Octopus and Related Protocols

Taral Joglekar
Ryan Wisnesky
CS 259

March 2006

1 Octopus Protocol

1.1 Introduction

Octopus [1] was proposed by Becker and Wille in 1998. The protocol aims to provide a method for establishing a shared key in ad hoc networks. It is a contributory key establishment scheme, in which all the participating nodes contribute some material for the final generated key. The aim behind constructing this protocol is to achieve a lower bound on the number of simple (unicast) messages required to be exchanged for an ad hoc network. In [1] the authors provide a basic protocol, the 2-d hypercube protocol, that achieves this goal for the nodes whose number is a power of two. Then they extend the hypercube protocol to build the Octopus protocol which works for any number of nodes.

1.2 Assumptions

During the specification of Octopus, the authors make some simplifying assumptions. They assume that some kind of authentication protocol has already completed and all the participating nodes have been authenticated. They also assume that a topology for the key exchange has already been decided, wherein each of the participating node is either a part of the 2-d core or is directly connected to one of the nodes in the core. The core members arrange themselves at the nodes of a d-dimensional hypercube, with each node having exactly d neighbors.

1.3 Forming the Network Topology

The octopus protocol does not specify the methods of forming the required topology.

To investigate the formation of the network topology, refer to [2]. This paper gives the adaptation of the CEDAR [3] protocol for formation of the topology. We will later investigate

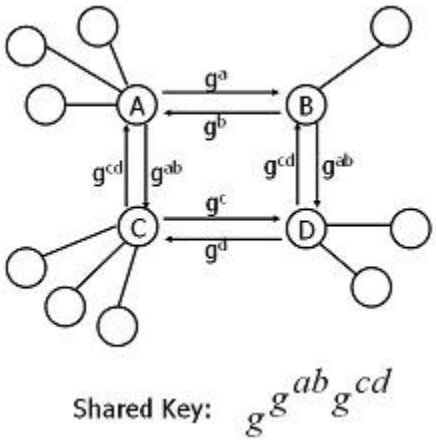

Figure 1.1

the attacks possible on this protocol as specified in [2] and the intruder model for the attacks to succeed.

1.4 Octopus Protocol Overview

The following describes the messages exchanged during the contributory key establishment phase of the Octopus protocol. The topology, as explained above, is assumed to have been established (see Figure 1.1). We show the key exchange for a 4 node core, but it is easily generalized to a 2-d node hypercube core. For this exchange, nodes A, B, C, D are distinguished and form the core. Other nodes attach themselves to one of these nodes and we will name them $Na_1, Na_2 \ldots, Nb_1, Nb_2, \ldots, Nc_1, Nc_2, \ldots Nd_1, Nd_2, \ldots$ respectively. We will call these attached nodes, peers, for a lack of a better term.

Note that all the key exchanges in this protocol use unicast messages and are 2-party DH key exchanges in themselves.

Phase 1: Peer to Core key exchange:

$$Na_1 \leftarrow DH \rightarrow A$$
$$Na_2 \leftarrow DH \rightarrow A$$
$$\ldots$$
$$Nb_1 \leftarrow DH \rightarrow B$$
$$Nb_2 \leftarrow DH \rightarrow B$$
$$\ldots$$
$$Nc_1 \leftarrow DH \rightarrow C$$
$$Nc_2 \leftarrow DH \rightarrow C$$

$$\ldots$$

$$Nd_1 \leftarrow DH \rightarrow D$$

$$Nd_2 \leftarrow DH \rightarrow D$$

$$\ldots$$

At the end of this phase there are keys $Kna_1, Kna_2 \ldots$ are established.

Phase 2: Core to Core key exchange:

Each core member now does a key exchange with its neighboring cores. For this case, each node has exactly 2 neighbors.

1. Each node calculates a key by multiplying all the keys it got from its peers, i.e. $Ka = Na_1 \cdot Na_2 \cdot \ldots, Kb = Nb_1 \cdot Nb_2 \cdot \ldots$, etc.

2. A and B exchange Ka and Kb as DH exponents, while C and D exchange Kc and Kd.

3. A and C as well as B and D exchange g^{KaKb} and g^{KcKd} as key exponents. This gives the final key as
$$g^{g^{KaKb} \cdot g^{KcKd}}$$

Phase 3: Each core distributes the shared key to its connected peers. For example A would send g^{KcKd} and $g^{KaKb \backslash \{Kna_j\}}$ to Na_j, where Kna_j is Na_j's key.

Finally all nodes calculate the same shared key, using their own secret DH value.

2 Modeling the Octopus Protocol

We decided to model the octopus protocol in Murphi, which is a finite state model checker. The aims of this model were

- To test for attacks on the key agreement between various nodes. Key agreement is defined over all the nodes and this means that the final shared key value that each node gets should be the same as any other node. In this context a successful attack would mean that at least a pair of nodes have different values for their shared secret. Though in practice, such an attack only prevents the nodes that have a key different from the other nodes from participating in the conversation, the other nodes continue exchanging messages. The disabled nodes might then cause a re-keying by trying to enter the conversation again.

- To see how type of authentication affects the power needed by an attacker to disrupt the system. In all wireless networks, creating a Denial Of Service by jamming signals at the physical level is always possible, but easily detectable. We do not concern ourselves with such attacks and only concentrate on attacks possible using valid (or

invalid) messages in the protocol framework. The system that evolves from a "rational construction" process would hopefully yield a message exchange strong enough to resist attacks from the most powerful attacker considered in the analysis. Alternately, it would show that there does exist attacks on the main protocol that authentication cannot fix. It is interesting to note that Becker and Wille state in their paper [1], that their aim in building this protocol was to ensure that the agents participating in the key exchange should not need to carry any secret information beforehand. Then they cite two-party DH key exchange as an example of a protocol that achieves this. It must be noted that the attacker that they consider in this model is a passive adversary and hence their claim to the security of the protocol may be true in that circumstance, but for an active adversary, this is certainly not possible as will be shown in the modeling of the protocol.

We decided, given the symmetry in the core, that modeling the interactions between one core member and its connected peers would be enough to decide on whether an attacker can successfully attack the protocol. This decision was motivated by noting that the interactions between any core and its peers is symmetric to any other core-peer interaction, and hence any attacks on one part of the protocol can only be replicated by considering all the nodes. As we will see, the power of an attacker interposing between a core and one of its peers only affects that peer and cannot affect the further key establishment. Also we assume that the cores are able to talk with each other in some secured manner. And even though they might be affected by similar problems as the core-peer interactions, we can always model the core-core interactions are peer-core DH key exchanges and apply the same security features we need to apply for the peer-core case to the core-core case. Another thing to note is that though the initial peer-core interaction is actually a DH key exchange, we can just as well combine both the messages from the core to the peer into one message as the first message from the peer to the core gives the core sufficient knowledge to complete the remaining protocol, without sending the DH exchange message to the peer.

2.1 Modeling the DH Key

Our initial obstacle was to come up with a method to represent the Octopus key along with its components in such a way that it would be possible to not only find meaningful attacks quickly, but also ensure that details in the octopus key exchange are not missed. For example, the secret components in the Octopus protocol are seen at various exponentiation levels in various messages and our primary concern was how important was it to reflect this in the model.

Our first thought was to model all the complexity of the key using a power-set method, with each key being a set in the power set and then writing key simplification rules that would allow each node to calculate as many keys as it can from the data its knows. We quickly rejected this approach, not only because the power set growth is exponential in the

number of nodes, but mainly because all that this would have helped analyze was the attacks possible by a passive adversary. Since it was already proved that a passive adversary cannot eavesdrop on the key (under standard cryptographic assumption that the DH problem is hard to solve), we did not want to include this complexity just to stress-test this aspect.

Other methods developed as we implemented the protocol, and the final method was to implement the key using an array that represents the knowledge of each node about the key contributions of all other nodes. The array was indexed by the node-id and contained the value that the owner node thought as having been contributed by that agent. A secondary array of booleans indicated whether the value contained in the first array was valid or invalid.

2.2 Adversary Model

The adversary could be modeled as:

1. being a part of the network with
    - being a core member
    - being a connecting node
2. being a passive or active agent proxying between a given core and its peers.

It was easy to see the manifestations of 1 without actual modeling:

1. As a core member the node has complete control on which peers it can allow to join to the network, which peers' keys finally contribute to the key exchange and which peers finally join the group by getting keys from it. Since the core is such a critical component in the system, its easy to see that a compromised core can affect the working of the protocol and prevent key agreement. It is interesting to note here that since each core is a part of the "core key exchange" it can always poison the key (by sending different keys to different core members) so that none of its neighboring the core members agree on a single key. This means that a compromised core can harm most of the network.

2. As a peer, it can send only one message and receive one message, the only thing possible for such an attacker is to try to poison the key in such a way that the final key is "weak" in the cryptographic sense. We did not pursue this thread of inquiry actively, though here as some thoughts: A good way for the adversary to "weaken" the key is to always select a small DH key exponent, that can be found by an active adversary by brute force. If the adversary can send 1 as its DH key component then the two-party DH key exchange would always yield the key to be 1. Though this results in a weak two-party DH key, the attack does not generalize to the Octopus protocol. Finally, the model that we pursued was that of an active adversary, interposed between one core member and its peers. Hence we checked for key agreement only between the peers connected to this single core member.

The initial adversary model was that of a passive attacker, snooping on the whole network and being able to intercept and replay messages. This we used until we had successfully modeled the key in the protocol.

The model of an active adversary we built was that of an adversary has control over the whole network. The reason for choosing this model was that we were trying to do "construction" and hence we decided that a new system should be designed with the strongest attacker in mind. Of course, the various abilities of an attacker do define a sort of componentization of the protocol features, with different message components put in to defend against various softs of attackers. To enumerate, the exact abilities of the attacker that we modeled were:

- Ability to eavesdrop and store messages. (Passive)
- Ability to interrupt the flow of any individual message.
- Ability to construct new messages as and when possible.
- Ability to modify only known contents of the messages.

As is obvious, the ability to disrupt flow and modify or corrupt any message would always lead to an attack in which the eavesdropper was successful in preventing the key establishment from ever happening. Hence we do not concentrate as much on the key establishment attacks, as we do on the key agreement attacks, i.e. if the protocol completes in the presence of an active attacker, how much harm can an attacker do to that run. Another justification for this approach comes from the fact that in an ad hoc network the amount of messages exchanged has to be kept to a minimum and hence any disruption of messages from a peer to a core (or vice versa) would be registered as that peer (or core) having left the system or never having entered the system. So this might result in livelocking the system, with all nodes always requesting re-keying to incorporate the network changes. We saw no strong defense against this attack through a protocol message exchange.

A shortcoming of the protocol that we realized was that though the protocol says that the shared key is "contributory", there is no direct contribution of the core nodes to the key! Specifically, if no node is connected to a given peer, then the secret that it uses for the core-core exchange, which is product of all the keys exchanged with its peers, is always 1, and hence a passive attacker that finds a lone core or an active attacker that is able to isolate a core node can effectively find the shared secret.

2.2.1 Modeling Progression

For a passive adversary, the model with no extra parameters gave key agreement. With an active adversary, it failed immediately as adversary could change any and all data. To avoid this we had to use the secrets established during authentication. These were of two types.

1. *Secret keys:* The keys are either weak or strong, but since the secret keys were to be used just once during Octopus, we assumed that there was no problem in encrypting the whole message using these keys. So now we can exchange $E_k(Peer_{Id}, Core_{Id}, \text{DH-component})$ from either side and achieve key agreement. But this cannot work as is because we need to put in the $peer_{id}$ in the message in the clear to let the core know which peer-secret key to use to decrypt the message. One attack is that the adversary can store messages and replay them during later re-keying. This attack comes under the general category of Replay attacks and ia hard to avoid in ad-hoc networks, because there are no authentication rounds. The general way to deal with this attack is to have a one time token between the communicating parties. This could either be a counter, a time-stamp, or a hash-chain (one time passwords). The standard method of challenge-response does not work with Octopus because of the message constraint.

   The other attack is a *message redirect*. If the messages do not contain the identity of the sender as well as the receiver, then an attacker can redirect the message to either a different core/peer.

   After fixing the protocol to handle these attacks, the communicating messages for a secret key protocol are:

   Peer → core:

   $$Peer_{id}, \{g^x, Core_{id}, Peer_{id}, \textit{one-time-password}, \textit{return-nonce}\}_{K_{secret}}$$

   Core → peer:

   $$\{\textit{Other-key-components}, Peer_{id}, Core_{id}, \textit{return-nonce}\}_{K_{secret}}$$

2. *Public Keys:* With public keys, there need not be exchange of any secret keys during authentication, except of course the one required to prevent replay attacks. So if we extend the same protocol developed with Secret Keys to Public Keys, we will need some way to encrypt both messages. If all the communicating parties can have public keys then we can use Public key encryption for both sides.

   Peer → core:

   $$Peer_{id}, \{g^x, Core_{id}, Peer_{id}, \textit{one-time-password}, \textit{return-nonce}\}_{K_{core}}$$

   Core → peer:

   $$\{\textit{Other-key-components}, Peer_{id}, Core_{id}, \textit{return-nonce}\}_{K_{peer}}$$

   But under the assumption that only a few members of the group can have public keys (namely core members) then we need some way to encrypt the out-going message. The peer can provide the key material in the first message.

Peer → core:

$$Peer_{id}, \{g^x, Core_{id}, Peer_{id}, \textit{one-time-password}, \textit{return-nonce}, K_{secret}\}_{K_{core}}$$

Core → peer:

$$\{\textit{Other-key-components}, Peer_{id}, Core_{id}, \textit{return-nonce}\}_{K_{secret}}$$

The next idea is that since the $Peer_{id}$ is no longer necessary for key disambiguation, why send it in the clear. But if we remove that, the core becomes susceptible to a DOS attack. Because then an adversary can keep on sending garbage messages to the core, and cause the core to decrypt all of them before it can discard the message. Since PKE consumes a lot of resources, this creates a DOS on the core. Hence we must keep the $Peer_{id}$ in the clear, which would assist the core to detect a possible DOS attack if the $Peer_{id}$ in the clear and that in the encrypted text do not match.

The next question is, what if the attacker can generate messages. Can he generate a valid message for a different peer? In this case the attacker can generate all other fields except for the one-time password field and hence it cannot generate a valid attack.

Can an attacker do a man-in-the-middle? The problem with an attacker being able to mount a MITM attack is that the public key of the core is obtained by a peer during the authentication phase, which works on a different channel than the wireless one. In fact this channel is mostly a physical channel that needs to be established just for the exchange of these keys [5] and hence we assumed that this channel is sufficiently secured.

Thus the final protocol that gave key agreement in presence of an active adversary was:

Using Secret Keys:

Peer → core:

$$Peer_{id}, \{g^x, Core_{id}, Peer_{id}, \textit{one-time-password}, \textit{return-nonce}\}_{K_{secret}}$$

Core → peer:

$$\{\textit{Other-key-components}, Peer_{id}, Core_{id}, \textit{return-nonce}\}_{K_{secret}}$$

Using Public Keys:

Peer → core:

$$Peer_{id}, \{g^x, Core_{id}, Peer_{id}, \textit{one-time-password}, \textit{return-nonce}, K_{secret}\}_{K_{core}}$$

Core → peer:

$$\{\textit{Other-key-components}, Peer_{id}, Core_{id}, \textit{return-nonce}\}_{K_{secret}}$$

3 CEDAR

CEDAR is used in conjunction with a generalized version of Octopus to provide dynamic core election and key establishment [2]. To look into CEDAR, we mostly used pencil and paper methods after building a Java model of the protocol. The original intention was to use the Java to determine future directions to be explored (for example, we ran the protocol on various topologies with random messages injected to see if we could cause an inconsistency. Such a result would only suggest that inconsistencies were possible.) After building the model, several obvious attacks occurred to us and we thus abandoned the model and thought about them offline. The Java code is included in our submission.

3.1 Overview

The CEDAR protocol is designed to calculate an approximation of the *Minimum Dominating Set (MDS)* of an ad-hoc network. An ad-hoc network is modeled as an undirected graph, where edges represent communication ability and the nodes represent agents. An undirected graph's *Dominating Set (DS)* is a set of nodes such that every node in the graph is either in the DS or is connected to a node in the DS. In other words, for a graph (E, N) with DS S:

$$\forall n \in N \exists d \in S \text{s.t.} (n, d) \in E$$

An MDS is a DS with the minimum number of nodes. A MDS for a graph is referred to as the *core* of the graph. Although a node may be connected to more than one node in the core, each node chooses only one node in the core to be that node's *dominator*. Thus, two distinct dominators will never share dominated nodes.

In terms of communication, if we assume that every node in the core has the ability to communicate with every other node in the core, then it is possible to route messages between any two points. One way this can be accomplished is to core-broadcast a query of which dominator is connected to the destination, and then send the data to that dominator. How these *virtual links* are established is a potential target for attack but is beyond the scope of our project.

For the purposes of Distributed Diffie-Hellman, because every node is either in the core or connects to exactly one node in the core, we can view the core nodes as the core nodes in a generalized Octopus protocol, where more than four nodes make up the core. In other words, Octopus requires connectivity between every node in the core and the rest of the nodes to be in disjoint sets where each set is connected to exactly one node in the core. These pre-requisites are exactly met by running CEDAR.

3.2 Algorithm

Before we can examine how CEDAR works, we need a few definitions:

Let $N_1(u)$ denote all the nodes that node u is connected to.

Let $N'_1(u)$ denote $N_1(u) - \{u\}$. This is often referred to as the *first deleted neighborhood*.

Let $dom(u)$ denote u's dominator.

Let $d(u) = |N'_1(u)|$. That is, $d(u)$ is the degree of the first deleted neighborhood.

Let $d^*(u)$ denote the number of u's neighbors that have chosen u as their dominator. This is referred to as u's *effective degree*.

Now, for the algorithm itself. The following algorithm is run at each node u:

1. Periodically, u broadcasts a beacon $(u, d^*(u), d(u), dom(u))$ to its neighbors.
2. If u does not have a dominator, it sets its dominator to be the node $v \in N_1(u)$ that has the largest value for $(d^*(u), d(u))$ in lexicographic order. u may choose itself as its own dominator.
3. u then sends v the message $(u, \{(w, dom(w)) | \forall w \in N'_1\})$. v then increments $d^*(v)$.
4. If $d^*(u) > 0$, u joins the core.

There are several key points to note about this algorithm:

- The algorithm approximates the MDS because when choosing a dominator, nodes prefer to select nodes that are already dominators, or, failing that, are well connected.
- The algorithm runs in constant time, and depends on the order of messages received. For instance, if a node doesn't hear from its neighbors before it must select a dominator, then potentially that node could choose itself. So, potentially, every node could choose itself.
- The only way for a node to leave the core is by listening to its neighbors beacons and discovering that its effective degree has become zero.
- A node only re-chooses its dominator if the dominator disappears.
- The core nodes do not know the identities of all the other core nodes. (More on this later).
- The generalized Octopus protocol relies on reliable core-broadcast. Thus, if there is inconsistent knowledge about who is in the core, then broadcast can fail, and then key-agreement may fail. Broadcast may also fail by other intruder actions, but these are not investigated here, mostly because of complexity.

3.3 Issues to Investigate

There are several issues we wanted to investigate:

1. Can an intruder prevent the formation of a stable core?
2. Can an intruder cause an inconsistency in which nodes do not agree on who is in the core?
3. Can an intruder cause an incorrect core to be formed?

3.3.1 A Weak Intruder

The answer to these questions depends a great deal on the intruder model. For now, assume that the intruder does not have the ability to interfere with communication; that is, the intruder can not cause cause messages to be dropped.

The answer to the first question is no. By point 4 above, a node only re-chooses its dominator if that dominator has disappeared. So if an intruder cannot make it look like a node's dominator has disappeared, then the intruder cannot force a node to change its dominator. That means that once dominators are chosen, they cannot be changed by the intruder. And because every node will eventually choose a dominator, there is no action the intruder can take to modify the core once that point is reached. So, an intruder cannot prevent the formation of a stable core.

The answer to the second question is a qualified yes. Because of the beacon messages, knowledge is self-correcting. In other words, suppose that $dom(a) = b$. Then b's beacon message to its neighbors includes the information that a dominates b. So, if one of b's neighbors was fooled into believing something else by an intruder, this information will be corrected during the next beacon broadcast. So, the intruder may cause temporary inconsistency, which may wreak havoc during specific times during message routing. But there is no steady state that the intruder can prod the network into where there is an inconsistency.

The answer to the third question depends on the phrasing of the question. There are two interpretations:

1. Can an intruder, who does not form part of the network topology, cause an incorrect core to form? The answer here is no. This is because each node will choose itself or one of its neighbors to join the core. Thus by definition, every node is within distance 1 of a core node, and so a DS will be formed. That being said, it is relatively easy for an intruder to cause many more nodes to join the core than are really required: the intruder just has to communicate to each node that it has chosen that node as a dominator, and that node will then join the core. Note that the intruder cannot permanently force a node out of the core that has been elected into the core, because that node's neighbor that isn't the intruder (of which there must be at least one), will

continue to broadcast that it needs that node as a dominator. If none of the neighbors did need that node in the core, then that node will essentially behave correctly: it will still choose itself or a neighbor as its dominator.

2. Can an intruder who is part of the topology cause an incorrect core to form? In this case, the intruder could communicate to a node that it (the intruder) has a very high effective degree, thus causing the node to choose the intruder as its dominator. In a very degenerate case, where the intruder is connected to every node, the intruder could end up being the sole core node. Technically, such a core is correct, but it is surely not a secure situation. The moral of the story here is that it is very easy for an intruder to cause itself to get elected to the core. However, there is another possibility here: the intruder can masquerade as another node. In such a case, a node would elect the intruder to the core, but the intruder would act as though it were another node in the core. In such a case, the configuration is incorrect because the nodes have an incorrect view of the network topology. It is unclear as to whether such a configuration is "technically" correct or not – See figure 3.3. Regardless, in such a configuration, routing would fail, and thus key-establishment would fail.

3.3.2 A Strong Intruder

These answers to these questions change when we assume that the intruder has the ability to drop messages.

In this new model, the answer to the first question is yes. This is because an intruder that can drop messages is essentially causing a change in the network topology, and changes in network topology cause a reconfiguration of the core set. A simple scenario is shown in figure 3.2. However, from a certain point of view, this scenario would also happen if the messages were dropped naturally, so the extent to which the intruder is causing this attack is debatable, much like in point 2 above.

The answer to the second question is also yes. As before, the intruder can send messages with incorrect dominator information, and then intercept and drop the beacon messages. This would result in inconsistency as to what the core is, and as a result, routing would fail, and thus so would key-establishment. Note that by core inconsistency, we mean neighbors agreeing on whom in their shared neighbors the core members are: no node has knowledge of the entire core set (as long as the core set isn't identical to the node's neighbors.)

The third question changes in the sense that having the ability to drop messages can force a node to remove itself from the core. That, coupled with an intruder's ability to replay messages, can result in an incorrect configuration. See figure 3.4 (a more animated display is included in our presentation slides). This attack is a *wormhole attack* [6].

3.3.3 Authentication

In the preceding discussion, no authentication was assumed. Thus it is natural to ask how the answers to the questions change assuming that there is an authentication mechanism in place. (One may also ask how these answers change if there is a nonce mechanism in place to prevent replay attacks, but nonce's are somewhat tricky in ad-hoc networks because when a node leaves, it is not necessarily sure where to restart its count of nonces. Therefore we omit such a possibility here.)

Suppose the intruder is limited to replaying messages it has already seen, and because of authentication, can only replay to node n messages that actually were addressed to node n. Then the answers to the questions change as follows.

For the first question, the answer is still a yes, because the intruder was not replaying messages to generate this attack.

For the second question, the answer is also still a yes. However, the intruder is limited to providing incorrect information that has already been sent; i.e. it must re-send messages from when the node's beacon was different. However, the effect is still the same. Without the ability to drop messages, the answer to the question is a qualified yes, as before.

For the third question, the interesting case is when the intruder may drop messages and only replay them. This is because without the ability to drop messages, the intruder may only replay messages, and thus it may not force changes in topology which cause a recomputation of the core set. The answer to this question is a yes, although the specific scenarios in which this occurs is more limited before. See figure 3.4.

4 A Digression on GDH-2

The A-GDH.2 protocol suite is a set of protocols that achieve the same results as the Octopus protocol: establishment of a shared secret in a multi-party environment. In [4], the authors examine the security of GDH.2 using a particular mathematical technique. Because GDH.2 is similar to Octopus, we might expect that we would be able to use the techniques from their paper to analyze Octopus. However, that is not the case, and this section shows why.

The GDH.2 protocol is a stepwise protocol in the sense that during each round of the protocol, exactly one node receives information from another node, processes it, and then sends the new result on to another node. The processing that occurs at each step is of the form $output = input^x$, where x is uniquely determined by the node. As such, each node may be viewed as providing a service of the form $s(g^\alpha) = g^{\alpha\beta}$, where g is the base of the Diffie-Hellman protocol. At the end of the protocol, the last node broadcasts the final result and each node is then able to compute the shared secret.

Because there is no authentication in the protocol, each node blindly waits to receive an input and then sends out its output; so, an intruder can essentially "play" the nodes to try to determine the shared secret. In other words, suppose the intruder knows g^x for some x. It can then contact node n_i and calculate g^{xn_i}; it can then contact node n_j and thus

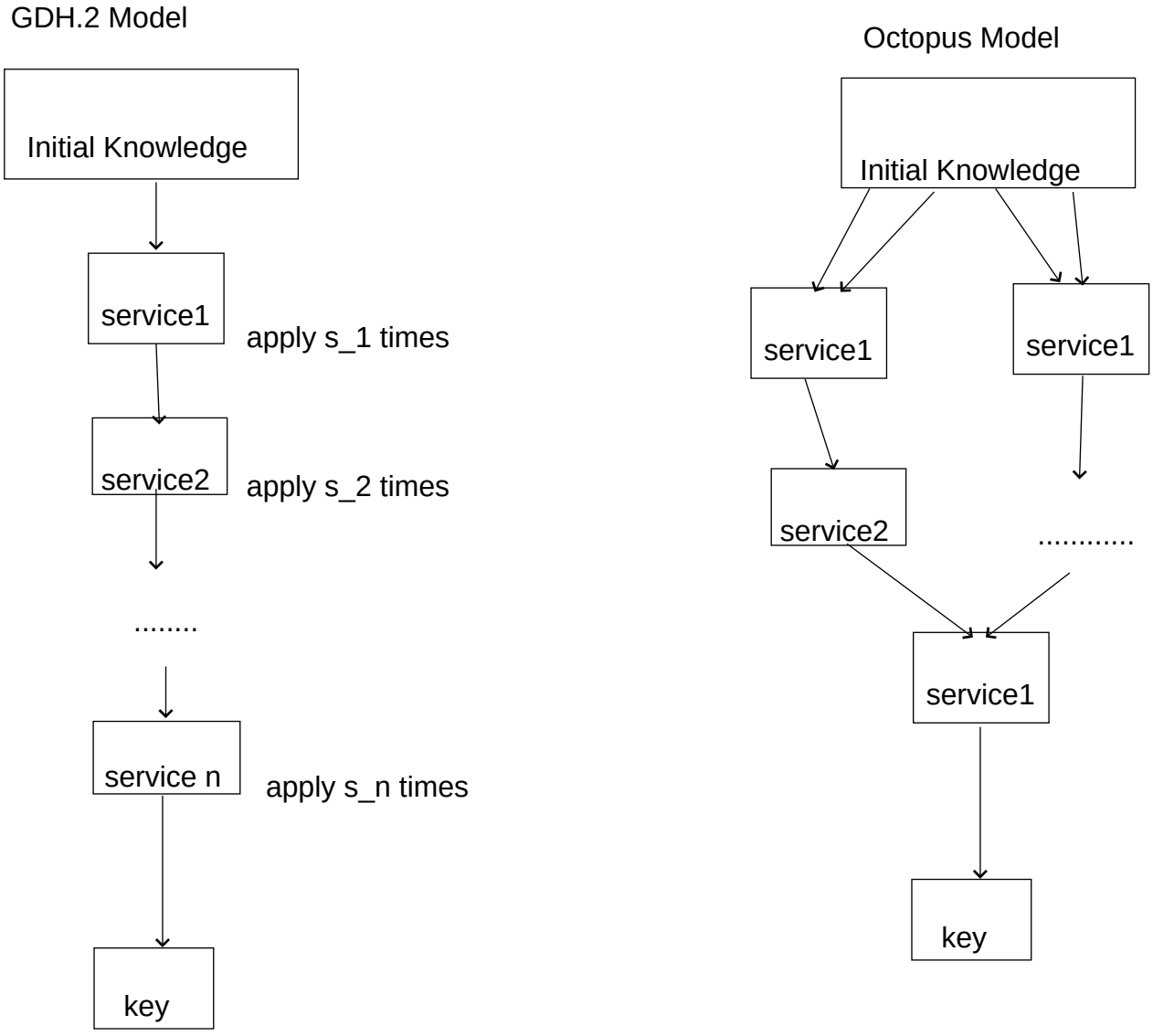
GDH.2 Model
Initial Knowledge
service1
apply s_1 times
service2
apply s_2 times
.....
service n
apply s_n times
key
Octopus Model
Initial Knowledge
service1
service1
service2
.....
service1
key

Figure 3.1

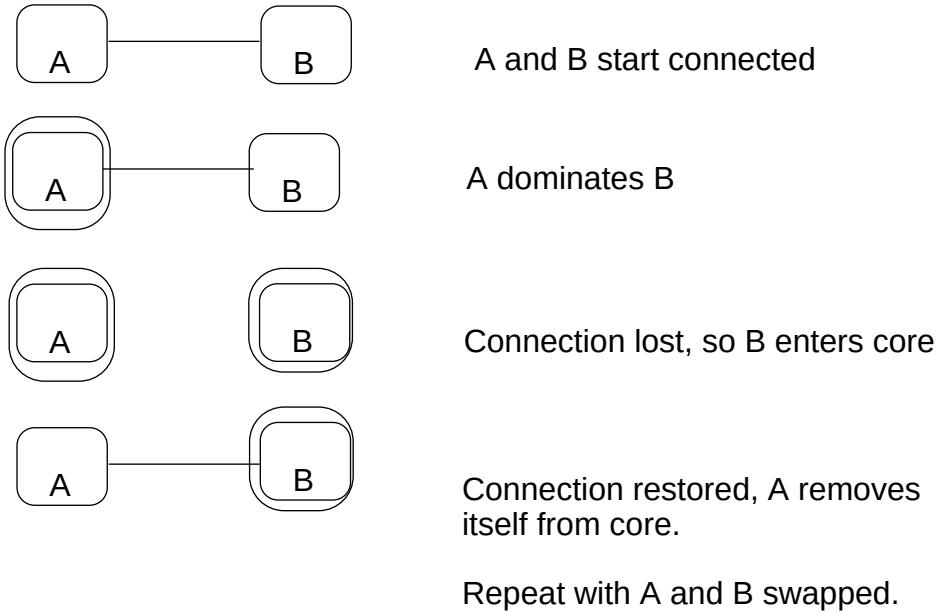

Figure 3.2

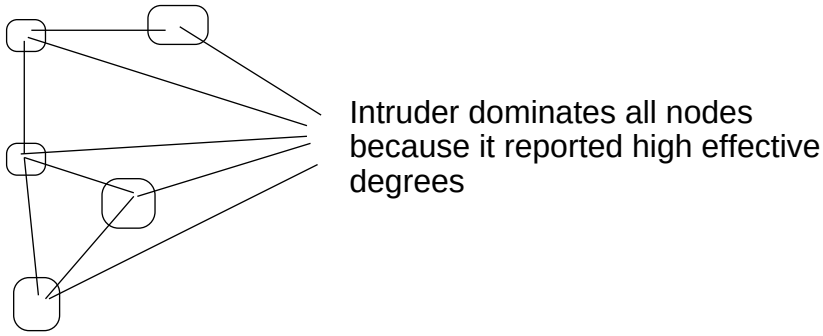

Figure 3.3

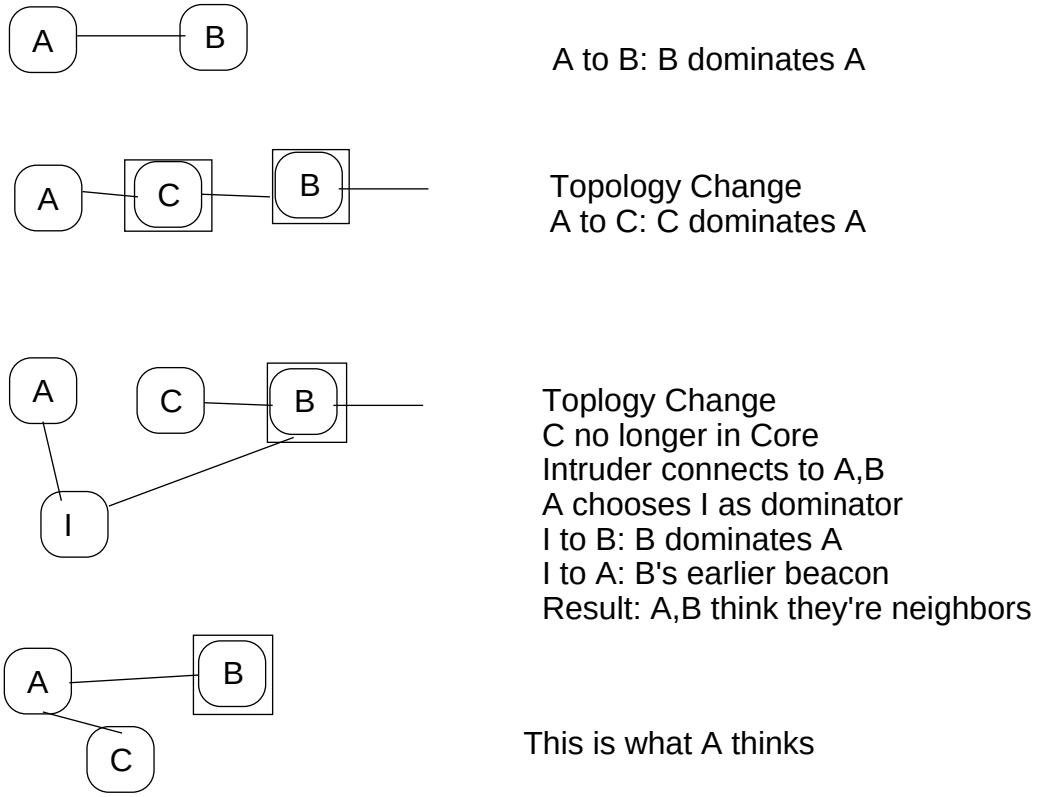
A
B
A to B: B dominates A
A
C
B
Topology Change
A to C: C dominates A
A
C
B
I
Toplogy Change
C no longer in Core
Intruder connects to A,B
A chooses I as dominator
I to B: B dominates A
I to A: B's earlier beacon
Result: A,B think they're neighbors
A
B
C
This is what A thinks

Figure 3.4

calculate $g^{xn_in_j}$, and so forth. (Additionally, the intruder obtains some extra information about inverses, but that avenue is not explored here). So, if the intruder can start from its initial knowledge and play the services to obtain the shared secret, then the protocol is not secure. So, we must determine when the services can be played such that the intruder can obtain the secret. Let e_i denote the exponent that node i raises its input to. Because of commutativity, when the agent uses service i s_i times (in any order), we end up with the equation

$$g^{(e_1)^{s_1}(e_2)^{s_2}...(e_n)^{s_n}=g^k}$$

where g^k is the shared secret. If we expand out the shared secret we get

$$g^{(e_1)^{s_1}(e_2)^{s_2}...(e_n)^{s_n}} = g^{(e_1)^{k_1}(e_2)^{k_2}...(e_n)^{k_n}}$$

And thus we can equate exponents to get a linear system

$$s_1 = k_1 \text{ and } s_2 = k_2 \text{ and } ... \text{ and } s_n = k_n$$

If this linear system has no solution, then there is no way the intruder can play the nodes to obtain the secret. (Actually, we think that this doesn't take into account the fact that the intruder also gains information about inverses, but we haven't explored it further.) In addition, the initial information that the intruder has must also be represented as a service, but that is inconsequential.

So, why can't this technique be used in our analysis Basically, our services do not take one input. For instance, one of our services is of the form $s(\alpha_1, \ldots, \alpha_n) = g^{\alpha_1 \cdots \alpha_n}$. (This is the initial exchange where the core node does pair-wise DH with each of its attached nodes and then communicates with its partner in the core). Therefore, we do not have commutativity when we try to form our equation. In other words, in GDH.2, because services are commutative, we can view them as all being applied one after another, and hence every different ordering of the services is equivalent to the ordering where all of service 1 is applied first, then all of service 2, etc. For instance, applying e_1 and then e_2 and then e_1 again is equivalent to applying e_1 twice and then e_2. This means that we only have to solve one equation to see if there is an attack, because that one equation represents all possible ways of using the services. (Of course, solving that equation involves solving a finite linear system of mini-equations of exponent equality.) Once we move to Octopus and have multiple inputs, we no longer have this property, and hence have to check a different equation for each possible configuration – there are infinitely many equations. In addition, each configuration may itself be arbitrarily long, leading the an intractable problem. Figure 0 illustrates the situation.

5 Conclusion

We found several non-trivial attacks on Octopus (the most interesting of which is the attacker swapping keys without changing the secret), and several on CEDAR (the most interesting of

which is a wormwhole replay attack). In the process we gained experience both with Murphi (which would like to see have the ability to do limited symbolic manipulation someday) and with straight-forward mathematical techniques for examining Diffie-Hellman. On the whole, we are very satisfied with the results. (And we also gained a lot of knowledge about a lot of difference protocols in our long search for the right one to analyze...)

References

[1] Becker et al. *Communication Complexity of Group Key Distribution*. ACM Conference on Computer and Communication Security, November 1998. Available online at `http://citeseer.ist.psu.edu/becker98communication.html`

[2] Du et al. *A group key establishment scheme for ad hoc networks*. Proceedings of the 17th International Conference on Advanced Information Netwoking and Applications. Available online at `http://doi.ieeecomputersociety.org/10.1109/AINA.2003.1192934`

[3] Sinha et al. *CEDAR: Core extraction distributed ad hoc routing*. Proc. of IEEE INFOCOMM '99, 1999. Available online at `http://citeseer.ist.psu.edu/article/sivakumar99cedar.html`

[4] Pereira et al. *A Security Analysis of the Cliques Protocols Suites*. In 14-th IEEE Computer Security Foundations Workshop. IEEE Press, June 2001. Available online at `http://citeseer.ist.psu.edu/pereira01security.htm`

[5] Stajano and Anderson. *The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks*

[6] Hu et al. *Packet Leashes: A Defense Against Wormhole Attacks in Wireless Ad Hoc Networks.* Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003), April 2003.