Rating Certificates

Eunjin (EJ) Jung
Department of Computer Science
University of Iowa
Email:ejjung@cs.uiowa.edu

Mohamed G. Gouda
Department of Computer Sciences
The University of Texas at Austin
Email:gouda@cs.utexas.edu

***Abstract*— We consider a system where each user has a public key and a private key. In this system, a certificate is a data item that is issued by one user u and contains the public key of another user v. A third user w that knows the public key of u can verify that this certificate has not been corrupted (by an adversary) since it was issued by u, and so can accept the public key in the certificate as the correct public key of v. User w can use this accepted public key of v in two ways. First, w can securely communicate with v. Second, w can obtain more public keys of other users, as it used the public key of u to obtain the public key of v. However, the safety of the second use is questionable if u, the issuer of the certificate, has concluded that it cannot trust v enough to accept a public key merely because v accepts it. To solve this problem, we propose that each certificate should have a "rating". The rating of a certificate describes how much trust the issuer puts on the subject concerning key acceptance. We present an algorithm for computing a subgraph $G.dst(src)$ of a certificate graph G, for a user src to find the correct public key of another user dst in G. The time complexity of this algorithm is $O(e)$, where e is the number of certificates in the system. This algorithm meets the lower bound of the worst case complexity.**

I. Introduction

We consider a system where each user u has a private key $R.u$ and a public key $B.u$. In this system, in order for a user u to securely send a message m to another user v, user u needs to encrypt the message m using the public key $B.v$, before sending the encrypted message, denoted $B.v\{m\}$, to user v. (This message may be a session encryption/decryption key for further secure communication.) This necessitates that user u know the public key $B.v$ of user v.

If a user u knows the public key $B.v$ of another user v in this system, then user u can issue a certificate, called a certificate from u to v, that identifies the public key $B.v$ of user v. This certificate can be used by any user in the system that knows the public key of user u to further acquire the public key of user v.

A certificate from user u to user v is of the following form:

$$\langle u, v, B.v, \texttt{info}, \texttt{sig}\rangle$$

This certificate is signed using the private key $R.u$ of user u, and it includes five items:

u	is the identity of the certificate issuer,
v	is the identity of the certificate subject,
$B.v$	is the public key of the subject v,
`info`	is other relevant information regarding this certificate such as expiration date, and
`sig`	is The digital signature of this certificate

The digital signature of a certificate is the encrypted message digest of this certificate. It is constructed by computing a message digest of all other four items in this certificate and encrypting the message digest with the private key $R.u$ of issuer u.

The certificates issued by different users in a system can be represented by a directed graph, called the *certificate graph* of the system. Each node u in the certificate graph represents a user u and its corresponding public and private key pair $B.u$ and $R.u$. Each directed edge (u, v) from node u to node v in the certificate graph represents a certificate $\langle u, v, B.v, \texttt{info}, \texttt{sig}\rangle$.

Any user x that knows the public key $B.u$ of user u can use $B.u$ to decrypt `sig` in (u, v). User x continues to compute the message digest of all other four items in the certificate. If the decrypted message matches the message digest computed by user x, then user x can accept the key $B.v$ in certificate (u, v) as the public key of user v.

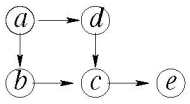

Fig. 1. A certificate graph example

Fig. 1 shows a certificate graph for a system with five users: a, b, c, d, and e. According to this graph,

user a issued two certificates (a, b) and (a, d)
user b issued one certificate (b, c)
user c issued one certificate (c, e)
user d issued one certificate (d, c)
user e issued no certificates.

A simple path $(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)$ in a certificate graph G, where the users $v_0, v_1, \cdots, v_k$ are all distinct, is called a *certificate chain* from v_0 to v_k in G of length k. v_0 is the *source* of the chain and v_k is the *destination* of the chain. Source v_0 of the certificate chain from v_0 to v_k may accept all the keys $B.v_1 \cdots B.v_k$ in these certificates as public keys of the users $v_1 \cdots v_k$ in this chain, respectively. For example, user a in Fig. 1 may use the certificate chain $(a, b)(b, d)$ to accept the public keys $B.b$ and $B.d$ of user b and user d.

The use of certificate chains to accept public keys of other users is based on the assumption that all the certificates in the

1-4244-0572-6/06/$20.00 ©2006 IEEE

chain are correct, i.e. all the keys in the certificates are indeed the correct public keys of the subjects. However, this may not be the case all the time. Some users may not take enough precautions before issuing certificates and as a result issue a certificate with a wrong public key for the subject. Some user may intentionally issue incorrect certificates to impersonate other users.

In a self-organized certificate system like Pretty Good Privacy (PGP) [1], any user can issue certificates and the problem of incorrect certificates may be worse. To mitigate the danger, a PGP user Alice may specify how much trustworthy another user is in issuing certificates. Alice puts the public keys of other users that she believes to be correct in her local *public key ring*. Each entry in the key ring contains an ID of another user, its public key and other information.

For each public key in her key ring, Alice can choose from four levels of trust, `completely trusted`, `marginally trusted`, `untrusted`, and `unknown` to assign to. (The default level of trust is `unknown`.) Note that this level of trust is *not* trust on whether the public key is the correct one for that user or not. In fact, having the public key in her key ring means that Alice believes that this is the correct public key of the user. Rather, the level of trust is trust Alice has on the public key as a signing key. In the default setting of PGP, for a user to accept a key $B.x$ as the public key of a user x, it needs to find either one `completely trusted` public key or two `marginally trusted` public keys in its own key ring, that sign certificates which has the same key $B.x$ as the public key of user x. Fig. 2 shows the example system with level of trust in each user's key ring. For example, David in the system in Fig. 2 `completely trusted` Carol's public key in his key ring, so David accepts Bob's public key in the certificate $(Carol, Bob)$.

PGP users can tune their own security settings by changing COMPLETES_NEEDED, MARGINALS_NEEDED, and CERT_DEPTH. The first two parameters indicate how many trusted keys are needed to accept a public key. CERT_DEPTH indicates how long a certificate chain can be to accept the public key in the last certificate in the chain.

However, there is a problem with the current level of trust scheme in PGP. First, the trust information is not shared with other users. In the example in Fig. 2, David `completely trusted` Carol's public key, and Carol has `untrusted` Bob's public key. There is a certificate chain from Carol to Alice, $(Carol, Bob)(Bob, Alice)$. However, Carol will not accept this public key of Alice because Carol has no trust on Bob's public key to introduce a new key. When David wants to obtain Alice's public key, David may want to use the certificate chain $(David, Carol)(Carol, Bob)(Bob, Alice)$. Even though David is relying on Carol's public key in the certificate chain $(David, Carol)(Carol, Bob)(Bob, Alice)$, David may make a different, in fact more vulnerable decision than Carol's and accept the public key in this chain as the public key of Alice.

One may reason that unclear semantics of the PGP trust levels cause this problem. When David assigns `completely trusted` to the public key of Carol, $B.Carol$, does it mean that David will accept any public key in a certificate signed by Carol, or David will accept any public keys in certificate chains starting with $(David, Carol)$? If it were the former, then David would not accept the public key of Alice in the certificate chain. However, this would prevent David from using any certificates issued by Bob. If it were the latter, then David would accept the public key of Alice, when Carol would not. The current specification of PGP does not specify the semantics clearly.

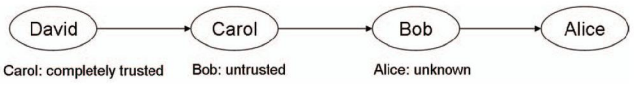

Fig. 2. An example of a PGP certificate system with level of trust

This example presents two challenges in a self-organized certificate system. First, users to share their trust level information with other users. David should be informed that Carol has assigned `untrusted` to Bob when he wants to use a certificate chain that includes $(Carol, Bob)$. This trust information needs to be signed so that other users can verify that this information is not tampered by an adversary. One way to share this trust information in a tamper-resistant manner is to add this trust information to certificates. In Section II, we propose a solution to this problem by adding "ratings" (trust information) to certificates. Second, we need efficient algorithms to aggregate the trust level information on many certificates. David uses three certificates $(David, Carol)$, $(Carol, Bob)$, and $(Bob, Alice)$ to find the public key of Alice. Our ratings would be useful for applications that are currently using PGP, as it is a small modification to current PGP mechanism and yet provides clearer semantics for trust information and efficient evaluation algorithm to evaluate the trust information. Also, these applications can keep their current settings of other security parameters such as `CERT_DEPTH` which limits the length of acceptable certificate chains.

In the following sections, we show what information can be added to certificates so that users can share their trust information about other users. We discuss the semantics of the added information. We present an algorithm that computes an optimal set of certificates that are necessary for a user to accept the public key of another user. Certificate revocation is an important part of a certificate system, but due to the space limitation the discussion is in [2]. In short, we could rely on Certificate Revocation List as PGP suggests, or any user can issue a "revocation certificate" to revoke one's own public key.

II. Ratings in Certificates

To tackle the first challenge discussed in the previous section, one could add trust information to a certificate. Consider the certificate graph in Fig. 3. David puts the level `untrusted` to the certificate $(David, Carol)$ so David can accept the public keys of Bob and Bill in the certificates issued by Carol. Carol puts `marginally trusted` on the certificates to Bob and Bill. Bob and Bill both issue certificates

for Alice that contain the same public key. By default, PGP users set MARGINALS_NEEDED to 2, which translates that Carol can accept the public key of Alice in the certificates issued by Bob and Bill, if Carol does not change the value of MARGINALS_NEEDED. However, it is not clear whether David should accept the public key of Alice or not.

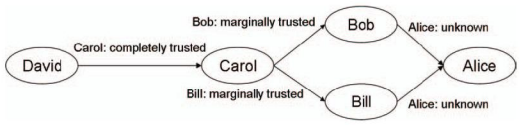

Fig. 3. An example of an extended PGP certificate system

Instead of adding trust information to a "signing" key as PGP does, we propose to add trust information as an "accepter". We add a new field called "rating" to a certificate. For example, if David completely trusts Carol as an accepter, then David accepts any public key accepted by Carol. This is different from PGP trust levels which is trust information that the issuer has on the key "signing" decision made by the subject. Now a certificate from user u to user v with a rating contains six items as follows:

$$\langle u, v, B.v, \texttt{rating}, \texttt{info}, \texttt{sig} \rangle$$

. u, v, $B.v$, info, and sig are the same as explained in Section I. rating is a rating which can be any one of the following that will be explained shortly:

accepted
independent
k-accepted

For simplicity, a certificate of a form

$$\langle u, v, B.v, \texttt{rating}, \texttt{info}, \texttt{sig} \rangle$$

is denoted $(u, v, \texttt{rating})$ where rating is

A	for	accepted
I	for	independent
$A(k)$	for	k-accepted

A certificate $(u, v, \texttt{rating})$ with rating provides two pieces of important information: the public key of user v and the rating of user v by user u. Any user w that knows the public key of user u can accept the public key of user v as long as they could verify the certificate. Moreover, user w can use the rating information in this certificate to decide whether to accept the public keys accepted by user v or not. Again, this is different from PGP trust levels, where user w can use the trust level information to decide whether to accept the public keys in certificates issued by user v or not.

1) accepted: A certificate of the form

$$\langle u, v, B.v, \texttt{accepted}, \texttt{info}, \texttt{sig} \rangle$$

indicates that u accepts $B.v$ as the public key of user v and that u also accepts any public key $B.x$ of a user x, if v accepts the same key $B.x$ as the public key of user x.

In Fig. 4, user u issues a certificate (u, v, A) and user v issues two certificates $(v, w, \texttt{rating})$ and $(v, x, \texttt{rating})$. Clearly v accepts both keys $B.w$ and $B.x$ as the public keys of users w and x, and the rating of the certificate (u, v, A) is accepted, so user u accepts the same keys $B.w$ and $B.x$ in certificates $(v, w, \texttt{rating})$ and $(v, x, \texttt{rating})$ as the public keys of user w and user x, respectively.

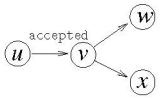

Fig. 4. An example of accepted certificates

Note that u will accept the public keys of users w and x in the certificate chains $(u, v, A)(v, w, \texttt{rating})$ and $(u, v, A)(v, x, \texttt{rating})$ regardless of the ratings in the certificates $(v, w, \texttt{rating})$ and $(v, x, \texttt{rating})$.

2) independent: A certificate of the form

$$\langle u, v, B.v, \texttt{independent}, \texttt{info}, \texttt{sig} \rangle$$

indicates that u accepts $B.v$ as the public key of user v. It also indicates that whether u accepts a key $B.x$ as the public key of a user x or not is independent of whether v accepts the same key $B.x$ as the public key of user x or not. In other words, user u does not accept any key $B.x$ as the public key of a user x just because v accepts the same key $B.x$ as the public key of user x.

In Fig. 5, user u issues two certificates (u, v, I) and (u, w, A), user v issues a certificate $(v, x, \texttt{rating})$, and user w issues a certificate $(w, x, \texttt{rating})$. Since user u issues a certificate to user v, u clearly accepts the public key $B.v$ of user v. However, user u would not accept $B.x$ as the public key of user x if there were only one certificate chain from u to x through v $(u, v, I)(v, x, \texttt{rating})$ in Fig. 5, because the rating of certificate (u, v, I) is independent. In the example certificate system in Fig. 5, there is another certificate chain from u to x $(u, w, A)(w, x, \texttt{rating})$, and the rating of (u, w, A) is accepted. Since w accepts $B.x$ as the public key of user x, u accepts $B.x$ as well.

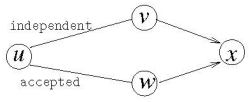

Fig. 5. An example of independent certificates

3) k-accepted: A certificate of the form

$$\langle u, v, B.v, k\text{-}\texttt{accepted}, \texttt{info}, \texttt{sig} \rangle$$

indicates that u accepts $B.v$ as the public key of user v. It also indicates that u accepts a key $B.x$ as the public

key of user x, if u has also issued certificates for users $v_1 \cdots v_{k-1}$,

$$(u, v, A(k)), (u, v_1, A(j_1)), \cdots, (u, v_{k-1}, A(j_{k-1})),$$

where each $j_i \leq k$, and each v_i accepts the same $B.x$ as the public key of user x. (Intuitively, the issuer has $\frac{1}{k}$ of the full trust on the subject.)

In Fig. 6, user u issues three certificates $(u, v, A(2))$, $(u, w, A(3))$, and $(u, y, A(3))$. User v issues a certificate $(v, x, \texttt{rating})$, user w issues a certificate $(w, x, \texttt{rating})$, and user y issues a certificate $(y, x, \texttt{rating})$. Clearly users v, w, and y accept the key $B.x$ in $(v, x, \texttt{rating})$, $(w, x, \texttt{rating})$, and $(y, x, \texttt{rating})$ as the public key of user x. User u issued three certificates $(u, v, A(2))$, $(u, w, A(3))$, and $(u, y, A(3))$, so it also accepts the key $B.x$ in the certificates $(v, x, \texttt{rating})$, $(w, x, \texttt{rating})$, and $(y, x, \texttt{rating})$ as the public key of a user x. Note that the certificates (v, x), (w, x), (y, x) must include the same key $B.x$ as the public key of user x for user u to accept it.

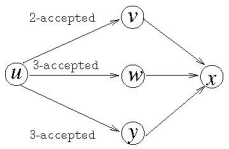

Fig. 6. An example of k-`accepted` certificates

These three ratings are very expressive and cover a wide range of trustworthiness. The `accepted` rating can be used for Certificate Authorities, while `independent` rating can be used for a user that cannot be trusted to issue any certificate at all. In fact, `accepted` is equivalent to `1-accepted` and `independent` is equivalent to ∞-`accepted`.[1] Note that the semantics of these ratings are for key acceptance, not for key signing. An issuer of a certificate can assign the rating for the subject depending on whether to accept a key just because this subject accepts the key or not. Even though this trust information will be shared with all other users, the decision itself is a local decision of the issuer.

III. Key Acceptance Examples

In the previous section, we defined three ratings for certificates, and showed their semantics. Consider the example certificate graph G in Fig. 7. Each edge (u, v) in G is labeled with the rating of the corresponding certificate $(u, v, \texttt{rating})$. Based on these ratings, we can compute the set C of users that can accept $B.dst$ as the public key of user dst.

The set C can be computed in three steps.

1) First, observe that each of the users x and y can accept $B.dst$ as the public key of user dst, since each of them has already issued a certificate declaring that the public key of dst is $B.dst$. Note that the rating in the certificate (x, dst, I) is `independent`. No matter what the rating is in the certificate (x, dst), x declared that the public key of dst is $B.dst$ in this certificate, so x accepts $B.dst$ as the public key of user dst.
2) Second, user v can accept $B.dst$ as the public key of user dst since v has issued two certificates with a rating $A(2)$ each, for users x and y (and x and y can accept $B.dst$ as the public key of user dst from the first step of our discussion). Note that user w cannot accept $B.dst$ as the public key of user dst even though w has issued a certificate for user x (and x can accept $B.dst$ as the public key of user dst, as discussed above), since the rating of the issued certificate (w, x, I) is `independent`.
3) Third, user u can accept $B.dst$ as the public key of dst since u has issued a certificate to v with rating A (and v can accept $B.dst$ as the public key of user dst from the second step of our discussion).

The set C is therefore:

$$\{dst, x, y, v, u\}$$

[1] Infinity can be implemented as a number that is larger than the number of users in the system.

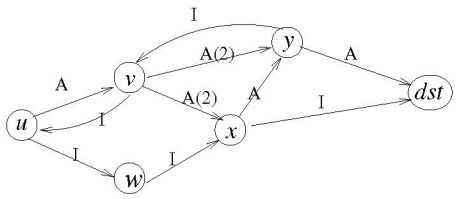

Fig. 7. An example of a certificate graph with ratings

In the following section, we present an algorithm that computes a subgraph $G.dst(src)$ of a certificate graph, for user src to decide whether to accept the public key of dst in $G.dst(src)$. Note that this input certificate graph does not have to be the full certificate graph of the system. For example, user u in the example above does not need the certificate chain $(u, w)(w, x)(x, dst)$, since the ratings in (u, w) and (w, x) are `independent`. In [2], we discuss how to compute the input certificate graphs for the following two algorithms.

IV. Key Acceptance Algorithms

The second challenge discussed in Section I was to design an efficient algorithm to aggregate trust information in many certificates. We assume that when a user src wishes to find a public key of another user dst, these two users would gather a set of certificates. The users may use locally stored certificates or send query messages to other users. We denote such set of certificates $G.dst$. Given a subgraph $G.dst$, Algorithm 1 computes a subgraph $G.dst(src)$ that can be used by user src to decide whether to accept the public key of user dst in $G.dst$ or not. If there are not enough certificates to accept the public key of user dst, $G.dst(src)$ will be null. In other words, if $G.dst(src)$ is not null, the returned $G.dst(src)$ is a proof for user src that src can accept the public key of user dst.

ALGORITHM 1 computes a subgraph $G.dst(src)$

INPUT: a certificate graph $G.dst$ and a user src in G
OUTPUT: a subgraph $G.dst(src)$

STEPS:
1: **if** $src \notin G.dst$ **then return** {}
2: **for** each user u in $G.dst$, $added[u]$:=**false**
3: $G.dst(src) := \{\}$; $Z := \{\}$;
4: **for** each certificate $(u, dst, \texttt{rating})$ in $G.dst$ **do**
5: $G.dst(src) := G.dst(src) \cup \{(u, dst, \texttt{rating})\}$;
6: add all the certificates $(x, u, \texttt{rating})$ issued for u in $G.dst$ to Z
7: **endfor**;
8: **for** each user u in $G.dst$, c[u] := 0;
9: **while** $added[src]$=**false**
10: **for** each certificate $(u, v, \texttt{rating})$ in Z **do**
11: remove $(u, v, \texttt{rating})$ from Z;
12: **if** $(u, v, \texttt{rating})$=(u, v, A) and $added[u]$=**false**
13: **then** $G.dst(src) := G.dst(src) \cup \{(u, v, A)\}$;
14: add all the certificates $(x, u, \texttt{rating})$ issued for u in $G.dst$ to Z
15: **else if** $(u, v, \texttt{rating})$=$(u, v, A(k))$ and $added[u]$=**false**
16: **then** c[u] := c[u]+$\frac{1}{k}$;
17: **if** c[u]$\geq$1
18: **then** add all the certificates issued by u in Z to $G.dst(src)$;
19: add all the certificates $(x, u, \texttt{rating})$ issued for u in $G.dst$ to Z
20: **endfor**;
21: **endwhile**;
22: **if** $added[src]$=**false then return** null
23: **else return** $G.dst(src)$

For example, assume that user v wants to find the public key of user dst in Fig. 7. User dst has the subgraph $G.dst$ shown in Fig. 8. [2] User dst can use Algorithm 1 to compute the subgraph $G.dst(v)$ shown in Fig. 9. $G.dst(u)$ for user u to accept the public key of dst is the same as $G.dst$. For users x and y, each of them needs only one certificate to accept the public key of dst. Fig. 10 shows $G.dst(y)$.

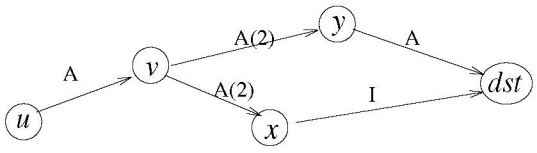

Fig. 8. $G.dst$ of the certificate graph in Fig. 7

The time complexity of Algorithm 1 is $O(e)$, where e is the number of certificates in $G.dst$. The proof of correctness and the complexity is in [2]. A certificate in $G.dst$ is added to $G.dst(src)$ at most once, so the complexity is $O(e)$. Also, the resulting $G.dst(src)$ is *optimal* in the sense that all the certificates in $G.dst(src)$ are necessary for user src to accept the public key of dst. Algorithm 1 includes the certificates issued by users that are on the chain from src to dst, so all the certificates in $G.dst(src)$ are necessary for user src to accept the public key of dst, i.e. $G.dst(src)$ is optimal.

[2]The algorithm that computes the subgraph $G.dst$ for user dst is in [2]

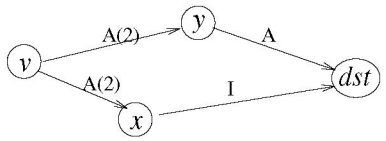

Fig. 9. $G.dst(v)$ computed by Algorithm 1

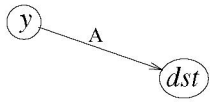

Fig. 10. $G.dst(y)$ computed by Algorithm 1

This algorithm is optimal in the sense that it meets the lower bound of the worst case complexities of any algorithms that compute the set of users that accept $B.dst$ as the public key of user dst. The example certificate graph in Fig. 11 shows when any algorithm has to consider all the three certificates to compute the set. In fact, any certificate graph that has only one certificate chain from any user to user dst, where all the certificates have `accepted` ratings, needs exactly e steps to find all the users that can accept the public key of user dst. Therefore, Algorithm 1 meets the lower bound of the worst case complexity.

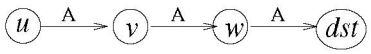

Fig. 11. A certificate graph with the worst case complexity

In [2], we present more algorithms that help computing necessary certificate subgraphs. In particular, we show an algorithm that computes a subgraph for each user in a distributed system so that users src and dst can compute $G.dst(src)$ more efficiently.

V. Related Work

SSL/TLS [3] uses X.509 certificates [4] which do not have any ratings in themselves, but in practice, users accept any certificates issued by trusted certificate authorities such as VeriSign. This is equivalent to having certificates with rating `accepted` from users to trusted certificate authorities.

The traditional PGP system is discussed in Section I. OpenPGP [6] is the most recent version of PGP specification. It has "level" in trust signatures. Generally, a level n trust signature asserts that a key is trusted to issue level $n-1$ trust signatures. This "level" information is to control the length of acceptable certificate chains.

SDSI/SPKI [7] supports a boolean delegation flag in their certificates to show that the subject of the certificate can authorize another principal. When this flag is false, no more certificates can be used after this certificate. This is equivalent to having only `accepted` and `independent` ratings in the certificates. Our ratings are more expressive with k-`accepted`.

KeyNote system [8] is a trust management system where certificates bind public keys with authorizations, instead of binding public keys to identities. It has a sophisticated language to describe policies, and each query evaluation includes Policy Compliance Value calculation. KeyNote system is more powerful in expressiveness, whereas our paper aims to help simpler certificate systems with finding the correct public keys.

Two frameworks in [9], [10] are based on certificates with probabilities to express the (un)trustworthiness of subjects. The semantics of these probabilities are defined and algorithms to evaluate certificate chains based on them are presented. The framework in [11] chooses from 6 levels of trustworthiness of subjects, and 6 levels of sureness of public keys in certificates.

Trustworthiness information in certificates is to increase the likelihood of accepting the correct public keys of other users. In addition to using the information in certificates, requiring multiple disjoint paths from a user to another user is suggested in [12]. Algorithm 1 returns a subgraph for user src to decide whether to accept the public key of user dst in the subgraph. The metric of multiple disjoint paths could be applied to this subgraph. In [13], the same authors also suggests adding financial evaluation to certificates.

Finding the correct public key of other users may be even more difficult in ad hoc networks due to mobility and unreliable availability. In [14], each user stores a set of certificates so that users do not need to rely on other users in search for certificate chains. In [15], each user issues certificates with *trust* and *confidence* values. The trust value is how much trust the issuer has on the subject being trustworthy and the confidence value is how much trust the issuer has on the public key in the certificate being correct. Users use these values to decide whether to accept public keys in certificate chains based on the theory of semirings.

VI. Conclusion

We have discussed the problem of trusting certificates in self-organized certificate systems, such as PGP, and proposed a solution to this problem by adding ratings to certificates. Table I shows a summary of comparisons among the solutions to the problem of trusting certificates in self-organized certificate systems. The first column shows whether each scheme can share the trust information among users or not. Except for PGP, every scheme supports the trust information by adding it as a field in a certificate. The second column shows the granularity of the trust information of each scheme. The ratings and the two schemes using probabilities support the issuer to specify partial trust for each certificate. However, in PGP a user has one parameter to apply to all the certificates for partial trust. SPKI/SDSI has only one boolean flag to share the trust information, so it does not support partial trust. The third column shows the complexity to use each scheme in evaluating trust information. PGP does not share the trust information, so the complexity is not specified. The complexity of using ratings, and SPKI/SDSI is low since they only require simple arithmetic operations and offer clear semantics. However, the two schemes based on probabilities require more complicated operations and it is hard for an average user to understand the semantics of these operations.

	shared trust info	**partial trust**	**complexity**
PGP	no	per user	not specified
OpenPGP	yes	per user	low
SPKI/SDSI	yes	no	low
probabilities	yes	per certificate	high
ratings	yes	per certificate	low

TABLE I
Comparison of Trust Information schemes

We also show an algorithm that computes a subgraph $G.dst(src)$ for user src, whin contains all the necessary certificates for src to verify and decide whether to accept the public key of dst or not.

References

[1] P. Zimmerman, *The Official PGP User's Guide*, MIT Press, 1995.

[2] Eunjin Jung and Mohamed G. Gouda, "Rating certificates," Tech. Rep. TR06-03, Dept. of Computer Science, University of Iowa, 2006.

[3] Tim Dierks and Eric Rescorla, "The TLS protocol version 1.1," Internet Draft (draft-ietf-tls-rfc2246-bis-08.txt), 2004.

[4] C. Adams, S. Farrell, T. Kause, and T. Mononen, "Internet X.509 public key infrastructure – certificate management protocol (CMP)," RFC 2510, 1999.

[5] Søren Peter Nielsen, Frederic Dahm, Marc Lüscher, Hidenobu Yamamoto, Fiona Collins, Brian Denholm, Suresh Kumar, and John Softley, "Lotus notes and domino r5.0 security infrastructure revealed," 1999.

[6] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP message format," RFC 2440, 1998.

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

[8] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote trust-management system version 2," RFC 2704, 1999.

[9] U. Maurer, "Modeling a public-key infrastructure," in *Proceedings of the European Symposium on Research in Computer Security (ESORICS '96)*. 1996, Springer-Verlag.

[10] T. Beth, M. Borcherding, and B. Klein, "Valuation of trust in open networks," in *Proceedings of the European Symposium on Research in Computer Security (ESORICS '94) LNCS 875*. 1994, pp. 3–18, Springer-Verlag.

[11] Alfarez Abdul-Rahman and Stephen Hailes, "A distributed trust model," in *NSPW '97: Proceedings of the 1997 workshop on New security paradigms*, 1997.

[12] Michael K Reiter and Stuart G. Stubblebine, "Resilient authentication using path independence," *IEEE Transactions on Computers*, vol. 47, no. 12, pp. 1351–1362, December 1998.

[13] Michael K. Reiter and Stuart G. Stubblebine, "Authentication metric analysis and design," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 2, pp. 138–158, 1999.

[14] Jean-Pierre Hubaux, Levente Buttyán, and Srdjan Capkun, "The quest for security in mobile ad hoc networks," in *Proceedings of the 2001 ACM International Symposium on Mobile ad hoc networking & computing*. 2001, pp. 146–155, ACM Press.

[15] George Theodorakopoulos and John S. Baras, "Trust evaluation in ad hoc networks," in *Proceedings of the 2004 ACM workshop on wireless security*, October 2004.