the malicious user who issued the false certificate must be in a certificate in *each* certificate chain whose head certificate agrees with this false certificate. This means, in the certificate graph the malicious user's true identity must be a label-cut for k_0, k_t (where k_t is the public key in the false certificate). Note there may be multiple such label-cuts.

Lines 7-10 Let y/j be a label which has not been found in a false certificate, but which has been found in a true certificate. A certificate conflicting with this true certificate, may itself be true or false. If it is true, the malicious user must be y. If it is false, the identity of the malicious user who issued this false certificate must be in every certificate chain ending at this certificate. In this case, the suspect set contains every identity which is a label-cut for k_0, k_t, plus y, where k_t is the public key in this undetermined certificate.

Lines 11-14 Let y/j found only in certificates not known to be either true or false. If a pair of conflicting certificates contain y/j, they may both be true, one may be true but not the other, or they may both be false. The suspect sets can be constructed for each case, and the union of them is taken to get a single suspect set.

The complexity of this algorithm is $O(\mid V\mid^4 \mid E\mid)$ (proof omitted).

This algorithm may generate a large number of suspect sets. We now explain how this information can be used.

5.2 Exploiting suspect sets (single or multiple non-colluding malicious users)

Suppose there is a single malicious user. Let L_s represent the set that results by intersecting all the suspect sets generated by the above algorithm. Then clearly the single malicious user's identity is in L_s.

If, on the other hand, there may be up to b non-colluding malicious users, we try to determine the

[5]For this algorithm, the key index number part of each label is ignored.

maximum disjoint sets (*MDS*) from all the suspect sets generated by algorithm 2. Two sets are called disjoint if they don't share any label in common.

Unfortunately, *MDS* is also NP-Complete, by transformation from the maximum independent set problem, MIS. First, each vertex in MIS is transformed into a set which contains just one distinguished label, representing that vertex. Then, for each edge joining two vertexes in MIS, a distinguished label is added to the two sets corresponding to those vertexes. A maximum disjoint set for this problem is also a solution for the transformed MIS problem. As before, we can apply known heuristics for MIS to solve this problem approximately.

Suppose a solution to MDS consists of $m \leq b$ suspect sets. Let L_m be the union of these m sets. It is clear that all b malicious users must be in L_m.

Given L_s or L_m, we can determine more certificates are true as follows. For each undetermined public key k_t, if there is only one malicious user, we simply test if any single label in L_s is a label-cut for k_0, k_t in the certificate graph. If not, k_t is determined to be true. If there are multiple non-colluding malicious users, we simply test if any single label in L_m is a label-cut for k_0, k_t. If not, k_t is determined to be true. For this computation, a modified BFS or DFS search suffices. The complexity for the algorithm is $O(|V||E|)$ for both cases.

5.3 Suspect sets (multiple colluding malicious users)

In the case of multiple colluding malicious users, we propose to use the following rules to construct suspect sets. These rules are presented in order of "narrowest" to "broadest".

suspect set rule 1 *Given a certificate chain whose head certificate is known to be false, construct a new suspect set that contains all the identities (except the identity in the head certificate) in the certificates of the chain.*

suspect set rule 2 *Given two certificate chains whose head certificates conflict with each other, if one of the head certificates is true and the other is undetermined, construct a new suspect set that contains all the identities in the certificates of the chain whose head certificate is undetermined.*

suspect set rule 3 *Given two certificate chains whose head certificates are conflict with each other, if both head certificates are undetermined, construct a new suspect set that contains all the identities in the certificates of the two chains.*

We do not describe an algorithm that implements these rules, due to space limitations. The algorithm is straightforward, and the rules are applied in order. It may be desirable to only apply rule 1, or apply only rules 1 and 2, if the later rules generate suspect sets that are too large.

To make use of the many suspect sets constructed by these rules to determine more true certificates, we try to find the maximum number of disjoint sets (*MDS*) from all the suspect sets. Suppose the number of maximum disjoint sets is found to be a.

Let L_c be the union of the a sets. It is clear that at least a malicious users are included in L_c. Delete all the edges with a label in L_c from the certificate graph. By doing this, the maximum number of malicious users with certificates in the certificate graph is reduced from b to no more than $b - a$. In this case, Theorem 3 can be applied to determine if the rest of the undetermined certificates are true, as follows. For each target public key k_t, if there exist $b - a + 1$ label-disjoint paths between k_0 and k_t, the head certificates of these paths are true. Algorithm 1 can be used to solve this problem.

In this section we have explained how to generate suspect sets, containing the identities of malicious nodes. We described how to use this information to prove additional certificates must be true. We now present experimental results about the use of conflicts.

6 Experimental results

To investigate the practicality and benefits of our conflict detection method, we implemented and tested it. Only results for the case of one legitimate public key per user are available at this time.

We emulated "typical" malicious user behavior, in order to contrast the performance before and after conflict detection.

For test purposes, we used actual PGP keyrings. These were downloaded from several PGP key-servers, and the keyanalyze [26] tool was used to extract many strongly-connected components. In addition, we synthetically generated keyrings to have a larger number of test cases to investigate. The synthetic data was generated by the graph generator BRITE [20]. We used the default configuration file for Barabasi graphs, which we believe are similar to actual keyrings. The undirected graphs generated by BRITE were converted to directed graphs by replacing each undirected edge with two directed edges, one in each direction. The number of vertexes in each synthetic key ring was set to 100. For each datapoint, 50 problem instances were generated (using a different random number generator "seed" for each); the values plotted are the average of these 50 instances.

Our first experiment applied the method of [25] to some synthetic data sets, in order to determine how the performance metric q is affected by varying the number of certificates issued per user. Figure 2 shows the results. Each curve represents a synthetic data set with the specified average number of certificates issued by each user, using a power-law distribution. From this figure it is clear that the performance decreases very quickly as the number of malicious users increases. In addition, performance is improved substantially by increasing the number of certificates issued per user.

In our second experiment, we used the synthetic data for the case in which each user issues on average 2 certificates. We emulated a single malicious user's behavior, as follows. We randomly picked a target, a malicious user, and n certificate issuers. Then the malicious user asked the n certificate issuers to certify n different public keys for herself. Using these n different public keys, the malicious user created n certificates, one per key, each binding the target's identity to the same false public key.

After emulating this behavior, we applied algorithm 1 for $b = 2$, and determined the maximum set of true certificates. The resulting performance is the performance *before* conflict detection. Then we applied algorithm 2 to find many suspect sets, from which we constructed L_s. For each of the remaining undetermined public keys, we made use of L_s and the method of section 5.1 to try to determine if it was true. The resulting performance is the performance *after* conflict detection. Each test was run 50 times and the averages are plotted.

Figure 3 shows how the performance q is affected by the number of false certificates n, from a performance of 2% with 1 false certificate, to a performance of 11% with 19 false certificates. This example illustrates a dilemma for the malicious user. While increasing the number of false certificates should increase the uncertainty about keys, it also makes it easier to narrow the list of "suspicious" users, thereby limiting the scope of the damage they can cause.

Figures 4 and 5 are for existing PGP keyrings, and demonstrate how performance is improved by conflict detection. In each PGP key ring, we emulated a single malicious user's behavior as follows. We randomly picked a target, a malicious user, and two certificate issuers. Then the malicious user asked for two different public keys, certified by the two certificate issuers. Using these two public keys, the malicious user created two public key-independent certificate chains to the target. After emulating this behavior, we applied algorithm 1 with $b = 2$ to determine the maximum set of true certificates. The resulting performance is the performance *before* conflict detection. Then we generated L_s by algorithm 2. By making use of L_s, we tried to determine more true certificates. The resulting performance is the performance *after* conflict detection. The bar chart shows a comparison of performance for the case before conflict detection and after conflict detection for some PGP keyrings. The table summarizes the performance of all 27 PGP keyrings, with and without conflict detection.

All of the experiments were performed on a Pentium IV, 2.0GHZ PC with 512MB memory. For the experiments on synthetic graphs with 100 vertexes, running times varied from 5 to 3 seconds. For figure 2, running times ranged from 30 to 150 seconds. For figures 4 and 5, running times ranged

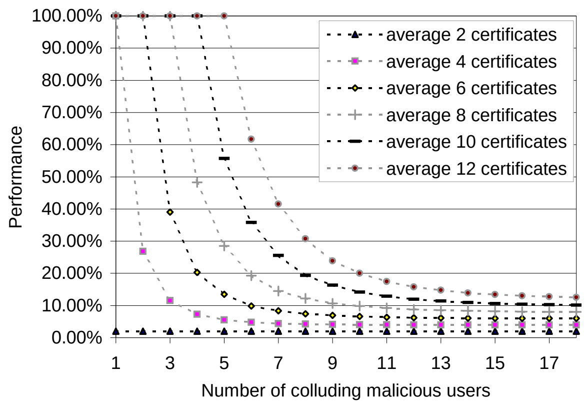

Figure 2: Performance for synthetic data before conflict detection

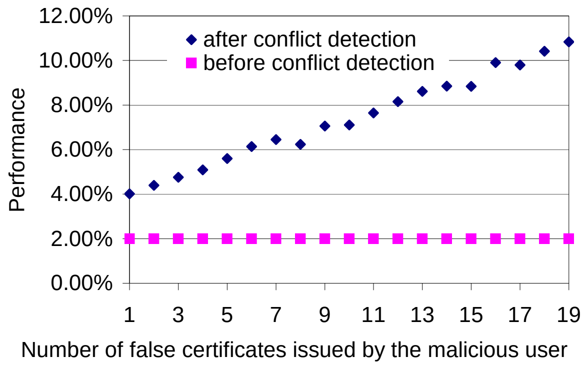

Figure 3: Improved performance for a sample synthetic data set with average 2 certificates after conflict detection

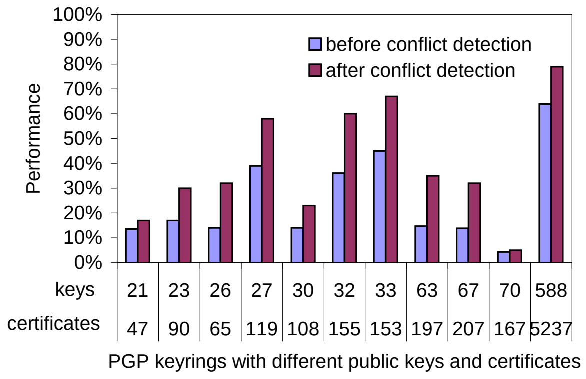

Figure 4: Performance comparison for PGP keyrings before and after conflict detection

Performance Range	0 ~ 25%	25% ~ 50%	50% ~ 75%	75% ~ 100%
Number of PGP keyrings before conflict detection	13	6	5	3
Number of PGP keyrings after conflict detection	4	10	6	7

Figure 5: Summary of performance improvement for 27 PGP keyrings

from 5 to 30 seconds, except for the graph with 588 vertexes, which required 10 hours of CPU time. Our implementations are in no way tuned for efficiency and can undoubtedly be substantially improved. We believe in most environments the algorithm for detecting true certificates will not need to be run frequently, and a running time of hours will be acceptable.

7 Conclusion and future works

In this paper, we investigated the problem of proving certificates are true when trust is distributed, as in PGP keyrings. This is a difficult problem because malicious users may falsify information. Under the assumption that users may legitimately have multiple public keys, we showed that redundant *identity-independent* certificate chains are needed. Previous methods based on *public key-independent chains* are not sufficient under this assumption. We proved the problem of finding the maximum number of identity-independent chains in a certificate graph is NP-complete. A heuristic method for solving it was proposed, as well as an optimal method when the problem size is reasonable.

In the case that certificate conflicts are detected, it is possible to exclude certain users from the set of possible malicious users. This allows additional certificates to be proved true.

Experimental results demonstrated that (a) the web of trust is seriously degraded as the number of malicious users increases, and (b) the use of conflict detection and redundant certificates substantially improves the ability to prove certificates are true.

There remain several aspects to be investigated, including how certificates should be distributed in a network to maximize the likelihood of proving certificates are true, and to aid in detection and identification of malicious users. Implementation of a practical solution is also needed, as web of trust approaches become more commonplace in distributed environments.

References

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows : theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, N.J., 1993.

[2] Dirk Balfanz, D. K. Smetters, Paul Stewart, and H. Chi Wong. Talking to strangers: Authentication in adhoc wireless networks. In *Network and Distributed System Security Symposium Conference Proceedings*, 2002.

[3] Thomas Beth, Malte Borcherding, and Birgit Klein. Valuation of trust in open networks. In *Proceeding of the 3rd European Symposium on Research in Computer Security (ESORICS 94)*, pages 3–18, 1994.

[4] M. Blaze and J. Feigenbaum. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173, Oakland CA USA, 6-8 May 1996.

[5] M. Burmester, Y. Desmedt, and G. A. Kabatianski. Trust and security: A new look at the byzantine generals problem. In *Proceedings of the DIMACS Workshop on Network Threats*, volume 38 of *DIMACS*. American Mathematical Society Publications, December 1996.

[6] Mike Burmester and Yvo Desmedt. Secure communication in an unknown network using certificates. In K. Y. Lam, E. Okamoto, and C. Xing, editors, *Advances in Cryptology ASIACRYPT '99*, pages 274–287, Singapore, November 1999. Springer-Verlag.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[8] Danny Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[9] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *Journal of the ACM (JACM)*, 40(1):17–47, 1993.

[10] John R. Douceur. The sybil attack. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, March 2002.

[11] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. RFC 2693: SPKI certificate theory, September 1999.

[12] Carl Ellison. Establishing identity without certification authorities. In *Proceedings of the 6th USENIX Security Symposium*, pages 67–76, San Jose, California, July 1996. USENIX Association.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman & Co., 1979.

[14] D.S. Johnson. Worst case behavior of graph coloring algorithms. In *Proc. Fifth Southeastern Conf. Combinatorics, Graph Theory, and Computing*, pages 513–527, 1974.

[15] Audun Josang. The consensus operator for combining beliefs. *Artificial Intelligence*, 141(1):157–170, 2002.

[16] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh V. Vazirani. On syntactic versus computational views of approximability. In *IEEE Symposium on Foundations of Computer Science*, pages 819–830, 1994.

[17] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[18] Butler Lampson, Martin Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems (TOCS)*, 10(4):265–310, 1992.

[19] Ueli Maurer. Modelling a public-key infrastructure. In *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS 96)*, pages 324–350, 1996.

[20] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user's perspective. Technical Report BU-CS-TR-2001-003, Boston University, 2001.

[21] S. Mendes and C. Huitema. A new approach to the X.509 framework: Allowing a global authentication infrastructure without a global trust model. In *Proceedings of the Symposium on Network and Distributed System Security, 1995*, pages 172–189, San Diego, CA , USA, Feb 1995.

[22] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography.* CRC Press, Boca Raton, Fla., 1997.

[23] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[24] M. Reiter and S. Stubblebine. Toward acceptable metrics of authentication. In *IEEE Symposium on Security and Privacy*, pages 10–20, 1997.

[25] M. Reiter and S. Stubblebine. Resilient authentication using path independence. *IEEE Transactions on Computers*, 47(12), December 1998.

[26] M. Drew Streib. Keyanalyze - analysis of a large OpenPGP ring. http://www.dtype.org/keyanalyze/.

[27] Anas Tarah and Christian Huitema. Associating metrics to certification paths. In Yves Deswarte, Gérard Eizenberg, and Jean-Jacques Quisquater, editors, *Computer Security - ESORICS 92, Second European Symposium on Research in Computer Security, Toulouse, France, November 23-25, 1992, Proceedings*, volume 648 of *Lecture Notes in Computer Science*, pages 175–189. Springer Verlag, 1992.

[28] Int'l Telecommunications Union/ITU Telegraph & Tel. ITU-T recommendation X.509: The directory: Public-key and attribute certificate frameworks, Mar 2000.

[29] Philip Zimmermann. *The official PGP user's guide.* MIT Press, Cambridge, Mass., 1995.

8 Appendix

Proof of theorem 4:
We sketch a proof that a known NP-Complete problem, 3-CNF-SAT[7], is transformable to our problem. The 3-CNF-SAT problem is described as follows. A boolean formula is in 3-CNF, or 3 conjunctive normal form, if it is expressed as an AND of clauses, each of which is the OR of three literals. A literal in a boolean formula is an occurence of a variable or its negation. For example, $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee x_5)$ is in 3-CNF form. It contains three clauses, each of which contains three literals.

Given a boolean formula ϕ in 3-CNF form, we are asked whether it is satisfiable. Let a *conflicting* pair of literals be a literal in one clause which appears in complementary form in another clause (for example, a literal x_1 appearing in one clause and $\neg x_1$ appearing in another clause). For each such conflicting pair, let there be a unique label assigned, and let the number of conflicting pairs be represented by m. We construct a directed flow graph G from ϕ as follows. Each clause in 3-CNF-SAT is represented as a subgraph of G. In each subgraph, there are three paths. Each path corresponds to a literal in the clause, and has as many edges as there are conflicting literals in the other clauses of ϕ. Each such edge is labeled with the unique label assigned for this conflicting pair. The subgraph has a source vertex and a sink vertex, and the three paths in the subgraph are each connected (in parallel) to this source and sink.

After constructing in this way as many subgraphs as there are clauses, a vertex s is created to be the source for the graph G, and a vertex t is created to be the sink of G. s is connected by a directed edge to the source of each subgraph, and the sink of each subgraph is connected by a directed edge to t. Each edge that connects the sinks of all the subgraphs to the vertex t is assigned the same label, which must be different from all existing edge labels.

If any edges remaining in G do not have labels, each such edge should be assigned a unique label.

For example, in figure 6 we show the transformation from a 3CNF formula $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee x_5)$ to the graph G by the method just described. There are three pairs of conflicting literals: x_2 in the first clause with $\neg x_2$ in the second clause, x_3 in the first clause with $\neg x_3$ in the second clause and $\neg x_3$ in the second clause with x_3 in the third clause. An edge is generated for each conflicting pair, and they are assigned the labels y, z and u respectively. For all other edges in G, since each edge has a unique label, the labels are not shown.

Given this graph G, it is possible to compute

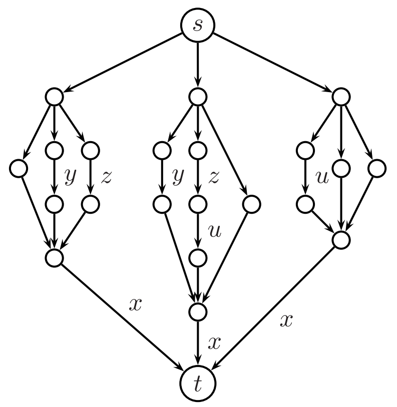

Figure 6: Graph constructed from the Boolean expression $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee x_5)$

the maximum number of label disjoint paths from s to t by some algorithm. If the result is m, then ϕ is satisfiable, otherwise, ϕ is not satisfiable. Since the transformation to G requires only polynomial time, computing the maximum label-disjoint flow in the graph G with unit capacity edges is NP-complete. $\square$