Hop Chains: Secure Routing and the Establishment of Distinct Identities

Rida A. Bazzi[1], Young-ri Choi[2], and Mohamed G. Gouda[2]

[1] School of Computing and Informatics
Arizona State University
Tempe, Arizona, 85287
bazzi@asu.edu

[2] Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
{yrchoi, gouda}@cs.utexas.edu

Abstract. We present a secure routing protocol that is immune to Sybil attacks, and that can tolerate initial collusion of Byzantine routers, or runtime collusion of non-adjacent Byzantine routers in the absence of collusion between adjacent routers. For these settings, the calculated distance from a destination to a node is not smaller than the actual shortest distance from the destination to the node. The protocol can also tolerate initial collusion of Byzantine routers and runtime collusion of adjacent Byzantine routers but in the absence of runtime collusion between non-adjacent routers. For this setting, there is a bound on how short the calculated distance is compared to the actual shortest distance. The protocol makes very weak timing assumptions and requires synchronization only between neighbors or second neighbors. We propose to use this protocol for secure localization of routers using hop-count distances, which can be then used as a proof of identity of nodes.

1 Introduction

In peer-to-peer networks, physical entities (or hosts) communicate with each other using pseudonyms or logical identities. Logical identities are assumed by software processes that execute on the hosts to provide or request services from other hosts. To the outside world, a host is identified with the software process that provides the logical functionality. In the absence of direct physical knowledge of a remote host, or certification by a central authority, it is not possible to tell whether or not two distinct logical identities reside on the same host (physical entity) and it is possible for one entity to appear in the system under different names or *counterfeit* identities. Douceur [4] was the first to thoroughly study this problem, and he says that an entity launches a *Sybil attack* when it appears under different identities. He claims that in the absence of a central certifying authority, the Sybil attack cannot be solved in practice.

Bazzi and Konjevod [2] proposed the use of geometric techniques to determine how many identities amongst a group of identities belong to distinct entities and thereby reducing the harm due to Sybil attack. Their work is based on existing evidence that roundtrip delays in the Internet exhibit geometric properties [9]. They provided solutions under a variety of adversarial assumptions including colluding entities and beacons without assuming a central certifying authority with direct knowledge of entities in the system. While the work of Bazzi and Konjevod is a significant step forward, it makes some restricting assumptions. For instance, the results about the geometry of roundtrip delays apply to systems in which routers are honest and they are not necessarily applicable in systems in which routers are corrupt. Also, their solutions require accurate measurements of roundtrip delays and clock synchronizations between routers that can be far apart physically. This is not always possible given the variability of network load and delays.[1]

In this work, our goal is to present a solution to the Sybil attack problem under the weakest possible system assumptions and in the absence of a central authority with direct knowledge of entities in the system. Our solution can tolerate stronger adversarial settings while making weaker system assumptions. In particular, we assume that routers can be dishonest and we allow for more collusion between the routers. Also, we require very rough synchronization between non-adjacent routers (in a sense that we will define precisely later). For some settings, we require synchronization, but only between adjacent (or almost adjacent) routers, which means that the synchronization we require is local. Relaxing the synchrony and synchronization assumptions is a major improvement over the results of [2] and we believe that it is an improvement that brings the results closer to a practical setting.

At the heart of our approach is a secure distance vector routing protocol that can tolerate Byzantine routers, Sybil attack by routers and collusion between routers. The protocol assumes that there are no shared keys between any two nodes, and that only the destination's public key is known a priori by nodes in the network. Under the assumption of no collusion between corrupt nodes, a first version of the protocol guarantees that no node can have a calculated hop-count distance, or simply distance, to destination that is shorter than its real or actual shortest hop-count distance to destination. In the presence of initial collusion, in which corrupt nodes can share information initially, but not afterwards, the second version of protocol guarantees that no honest router can have a calculated hop-count distance to destination that is shorter than its real or actual shortest hop-count distance to destination. In the presence of initial collusion between any two nodes and runtime collusion between adjacent corrupt nodes, in which corrupt nodes can communicate with each other at any time, the second protocol guarantees the following: for any path P from destination to an honest node u, the calculated hop-count distance of node u is not less than the number of honest nodes on P plus the number of corrupt components of P (a corrupt component is a maximal subpath that contains corrupt nodes). In other words, every sequence of adjacent corrupt nodes on a path can appear to be one node. In the presence

[1] In their work, they propose approaches to handle inaccuracies, but these approaches are incomplete.

of remote collusion between nodes and if there are no colluding adjacent nodes, then a further modification guarantees that the hop-count distance calculated by an honest node is not shorter than the shortest distance from destination to the node. The protocol has two basic components. The first is a practical and simple protocol that enables a node to determine if another node is its physical neighbor. The second is a novel use of key chains, which we call *hop-chains*, that enable the destination to certify *remotely* its distance to nodes in the network. To tolerate initial collusion, we introduce *mistrust hop-chains* to prevent nodes from cheating by initially agreeing on keys. This secure routing protocol we present is a significant contribution on its own. The protocol is more secure and requires less assumptions than other secure routing protocols in the literature (see Section 9).

Our solution to the Sybil attack problem proposes to use the secure routing protocol in order to come up with a secure localization protocol for networks in which hop-count distances from a number of beacons (or anchor points) can be used to localize nodes. This is along the lines of the approach of [2], but replacing roundtrip delays with hop-count distance.

2 Identities and Public Keys

In our model, only the destination has a public key that needs to be known by all other nodes. Other nodes need to have identities that cannot be forged by faulty nodes. This can be achieved by having each node randomly choose its own public key and corresponding private key. We assume that the keys are large enough so that corrupt nodes can with negligible probability guess or generate keys identical to those of honest nodes. Also, correct honest nodes generate different keys with high probability. This guarantees that with high probability nodes cannot forge messages, but does not rule out that corrupt nodes can replay messages.

In our framework, the identity of a node is its public key. For an honest node, this identity is unique and does not change over time. For a corrupt node, there can be multiple identities, one for each public key that the node chooses. We spell out our assumptions about keys in Section 5.

3 Neighbor Computation

The ability of a node to determine whether another node is its neighbor is an important ingredient for our secure routing protocol. Before we explain how that determination can be done, we need to precisely define what we mean by "determining if a node is the neighbor of another node".

We say that a node is the neighbor of another node if the two nodes can communicate directly and not through an intermediate node. In wireless networks, this requires that the nodes are in each other's range. In wired networks, this requires that the nodes either have access to a shared communication link or share a private link. In our model, nodes are known to other nodes through their public keys. So, determining whether a node is the neighbor of another node reduces to determining if the owner of the private key corresponding to a given public key is in the neighborhood of the node. What we determine is something subtly different from the foregoing. We determine if a node with *access to* the private key corresponding to a given public key is in the neighborhood of the node. The distinction is subtle, but important. A node has *access to* the private key if it has the private key or it is in collusion with a node that has the private key. If there is no collusion other than initial collusion between nodes, then having access to a key and having a key are the same thing.

3.1 Immediate Neighbors

A first step in neighbor determination is to broadcast a message requesting from neighbors to provide their public keys. The goal of neighbor determination is to determine if a neighbor of the node has access to the private key corresponding to the provided public key.

A naive approach for determining whether a node is the neighbor of another node is to send a request message and wait for a reply within a short period of time. The reply should allow for the transmission time, roundtrip delay and any local computation at the node to encrypt and decrypt messages exchanged to prevent third parties for interfering with the communication. Unfortunately, the time for computations can be substantial especially if public key encryption is involved which makes the approach vulnerable to a man-in-the-middle attack.

A better approach is similar to the one taken by [3] in which communication is done in the clear to eliminate high processing time. In a first phase, a node sends a random bit in a message encrypted with the destination's key. The destination decrypts the message and recovers the bit. In a second phase, the node broadcasts a message in the clear to all its neighbors. Upon receipt of the message, the destination performs XOR on the first bit of the message with the random bit and resends the message in the clear to the sender. The extra processing time is minimal. The probability that the destination sends the correct answer without knowing the random bit is 1/2. This probability can be made arbitrarily small by repeating the two phases multiple times. A corrupt node B cannot compromise this scheme by launching a man-in-the-middle attack in a timely manner. But, a corrupt node B can execute the first phase by colluding with another node C, which decrypts the first phase message and provides the value of the bit to B. This way, B can execute the second phase. Thus, this two-phase approach guarantees that B *has access* to the private key corresponding to the public key it sends to A.

3.2 Neighbors of Neighbors

To tolerate runtime collusion between non-adjacent nodes and if there are no colluding adjacent nodes, our routing protocol requires the ability for a node to determine if another node is the neighbor of its neighbor. We propose to use the same approach we propose for determining neighbors to also determine neighbors of neighbors by allowing more time for the message to be forwarded to a neighbor of a neighbor and then sent back.

3.3 Effects of Congestion

If two nodes have a dedicate link between them, then the determination of neighbors can be done without interference by other nodes. In a wireless medium, other nodes can interfere in the communication by launching denial of service attacks. We do not address denial of service attacks in this paper. Our assumption about congestion is fundamentally different from the assumptions in [2]. In our work, we make the realistic and practical assumption that two *adjacent* (immediate neighbors) nodes can communicate with no congestion for some periods of time, whereas in [2], a similar assumption is made for nodes that are many hops apart.

3.4 Timing Consideration

In our neighbor computations, we assume that the dominant factor in the delay is due to transmission and processing, but not propagation delay. The transmission rate between two adjacent nodes is determined by their hardware and it is not unreasonable that the nodes can measure time to an accuracy of 1 bit. In wireless networks, speeds of 100 Mbps can be considered high. At this speed, a 4 KByte frame takes around 0.32 msec. During that time a signal can propagate up to 96 km which is way beyond the range of node to node transmission in ad-hoc networks. In wired networks, propagation delay can be substantial for transatlantic communication, but such communication has to go through known entities that charge for their services and cannot be part of any ad-hoc network.

4 Distance Vector Routing and Its Vulnerabilities

In a traditional distance vector routing protocol, nodes in a network collaborate to build a spanning tree whose root is the ultimate destination node d. Initially, no node u other than d has a parent in the routing tree, and the distance of node u is infinite. Only the ultimate destination node d is in the routing tree, and it periodically broadcasts an advertisement message of the form $adv(d, 0)$ to its neighbors.

When a node u whose current distance is greater than $s + 1$ receives an $adv(v, s)$ message, node u makes node v its parent in the tree, and sets its distance to be $s+1$. Once node u has a parent in the tree, node u becomes connected to the tree and starts sending an $adv(u, s')$ message periodically, where s' is the current distance of u. When u stops receiving advertisement messages from its parent for a certain time period, it stops sending advertisement messages.

A node can cause harm if it drops packets it is supposed to forward[2] or if it reports a false (shorter) distance to destination. Such misbehavior is illustrated in Figure 1 where the ultimate destination is node (0,0) and node (8,8) misbehaves. The left side of the figure illustrates the case where node (8,8) drops packets and the right side illustrates the case where node (8,8) reports a distance of 2 to

[2] There are techniques to detect nodes that do not forward messages [1], but in this paper we do not consider the problem of detecting such nodes.

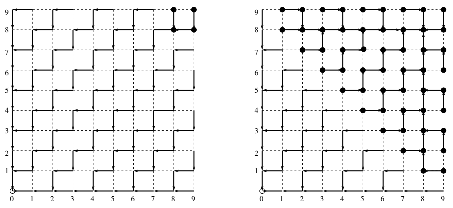

Fig. 1. Routing tree when (8,8) does not lie (left) and when (8,8) lies (right)

destination. The black circles illustrate the affected nodes and node (8,8) (in general, the number of affected nodes depends on the difference between the reported distance and the real distance). Our goal for a secure distance vector routing protocol is to prevent corrupt nodes from reporting distances that are smaller than their actual distances to destination. The proposed secure routing protocol tolerates strong adversaries. We consider the following failures.

Byzantine failures. Corrupt nodes can behave arbitrarily. In particular, they can advertise multiple public keys (attempt Sybil attacks) and they can replay or resend messages received from others.

Initial collusion of nodes. Corrupt nodes that initially collude can share information before the execution of the protocol, but they cannot communicate information that they learn during the execution of the protocol. To our knowledge, this model has not been considered by others. It makes sense to consider it in our setting because nodes are known to others through their public keys. It is not clear how a node can find or *trust* another node to collude with. If a node is corrupted with a virus, for example, then two corrupted nodes share the common information that the virus carries and therefore they initially collude.

Runtime collusion of adjacent nodes. Even though it is not clear how nodes can find other nodes to collude with, it makes sense to consider adjacent nodes that are colluding at runtime. In fact, if two adjacent nodes are initially colluding, they could discover that they have the same keys by communicating with each other and then decide to collude. Our protocol tolerates run-time collusion between adjacent nodes (or connected component consisting of corrupt colluding nodes).

Runtime collusion of non-adjacent nodes. We consider runtime collusion between non-adjacent nodes, but in the absence of collusion between adjacent nodes. We assume that non-adjacent corrupt nodes can communicate information with each other at any time.

5 Tolerating Non-colluding Byzantine Failures

Every node u in the network creates its own public key(s) (BK_u), and corresponding private key(s) (RK_u). The public key BK_d is known by all nodes. We denote a message m encrypted with a key BK_u with $BK_u\langle m\rangle$. Similarly,

$BK_u\langle m\rangle$ is the decryption of m using the public key of u, and $BK_u\langle RK_u\langle m\rangle\rangle = m$. To reduce the size of messages, we assume the existence of a message digest, or one way hash, function MD. The use of MD is not needed for the correctness of the protocol. If a node receives a message (m, m') such that $BK_u\langle m'\rangle = BK_u\langle RK_u\langle m\rangle\rangle$, then m' must have been encrypted by u, which has RK_u. Similarly if a node receives a message (m, m') such that $BK_u\langle m'\rangle = BK_u\langle RK_u\langle MD(m)\rangle\rangle$, then m' must have been encrypted by u.

In the protocol, each node u that is connected to the routing tree maintains a hop-chain that verifies its hop-count distance to destination. The hop-chain contains a sequence of public keys and a sequence of certificates. The public keys are supposed to be keys of a sequence of nodes $d = u_0, u_1, \ldots, u_k = u$ that form a path from d to u. The certificates vouch that every node in the sequence is the neighbor of the next node in the sequence. Finally, a hop-chain has a date dt that specifies the period of validity of the chain. The date changes infrequently relative to the communication delays in the network, and it does not require any tight synchronization between the nodes. A hop-chain has the following format: $\langle dt, BK_{u_0}, BK_{u_1}, \ldots,\ BK_{u_{k-1}}, BK_{u_k}, C_{u_0}, C_{u_1}, \ldots C_{u_{k-1}}, C_{u_k}\rangle$, where $C_{u_0} = RK_{u_0}\langle MD(dt, BK_{u_0})\rangle$, and $C_{u_i} = RK_{u_{i-1}}\langle MD(dt, BK_{u_i})\rangle$, $0 < i \le k$.

Definition 1. *The* length *of a hop-chain H_u of a node u, denoted $len(H_u)$, is the number of certificates in H_u.* [3]

It is straightforward to see that only a node that has RK_{u_0} can generate C_{u_0}, and only a node that has $RK_{u_{i-1}}$ can generate C_{u_i}. We say that a hop chain is valid if it is of the form $\langle dt, K_0, K_1, \ldots, K_k, C_0, C_1\ , \ldots, C_k\rangle$ and

- $K_0 = BK_d$ and $BK_d\langle C_0\rangle = MD(dt, BK_d)$
- $BK_{i-1}\langle C_i\rangle = MD(dt, K_i)$, $0 < i \le k$.

A node u that receives a hop-chain can locally check its validity. The protocol ensures that the owners of successive keys in the sequence are neighbors or the same node (creating successive bogus nodes). It will follow that the hop-chain length of node u minus one cannot be less than the shortest distance from u to d.

Initially, no node u other than d has a parent in the routing tree, and the distance of node u is infinite. Only node d is initially connected to the routing tree.

Each hop-chain contains a date field dt that indicates the date (time) at which the chain is generated. The root of the routing tree, node d, periodically updates dt every P seconds. Thus, node d periodically recomputes its chain $\langle dt, BK_d, RK_d\langle MD(dt, BK_d)\rangle\rangle$ every P seconds. The period of time P is chosen to be larger than the delay between nodes in the network.

A node u that is connected to the tree periodically broadcasts an advertisement message to its neighbors every p seconds where $p \ll P$. The advertisement message of node u has the form $adv(BK_u, dt, H_u)$, where dt is the latest date that u is aware of, and H_u is the latest chain that u has.

When a node u receives an advertisement message, $adv(bk, t, h)$, where bk is a public key, t is a date, and h is a hop-chain, u ignores the message if t is smaller

[3] Note that $len(H_u)$ minus one is equal to the hop-distance from destination to u.

```
1:  var  $dt$           : integer,     // date
2:       $H_u, ph$      : integer,     // current/potential hop-chain
3:       $ds$           : 0..dmax+1,   // distance, initially dmax+1
4:       $BK_p$         : integer,     // parent's permanent public key
5:       $trc$          : 0..tmax,     // time to remain connected
6:       $wait$         : boolean,     // wait for ack msg or not, init. false
7:       $pdt, t$       : integer,     // potential and received date
8:       $h, c$         : integer,     // received hop-chain/certificate
9:       $bk, bk'$      : integer      // received keys
```

Fig. 2. Variables of a node u

than the latest date u is aware of – the message is too old. If the message has a date that is more recent than the latest date at node u, u verifies that the message is valid. If the message is valid, u tentatively decides to use the received hop-chain to calculate its distance to destination. An *adv* message is valid, if bk is the public key of a neighbor of u, the date of the message is the same as the date of the hop-chain h, the hop-chain h is valid, and the last key in the chain is equal to bk. In the case that the date of the message is the same as the most recent date u is aware of, if the message is valid, and the length of h is smaller than the current distance of u to destination, u tentatively decides to use the received hop-chain to calculate its distance to destination. When a node u tentatively decides to use the received chain to calculate its distance to destination, u makes bk its potential parent, assigns h to the potential chain ph, and assigns t to the potential date pdt. Finally, u sends a reply message to node bk and waits to receive a certificate from bk – its potential parent. The reply message is of the form $rpl(bk, BK_u, t)$. While node u is waiting to receive a certificate, u ignores any advertisement messages u receives until u receives a certificate or u times out. (This is only to keep our code easy to follow.)

There is no loss in ignoring advertisement messages, since advertisement messages will be sent periodically, and so node u can receive them later.

When a node u that is connected to the routing tree receives a reply message $rpl(bk, bk', t)$, where $bk = BK_u$, and t equals to dt in its own hop-chain, node u first computes a certificate $c = RK_u\langle MD(dt, bk')\rangle$, and then u sends an acknowledgment message $ack(bk', t, c)$ to node bk'.

When a node u that is waiting to receive a certificate from its potential parent receives an acknowledgment message $ack(bk', t, c)$, where $bk' = BK_u$, and $t = pdt$, node u checks the validity of the certificate in the message. The certificate c is valid if it is encrypted with the corresponding private key of the last key $tpbk$ in the potential hop-chain ph (the key of its potential parent) and so $c = tpbk\langle MD(dt, BK_u)\rangle$. If the certificate in the message is valid, u makes node $tpbk$ its parent in the routing tree and updates its distance to destination. Finally, u computes its (new) hop-chain by adding BK_u and c to the hop-chain of its parent, ph.

When a node u has a parent and does not receive any valid advertisement message from its parent for a time period of $tmax \times p$ seconds, u concludes that it is not connected to its parent anymore. Thus, u disconnects from the tree by making its distance infinite, and stops sending advertisement messages.

After node u sends a reply message to its potential parent v, u starts a timer and times out after w seconds, at which time u no longer considers v as its potential parent. The value of w is chosen to be large enough to accommodate roundtrip delay to a neighbor including time for public key encryption and decryption. The protocol variables and specification of a node u are given in Figures 2 and 3. We state without proof the properties of the protocol.

Lemma 1 (len(hop-chain) $\geq$ len(shortest path)). *For every honest node u, $len(H_u) \geq len(S)$, where S is the shortest path from node d to node u and $len(S)$ is the number of nodes in path S.*

Lemma 2 (len(hop-chain) $\leq$ len(good path)). *For every honest node u, if there exists a "good" path G from node d to node u such that each node in the path is honest, then eventually $len(H_u) \leq len(G)$ holds.*

6 Tolerating Initial Collusion

The protocol of the previous section is vulnerable to initial collusion. Consider two nodes u and v that share the public and private keys BK_u and RK_u. Assume that v is a farther node from the destination. Since v has u's keys, the neighbors of v will consider the owner of the private key RK_u to be their neighbor. If at some point, node v receives an advertisement message with a chain that contains the public key of u, v can *cut* the chain, only keep the portion of the chain that is identical to u's chain, and present that portion to its neighbors. Node v can then advertise u's chain to its neighbor, and the neighbors of v will find the received chain to be valid, in effect v manages to claim a distance to destination that is shorter than its actual distance to destination.

The reason for the success of this attack is that a node is certified based only on the initial information of the node, and initially colluding nodes share all their initial information. To get around this difficulty, we need to certify nodes based on information that they do not have initially. This can be achieved by having a parent in the routing tree create public/private key pairs for its children.

These *temporary* keys will be used alongside the *permanent* keys of a node. We say that they are temporary because their values depend on the identity of the parent of a node at a given time. For a node u, we denote these keys with TBK_u and TRK_u. For the destination node d, we really need no temporary keys, but we introduce them to make the protocol more uniform.

In the modified protocol, nodes use temporary keys and permanent keys to check the validity of a certificate and therefore of a chain. The *mistrust* hop-chain of a node has the format: $\langle dt, (BK_{u_0}, TBK_{u_0}), \ldots, (BK_{u_k}, TBK_{u_k}), C_{u_0}, C_{u_1}, \ldots, C_{u_{k-1}}, C_{u_k}\rangle$, where $C_{u_0} = RK_{u_0}\langle TRK_{u_0}\langle MD(dt, BK_{u_0}, TBK_{u_0})\rangle\rangle$, and $C_{u_i} = RK_{u_{i-1}}\langle TRK_{u_{i-1}}\langle MD(dt, BK_{u_i}, TBK_{u_i})\rangle\rangle$, $0 < i \leq k$.

It is straightforward to see that only a node that has RK_{u_0} and TRK_{u_0} can generate C_{u_0}, and only a node that has $RK_{u_{i-1}}$ and $TRK_{u_{i-1}}$ can generate C_{u_i}.

```
   timeout DATE expires → // $d$ periodically updates date
                          if ($u = d$) then
                               $dt$ := UPDATE_DT;
                               $H_u := \langle dt, BK_u, RK_u\langle MD(dt, BK_u)\rangle\rangle$;
                               timeout DATE after $P$

[] timeout ADV expires →  // $u$ periodically sends advertisement
                          if ($u = d$) then
                               send $adv(BK_u, dt, H_u)$;
                               timeout ADV after p
                          elseif ($u \neq d$) then
                               $trc$ := MAX($trc - 1, 0$);
                               if ($trc > 0$) then
                                    send $adv(BK_u, dt, H_u)$;
                                    timeout ADV after $p$
                               elseif ($trc = 0$) then
                                    $ds$ := dmax+1

[] timeout RPL expires→   // no longer wait for ack from potential parent
                          $wait$ := false

[] rcv adv($bk, t, h$)  → // if valid adv received from a node closer to d
                          // update potential parent and reply to sender
                          if $\neg wait \wedge (t > dt \vee (t = dt \wedge len(h) < ds))$
                          $\wedge$ valid($adv(bk, t, h)$) then
                               $pdt := t$; $ph := h$;
                               $wait$ := true;
                               send $rpl(bk, BK_u, t)$ to $bk$;
                               timeout RPL after $w$
                          // if valid advertisement received from parent
                          // stay connected for a longer period
                          if $((trc > 0) \wedge (bk = BK_p) \wedge (len(h) = ds)$
                          $\wedge$ valid($adv(bk, t, h)$)) then
                               $trc$ := tmax

[] rcv rpl($bk, bk', t$) → // if valid reply received from a node
                          // compute a certificate and send it to sender
                          if $((BK_u = bk) \wedge (t = dt) \wedge (trc > 0)$ then
                               send $ack(bk', dt, RK_u\langle MD(dt, bk')\rangle)$ to $bk'$

[] rcv ack($bk, t, c$)  → // if valid ack received from potential parent
                          // update its parent, distance, chain, and send adv
                          if $wait \wedge (BK_u = bk) \wedge (pdt = t)$
                          $\wedge$ valid($ack(bk, t, c)$) then
                               $dt := pdt$;
                               $H_u$ := COMP_CERT($ph, c$);
                               $ds := len(ph)$;
                               $BK_p$ := GET_BKP($ph$);
                               $wait$ := false;
                               $trc$ := tmax;
                               send $adv(BK_u, dt, H_u)$;
                               timeout ADV after p
```

Fig. 3. A specification of a node u

We say that a mistrust hop-chain is valid if the chain is of the form $\langle dt, (K_0, T_0), (K_1, T_1), \ldots, (K_k, T_k), C_0, C_1, \ldots, C_k\rangle$ and

– $K_0 = BK_d$ and $TBK_d\langle BK_d\langle C_0\rangle\rangle = MD(dt, K_0, T_0)$
– $TBK_{i-1}\langle BK_{i-1}\langle C_i\rangle\rangle = MD(dt, K_i, T_i)$, $0 < i \leq k$.

Using permanent and temporary keys, the protocol ensures that the owner of every pair of permanent and temporary public keys in the sequence encrypted the certificate of the owner of the next pair of public keys in the sequence. Thus,

```
[] rcv rpl($bk, bk', t$) →    // if valid reply received from a node
                               // choose temp. keys and send a cert. to sender
                               if (($BK_u = bk$) ∧ ($t = dt$) ∧ ($trc > 0$) then
                                    ($tcbk, tcrk$) := GEN_KEYS;
                                    send $ack((bk', tcbk), dt, bk'\langle tcrk\rangle,$
                                    $RK_u\langle TRK_u\langle MD(dt, bk', tcbk)\rangle\rangle)$ to $bk'$

[] rcv ack($bk, bk', t, r, c$) →  // if valid ack received from potential parent
                                   // update its parent, distance, chain, and send adv
                                   if $wait$ ∧ ($BK_u = bk$) ∧ ($pdt = t$)∧
                                   valid($ack(bk, bk', t, r, c)$) then
                                        $dt := pdt$;
                                        $TBK_u := bk'$; $TRK_u := RK_u\langle r\rangle$;
                                        $H_u$ := COMP_CERT($ph, c$);
                                        $ds := len(ph)$;
                                        $BK_p$ := GET_BKP($ph$);
                                        $wait$ := false;
                                        $trc$ := tmax;
                                        send $adv(BK_u, dt, H_u)$;
                                        timeout ADV after p
```

Fig. 4. *rpl* and *ack* processing to handle initial collusion

a corrupt node u that initially colludes with another node v closer to destination cannot use the hop-chain of v, since u has no access to the temporary private key of v.

In the modified protocol, *rpl* and *ack* processing is modified as in Figure 4. When a node u that is connected to the routing tree receives a reply message $rpl(bk, bk', t)$, where $bk = BK_u$, and t equals to dt in its own hop-chain, node u randomly chooses temporary public/private key pair $(tcbk, tcrk)$ for node bk'. Node u then computes a certificate $c = RK_u\langle TRK_u\langle MD(dt, bk', tcbk)\rangle$, and also computes an encrypted temporary private key $r = bk'\langle tcrk\rangle$. Finally, node u sends an acknowledgment message $ack((bk', tcbk), t, r, c)$ to node bk'.

When a node u that is waiting to receive a certificate from its potential parent receives an acknowledgment message $ack((bk', bk''), t, r, c)$, where $bk' = BK_u$, and $t = pdt$, node u first checks the validity of the certificate in the message. The certificate c is valid if it is encrypted with the corresponding permanent and temporary private keys of the last key pair $(pbk, tpbk)$ in the potential chain ph (the key of its potential parent) and so $tpbk\langle pbk\langle c\rangle\rangle = MD(t, BK_u, bk'')$. If the *ack* message is valid, u makes node pbk its parent in the routing tree and updates its distance to destination. Finally, u computes its temporary public and private keys, TBK_u and TRK_u by assigning bk'' to TBK_u, and $RK_u\langle r\rangle$ to TRK_u, and then computes its (new) hop-chain by adding (BK_u,TBK_u) and c to the hop chain of its parent, ph. We state without proof the properties of the protocol.

Lemma 3 (Initial collusion: len(hop-chain) $\geq$ len(shortest path)). *For every honest node u, $len(H_u) \geq len(S)$, where S is the shortest path from node d to node u.*

Lemma 4 (Initial collusion + Collusion of adjacent nodes). *For every honest node u, if there exists a path P from node d to node u, then $len(H_u) \geq len(P) -$ $(|cor(P)| - |cor\_comp(P)|)$, where $cor(P)$ is the set of corrupt nodes in P and $cor\_comp(P)$ is the set of maximal connected components of P whose elements are all corrupt.*

7 Tolerating Runtime Collusion of Non-adjacent Nodes

The protocol in the previous section will not work if two colluding nodes can share both their permanent as well as their temporary keys provided by the parent of a node. In order to tolerate runtime collusion of non-adjacent nodes, we make the parent of a node u keep the temporary private key of node u. However, implementing this idea is not straightforward. Consider the following modification. The hop-chain format does not change, except that the temporary private key of u is kept at the parent of u. The *rpl* message sent by u contains a nonce that the potential parent v of u should forward to its parent to encrypt with the temporary private key of v. When v receives this *rpl* message, v forwards the nonce to its parent (the potential grandparent of u). Later when v receives the encrypted nonce from its parent, v forwards it to u. Clearly, this will not work because two non-adjacent colluding nodes v and v' can cheat as follows. The farther node v forwards the nonce sent by u to node v' closer to destination, and in turn v' forwards it to its parent. When the parent of v' sends the encrypted nonce to v', v' forwards it to v that forwards it to u.

In the above modification, we need to ensure that the parent of v, the grandparent of u, is a neighbor of v. Thus, we need to run the *neighbor of neighbor* determination protocol described in Section 3 for the owner of the temporary private key of v, which should be a neighbor of a neighbor of u.

The protocol is modified as follows. First, an *ack* message sent from u to v does not contain the temporary private key of v generated by u. Second, when u receives an $adv(bk, t, h)$ message, u needs to check that the owner of the last temporary private key in the chain is a neighbor of a neighbor of u, as well as the validity of the received hop-chain.

Lemma 5 (Collusion: len(hop-chain) $\geq$ len(shortest path)). *For every honest node u, $len(H_u) \geq len(S)$, where S is the shortest path from node d to node u.*

8 Preventing and Mitigating the Sybil Attack

We say that a solution prevents Sybil attacks if no entity can successfully pretend to have more than one identity. We say that a solution mitigates Sybil attacks if the solution limits the number of identities that an entity can have. This is done under the assumption that nodes have vast resources, but we assume that corrupt nodes cannot break the public key encryption scheme in use. In the following sections, we show how the routing protocol can be used to mitigate or prevent Sybil attacks under various assumptions about the network. We start by considering restricted settings, and then we consider a general network.

8.1 Sybil Attack in an Immediate Neighborhood

In the simplest setting, we are interested in determining if two identities that are the immediate neighbors of a node reside on distinct nodes. If the nodes are connected with physical links, then it is easy to distinguish nodes because identities that appear on the same link can be considered to be identical. In this case, the number of physical links determines the number of neighbors.

If the nodes are connected through a wireless link, then the situation is similar to the situation described in Douceur's work [4]. Roundtrip delays cannot be used to differentiate two nodes and it seems that the only way to distinguish nodes is through an approach similar to that of Douceur [4], namely requiring the node to prove that it has the resources of multiple nodes.

8.2 Sybil Attack in a Line

In this section, we consider a number of nodes connected in a line. This case is the basis for our treatment of Sybil attacks in general networks. We only consider how to detect multiple identities that correspond to the same entity or how to detect that a number of identities above some threshold correspond to the same entity. We are not concerned with a corrupt node that drops messages from nodes on its two sides thereby disconnecting them. *The result of the section will be to determine conditions under which corrupt node can successfully launch Sybil attack in a line.* We only describe the approach for the case of initial collusion and collusion between adjacent nodes.

Initial collusion. Consider a sequence of nodes $A = A_0, A_1, \ldots, A_n = B$. These nodes are such that the actual distance from A to A_i is equal to i. Also, the minimum distance from A_i to B is $n - i$. We assume that A and B are honest, the distance between A and B is n, and their public keys are known by all nodes in the line. Nodes A and B are beacons used to locate nodes in the line. Finally, we assume that only initial collusion exists in the network.

Under these assumptions, by Lemma 3, it follows that the length of a hop-chain from beacon A to A_i, $len_A(H_{A_i})$ is greater than or equal to i. Similarly, the length of a hop-chain from beacon B to A_i, $len_B(H_{A_i}) \geq n - i$. If there are no corrupt nodes, then $len_A(H_{A_i}) + len_B(H_{A_i}) = n$. It follows that a corrupt node cannot insert any bogus nodes on a hop-chain without being detected because adding bogus nodes will make the sum greater than n.

If the distance between A and B is not known, and only a lower bound n_{low} and an upper bound n_{high} on the distance are known, any node A_i such that $len_A(H_{A_i}) + len_B(H_{A_i}) \leq n_{high}$ would be accepted. If the actual distance between A and B is n_{low}, then a corrupt node can insert up to $n_{high} - n_{low}$ bogus nodes without being detected. The corrupt nodes as a group cannot insert more than $n_{high} - n_{low}$ bogus nodes (identities) without being detected. Since nodes are colluding only initially, this should also create a dilemma as to which nodes should be the ones to insert the bogus identities.

Initial collusion and collusion of adjacent nodes. Consider adjacent colluding nodes. In this case, using the protocol of Section 6, the nodes can shrink the path, but only by the number of corrupt nodes minus the number of corrupt components. This will not affect the above results, because the number of identities that can be created by corrupt nodes would still be $n_{high} - n_{low}$. The disappeared corrupt nodes can be replaced by other corrupt nodes that appear elsewhere on the line, but the corrupt nodes cannot add more bogus identities than $n_{high} - n_{low}$.

We summarize the results of these sections with the following lemma.

Lemma 6. *On a line, the number of new identities that can be added is not more than $n_{high} - n_{low}$, where n_{high} and n_{low} are upper and lower bounds on the hop-count distance between the end nodes assumed honest.*

8.3 Sybil Attack in a Network

In a general network, under the assumption of no collusion between adjacent nodes, the path from a beacon node to any node cannot be made shorter than it really is. Also, the length of a path from a beacon to any node is not more than the length of the shortest good path. If the network has enough redundant paths, then the distances between nodes are not affected by corrupt routers. In this case, we propose to use hop-count distances of a node from a number of beacons as the identity of the node. There is already a good amount of work on hop-count based coordinates (see [5] for example), and it is not our goal in this paper to study this topic. We simply propose to use our secure routing protocol in conjunction with hop-count based coordinates in order to assign identities to nodes. If the number of corrupt nodes is not large, then corrupt nodes cannot practically affect changes in the location of other nodes.

9 Secure Routing Related Work

Secure distance vector routing protocols have been proposed by many researcher [11,10,12,6,7]. Existing protocols are based on assumptions that are stronger than the ones we make. The protocol proposed in [11] uses a set of the intrusion detection sensors to detect routing attacks, and requires knowledge of the network topology and sensor positions. SEAD [6] does not prevent corrupt nodes from replying advertisement messages, and does not consider colluded attackers. In [7], nodes are assigned to unique identifiers (by a central authority), the destination knows all these identifiers, and the clocks of all nodes are tightly synchronized (to use [8]). RIP-RT [10] assumes that corrupt nodes cannot modify the value of the time-to-live field in a probing message and that any two nodes share a key. Moreover, this protocol assumes that each node knows the identifiers of adjacent nodes. The problem of detecting misbehaving nodes was considered in [1], which also proposes an on-demand secure routing protocol. Our routing protocol focuses on reducing harm caused by corrupt nodes that lie their distances, and does not consider to detect such nodes.

References

1. B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens. An on-demand secure routing protocol resilient to byzantine failures. In *WiSE '02: Proceedings of the 3rd ACM workshop on Wireless security*, pages 21–30. ACM Press, 2002.

2. R. Bazzi and G. Konjevod. On the stabilishment of distinct identities in overlay networks. In *Proceedings of ACM Symposium on Principles of Distributed Computing.*
3. S. Brands and D. Chaum. Distance-bounding protocols (extended abstract). In *Proceedings of EUROCRYPT*, pages 344–359, 1993.
4. J. Douceur. The sybil attack. In *Proceedings of IPTPS*, pages 251–260, 2002.
5. R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI 2005)*, 2005.
6. Y.-C. Hu, D. B. Johnson, and A. Perrig. Sead: Secure efficient distance vector routing for mobile wireless ad hoc networks. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002).*
7. Y.-C. Hu, A. Perrig, and D. B. Johnson. Efficient security mechanisms for routing protocols. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS 2003)*, February 2003.
8. Y.-C. Hu, A. Perrig, and D. B. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, April 2003.
9. T. Ng and H. Zhang. Predicting internet network distance with coordinate-based approaches. In *Proceedings of INFOCOM*, 2002.
10. D. Pei, D. Massey, and L. Zhang. Detection of invalid routing announcements in the rip protocol. In *Proceedings of GLOBECOM 2003*, 2003.
11. V. M. tal and G. Vigna. Sensor-based intrusion detection for intra-domain distance-vector routing. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 127–137. ACM Press, 2002.
12. T. Wan, E. Kranakis, and P. V. Oorschot. S-rip: A secure distance vector routing protocol. In *Proceedings of Applied Cryptography and Network Security*, 2004.