Improving Robustness of PGP Keyrings by Conflict Detection*

Qinglin Jiang, Douglas S. Reeves, and Peng Ning

Cyber Defense Lab
Departments of Computer Science and Electrical and Computer Engineering
N. C. State University Raleigh, NC 27695-8207 USA
{qjiang,pning,reeves}@ncsu.edu

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 using a *web of trust*. If 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 address two issues not previously considered. First, we solve the problem of users who claim multiple, false identities, or who possess multiple keys. Secondly, we show that *conflicting* certificate information can be exploited to improve trustworthiness. Our methods are demonstrated on both real and synthetic PGP keyrings, and their performance is discussed.

1 Introduction

Authentication is one of the most important objectives in information security. Public key cryptography is a common means of providing authentication. Some examples are X.509 [19] and PGP [20]. 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 an essential requirement in this approach. 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 [19], we usually assume that the certificates contain true information, because the authority is secured and trusted. In non-hierarchical systems, such as PGP keyrings [20], each user becomes an authority. Such systems are referred to as *webs of trust*. With webs of trust, it may be risky to expect all certificates to contain true information, because not all users are fully secured and trusted. Accepting a false public key (i.e., believing it contains true information) undermines the foundation of authentication. For this reason, a

* This work is partially supported by the U.S. Army Research Office under grant DAAD19-02-1-0219, and by the National Science Foundation under grant CCR-0207297.

method that can be used to verify a user's public key (i.e., to detect and reject false certificates) is very much needed.

The goal of this paper is to develop a robust scheme to determine if a certificate is trustworthy. Our method uses redundant information to confirm the trustworthiness of a certificate. Previous work [16] has shown that redundancy (in the form of multiple, independent certificate chains) can be used to enhance trustworthiness. We show that in some circumstances multiple certificate chains do not provide sufficient redundancy. This is because a single malicious user may (legitimately) 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 determining if certificate chains are independent.

In addition, we investigate the implications of *conflicting certificates* (i.e., certificates which disagree about the public key of a user). Conflicts are simple to detect. We show that such conflicting information can be used to help identify 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. Section 2 summarizes related work. Section 3 defines our problem precisely. Section 4 presents solutions without using conflicting certificate information. Section 5 shows how conflicting certificates information may be used to improve performance. Section 6 presents our experimental results, and section 7 concludes.

2 Related Work

In addition to X.509 and PGP, discussed above, there are other public key infrastructures, such as SPKI/SDSI [6] and PolicyMaker [3]. These 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.

Existing work on improving the trustworthiness of webs of trust can be classified into two categories. In the first category, *partial trust* is used to determine trustworthiness of a target public key. In [18] and [13], this trustworthiness is based on the trust value in a single certificate chain. Multiple certificate chains are used in [2] and [11] to boost confidence in the result. [9] tries to reach a consensus by combining different beliefs and uncertainties. In [15], insurance, which may be viewed as a means of reducing risk, is used to calculate the trustworthiness 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 on the number of participants that may be malicious is assumed. An example is [4], which requires a bound on the minimum network connectivity in order to reach a consensus. Another important method in this category is [16]. This method suggests using multiple public key-independent certificate chains to certify the same key. The authors showed that if at most n public keys are compromised, a public key is provably correct if there are $n + 1$ or more public key-independent

certificate chains certifying it. Computation of the number of indepedent chains is accomplished by computing network flow in the certificate graph.

Methods in the first category (partial trust) are based on probabilistic models. We believe such models are more appropriate for computing the reliability of faulty systems than they are for computing trustworthiness of information provided by users. For example, such approaches require proper estimation of the trustworthiness of each user. If such estimates are incorrect, or a trusted user is compromised, then the output produced by these methods will be misleading. We believe methods in the second category are more suitable for the case of (intentionally) malicious users. That is, it should be much easier to bound the number of users who are malicious than to specify how trustworthy each user is.

A limitation of 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.

All methods for 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 one another. In [5], 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 by 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.

The next section presents definitions and assumptions, and a statement of the problems to be solved.

3 Problem Statement

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 (or valid) identity*. An identity which does not belong to a real user is a *false identity*.

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 true public key. If the user with identity x only has a single public key, j will be omitted for the sake of convenience.

Our definitions of *public key certificate* and *certificate chain* follow [14]. 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. 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 says k is her public key, 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*.

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 (i.e., the user with the corresponding identity says both of the two keys are her public keys, with the same index number). This may happen when a user x intentionally has two conflicting certificates issued to herself, by two separate parties, for the purpose of possessing more public keys. We expand our definition of a malicious user to also include x in such a case; the issuers of the conflicting certificates, however, are not considered to be malicious on this count, and the certificates are defined to be true.

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. We do not discuss further in this paper how certificates are distributed, which is an open problem.

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. 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* $G^x(V^x, E^x)$. 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. In this case, the two different head vertexes are called *conflicting vertexes*.

For the sake of simplicity, we add to the certificate graph an "oracle" vertex k_0. There is a directed edge in G^x from k_0 to every key which is assumed to be true (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 "paint" vertexes with different colors. A white vertex represents a public key that is either assumed or determined to be true for the identity on the edges directed

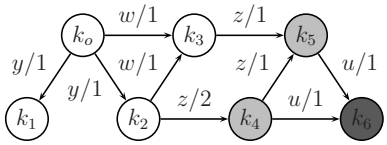

Fig. 1. User x's certificate graph

towards that vertex. A dark gray vertex represents a public key which is known to be false for the corresponding identity. If a public key is neither assumed to be true nor proved (yet) to be true for the corresponding identity, we paint it light gray.

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 and k_2 are two conflicting vertexes because the labels on their incoming edges are the same. k_5 is z's public key, with index number 1, and k_4 is z's public key, with index number 2. k_6 is a false public key.

When there is more than one malicious user, there may be a relationship between these users. Two malicious users who cooperate with each other to falsify information are said to be *colluding*. We say that two users x and y are colluding if either: (i) there exist two false certificates, one issued by x and one by y, and they agree with each other; or, (ii) x issues a false certificate upon y's request, or vice versa.

3.1 Problem Description

We now define the problems to be solved. First we consider the case in which malicious users do not collude, followed by the colluding case. The goals are the same, that is, to maximize the number of certificates which can be proved to be true.

Problem 1 *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.*

Problem 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* may collude*, maximize the number of certificates which can be proved to be true.*

It is necessary to have a metric to evaluate the performance of proposed solutions for the above problems. 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 for user x. K_a^x is the maximum set of true public keys that any method could determine to be true by means of certificate chains in this graph. Let K_0^x be the set of public keys that are assumed to be

true initially by user x, and K_M^x 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 metric definition now follows.

Definition 1 *Given a solution s to Problems 1 or 2 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, we propose using the weighted average of each user's performance:

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

4 Solutions to Problems 1 and 2

In this section, we present methods for solving problems 1 and 2 under two assumptions: (a) when there is only one public key per identity, and (b) when there may be multiple public keys per identity. We assume there is an upper bound on the number of users who may be malicious, and use redundancy to determine the certificates that must be true. We ignore in this section the case in which there are conflicting certificates, which is considered in section 5.

Two certificate chains are *public key-independent* if their head certificates agree, and their remaining certificates have no *public keys* in common. Two certificate chains are *identity-independent* if their head certificates agree, and their remaining certificates have no *identities* in common. We state the following theorems without proof (see [8]).

Theorem 1 *Given two identity-independent certificate chains and any number of* non-colluding *malicious users, the head certificates must be true.*

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

Theorem 2 *Given $n+1$ identity-independent certificate chains, if there are at most n colluding malicious users, then the head certificates must be true.*

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

4.1 Maximum of One Public Key Per Identity

Reiter and Stubblebine [16] considered and solved this problem. We summarize their results for the convenience of the reader. 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 tail vertex and 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 maximum number of vertex-disjoint paths from k_0 to k_t, in running time $O(|V||E|)$.

It can be shown that all certificates (edges) ending at k_t must be true if: (i) 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, (ii) 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 $n + 1$.

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. For example, in figure 1 there are two vertex-disjoint paths from k_0 to k_6. However, user z has two public keys, and the maximum number of identity-independent paths to k_6 is only 1. Therefore, it is not safe to conclude k_6 is u's true key if z may be malicious. This issue has not previously been addressed.

We still wish to use the notion of redundant, identity-independent paths to nullify the impact of 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*.

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 directed graph:

Theorem 3 *Given a certificate graph G^x and a vertex k_t. If one identity may legitimately be bound 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 can be found in [8]. Therefore, to solve this problem exactly is computationally expensive in the worst case.

A heuristic for this problem follows an idea from [16]. 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*)[7] 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, and can be found in [8]. The size of the maximum independent set in the transformed graph is the maximum number of label-disjoint paths 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 [10].

Alternatively, an exact solution may be computationally tractable if the required number of label-disjoint paths is small. Suppose we wish to solve the problem of whether there exists at least b label-disjoint paths in G^x from k_0 to k_t. 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. If we enumerate all subsets of labels in G^X with b or fewer labels, and no label-cut with b or fewer labels exists, then k_t is determined to be true. The algorithm runs in $O(|E||V|^b)$ (proof omitted).

In this section, we solved problems 1 and 2, without considering the possibility of conflicting certificates. We now turn to this problem.

5 Dealing With Conflicting Certificates

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. Stubblebine and Reiter [16] pointed out that conflicting certificates represent important information, but did not suggest how they could be used. We propose below a method of doing so, based on the notion of the *suspect set*:

Definition 2 *A suspect set is a set of identities that contains at least one malicious user. A member of the suspect set is called a suspect.*

We now describe how to construct suspect sets of minimum size, and how they can be used to determine more true certificates.

5.1 Constructing Suspect Sets (Non-Colluding Malicious Users)

Suppose we know or have determined a certificate is false by some means. If there is only a single malicious user, or multiple malicious users who are non-colluding, the true identity of the issuer of this false certificate must appear in

every chain ending with a certificate that agrees with this false certificate. Using this insight, we propose to construct suspect sets using the following algorithm. The algorithm takes a certificate graph G^x as input.

Algorithm 1 *constructing suspect sets*

```
1: for each label y/j in the certificate graph do
2:    for each dark gray vertex k_i whose incoming edge has a label y/j do
3:       construct a new suspect set consisting of the set of labels, each of which is a
         label-cut for k_0, k_i.
4:    end for
5:    for each light gray vertex k_i with an incoming edge labeled y/j, conflicting with
      a white vertex with an incoming edge labeled y/j do
6:       construct a new suspect set consisting of {the set of labels each of which is a
         label-cut for k_0, k_i} ∪ y
7:    end for
8:    for each pair k_i, k_h of light gray vertexes whose incoming edges both have a label
      y/j do
9:       construct a new suspect set consisting of {the set of labels each of which is a
         label-cut for either k_0, k_i or k_0, k_h} ∪y
10:   end for
11: end for
```

The intuition behind algorithm 1 is as follows. For each certificate known to be false, the malicious user's true identity must be a label-cut between k_0 and the false certificate. For each certificate conflicting with a true certificate, a malicious user has either purposely requested the conflicting certificate be issued to herself, or has issued the conflicting certificate. For each pair of conflicting but undetermined certificates (neither known to be true), any of either, both, or neither being true must be considered possible. The complexity of algorithm 1 is $O(|V|^3|E|)$ (proof omitted). We now explain how suspect set information can be useful.

5.2 Exploiting Suspect Sets (Non-Colluding Malicious Users)

Consider the case where there is a single malicious user. Let L_s represent the intersection of all the suspect sets generated by algorithm 1. 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 must determine the maximum disjoint sets (*MDS*) from all the suspect sets generated by algorithm 1. Two sets are called disjoint if they do not have any members (labels) in common. Suppose a solution to MDS consists of m suspect sets, and $m = b$. Let L_m be the union of these m sets. It is clear that all b malicious users must be in L_m.

Unfortunately, *MDS* is also NP-Complete, by transformation from the maximum independent set problem (proof omitted). As a result, the solution may only be approximated.

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 breadth-first search suffices, with a complexity of $O(b \cdot |V||E|)$.

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 decreasing order of priority.

Suspect set rule 1 *Given a certificate chain whose head certificate is 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 are conflicting 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 conflicting 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. To make use of the suspect sets constructed by these rules, 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. Next, all the edges with a label in L_c are deleted 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 greater than $b-a$. In this case, Theorem 2 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. The algorithm described previously for computing the minimum label-cut can be used to solve this problem.

We now present experimental evidence about the benefits of using certificate conflicts.

Table 1. Performance for PGP keyring before and after conflict detection, for a total of 10,000 public keys

# of colluding malicious users	1	2	3	4
# of provably true certificates before conflict detection	78	54	40	33
# of provably true certificates after conflict detection	9992	77	53	40

6 Experimental Results

We implemented and tested our conflict detection method to investigate its practicality and benefits. These experiments only considered the case of one legitimate public key per user. 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 keyservers, and the keyanalyze [17] tool was used to extract strongly-connected components. In addition, we synthetically generated keyrings to have a larger number of test cases. The synthetic data was generated by the graph generator BRITE [12]. 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 data point we report, 50 problem instances were generated (using a different random number generator seed each time); the values plotted are the average of these 50 instances.

The first experiment compares our method with the method of [16] on one of the largest PGP keyrings. The graph of this keyring contains 15956 vertexes (users) and 100292 edges (certificates). For this PGP keyring we emulated the behavior of n colluding malicious user as follows. First we randomly picked a target, and then n malicious users were randomly selected. The n malicious users issued n false certificates, one per malicious user, each binding the target's identity to the same false public key.

After emulating this behavior, we applied the method of [16] to determine the maximum set of true certificates. The resulting performance is the performance *before* conflict detection. Then we applied the suspect rules of colluding malicious users (from section 5.3) to find many suspect sets, from which we constructed L_c. For each of the remaining undetermined public keys, we made use of L_c and the method of section 5.3 to determine if it was true. The resulting performance is labeled the performance *after* conflict detection. Table 1 shows the results.

For this very large PGP keyring, it is not practical to evaluate the performance for all users. Instead, we randomly picked 200 users. For each user, we randomly selected 50 public keys on which to test our method. Each user's certificate graph was the entire keyring. The figure shows how many public keys can be determined to be true when there are different numbers of malicious users. This figure shows that when there is only a single malicious user, performance is greatly improved (by two orders of magnitude) with the use of conflict detec-

tion. In these experiments, the suspect sets turned out to be quite small. The performance with conflict detection dropped dramatically, however, for the case of two or more colluding malicious users, because of lack of sufficient redundancy in the certificate graph.

For our second experiment, we synthetically generated PGP keyrings. We emulated a single malicious user's behavior, as follows. We randomly picked a target, a malicious user, and n certificate issuers. The malicious user was assumed to ask 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 the algorithm for computing the minimum label-cut for the case of $b = 2$, and determined the maximum set of true certificates. The resulting performance was the performance *before* conflict detection. Then we applied algorithm 1 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. This gave the performance *after* conflict detection. Each test was run 50 times to obtain averages. The results were:

- Performance before conflict detection was 2%, regardless of the number of false certificates.
- Performance after conflict detection steadily increased from 4% (with 1 false certificate) up to 11% (with 19 false certificates).

The malicious user is faced with a dilemma (fortunately!). While increasing the number of false certificates should increase uncertainty about keys, it also becomes easier to narrow the list of "suspicious" users, thereby limiting the scope of the damage the malicious user can cause.

Our final experiment was for real PGP keyrings, and demonstrates how performance is improved by conflict detection. The results are shown in Figure 2. For all cases, performance increased when conflict detection was used. 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 again applied the method for determining if a label-cut of size b of less exists, for $b = 2$, to determine the maximum set of true certificates. The resulting performance is the performance *before* conflict detection. Then we used the method of section 5.3) to obtain the performance *after* conflict detection.

All experiments were performed on a Pentium IV, 2.0GHZ PC with 512MB memory. Running times for figure 2 ranged from 5 to 30 seconds, except for the graph with 588 vertexes, which required 10 hours of CPU time. We believe analysis of keyrings for robustness will be done infrequently, in which case these execution times should be acceptable.

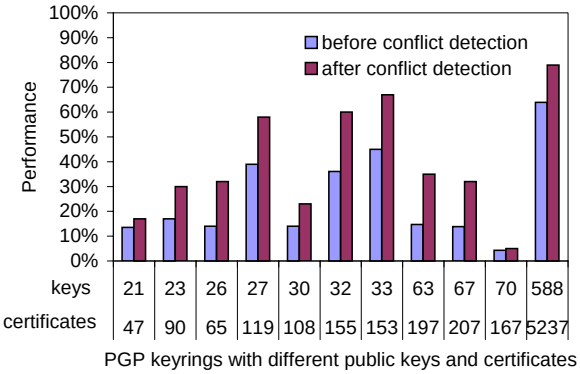

Fig. 2. Performance comparison for PGP keyrings before and after conflict detection

7 Conclusion and Future Work

In this paper, we described the problem of proving certificates are true in webs of trust (such as 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 necessary. Previous methods based on *public key-independent chains* are not sufficient under this assumption.

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.

Our results show that current PGP keyrings are not particularly resistant to attacks by malicious users, particularly colluding users. We are currently investigating ways to increase the robustness of webs of trust, such as PGP keyrings.

References

1. R. Ahuja, T. Magnanti, and J. Orlin. *Network flows : theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, N.J., 1993.
2. 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.
3. 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.

4. 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.
5. 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.
6. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. RFC 2693: SPKI certificate theory, September 1999.
7. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman & Co., 1979.
8. Qinglin Jiang, Douglas S. Reeves, and Peng Ning. Improving robustness of PGP keyrings by conflict detection. Technical Report TR-2003-19, Department of Computer Science, N.C. State University, October 2003.
9. Audun Josang. The consensus operator for combining beliefs. *Artificial Intelligence*, 141(1):157–170, 2002.
10. 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.
11. 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.
12. 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.
13. 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.
14. A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press, Boca Raton, Fla., 1997.
15. M. Reiter and S. Stubblebine. Toward acceptable metrics of authentication. In *IEEE Symposium on Security and Privacy*, pages 10–20, 1997.
16. M. Reiter and S. Stubblebine. Resilient authentication using path independence. *IEEE Transactions on Computers*, 47(12), December 1998.
17. M. Drew Streib. Keyanalyze - analysis of a large OpenPGP ring. http://www.dtype.org/keyanalyze/.
18. 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.
19. Int'l Telecommunications Union/ITU Telegraph & Tel. ITU-T recommendation X.509: The directory: Public-key and attribute certificate frameworks, Mar 2000.
20. Philip Zimmermann. *The official PGP user's guide*. MIT Press, Cambridge, Mass., 1995.