Secure Pebblenets

Stefano Basagni and Kris Herrin
Department of Computer Science
The University of Texas at Dallas
E-mail: basagni@utdallas.edu
kherrin@utdallas.edu

Danilo Bruschi and Emilia Rosti
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano, Italy
E-mail: bruschi@dsi.unimi.it
rosti@dsi.unimi.it

ABSTRACT

We consider the problem of securing communication in large ad hoc networks, i.e., wireless networks with no fixed, wired infrastructure and with multi-hop routes. Such networks, e.g., networks of sensors, are deployed for applications such as microsensing, monitoring and control, and for extending the peer-to-peer communication capability of smaller group of network users. Because the nodes of these networks, which we term *pebbles* for their very limited size and large number, are resource constrained, only symmetric key cryptography is feasible. We propose a key management scheme to periodically update the symmetric keys used by all pebbles. By combining mobility-adaptive clustering and an effective probabilistic selection of the key-generating node, the proposed scheme meets the requirements of efficiency, scalability and security needed for the survivability of networks of pebbles (*pebblenets*).

Keywords

Wireless sensor networks, Ad hoc networks, Security, Key management.

1. INTRODUCTION

Large collections of very small communication devices networked in an ad hoc fashion will play a growing role in future tactical and commercial telecommunication scenarios. Such a class of networks is critical and unavoidable for many important applications, both in the military and the civilian fields, as they extend the ability to collect data, and to monitor, and control the physical environment from remote locations. As an example, consider a network of sensors submerged in an ocean bed to detect debris after a plane crash, or dispersed on the ground to collect data in a contaminated area. Wireless connections among heterogeneous devices such as cellular telephones and laptops, printers, PDAs, home appliances, etc., are another area where such networks are gaining increasing popularity.

MobiHOC 2001, Long Beach, CA, USA

Common features of all of these scenarios are the number (potentially thousands) of small, inexpensive, wireless nodes, randomly scattered in 3-D space, and the dynamic changes of the topology of the network. Topology changes are not only due to the random and unpredictable movement of the nodes, but also to nodes failure due to depletion of on-board power or, in a military scenario, destruction by the enemy. Nodes may also be added to replace the exhausted, or destroyed, ones.

1.1 Pebblenets

Because of their potentially very limited size and large number, we will call the nodes of such large ad hoc networks *pebbles*, and the corresponding networks *pebble networks* or *pebblenets*. We anticipate each pebble to have embedded processing, storage, communication, and positioning capability, the latter by means of on-board GPS devices or other relative positioning mechanisms.[1] The processing capabilities as well as the storage capacity are limited, so complex computations (e.g., those required by strong asymmetric cryptography) are not possible. Pebbles are battery powered, therefore all their activities must be energy-efficient. Pebbles are tamper-resistant devices, thus the data they store cannot be recovered if attempts are made to physically break them. They are able to communicate with neighboring nodes in order to exchange data (short-range wireless communications), thus "covering" a limited area surrounding it. From a communication perspective, pebblenets are expected to be deployed as support (transport) networks to extend peer-to-peer communication capability among mobile users that cannot directly communicate with each other, either for technological or detection reasons. For instance, squads of robots sent to investigate a large area will use the pebblenet infrastructure to communicate with each other in case they are too far apart to communicate directly with one another.

The nature of pebblenets and their applications, even in the civilian scenario, e.g., industrial process control, environmental or disaster monitoring, medicine, calls for secure communications. The requirements of secure communication are confidentiality, integrity (and/or authenticity), and availability. Confidentiality is the requirement that all com-

[1] GPS devices are not strictly necessary for the protocol presented in this paper. However, because of the domain of applicability of pebblenets, we believe them to be a useful complement of a node.

munications be intelligible by authorized principals only. Integrity is the requirement that all communications be modifiable by authorized principals only, and authenticity is the associated requirement that communications be generated by authorized principals only. Finally, availability is the requirement that the service be available to its users when expected.

1.2 Related Work

Various solutions for securing ad hoc networks have been presented in the literature [15, 7, 13, 9]. In [15] the focus is mainly on protecting routing from attacks such as denial of service and service disruption by forged routing information maliciously injected in the network. The distributed nature of ad hoc networks and the use of cryptography provide the solution to the above problems. The key management scheme proposed to support cryptographic operations is based on a public key infrastructure (PKI) with a distributed certification authority. A set of n nodes cooperate to implement a threshold cryptography scheme that allow them to share the ability to sign certificates, thus tolerating up to a certain number of corrupted nodes. Such a scheme is too heavy for the type of nodes we consider, both from the computational point of view and the storage point of view, i.e., public key cryptography is computation intensive and server nodes are supposed to store all the network nodes certificates. An architecture for authentication in mobile ad hoc networks is proposed in [7] for routers to authenticate management messages. Authentication methods are available ranging from secret key to public key cryptosystems with various types of certificates. The same considerations of the previous case apply here too. In [13], "peanut nodes" with small CPUs and batteries are considered for which only strong symmetric cryptography is feasible. The issues of availability, authenticity, integrity, and confidentiality are examined. "Secure transient associations" are introduced to address the issue of authenticity, as the association existing between pairs of principals, such as controllers and devices, as long as there is a relationship between the two, such as ownership or membership in a system. Associations are transient so as to allow a device to change the system it is part of without destroying the device. In [9] the authors present an application of the PLAN active network programming language [8] based on reflection, namely, the ability of any programming language to treat pieces of code as data or as code, depending upon the context. PLAN is used to implements the VersaKey multicast key management scheme [14]. However, because the devices considered in [9] are more powerful than those comprising our pebblenets, we do not consider this solution here.

1.3 Our Contribution

In this paper we propose a key management scheme for pebblenets that allows for the efficient exchange and update of keys to secure traffic in a pebblenet. Similarly to [13], because of the limited computing power of the pebbles, we only consider symmetric cryptography as feasible in pebblenets. Furthermore, because of the nature of the nodes, their large number, and the type of applications where they are deployed, it is sufficient to guarantee group membership of the nodes rather than individual identity. Such a restriction improves on system efficiency since group membership is verified using a single symmetric key for the entire group, thus saving on storage space and search time, rather than as many individual keys as there are members in the group as required by individual authentication.

Pebblenets define dynamic groups as the number of pebbles in a network may change due to "leaves," i.e., pebbles disappearing from the network because of exhaustion of their battery power, etc., and "joins," i.e., additions of new pebbles to replace the exhausted ones or to extend the network coverage. Power limitations of pebblenets suggest to restrict key management activities in order to preserve power. Under these circumstances, periodic key updates are preferable to those triggered by membership changes, e.g., leaves and joins, when such changes occur quite frequently, especially in large and highly dynamic groups [12]. By adequately tuning the key update period, a trade-off is attained between the conflicting goals of keeping a high security level and minimizing the pebble power consumption. More secure systems may require more frequent key updates, which implies more activity, hence more power consumption. Mixed strategies, where key updates occur periodically and also upon joins of batches of new pebbles, are also possible.

Each pebble is equipped with a group identity key, which guarantees group membership. Once a pebble is recognized as part of the group, it participates in the key management process, which is a network wide operation that involves all the pebbles. Data traffic is protected using a global key shared by all nodes, the *Traffic Encryption Key*, or TEK. TEKs are periodically refreshed, or updated, with a period whose duration is a function of the security level required. The key management scheme we propose realizes the periodic generation of a new TEK for securing communication throughout the pebblenet, and its secure distribution to all the pebbles.

At each new re-keying one of the pebbles emerges as a key manager that generates and distributes the new TEK to all the other pebbles. Given the large number of pebbles, it is not feasible to select the key manager among all pebbles. Consequently, we partition the network into clusters, among whose leaders the key manager is selected. The protocol is comprised of two phases. First, the network nodes organize into clusters with clusterhead and ordinary nodes. The clusterheads, which are only a small fraction of the total number of pebbles, subsequently organize into a backbone. Finally, a fraction of the clusterheads of the backbone is selected among which the pebble is found that will generate the new TEK.

The strength of our method relies on a secure, scalable clusterhead selection protocol which selects only a small fraction of the pebbles in a distributed way. Thus, disregarding the increasing number of pebbles, the key manager is selected only among very few nodes. We actually reduce the number of candidates for the key manager position further, by allowing only some of them to participate in the key generation and distribution process.

Given the nature of pebblenets, it is computationally unfeasible to use centralized algorithms, i.e., algorithms that require the knowledge of the entire network topology. A

solution has to be found that allows the clusterheads to decide whether to participate in the key management protocol knowing only little information about the current topology of the network. This may prevent us from selecting a unique clusterhead to be the key manager. Our proposed key management scheme succeeds in producing the needed unique TEK by implementing an effective probabilistic mechanism for the selection of the key-generating node. In the rare case that more than one TEK is produced in the same re-keying by different nodes, we propose an efficient method that enables each pebble to retain the same one.

The proposed, periodic backbone construction mechanism is general with respect to the communication protocols used to deliver data in the pebblenet. It is to be used only for the re-keying. The generated TEK is then used to encrypt data traffic which is delivered where intended according to communication protocols that may not need a backbone-like pebble organization, or may need a different pebble organization.

This paper is organized as follows. Section 2 illustrates the problem considered in this paper and the systems it applies to. Section 3 describes the proposed key management in detail. Section 4 describes how network splits and additions of pebbles are managed. Section 5 summarizes our contributions and concludes the paper.

2. PROBLEM DEFINITION

2.1 The System

Pebbles are born equal since there is no way to guarantee that a specific pebble will survive once they "hit the ground." Each pebble is equipped with a secret *group identity* key K_{GI}, the same for everyone, a one-way hash function h, a secure key generation algorithm, and a strong symmetric encryption algorithm [11]. The group identity key guarantees the authenticity of a pebble as a member of a group, not its individual identity. Because pebbles are small devices with limited computational capability, we believe that group membership is sufficient for the uses of this type of network. Were individual pebble identity a system requirement, more powerful devices should be envisioned as system components. Furthermore, we assume that pebbles are honest and there is no malicious pebble in the group.

Pebbles have a local identifier (ID) and are able to compute a *weight*, i.e., a numerical value that expresses what the current status of that pebble is. Such weights should account for node parameters such as mobility, battery power level, distance from the other nodes, values related to the surrounding environment (terrain, temperature, luminosity, etc.) and many others, and their variations in time. For instance, as introduced in [1, 2, 5], we assume the following expression for the computation of node v's weight:

$$w_v = \sum_{i \in I} c_i P_i$$

where the c_is are the (constant) weighing factors for the $|I|$ system parameters of interest P_i.

The tamper-resistance properties of the pebbles [10], imposed by the type of applications that will deploy them, protect the network from adversaries trying to insert malicious nodes in the network after capturing honest ones. An adversary trying to gain access to the identity key by physically extracting it from the pebble, would definitely damage it thus thwarting the attempt. Malicious nodes could not then be recognized, therefore accepted, by the group, as they do not have the identity key.

Tamper-resistant devices provide for a "safe" leave in that when a pebble is no longer part of the network, for any reason, it will not be able to access the traffic exchanged after it left. This may happen either because the pebble is no longer operative (i.e., it ran out of power) or because an adversary broke it by tampering with it trying to gain access to the group identity key. For this reason, the protocol we propose in the next section will not consider leaves.

On the contrary, joins will be considered to account for group enlargements aiming at replacing non-operative pebbles or adding new ones. The problem of merging different sets of pebbles (such as those that occur after the network splits) will also be considered.

2.2 The Problem

Secure pebblenets should guarantee data traffic confidentiality and authenticity. By data traffic we mean all the traffic on the network, be it routing or application oriented traffic, as opposed to control traffic, by which we mean messages exchanged during the key management protocol.

Data traffic is encrypted by all pebbles using the same key, the Traffic Encryption Key (TEK), which changes during the network lifetime in order to protect the system from cryptanalytic attacks and to cope with changes in the pebble set due to "natural" pebble extinction. The problem we consider in this paper is that of securely distributing and updating the TEK among the pebbles, thus ensuring confidentiality. As for pebble authenticity, this is provided by the imprinted group identity key K_{GI}. For this reason, as we will show in the next section, we limit as much as possible the use of K_{GI} in order to minimize its exposure to cryptanalytic attacks and generate keys from it that we use to securely distribute the TEK. In general, it is not a safe practice to have a single network-wide key from which a set of other keys are derived. However, because of the limited resources available at each pebble and the very short lifespan of the derived keys—a round of the re-keying algorithm each—we deem the use of a globally shared key cost-effective and safe.

The dynamic nature of the group, due to pebble additions and losses, imposes that a dynamic protocol be designed. Such a protocol should offer provision for dynamic key manager (a pebble) selection at start-up time and during the life of the network. Because of the large number of pebbles in the network, some form of complexity reduction is necessary in order to minimize the amount of control traffic generated by the key management operations. To this aim, we propose a scalable key management mechanism based on mobility-adaptive clustering, and an effective probabilistic selection of the key-generating node.

3. THE RE-KEYING PROTOCOL

In this section we describe in detail the key management protocol to secure communication in large pebblenets. Because of the nature of the pebblenet, the key manager must be dynamically designated at run-time and may change each time the protocol is executed. The protocol is performed periodically, with the period being decided according to the needed level of security. Selecting a different key manager each time allows us to always select it among the fittest nodes for the role. This also increases the overall security of the pebblenet since the selection of the manager cannot be predicted.

The protocol is comprised of two phases.

1. The network nodes organize into clusters on which a backbone is superimposed.
2. A small number of the backbone nodes start generating the new keys, which are then broadcast to all nodes. Within finite time, each pebble has received the same unique TEK.

The protocol operations of the first phase comprise the following steps.

1. Based on pebble weights recomputed each time the protocol is executed, the pebblenet is partitioned into *clusters*. Each cluster consists of one *clusterhead*, selected among the nodes with the highest weight, hence expectedly most suitable for an active role in the key management process, and a (possibly empty) set of non-clusterhead, or simply *ordinary*, nodes.
2. The clusterheads organize themselves in a *backbone*, which is guaranteed to be connected (if the pebblenet is) in the face of node mobility and node/link failures.

The protocol operations for the second phase concern key generation and distribution.

1. Only a few clusterheads in the backbone start the key generation process. Possibly more than one (but at least one) among these clusterheads eventually generates the new TEK, but exactly one TEK will become the one that every pebble will use.
2. Once it is generated, the new TEK is broadcast securely along the backbone to all the clusterheads, which in turn securely transmit it to the ordinary nodes in their clusters.

In the following we describe the details of the two phases.

3.1 First Phase

3.1.1 Secure Clusterhead Selection

Once the pebbles have settled, they start executing a secure version of the clusterhead selection protocol. The original protocol [1, 2], based on the construction of a *maximal weighted independent set* in wireless networks [3], constructs a partition of the pebblenet into clusters whose diameter is at most two and in which no two clusterheads are neighbors. Here, it has been modified in order to make use of secure communications.

The clusterhead selection protocol is designed in such a way that each pebble, by just knowing its own identifier (ID) and weight, as well as the IDs and weights of its one-hop neighbors, can autonomously decide its role, i.e., whether it will be a clusterhead or an ordinary node. The basic rule of the protocol is that a node can decide its role when all the nodes with larger weight have already decided their own role.

The whole clusterhead selection procedure comprises three steps:

1. neighbor discovery;
2. clusterhead selection, and
3. "neighboring clusterheads," discovery.

The three steps are described here for the generic ith re-keying.

In the first step, each pebble makes its active neighbors aware of its presence by broadcasting an initial HELLO message, containing its ID and weight, encrypted using $K_H^i = h(K_B^{i-1})$, i.e., the key obtained by applying the one-way function h to the backbone key K_B^{i-1} generated in the previous re-keying (the $(i-1)$th) and initialized as $K_B = K_{GI}$. This way we minimize K_{GI} exposure to cryptanalytic attacks by circulating keys derived from it, rather than K_{GI} itself. Note that because the K_H^i is obtained by applying the one-way function h, recovering K_{GI} from it is practically impossible. Pebbles may keep a local counter to know how many times to apply h to K_{GI} in order to optimize key versioning.

Each pebble p advertises its presence by broadcasting the following message to all its neighbors N_p:

$$p \Rightarrow N_p : E_{K_H^i}(w_p|ID_p|MAC(K_{GI}, \langle w_p|ID_p\rangle)).$$

Here $MAC(k, \langle x\rangle)$ is a keyed message authentication code applied to message x using key k, and $E_{K_H^i}(x)$ is the encryption of x using key K_H^i. The symbol $|$ indicates the concatenation of two messages. Using the group key to compute the MAC guarantees the integrity of the message and the authenticity of its origin, i.e., the membership of the transmitter pebble to the network. Note that the mere encryption of the message using K_H^i does not fully guarantee the sender's membership to the network, because an adversary who had managed to recover K_H^i from a cryptanalysis of the traffic, could inject a similar encrypted forged message. On the contrary, computing the MAC requires K_{GI} that only network members have.

Notice that pebbles may also use HELLO messages to keep their knowledge of the local topology up to date according to the requirements of a given communication protocol. Those HELLO messages should not be confused with the (secure) ones described above, which instead are only used (for topology discovery) to implement part of the first phase of the proposed protocol.

Once the nodes have gathered information about their neighbors, the *second step* begins. The clusterhead selection protocol proceeds now securely at each node using the following rule: A node p broadcasts a message that communicates its role only after each of its neighbors with larger weight have decided their own role. The message used in this case is the following:

$$p \Rightarrow N_p : E_{K_H^i}(w_p|ID_p|R|MAC(K_{GI}, \langle w_p|ID_p|R\rangle)).$$

This message is similar to the HELLO message, except for the parameter R which defines the type of message. The selection protocol makes use of two types of messages.

1. A message with $R = CH$ is sent by a node p that decides to be a clusterhead. This happens when p's weight is the biggest among those of its neighbors that have not decided their role yet. This means that nodes whose weight is bigger than p's (if any) have already decided that they will be ordinary nodes.
2. A message with $R = v$, v being a pebble different from the sender, is broadcast by a node that is going to be an ordinary node affiliated with the clusterhead v. This always happens when a neighboring node v with bigger weight has sent a message with $R = CH$. In this case v is the sole clusterhead with which that node is affiliating. (This implies non overlapping clusters, i.e., the fact that a node belongs only to one cluster.)

The total ordering of the pebbles' weights guarantees that there is always a node that starts the protocol, and also its termination when a role has been assigned to each pebble.

Once the clusterheads have been selected, forming the set $\mathcal{C}$, each clusterhead c in $\mathcal{C}$ creates and communicates to its cluster members M_c a cluster key K_c^i via the following message:

$$c \Rightarrow M_c : E_{K_H^i}(ID_c|K_c^i|MAC(K_{GI}, \langle ID_c|K_c^i\rangle)).$$

The key K_c^i will be used for all intra-cluster operations, in particular it will be used to communicate the new TEK from the clusterhead to each of its ordinary nodes.

The cluster key is also used in the *third step* of the clusterhead selection protocol, where the ordinary nodes communicate to their clusterhead information about other clusterheads at most three hops away. Clusterheads at most three hops away from each other are considered "neighboring clusterheads" (this being neighbors, in general, may be non physical, i.e., two neighboring clusterheads may not be able to receive directly each other transmissions). This information is needed for the (secure) construction of a connected backbone of clusterheads, and it requires each ordinary node to broadcast a message to its own clusterhead.

In the case an ordinary node p is neighbor of two (or more) clusterheads c_1 and c_2, having received their messages with $R = CH$, it can now send (securely) this information to its clusterhead, say c_1. Furthermore, since c_2 has previously received a message from p with $R = c_1$, it is aware of the presence of c_1 two hops away.

In the case the ordinary node p_1 that belongs to the cluster of a clusterhead c_1 is neighbor of another ordinary node p_2 belonging to the cluster of the clusterhead c_2, it received the message from p_2 with which p_2 declared that it was joining the cluster of c_2. Thus, as before, it can (securely) send this information to c_1, making him aware of the presence of c_2 three hops away.

We illustrate the operations of the second step (the somewhat more involved step) of the described selection protocol with the following example. We consider the 23 node pebblenet depicted in Figure 1 where the nodes IDs are identified with their weights. (For the sake of clarity, we assume here that all the nodes are turned on at the same time.) Initially, nodes 18, 20 and 23 are the only ones that can make a decision since there are no other pebbles in their neighborhood with a larger weight. Thus, they declare themselves as clusterheads by broadcasting a corresponding (secure) message to their immediate neighbors. Upon receiving this message, nodes 15 and 16 join the cluster led by node 18, nodes 1, 14 and 19 affiliate with node 20 and nodes 2, 10 and 22 join the cluster led by node 23. The join operation is performed by a node by broadcasting a corresponding message to *all* its neighbors (not just to the selected clusterhead). At this point, nodes 12, 13 and 21 know that neighboring nodes with larger weight have decided their role (namely, they are ordinary nodes) and broadcast a "clusterhead message" to all their neighbors. Knowing the largest clusterhead in their neighborhood, nodes 3, 8 and 11 join node 12's cluster, nodes 7 joins the cluster of node 13 and nodes 4 and 17 affiliate with node 21. Node 9, that was waiting for a decision from node 11, is now aware that node 11 is not going to be a clusterhead and broadcast that it is going to be a clusterhead. Finally nodes 5 and 6 can make their decision: they broadcast a message declaring that they affiliate with node 9 as ordinary nodes. The selected clusterheads are the squared nodes in the figure.

Notice that having different cluster keys, two neighboring nodes in different clusters cannot communicate with each other. As a matter of fact, there is no need for them to communicate to implement the described key management protocol. The two nodes may need to communicate in order to forward data. In this case, they will use the latest TEK generated.

Message and time complexity. The clusterhead selection protocol described so far requires the transmission of n mes-

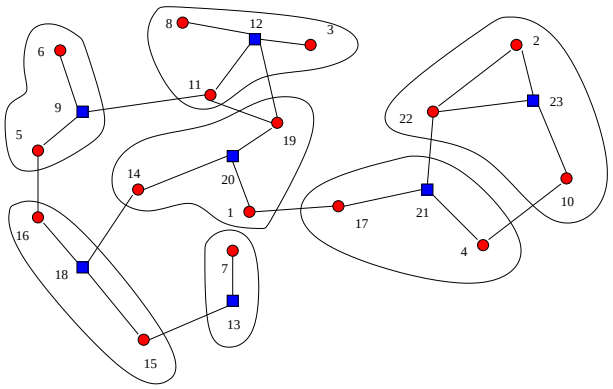

Figure 1: A pebblenet is partitioned into clusters (the squared nodes are the clusterheads).

sages for the neighbor discovery phase, n messages for selecting the clusterheads [1], k messages from the $k < n$ clusterheads to their ordinary nodes to communicate the cluster key, and $n - k$ messages for the communication from the ordinary nodes to their clusterhead about the clusterheads at most three hops aways. This sums up to a message complexity which is proportional to the number of pebbles (more precisely, the number of messages is $3n$). These messages are also very short in length. Provided that the nodes' weight is polynomial in the number n of the nodes in the pebblenet, to transmit the needed information a number of bits proportional to $\log n$ is sufficient, with a small proportionality constant.

In the case the network is synchronous and the nodes start their operations all at the same time, it has been proven that the number of rounds[2] needed to complete the clusterhead selection protocols is bounded by twice the number of selected clusterheads [3].

3.1.2 *Secure Backbone Construction*

The selected clusterheads start communicating to build a backbone of clusterheads ($\mathcal{CB}$) using $K_B^i = h(K_H^i)$ as the encryption key for this phase. Note that because a one-way function is used to derive a new key for this phase, forward secrecy may be jeopardized if anyone were to compromise a key at some point. On the other hand, backward secrecy, and ultimately the group identity key, are guaranteed by the one-way nature of the function. This allows the group to safely use the group identity key K_{GI} for MAC computations without exposing it to direct cryptanalytic attacks.

In order to guarantee a connected backbone, each clusterhead needs to be connected to all its *neighboring clusterheads*, namely, the clusterheads that are two or three hops away. Connecting clusterheads that are at most three hops away from each other is necessary and sufficient condition to guarantee that, given a connected pebblenet, the resulting backbone is connected as well, as proven in [6].

As described earlier, the information about the neighboring clusterheads is provided to a clusterhead by its ordinary nodes. This information is easily gathered during the secure clusterhead selection step by the ordinary nodes and its transmission to their clusterhead c is secured by using the cluster key K_c.

The backbone is then obtained by connecting each clusterhead, either virtually (by implementing the connection over at most three physical one-hop links) or physically (by using directional antennas), with all its neighboring clusterheads. The backbone resulting by connecting the clusterheads in Figure 1 is depicted in Figure 2 (the thicker links are the backbone links).

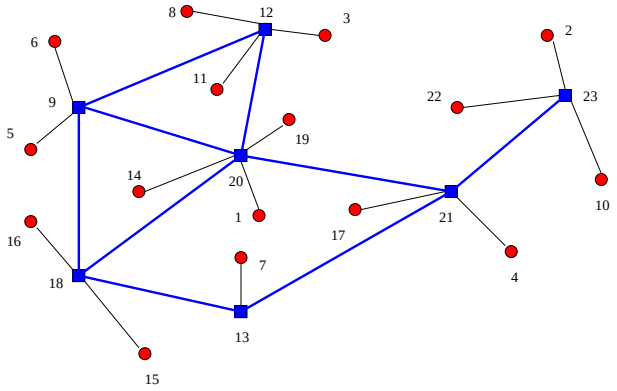

Figure 2: The secure communication structure imposed by the backbone of clusterheads.

3.2 Second Phase: Key Generation and Distribution

The TEK used to encrypt data traffic is changed on a regular basis to ensure security. Furthermore, to increase the security level, an indirection level may be used so that the current TEK is actually used to encrypt a session key, which is used to encrypt the data payload. Since the session key is generated for each message and distributed along with the payload it encrypts, we only consider the problem of updating the TEK. The frequency of TEK changes depends on the level of security desired in the network. We assume that such a value, being of interest for the user, is derived from a coded parameter each pebble is equipped with.

A key manager is necessary that initiates the procedure. For the reasons explained above, the key manager should be one of the nodes better equipped for this role, which is expressed by a node's current weight. Hence, the competition for being a key manager is among the clusterheads only. A natural choice would be to have the clusterhead with the largest weight to take care of the TEK generation. However, as a result of the previous phase, no clusterhead knows who such an "absolute leader" is, since each clusterhead has only a local knowledge of the backbone topology. The most a clusterhead can check is whether it can be a *potential key manager* (PKM) based on its weight being larger than its backbone neighbors'. For instance, in the pebblenet of Figure 2, only node 23 will be a PKM, since all the other clusterheads have at least a backbone neighbor with larger weight. However, node 23 is only aware of hav-

[2] A round is the time required to broadcast a message from a node to all its one-hop neighbors.

ing a weight bigger than its only backbone neighbor (node 21). Since gathering information about which clusterhead has the largest weight among all the clusterhead can be too expensive in terms of network and pebbles' resources, each PKM is eligible to generate and distribute the new TEK. In an attempt to avoid to have multiple PKMs generating the TEK, the key generation is started after an exponential delay with average Δ, which is a coded parameter each pebble is equipped with. Once the TEK is generated it is immediately broadcast to all the clusterheads in the backbone. There is a chance are, of course, that two or more PKMs can generate and start the distribution of a new TEK before receiving each other's TEKs. We call this event a *collision*. Since each clusterhead generates a random number drawn from an exponential distribution with average Δ, the probability of a collision, is very low.[3] However, because of the network size, two or more distant PKMs might independently initiate the procedure before they become aware of each other having generated a new TEK. In this case the collision is resolved as follows.

At the end of the ith secure backbone construction (which is related to the ith rekeing), and after waiting for its exponentially distributed delay, a PKM c generates the new TEK t^i and broadcasts it to the other clusterheads encrypting the corresponding message using the current backbone key K_B^i:

$$c \Rightarrow \mathcal{CB} : E_{K_B^i}(ID_c|w_c|t^i|MAC(K_{GI}, \langle ID_c|w_c|t^i\rangle)).$$

(With $c \Rightarrow \mathcal{CB}$ we indicate the secure, multi-hop broadcast of the new TEK t^i throughout the backbone.) A clusterhead c' that receives the message above reacts in the following way. If c' is not a PKM or it is a PKM whose local timer has not gone off yet (i.e., the exponential delay has not expired yet) or it has been reset, then it will retain t^i, considering it as the new TEK, only if w_c is larger than the weight of all the PKMs that have generated the TEKs in the current (ith) re-keying and received so far (if any). If c' is a PKM and it has not reset its local timer, it resets it at this time. In the case c' is a PKM that has already generated and sent out a TEK, it will ignore the received one if $w_{c'} > w_c$, otherwise it will keep the received one as the current TEK. In case the two weights are the same, we can use the largest node ID as the tie breaker. Because the backbone is connected, all clusterheads will eventually receive a TEK that will be the same for everyone. At this point, each clusterhead c distributes to the ordinary nodes in its cluster M_c the new TEK t^i by using the cluster key K_c^i:

$$c \Rightarrow M_c : E_{K_c^i}(ID_c|t^i|MAC(K_{GI}, \langle ID_c|t^i\rangle)).$$

Simulation results in pebblenets with up to 2000 nodes show that the potential key managers are a very small fraction of the clusterheads (which, in turns are a very small fraction of the number of the pebbles [4]). Figure 3 shows the average number of PKMs. This number is basically dominated by the logarithm of n, n being the number of pebbles.

[3] The exponential delay adopted here is like Ethernet exponential back-off after a collision.

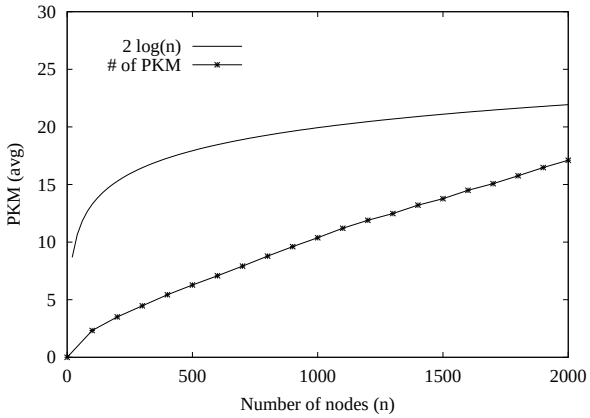

Figure 3: Average number of potential key managers (PKM) in pebblenets with 2000 nodes.

(Simulation results with confidence level of 95% with a precision within 5% refer to connected networks of pebbles randomly and uniformly scattered in the plane.)

In this section we have focused on the generation and secure distribution of the TEK. Since the keys used for its distribution and for the cluster construction are derived from K_{GI} by applying h an adequate number of times, there is no need to distributed them. Each node can recompute the key it needs based on the counter i of the number of times it has executed the protocol.

4. SPLITS AND JOINS

In this section we briefly outline solutions for the problems of when a network is partitioned into multiple connected components (splits) and when one or more pebbles are added to the network (joins).

In the first case, the re-keying process described above works correctly in each of the connected components into which the pebblenet is divided. (A connected component is a separate network with a smaller number of pebbles.)

When a new set of pebbles join the network, they initiate a new round of the protocol by sending out a HELLO message as described in Section 3.1.1. The nodes from the previous batch that are still active will reply using the current version of K_H, which the newcomers can derive by applying h to K_{GI} a sufficient number of times. Once they have synchronized with the other nodes, the clusterhead and backbone construction proceed as described.

A problem may arise with splits when, with the introduction of new pebbles, two formerly separated connected components become a single component where the nodes are now using different TEKs. In this case the problem can be solved by securely disseminating the TEK generated by the pebble with the largest weight among the key managers of the connected components. In this case too, the TEK can be securely broadcast by encrypting it with the key $h^m(K_{GI})$, where m is the largest exponent currently used in either group for the respective series K_H, K_B.

5. CONCLUSIONS AND FUTURE WORK

The paper introduced a solution for the problem of secure communication in large ad hoc networks of small, resource-constrained nodes, termed pebbles. For these networks that require symmetric key encryption as the only feasible solution for secure communication, we proposed a key management scheme for the periodic generation of a new traffic encryption key (TEK). To this aim, among the large number of pebbles, only a small fraction of nodes is selected among which a further small set is selected whose nodes compete for being the key manager. This competition is finally decided probabilistically, by having each competing node using an exponential delay function to determine when and if to start the key generation and distribution process. Efficient solutions for the case in which more than a TEK is traveling the network, as well as for "splits" and "join" situations, are also provided.

The presented work concerns the definition and a preliminary study of the problem of securing communication in pebblenets. Future research directions include a thorough evaluation of the secure backbone construction phase with an emphasys on the corresponding overhead, and of the resources needed for computing and distributing the new generated keys. Other issues concern the test of the effectivity of our probabilistic mechanism in selecting only one key manager at each re-keying.

6. REFERENCES

[1] S. Basagni. Distributed clustering for ad hoc networks. In A. Y. Zomaya, D. F. Hsu, O. Ibarra, S. Origuchi, D. Nassimi, and M. Palis, editors, *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, pages 310–315, Perth/Fremantle, Australia, June 23–25 1999. IEEE Computer Society.

[2] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *Proceedings of the IEEE* 50*th International Vehicular Technology Conference, VTC 1999-Fall*, volume 2, pages 889–893, Amsterdam, The Netherlands, September 19–22 1999.

[3] S. Basagni. Finding a Maximal Weighted Independent Set in Wireless Networks. *Telecommunication Systems*, Special Issue on Mobile Computing and Wireless Networks. In print.

[4] S. Basagni, D. Turgut, and S. K. Das. Mobility-Adaptive Protocols for Managing Large Ad Hoc Networks. In *Proceedings of the IEEE ICC 2001*, Helsinki, Finland, June 11–14 2001.

[5] M. Chatterjee, S. K. Das, and D. Turgut. An on-demand weighted clustering algorithm (WCA) for ad hoc networks. In *Proceedings of IEEE Globecom 2000*, pages 1697–1701, San Francisco, CA, November 27–December 1 2000.

[6] I. Chlamtac and A. Faragó. A new approach to the design and analysis of peer-to-peer mobile networks. *Wireless Networks*, 5(3):149–156, May 1999.

[7] S. Jacobs, M.S. Corson. MANET authentication architecture. Internet draft: draft-jacobs-imep-auth-arch-00, March 1999.

[8] P. Kakkar, M. Hicks, J. T. Moore, and C. A. Gunter. Specifying the PLAN networking programming language. *Higher Order Operational Techniques in Semantics*, Electronic Notes in Theoretical Computer Science, vol. 26, September 1999.

[9] P. Kakkar, and C.A. Gunter. Cryptographic reflections. Available at: http://www.cis.upenn.edu/ gunter/wip.html.

[10] NIST. FIPS PUB 140-1, Security Requirements for Cryptographic Modules. *National Institute of Standards and Technology*, January 1994.

[11] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 2nd Edition, 1996.

[12] S. Setia, S. Koussih, S. Jajodia, E. Harder. Kronos: a scalable group re-keying approach for secure multicast. *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.

[13] F. Stajano and R. Anderson. The resurrecting duckling: security issues for ad hoc networks. B. Christianson, B. Crispo, M. Roe eds., *Proceedings of Security Protocols: 7th International Workshop*, Lecture Notes in Computer Science n. 1796, pages 172–194, 1999.

[14] M. Waldvogel, G. Caronni, D. Sun. N. Weiler, B. Plattner, The VersaKey framework: versatile group key management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614-1631, September 1999.

[15] L. Zhou and Z. J. Haas. Securing ad hoc networks. in *IEEE Network Magazine*, 13(6), November/December 1999.