Secure communication in an unknown network using certificates

Mike Burmester *
Information Security Group
Department of Mathematics
Royal Holloway – University of London
Egham, Surrey TW20 OEX, UK
E-mail: m.burmester@rhbnc.ac.uk

Yvo Desmedt *
Center for Cryptography, Computer and
Network Security, Department of EE & CS
University of Wisconsin – Milwaukee
P.O. Box 784, WI 53201 Milwaukee, USA
E-mail: desmedt@cs.uwm.edu

Abstract

We consider the problem of secure communication in a network with malicious (Byzantine) faults for which the *trust graph*, with vertices the processors and edges corresponding to certified public keys, is *not* known except possibly to the adversary. This scenario occurs in several models. For example, in survivability models in which certifying authorities may be corrupted, or in networks which are being constructed in a decentralized way. We present a protocol with which secure communication can be achieved in this case, provided the trust graph is sufficiently connected.

Keywords: Network security, Byzantine faults, Privacy, Integrity, Reliability.

1 Introduction

Secure communication in an open and dynamic network in the presence of a malicious adversary can only be achieved when the messages are authenticated. For this purpose we use authentication channels. There are several ways to establish such channels. For example, we can use dedicated communication lines in the network. Alternatively, shared secret keys or public keys can be used. The graph with vertices the processors in the network and edges the authentication channels is called a *trust graph* [4]. If the sender is connected to the receiver by an edge in this graph then the messages can be authenticated through the corresponding channel. Otherwise we may use authentication paths through intermediary processors in the trust graph [18,4].

The interplay of network connectivity and communication security has been studied extensively in recent years (see *e.g.* [6,13,5,1,9,20]). Dolev [6] and Dolev–Dwork–Waarts–Yung [7] have shown that if the number of malicious (Byzantine) faulty nodes is bounded by u then secure communication can only be achieved if the network is at least $(2u+1)$ connected. Even if the faults are not malicious, for reliable communication the network must be at least $(u+1)$ connected [13].

*Research supported by DARPA F30602-97-1-0205.

In this paper we deal with the case when the trust graph is not known except possibly to the adversary. It is well known [2] that in this case when there are no malicious faults, we can achieve secure communication if there are at least $(2u+1)$ vertex-disjoint paths which connect the sender and receiver, where u is the number of faulty nodes. Recently this result has been extended to include the case when the faults are malicious (Byzantine), provided the trust graph is known to at least one non-faulty processor [4]. However the case when the trust graph is not known to any (non-faulty) processor and when there are malicious faults has not been investigated. In particular, no efficient algorithm for constructing the trust graph in this case has been presented so far.

In this paper we focus on the case when the authentication in the trust graph $G^* = (V^*, E^*)$ is based on public keys (using signatures), with edges $(x, v) \in E^*$ corresponding to certificates in which v certifies the public key of x (by signing it). We consider the problem of secure communication when the public key of the sender is not certified by the receiver [17], and when the structure of the trust graph is not known to the sender or the receiver. We describe an algorithm for this setting which makes it possible for the sender to compute a good approximation of the trust graph in polynomial time if the vertex-connectivity of the trust graph is at least $2u+1$, where u is an upper bound on the number of faulty (Byzantine) processors.

Related work and motivation

This problem is an extension of the classic Byzantine generals problem [16,6,14,7] and is related to dependable computation. Authentication in open networks was considered by Beth-Borcherding-Klein [3] and Maurer [15]. Reiter-Stubblebine [18] consider trust-paths for authentication, and similarly Burmester-Desmedt-Kabatianski [4], but they use a slightly different model.

Goldreich–Goldwasser–Linial [12], Franklin–Yung [10] and Franklin–Wright [9] have studied broadcast (multi-recipient) models. Franklin–Wright [9] have shown that if the number of Byzantine faults is bounded by u then we have secure communication if the number of broadcast lines is greater than $\lceil 3u/2 \rceil$. This bound has been recently lowered to u [20].

In our scenario the sender has only local information about the the trust graph, and in order to communicate securely with other processors, must find appropriate communication paths through possibly corrupted processors. This situation occurs in several models, as for example in survivability models for which the certifying authorities may be corrupted and only local data is reliable. In this case the certifying authorities can provide erroneous (made-up) keys in order to decrypt private messages and to sign fraudulent messages [17]. This model is used by Zimmermann [21], Reiter–Stubblebine [18] and Rivest–Lampson [19]. We use a similar model, only for us the trust graph is not known (in [21] and [19] the trust graph is known).

2 Model and primitives

A communication system consists, essentially, of several linked processors, such as servers, programs, hardware units etc. A basic requirement is that the system should be reliable and dependable (robust). Reliability usually deals with faults which follow a random pattern, *e.g.,* accidental faults. These faults follow predictable patterns and are usually independent of each other at their origin [4]. They can be controlled by using redundancy (replication). Dependability deals with malicious (Byzantine) faults which are more difficult to deal with. The usual scenario is for an adversary to control all the faulty processors, according to some plan which may exploit the possible weaknesses of the system. The adversary has at least as much power and knowledge of the state of the system as the non-faulty processors (excluding the secret keys), and possibly more. In our case, for example, the adversary may know the structure of the trust graph, whereas the receiver will not. The adversary may try to use such information and forward misleading messages to non-faulty processors in an attempt to make the system fail, as in the case of the *bogus paths* attack [4].

Modeling a scenario in which the adversary is malicious should allow for a dynamic topology in which changes in the system may take place without the (non-faulty) processors being aware of it. It should allow for the most general type of processor which could represent a simple gate, a software package, or a powerful computer. Also, the model should describe the structure of the system at an appropriate level of abstraction: it must distinguish those aspects which are relevant to the computation and abstract out those aspects which are not essential.

2.1 The trust graph

The *trust graph* $G^* = (V^*, E^*)$ is a directed graph with vertices the processors of the network and edges the authentication channels. These channels can be quite general. They can be physically secure (dedicated) communication lines. Alternatively, they can correspond to conventional authentication channels with shared secret keys, or to authentication channels with public keys. We are not interested in how these channels are implemented, or how the underlying (real) communication network is used, and we do not make any specific requirements other than that these channels are reliable and that we have synchrony (delays are bounded). This issue will be discussed in Section 6.1.

We are concerned with the secure communication between the (non-faulty) processors of the trust graph. Since this can be achieved when this graph is known [4], or when it is completely connected, we focus on the case when its structure is not known and when it has a weaker connectivity. Our goal is to find a polynomial time algorithm to construct the trust graph, or at least a good approximation of it. We summarize our basic requirements for this model below.

General assumptions

- The authentication channels are reliable and we have synchrony (delays are bounded).
- All processors including the adversary are polynomially bounded (for unconditional security we allow the adversary to have unlimited computer power).
- The number of faulty processors is bounded by u, and the trust graph is $2u + 1$ vertex-connected [8].
- Every processor has a unique identifying label.

In this paper we focus on the particular case when the authentication in the trust graph $G^* = (V^*, E^*)$ is based on public keys, with signatures. Each edge $(x, v) \in E^*$ is labeled by a certificate $c_{xv} = (x, v, k_x, k_v, sign_{k_v}(x, k_x))$, where k_x, k_v are the public keys[1] of x, v, and $sign_{k_v}(x, k_x)$ is the signature of v with key k_v on (x, k_x).

[1] We shall assume that the length of the public keys is superpolynomial in u.

Observe that a processor v can be identified by its public key k_v. This does not restrict the generality of our approach, since we are assuming that the processors have unique identifying labels. When there is no ambiguity we may identify the processor v with the key k_v. It is important however to note that in our model there is a clear distinction between the label of a vertex v and its public key k_v. Indeed in our scenario a processor b wishes to communicate with a (possibly known) processor v which has not certified the public key of b. A variant scenario is when the processor b wishes to communicate with a processor v which does *not know* b's public key. This issue will be discussed in more detail in Section 6.2. We just mention here that if a trust graph (or a good approximation of it) has been constructed with vertices labeled by public keys, then it is easy to find the "true" labels of the vertices, provided the trust graph is sufficiently connected (by querying each "public key" for its label).

2.2 A good approximation of the trust graph

The vertices of the trust graph correspond to faulty and non-faulty processors. Similarly, the edges of the trust graph correspond to faulty and non-faulty channels. Faulty processors (channels) are real processors (channels) which are under the control of the adversary. It may not be possible in the general case for a non-faulty processor to construct the true trust graph, because its faulty neighbors under the control of the adversary can always lie about their neighbors (and the neighbors of their neighbors, ...), *e.g.* by claiming or disclaiming incident edges. That is, faulty processors can fabricate non-existing processors and channels to prevent non-faulty processors from finding the trust graph. We call these processors and channels, *fake*. Fake processors (channels) do not belong to the trust graph. They correspond to vertices (edges) of a virtual graph which is an extension of the trust graph.

We say that a graph $G' = (V', E')$ is a *good approximation* of the trust graph $G^* = (V^*, E^*)$ if $V' = V^*$ and $E' \subseteq E^*$, where the edges in $E^* - E'$ are directed into a faulty vertex. Good approximation graphs are adequate for secure communication, provided our assumptions in Section 2.1 are satisfied. Observe that the adversary can reduce the connectivity of the trust graph from $\lfloor \frac{5}{2}u + 1 \rfloor$ to $(2u+1)$ by removing (disclaiming) $\lfloor \frac{u}{2} \rfloor$ edges which connect u faulty processors in pairs. For example, in Figure 1 two faulty vertices f_1, f_2 can reduce the connectivity between the two vertices v and b from 5 to 4 in such a way that they can still control 2 out of 4 vertex-disjoint paths linking v to b (if f_1 disclaims the edge (f_1, f_2)). This implies that if a message is sent through these paths, then a majority vote on the received communication may not be decisive. Of course there may be other sets consisting of 4 vertex-disjoint paths which also link v, b for which f_1, f_2 have a minority vote, but it may be hard to find these in the general case.

Any further removals by the adversary will however only reduce the number of faulty or fake paths, and will not affect the $(u+1)$ vertex-disjoint authentication paths which are not faulty.

2.3 Virtual paths

A path $\pi = (b_1, b_2, \ldots, b_n)$ is *virtual* if every processor b_ℓ, $\ell > 1$, in π has certified (signed) with its public key k_{b_ℓ}, its parent $b_{\ell-1}$ and the key $k_{b_\ell - 1}$. The description of a virtual path π must include all the certificates $c_{b_{\ell-1}b_\ell} = (b_{\ell-1}, b_\ell, k_{b_{\ell-1}}, k_{b_\ell}, sign_{k_{b_\ell}}(b_{\ell-1}, k_{b_{\ell-1}}))$, $\ell = 2, \ldots, n$. To authenticate a message m through π, each processor b_ℓ in turn signs (m, π) and forwards this, together with the signatures of its ancestors to its descendants. The authentication is initialized by processor b_1

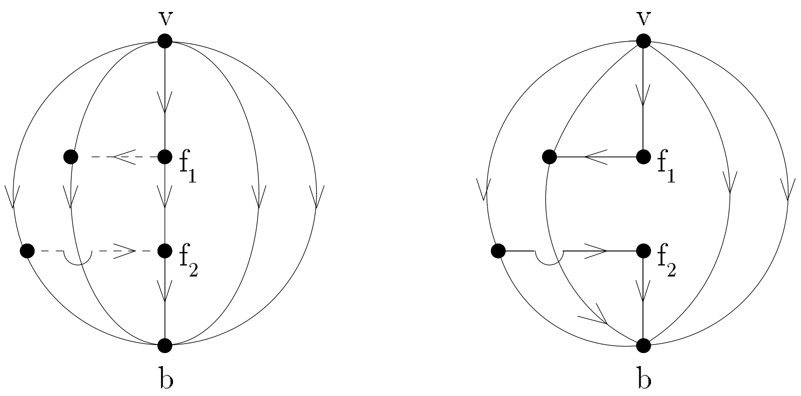

Figure 1: Two faulty vertices f_1, f_2 reduce the connectivity between v and b from 5 to 4 in such a way that they control 2 out of 4 vertex-disjoint paths from v to b.

which forwards $\text{sign}_{b_1,\pi}(m) := (m, \pi, \text{sign}_{b_1}(m, \pi))$ to b_2. Each processor b_ℓ in π, $\ell = 2, \ldots, n-1$, on receiving $\text{sign}_{b_{\ell-1},\pi}(m)$ checks it for correctness, *i.e.*, that the keys certify what they are supposed to and that the signatures are valid. If not, the message is not forwarded. Otherwise, processor b_ℓ appends to the message its signature $\text{sign}_{b_\ell}(m, \pi)$ and then forwards the resulting list $\text{sign}_{b_\ell,\pi}(m)$ to $b_{\ell+1}$. The message m is authenticated through the virtual path π when processor b_n receives a valid $\text{sign}_{b_{n-1},\pi}(m)$. A path whose processors belong to the trust graph, *i.e.,* are not fake, is an *authentication* path. A message authenticated through such a path with no faulty processors is authentic. However if some processors are faulty then the message (*e.g.*, a certificate) may be a forgery.

If a path π has faulty or fake processors then it is not certain if the message will ever reach processor b_n. A faulty processor b_ℓ can claim to its descendants that a message has been authenticated by its ancestors $b'_{\ell-1}, \ldots,$ but processor b_n cannot be certain, (i) that the message, if any, has been substituted and, (ii) that some of the processors $b'_{\ell-i}$ and edges $(b'_{\ell-i}, b'_{\ell-i+1})$ on the claimed path are not fake (not in the trust graph).

Definition. A *fake path* is a virtual path some of whose vertices are fake.

The following result will be needed later.

Lemma 1 *Let $\pi = (b_1, b_2, \ldots, b_n)$ be a virtual path. If the vertex b_n is not faulty and π is fake, then at least one ancestor b_ℓ of b_n in π must be faulty.*

Proof. The public key of a fake processor will only be certified by a faulty processor. $\square$

Virtual paths can be used for secure communication if no more than u processors are faulty and if $(2u+1)$ vertex-disjoint such paths connect the sender to the receiver. By the Lemma, if the sender and receiver are not faulty, at least $(u+1)$ of these paths are non-faulty authentication paths. A majority vote on the received communication can then be used.

Another way to communicate in a network which is under the control of a malicious adversary is by flooding.

2.4 Flooding

Flooding [2] is a broadcasting method in which a processor v sends a message to its neighbors, which then relay it to their neighbors, and so on, until the message reaches all the processors in the trust graph (our connectivity assumption guarantees this). To limit the number of transmissions, a processor does not relay back a message to the processor which sent it. Also, transmissions are not repeated (by using sequence numbers). If the the adversary does not make any fake processors or channels (*e.g.*, if the faults are not malicious) then the total number of transmissions for one query through the trust graph is bounded by $2|E^*|$, where E^* is the edge set of the trust graph [2].

In our case the faulty processors are under the control of the adversary and will make fake processors and channels, and furthermore they may try to jam the system by claiming to have a large number of (mostly fake) neighbors. To prevent this we introduce Round Robin Flooding [2]. In this, each processor v allocates "equal-time" to all its edges. For convenience we take the delay-time of the authentication channels (the edges) in the trust graph G^* to be bounded by 1. Then v will allocate to its incident edges time bounded by $deg(v)$, the degree of v in G^*. This means that the time taken for a query of a non-faulty processor to reach any other processor in G^* is bounded by n^2, where $n = |V^*|$, provided G^* is $(u+1)$ vertex-connected, with u an upper bound on the number of faulty processors. Since we only use Round Robin Flooding, from now on we shall refer to this simply as flooding.

3 Secure communication with Byzantine faults

We can formulate our problem of secure communication in terms of communication networks. These networks can be represented by graphs $G = (V, E)$ in which communication is possible only through the edges of G (we assume that these are reliable). In our case up to u vertices in G may be faulty and under the control of the adversary. Communication through these may be corrupted. In particular, a faulty vertex can lie about its neighbors. If the graph has sufficient connectivity then secure communication can be achieved through $(2u+1)$ vertex-disjoint communication paths, since the adversary can only occupy less than half of these. However in our case the structure of G is not known. The problem is to find an efficient algorithm to construct G, or at least a good approximation of G, in the case when the adversary can control up to u vertices. There are two different ways in which this problem can be formulated.

CN1 – *Constructing a communication network with up to u faulty vertices and* $Adversary_u$.
Instance: A directed graph $G = (V, E)$, $b \in V$, the set N_b of neighbors of b in V, the set E_b of edges in E incident to b, the $Adversary_u$ which can control up to u vertices in G.
Question: Can a good approximation of G be constructed given as input only $b \in V$, N_b and E_b, in the presence of $Adversary_u$.

In this problem, vertex b only knows its neighbors in N_b and the corresponding edges E_b of the communication graph, and does not have access to the program of $Adversary_u$. It can find out information about the graph G by communicating through its neighbors (or by guessing). We assume that the adversary has a *fixed* program. This restriction is removed in the next problem.

CN2 – *Constructing a communication network with up to u faulty vertices and any adversary.*
Instance: A directed graph $G = (V, E)$, $b \in V$, the set N_b of neighbors of b in V, the set E_b of

edges in E incident to b.
Question: Can a good approximation of G be constructed given as input only $b \in V$, N_b and E_b, in the presence of *any* adversary which can corrupt up to u vertices in G.

This problem addresses malicious faults in a general way. It gives more power to the adversary who can change dynamically her program, whereas the non-faulty processors are bound by their programs.

We will show that both problems can be solved in polynomial time. We discuss the first one in the following section. In Section 4 we deal with the second problem.

3.1 Problem CN1 – a simplistic solution

Suppose that vertex b wants to construct a good approximation of the trust graph $G^* = (V^*, E^*)$ by querying all its neighbors in G^*, the neighbors of the neighbors, etc, for a signed list of the labels (the certificates) of their incoming edges. The query is flooded, and the neighbor list of vertex v is $L_v = (v, E^*_{inc,v}, \text{sign}_{k_v}(E^*_{inc,v}))$, where $E^*_{inc,v}$ is the list of labels of the incoming edges of v in E^*. Lists are only forwarded or accepted if they are correct (the signature in L_v must authenticate the labels with the key k_v of v, and the labels must be valid). Let n be the order of the graph G and let n^c be an upper bound on the complexity of *Adversary*$_u$. Vertex b will receive at most,

$$(n-u) + un^c \leq n^{c+1}$$

edge lists. This is a *first approximation* $G'' = (V'', E'')$ of the trust graph G^*. By our connectivity requirements on G^*, G'' must contain a good approximation of G^*.

The next stage is to remove from G'' all fake processors. Let $v \neq b$ be any vertex in V'', and $c(v,b)$ be the connectivity from v to b in G'', that is the maximum number of vertex-disjoint paths connecting v to b in G''. If the vertex v is fake then $c(v,b) \leq u$ by Lemma 1. On the other hand if v is not fake then $c(v,b) \geq u+1$, by our connectivity assumption. Checking this connectivity can be done by using a Max Flow algorithm[2] with complexity $O(|V''|^{\frac{1}{2}} \cdot |E''|)$ (Dinic's algorithm [8]). The complexity of finding all the non-fake vertices is therefore $O(n^{\frac{5}{2}(c+1)})$. Let V' be the set of vertices in V'' which belong the trust graph (the non-fake vertices in V'') and let E' the set of edges in E'' which belong the trust graph (the non-fake edges in E''). Then $G' = (V', E')$ a good approximation of the trust graph. It follows that vertex b will get a good approximation of the trust graph in polynomial time by using this procedure. Of course we must assume that the signature scheme will remain secure in this adversarial scenario.

This construction is not really satisfactory because its complexity is a function of the complexity of *Adversary*$_u$. In particular, it requires that non-faulty vertices work harder than faulty vertices.

4 Problem CN2 – the general case

4.1 Discussion

The main problem with the construction in Section 3.1 is that there is no halting strategy. The construction goes on until *Adversary*$_u$ is exhausted, which of course is too late. The trust graph

[2]Crucial to our argument is the fact that the adversary cannot make fake labels for edges (x,v) with non-faulty vertices.

will be completed long before this construction has ended, and what is constructed includes mainly fake vertices and edges. What we need is some means to recognize this.

There are also other problems with this construction. A neighbor list of a faulty vertex can be enormous, consisting mainly (or entirely) of endless lists of fake vertices and edges. If such a vertex were given equal time to a non-faulty vertex, this would allow the adversary to take control over most of the construction. The easiest way to sort out this problem is to share time equally, and ask each vertex to send only one edge label at-a-time.

Finally, by having no bound on the order of the graph, the description of some of the vertices may be much longer than needed. We shall assume that the description of the vertices of the trust graph is short. The problem with the description of fake vertices is dealt with by using a subprotocol (*Round Robin*) *packet_flood* in which the sender sends one packet at-a-time. The receiver will use these only after an EOT (end of transfer) has been received.

4.2 An informal approach

The first part of our construction is similar to the previous one, modified to take into acount our remarks in the previous section. Vertex b floods a query in the trust graph G^* to all other vertices for signed labels of their incoming edges, one-at-a-time. At some stage b will start receiving signed labels of edges (v, w) from which it can begin to build a graph G''. After some time, some of the vertices will have completed their lists. These vertices are labeled "*replied*". The others, in the process of replying, are labeled "*replying*". Eventually some of the vertices will be linked to b. These are labeled "*linked*". The others are "*not_yet_linked*".

Suppose that the graph G'' under construction has reached the point when $V' \subseteq V''$, *i.e.*, the vertices in the good approximation graph G' have all been found. Then the vertices v'' in V'' which are still sending new vertices v''', *i.e.*, not yet found, must be under the control of the adversary. These vertices v'' are either faulty or fake. Let G''_{aux} be the graph obtained from G'' by adding one new vertex v_{aux} and new edges connecting v_{aux} to all the vertices in G'' labeled *replying*. It is easy to see that $V' \subseteq V''$ if and only if $c(v_{aux}, b) \leq u$. Indeed there are at most u faulty vertices. We will use this test for our halting procedure.

Halting routine: *is_graph* $(*)$
Argument: $G'' = (V'', E'')$, $b \in V''$, a list of vertices $v_i \in V''$ labeled *replying*.
Value: *satisfactory* if $c(v_{aux}, b) \leq u$, else *not_satisfactory*.

Observe however, that this does not guarantee that G'' contains a good approximation graph. We only know that all vertices in G' have been found. Processor b will now ask all vertices v labeled "*replying*" to give, a new incoming edge, one at-a-time, as before. If (x, v) is such an edge, but $x \notin V''$, then b stops asking v for new edges.

Find missing edges routine: *missing_edges* $(*)$
Argument: $G'' = (V'', E'')$, $b \in V''$, a list of vertices $v_i \in V''$ labeled *replying*.
Value: a graph G''', containing a good approximation graph.

Now suppose that processor b has constructed a first approximation of the trust graph G^*, that is a graph $G'' = (V'', E'')$ which contains a good approximation G' of G^*. G' will also contain fake vertices and edges. Fake vertices v can be traced as in Section 3.1, because $c(v, b) \leq u$. Similarly fake edges can be traced. Discarding these from G'' will give us a good approximation of the trust graph.

Cleaning up routine: *clean_up* $(*)$
Argument: $G'' = (V'', E'')$, a first approximation graph.
Value: G', a good approximation graph.

4.3 The Protocol

Vertex b in the trust graph $G^* = (V^*, E^*)$ wants to construct a good approximation of G^* given the set N_b^* of neighbors of b in V^* and the set E_b^* of incident edges of b in E^*.

The setting

The vertices in N_b^* and all the vertices under construction have $link\_status \in \{linked, unlinked\}$ and $reply\_status \in \{not\_replied\_yet, replying, replied\}$.
Initially $link\_status(x) := linked$, and $reply\_status(x) := not\_replied\_yet$, for all $x \in N_b^*$. Each (non-faulty) vertex $v \neq b$ in V^* makes an ordered list $edge\_list\,(v)$ of its incoming edges. Each time v replies to a query of b requesting its incoming edges, v will send the first edge *first_edge* (*edge_list* (v)) in this list, and then remove this edge from the list. Edges are sent one-at-a-time. Initially $edge\_list(v) := E^*_{inc,v}$, where $E^*_{inc,v}$ is the complete list of incoming edges of v in E^*. Finally, a label of an edge (x, v) is $label\_(x, v) := c_{xv}$, the certificate in which v certifies the public key of x.

Protocol $(input = (b,\ N_b^*, E_b^*))$

until *is_graph* (G'') = *satisfactory* **do**
  begin
    b floods a query: "all vertices v send a *label* (x, v) of a new edge (x, v)";
    vertex $v \neq b$
      if $\mid$ *edge_list* $(v) \mid > 1$ **then** *reply_status* (v) = *replying*;
      if $\mid$ *edge_list* $(v) \mid = 1$ **then** *reply_status* (v) = *replied*;
      packet_flood to b: $(data(v),\ sign_{k_v}(data(v)))$,
          where $data(v)$= (*first_edge* (*edge_list*(v)), *reply_status* (v));
      edge_list (v) :=*edge_list* (v) − *first_edge* (*edge_list* (v)) :
    vertex b
      if there is a path from b to v in G'' **then** *link_status* (v) := *linked*
        else *link_status* (v) := *unlinked* ;
      if *link_status* (v) := *linked* **and** $(data(v), sign_{k_v}(data(v)))$ is correct
        and *label* (x, v) is correct **then** *add_to_G''* (*label* (x, v));
  end;
  $G''' := missing\_edges\,(G'')$;
  $G' := clean\_up\,(G''')$

5 Proofs

We shall now prove that this protocol is efficient.

Lemma 2 *The construction above will halt in $O(n^4)$ communication time, where n is the order of the trust graph G^*.*

Proof. Let v be any vertex in G^* and let $\pi = (v = v_1, v_2, \ldots, v_r = b)$ be a path in G^* all of whose vertices are not faulty. The time taken to send the label of an edge (x, v) through the channel (v_i, v_{i+1}) of π is bounded by the number of neighbors of v_{i+1} in V^*, which is bounded by n. The length of π is bounded by n, so b will get the label of (x, v) in time n^2. Since v cannot have more than n incident edges, b will get the complete list of edges $E^*_{inc,v}$ in time n^3. It follows that b will get a first approximation of G^* in time $O(n^4)$. By this time, of course, b may also get up to $n^4 - n$ fake vertices.

Next let us consider the cost of the halting and clean up tests. For the halting test we use a Max Fow algorithm with time complexity $O(|V''|^{\frac{1}{2}}|E''|)$. In our case $|V''| = O(n^4)$, $|E''| = O(n^4)$. The complexity for this test is then $O(n^2 \cdot n^4) \cdot O(n^4) = O(n^2 \cdot n^4 \cdot n^4) = O(n^{10})$. A similar argument applies for the clean up test with bound $O(n^{10})$. Observe that one should distinguish between the first complexity and the other two. The first one involves communication in the network, whereas the others are essentially off-line. □

Lemma 3 *The constructed graph G' is a good approximation of the trust graph G^*.*

Proof. Suppose that a vertex v in G^* is not in the constructed graph G'. Let π_i, $i = 1, 2, \ldots, 2u+1$, be vertex-disjoint directed paths in G^* from v to b, and let $v_i \in \pi_i$, $i = 1, 2, \ldots, 2u+1$, be vertices in G' whose ancestors in π_i are not in G'. The *reply_status* of the vertices v_i must be *replying*. Apply the halting test to these vertices to get a contradiction. The *missing_edges* routine guarantees that no edges are missing. □

We therefore have,

Theorem 1 *If the trust graph is $(2u + 1)$ vertex-connected and if all the other assumptions in Section 2 hold, then the Protocol above will construct a good approximation of the trust graph in polynomial time.*

6 Discussion

6.1 Reliability and the communication network

In our model we assume that the edges of the trust graph $G^* = (V^*, E^*)$ correspond to reliable channels. These can be regarded as virtual channels in a communication network $\bar{G} = (\bar{V}, \bar{E})$ for which $V^* \subset \bar{V}$. An edge $(v, w) \in E^*$ could correspond to some path in $\bar{G}$ linking v, w, but not necessarily to a fixed path. Alternatively it could correspond to several paths, possibly vertex-disjoint. This would allow for the possibility of the channel (v, w) not being reliable, but reliability for the system may still be achieved through other paths in G^*.

This model is more general. However, at a high level we can add virtual edges whenever we get a reliable channel. So there is no real difference. Indeed one could argue that the general goal of secure communication is to extend the trust graph to a completely connected graph.

6.2 Identifying labels of processors

The following is an interesting scenario. A faulty processor may try to use different public keys, that is pseudonyms. The other faulty processors may be willing to support it, but a more successful strategy would be to get some non-faulty processors to accept the pseudonym. The effect of this would be to give the adversary more power, by controlling more parties. If no more than u non-faulty processors have certified a pseudonym p and if the trust graph is known then there is no problem. The pseudonym can be traced by observing that the connectivity $c(p, b) \leq 2u$ for every non-faulty processor b.

However if the trust graph is not known then there is a problem. This is because is not possible in our general adversarial model to construct a good approximation graph given only a vertex $b \in V^*$, the neighbor set N_b^* and the edge set E_b^*. Indeed if the adversary controls $(u+1)$ processors (u faulty processors and a pseudonym) then there is no way of preventing her from taking over the construction (in the general setting of Section 4).

This remark also suggests a method for extending the trust graph. A new processor will be "allowed to join" if at least $(2u+1)$ processors are prepared to certify it. In the final version of the paper we will discuss this issue in more detail.

6.3 Double certificates – undirected trust graphs

In our model the edges (v, w) correspond to single certificates $c_{vw} = (v, w, k_v, k_w, sign_{k_w}(v, k_v))$ in which w certifies v's public key. If v also certifies w's public key with the certificate $c_{wv} = (w, v, k_w, k_v, sign_{k_v}(w, k_w))$ then we get an undirected edge (v, w). So the trust graph is undirected. All our results can be extended to this case.

References

[1] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness theorems for non-cryptographic fault-toerant distributed computing. In In *Proc. ACM STOC*, ACM Press, pp. 1–10, 1988.

[2] D. Bertsekas and R. Gallager. *Data networks.* Prentice Hall, second edition, 1992.

[3] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. Proceedings, *Computer Security - ESORICS 94, Lecture Notes in Computer Science* **875**, Springer-Verlag, 1994, pp. 3–18

[4] M. Burmester, Y. Desmedt, and G. Kabatianski. Trust and security: A new look at the Byzantine generals problem. In R. N. Wright and P. G. Neumann, editors, *Network Threats, DIMACS, Series in Discrete Mathematics and Theoretical Computer Science*, **38**, AMS, 1998.

[5] D. Chaum, C. Crepeau and I. Damgaard. Multiparty unconditional secure protocols. *Proc. ACM STOC*, ACM Press, pp. 11–19, 1988.

[6] D. Dolev. The Byzantine generals strike again. *J. of Algorithms*, **3**(1), pp. 14–30, 1982.

[7] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *Journal of the ACM*, **40**(1), pp. 17–47, 1993.

[8] S. Even. *Graph algorithms.* Computer Science Press, Rockville, Maryland, 1979.

[9] M. Franklin and N. Wright. Secure communication in minimal connectivity models. In *Advances of Cryptology, Eurocrypt 98, Lecture Notes in Computer Science* **1403**, Springer-Verlag, 1998, pp. 346–360.

[10] M. Franklin and M. Yung. Secure hypergraphs: privacy from partial broadcast. In *Proc. ACM STOC*, ACM Press, pp. 36–44, 1995.

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

[12] O. Goldreich, S. Goldwasser and N. Linial. Fault tolerant computations in the full information model. *SIAM J¿ Comput.* **27**(2) pp. 506–54, 1998.

[13] V. Hadzilacos. Issues of fault-tolerance in concurrent computations. Ph.D. Thesis, Harward University, Cambridge, MA, 1984.

[14] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on programming languages and systems*, **4**(2), pp. 382–401, 1982.

[15] U. Maurer. Modeling public-key infrastructure, Proceedings, *Computer Security - ESORICS 96, Lecture Notes in Computer Science* **1146**, Springer-Verlag, 1996, pp. 325–350.

[16] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults, *Journal of ACM* **27**(2), pp. 228–234, 1980.

[17] G. J. Popek and C. S. Kline. Encryption and secure computer networks. *ACM Computing Surveys*, **11**(4), pp. 335–356, 1979.

[18] M. K. Reiter and S. G. Stubblebine. Path independence for authentication in large scale systems. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pp. 57–66, 1997.

[19] R. L. Rivest and B. Lampson. SDSI–A Simple Distributed Security Infrastructure. http://theory.lcs.mit.edu/ cis/sdsi.html

[20] Y. Wang and Y. Desmedt. Secure communication in broadcast channels: the answer to Franklin and Wright's question. In J. Stern, Ed, *Advances of Cryptology, Eurocrypt 99, Lecture Notes in Computer Science* **1592**, Springer-Verlag, 1999, pp. 446–458.

[21] P. R. Zimmermann. The Official PGP User's Guide. MIT Press, Cambridge, Massachussets, 1995.