Securing Personal Network clusters

Assed Jehangir
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
The Netherlands
jehangira@cs.utwente.nl

Sonia M. Heemstra de Groot
Twente Institute for Mobile and
Wireless Communications, and
Delft University of Technology
The Netherlands
Sonia.Heemstra.de.Groot@ti-wmc.nl

***Abstract*— A Personal Network is a self-organizing, secure and private network of a user's devices notwithstanding their geographic location. It aims to utilize pervasive computing to provide users with new and improved services. In this paper we propose a model for securing Personal Network clusters. Clusters are ad-hoc networks of co-located personal devices. The ad-hoc makeup of clusters, coupled with the resource constrained nature of many constituent devices, makes enforcing security a challenging task.**

We propose a novel way of imprinting personal devices and pre-deploying tickets for future authentication. Our key management framework is based on a combination of existing models, albeit modified to match the constraints and assets of Personal Networks. The mechanisms for self organization are original since they need to be tailored to our proposed security framework. All models are designed to be suitable for resource constrained devices yet robust enough for self organization and secure communication.

Keywords- Personal Networks; security; constrained devices; symmetric cryptography; key management; self organization

I. INTRODUCTION

A **Personal Network** (PN) [1] [4], is a paradigm that utilizes pervasive computing to provide its users with new and improved services. A PN comprises a core consisting of a **PAN** (Personal Area Network) which can be transparently extended to include other devices belonging to the user, both in his vicinity and those at remote locations. PNs are composed of heterogeneous personal devices and are dynamic entities due to the mobile nature of their constituents.

Figure 1 outlines the network layer PN architecture envisioned in the QoS for PN@Home project [2]. Personal devices initially organize themselves in the form of **clusters**. Clusters are ad-hoc networks of personal devices that can communicate amongst each other without using any non-personal devices. One can think of a cluster as a secured network of co-located personal devices. Forming clusters facilitates in securing multi-hop communication and access to the common pool of resources. The otherwise isolated clusters are interconnected using secure dynamic tunnels, created between **gateway** devices, resulting in a network of personal devices that are geographically dispersed. In this work we focus on securing clusters in which constrained mobile devices communicate over unreliable channels, such as short range radio links. When communication occurs over reliable channels such as wired Ethernet, security can be enforced using simpler techniques such as firewalls at network entry points.

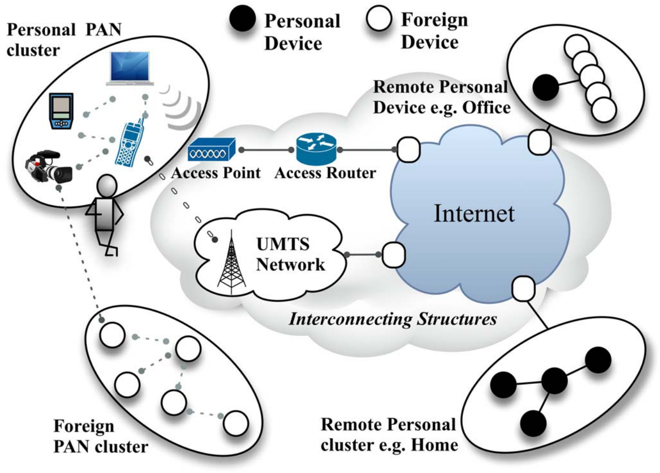

Figure 1. An instance of a Personal Network (PN)

We envision a PN to contain a wide range of device types, from resource constrained sensors that must last months on a pair of AA batteries, to powerful devices like laptops and PDAs that are recharged often. The main challenge in securing PNs is in using mechanisms that are suitable for resource constrained devices yet robust enough for secure communication and self organization. This work focuses on securing constrained devices since any mechanism suitable for them will also function with more powerful devices.

We begin in Section II by examining the limitations imposed on our design by constrained personal devices. Section III presents related works, some of which are based on similar requirements. In Section IV we give an overview of our architectural framework where we introduce components used in our security model. Section V presents insight on how personal devices self-organize. Section VI gives an overview of our security model and Section VII presents the associated mechanisms in detail. Section VIII analyses the possible attacks on our security model and Section IX concludes the paper with a summary of our contribution.

II. Constrained devices

Designing a security framework for constrained devices has generally been problematic especially since attackers have none of these constraints. Personal devices such as cameras, Bluetooth headsets, biomedical sensors, wrist watches, belt computers etc. may be constrained in their:

- *Energy*: Many personal devices are designed to be mobile. Their small batteries need to last months without recharging or replacement. We consider energy to be the scarcest resource in our system and our security mechanisms must be frugal in their power consumption.
- *Processing Power*: Processing abilities are extremely limited, in line with the constraints imposed on their power usage. A typical example of a constrained personal device is that of a Berkeley Mica Mote sensor. It features an 8-bit 4 MHz Atmel ATmega 128L processor with 128 Kbytes program store, and 4 Kbytes SRAM. The processor only supports a minimal instruction set, without support for multiplication or variable-length shifts or rotates.
- *Storage space*: A limited storage space means that only a limited number of cryptographic keys can be stored in the device. Also, any proposed security framework must be compact in its implementation.
- *User Interface*: Some devices may only present their users with a few buttons and possibly a LED for indicating their state e.g. sensors, wireless headsets, cameras etc. Our design cannot make any assumptions about minimum user interface requirements.
- *Cost*: Finally, the cost of a device will likely play a critical role in its success. Any cost overhead of security must be proportional to its benefit.

In view of these constraints, the remainder of the paper will present details on our proposed mechanisms for key management and self organization in PNs.

A. *Symmetric or Asymmetric cryptography*

Carman et al. acknowledge in [13] that although security solutions based on symmetric key cryptography are attractive due to their low energy and processing requirements, they also have limitations. For instance, two well known security protocols SNEP [6] and TESLA [10] provide secure broadcast authentication using symmetric cryptography by delaying the disclosure of authenticable secret keys. Emulating asymmetry thus requires that each device is time synchronized with the sender, performs key management functions and has sufficient buffer capacity. Key management (requiring broadcasts) uses energy. If the number of messages being authenticated in each time period is large, the receiver also needs to have sufficient buffer capacity to store them before the corresponding key is released and they can be authenticated.

Using public key cryptography instead would simplify protocols, since only the public key of the certification authority would have to be embedded into the devices and they could perform mutual authentication using signed certificates. Unfortunately, in the context of constrained devices the challenge with public key cryptography is to overcome the considerable computational complexity of asymmetric algorithms. Recent advances in efficient elliptic curve based algorithms such as EC-DH and EC-DSA show that it is *computationally* possible to use ECC based solutions in constrained devices like sensors [11] [12]. However they suffer from increased energy usage due to increased computational complexity and higher transmission costs as a result of transferring certificates and digital signatures. Ultimately, without a minimum capability criterion for constrained personal devices (and the cost of including public key cryptography for the single purpose of key establishment) we have concluded that our security mechanisms must only rely on lightweight symmetric cryptographic primitives. As a result we must also reject mechanisms based on the Diffie-Hellman protocol.

In summary, our aim is to develop a security framework for PNs based solely on symmetric cryptographic primitives. Where possible we will use or modify existing well established security models that utilize the inherent assets of PNs, principally that personal devices fall under one administrative domain and that the heterogeneity of device types allows us to delegate tasking requiring larger resources to more able devices.

III. Related Work

The work presented in this paper is a continuation of [3] where we presented general ideas of a security framework for PNs. Our aim was to use secure but lightweight mechanisms suitable for resource constrained devices and wireless communication. We proposed pair-wise keys for secure cluster formation and symmetric group keys for securing intra-cluster communication. Although the amount of overlapping material with this work is nominal, [3] does serve to put some of the issues we present here in a broader context. Note that since our proposals for self organization need to be tailored to our proposed architecture, the remainder of this section looks at related work only in the context of key management among constrained devices.

A large part of the research done in the field of key management among constrained devices has focused on sensor networks [6] [16] [17]. Although some devices in our network are similarly constrained, key management in PNs presents a different set of requirements due to the dynamic nature of the network (personal devices are mobile and so clusters are dynamic and often transient), the heterogeneity of personal devices (from wrist watches to laptop computers) and the difference in size and scope. When compared to the typically large and unattended sensor network, lost/compromised devices in a PN are easier to detect and subsequently blacklist. Nevertheless many of the conclusions drawn from research in securing sensor networks [5], particularly on the suitability of specific cryptographic algorithms [15] are directly relevant.

The constrained energy and communication capabilities of large sensor networks meant that protocols such as TLS and Kerberos (originally developed for wired networks) were

This work was supported by the Dutch Ministry of Economic Affairs under the Innovation Oriented Research Program (IOP GenCom, QoS for Personal Networks @ Home) and the Freeband PNP 2008 project.

deemed impractical. Most current approaches use some variation of pre-deploying symmetric keys. Amongst common proposals are those that use a global key shared by all nodes [18] [13], those in which every node shares a secret key with a base station [6], and those based on random key sharing [17].

Also relevant are models and protocols (based on symmetric cryptography) designed for securing collaborative multicast communities [10] [14] and other long term communities [7] [9]. These form the building blocks of our proposed security framework for PNs and we will look at each of them in more detail later in this paper.

To the best of our knowledge, the only other security architecture designed specifically for PNs is in the MAGNET project [4]. As such it is the only related work with which we can make a thorough comparison. Unlike our centralized approach to cluster formation, MAGNET devices wanting to join a cluster require a security association with at least one neighboring clustered device. However devices eventually need to establish separate security associations with all neighboring devices, something that results in increased communication overhead. Furthermore the use of pair-wise keys to secure both unicast and broadcast traffic increases processing overhead because of repeated decryption and re-encryption for packets traveling multiple hops. Our approach of using a group key reduces the overhead of key management and the centralized nature of our security architecture can be viewed as an asset as it provides the user with a higher degree of control.

IV. Architectural Framework

Our aim is to establish a line of defense between the PN and the outside world, so it can be a truly *personal* network. Since devices belonging to a cluster are (by its very definition) connected and long-term connectivity between various clusters of a PN is not feasible, our security mechanisms are based on a separation of trust at the cluster level.

A. *Security Manager*

A security manager is the cyber-representative of the user within the PN. Each new device needs to be associated with the security manager before it can become part of the PN. Consequently the security manager maintains a database of secret keys, one for each personal device. This allows it to perform certain management tasks; such as configuring access rights, blacklisting devices, etc. As the controlling entity for all personal devices, it is important that the user maintains a backup of its secret keys. In case the security manager breaks down, they can be restored to a new device. However if the security manager gets lost or compromised, these keys can be used to quickly and easily re-imprint all devices with a new security manager.

B. *Security Agents*

Security within the cluster is managed by a device with the role of a security agent. A cluster can have only one security agent at a time; however the device acting as a security agent may also have other roles related to e.g. service discovery. Along with other management tasks, the security agent is responsible for advertising its cluster, authenticating new cluster members and initiating group key updates. Furthermore, in order to reduce risk from cryptanalytic attack, the security agent updates the cluster key from time to time. Section VII.D will look at the re-keying mechanism in more detail.

Security agents periodically broadcast **cluster advertisements**, containing the **PN ID** and the security agent ID. The PN ID is a random value assigned to a PN when it is first created and is used by devices to distinguish advertisements belonging to their own PN from others. In order to reduce the possibility of ID collisions the PD ID should be calculated randomly over a sufficiently large space. Other cluster members may[1] rebroadcast non-duplicate advertisements, in effect reaching outside devices within the range of even peripheral cluster members. These advertisements serve a dual purpose; they update cluster members to the presence of their security agent and also allow outside devices to discover the cluster.

Using a multicast tree for broadcasting large number of messages to devices in an ad hoc network is more efficient than blind flooding [26], since it ensure that each device receives the message only once. However since constructing and maintaining such a delivery infrastructure has an overhead, further work is required to evaluate their suitability for disseminating cluster advertisements. This is especially true because unlike a typical multicast delivery tree where messages are only meant for tree members, cluster advertisements are also meant for outside devices that may wish to become members. Since we wish clusters to extend with devices that are within the range of even peripheral cluster members, our broadcast routing protocol should be optimized for peripheral members advertising. Consequently, at the moment we use blind flooding due to its simplicity, robustness and the fact that it gives us an idea of the worst case overhead for disseminating cluster advertisements.

If outside devices hearing cluster advertisements are un-clustered and belong to the same PN, they authenticate with the security agent of the cluster through the advertising device. To that end, cluster members enable IEEE 802.1X *based* port authentication. Since clustered devices are already part of another cluster, they merely forward such advertisements to their own security agent. The two security agents then authenticate each other, with the aim of merging the two clusters into one. **Cluster merging** is the process whereby two clusters of the same PN (within each other's transmission range) merge to form one cluster. Using IEEE 802.1X based port authentication allows two clusters to merge when their periphery overlaps and not only when the transmission range of the two security agents overlap. For more details refer to [3] [23].

Merging the two clusters will allow devices from one cluster to access resources in the other cluster directly, instead of going through respective gateways. However before clusters can be merged, they need to have a common cluster key and security agent. Therefore during authentication one of the security agents will step down after updating the cluster key,

[1] Depending on cluster policy and device's routing capabilities

cluster policy etc. of its cluster with that of the cluster being merged with.

In terms of security agent functionality, Security Agent Capable (**SAC**) devices have the capabilities to function as security agents whereas Security Agent In-capable (**SAI**) devices do not. SAI devices such as sensors are less sophisticated and typically only useful when networked with other smarter devices in a cluster.

C. Securing Cluster Communication

As PN constituents share a common trust, we use group authentication to improve system efficiency. Using a shared group key, henceforth called the **cluster key**, to verify cluster membership instead of as many keys as members in the cluster reduces the overhead associated with key management. Consequently clustered device can verify that messages were generated by trusted devices and not modified in transit by any un-trusted devices. As a result, any invalid packet injected by an adversary will be dropped.

Clustered devices append a MAC (message authentication code) calculated using the key $\mathbf{K_{mac}}$ to all intra-cluster traffic that they generate. K_{mac} is generated from the cluster key using a globally known one way hash function. This is because the existing cluster key is used to encrypt the next cluster key during key update, so we limit its further direct use in order to minimize exposure to attack. Appending a MAC increases the communication overhead so it should not be overly conservative. Although depending on the bandwidth of the links and the cluster key validity period, we assume a MAC size of 4 bytes [5] to be sufficient. This is also the value used in our simulations.

Optionally, devices may encrypt sensitive messages at the link layer by using an encryption key ($\mathbf{K_{encr}}$). Similarly to K_{mac}, the encryption key is also derived from the cluster key using a globally known one-way transformation. By default, cluster advertisements are not encrypted but only appended with a MAC (unless a cluster wishes to stay anonymous and undiscoverable).

Lastly, we would like to point out that although evaluating the cryptographic algorithms for securing communication amongst constrained devices is important, it is not a part of our research because a lot of research has already been done in this area in the context of (resource constrained) sensor networks [15] [5].

D. Bootstrapping Cluster Formation

When an SAC device powers up it starts functioning as security agents in a cluster that only contains itself. As a security agent it will periodically broadcast cluster advertisements. On the other hand, SAI devices unable to create their own clusters start out in the orphan state. Devices in the orphan state do not belong to any cluster so do not have a cluster key. Such devices only know their PN ID, which was configured during the imprinting process.

SAI devices attempt to authenticate with the first cluster of their own PN whose advertisement they receive. After a successful authentication they are able to join that cluster and take part in intra-cluster communication. Similarly, two clusters within each other's transmission range merge to form one cluster. As a result, co-located personal devices will systematically group themselves into a single cluster.

V. SELF ORGANIZATION

We have said that devices listen for cluster advertisements in order to discover clusters within their transmission range. Cluster members use such advertisements to monitor the state of their security agent. We define a **live cluster** as one that has a functioning security agent and can grow by adding new members. A **zombie cluster** has lost its security agent, however communication between existing cluster members is still possible until the cluster key expires.

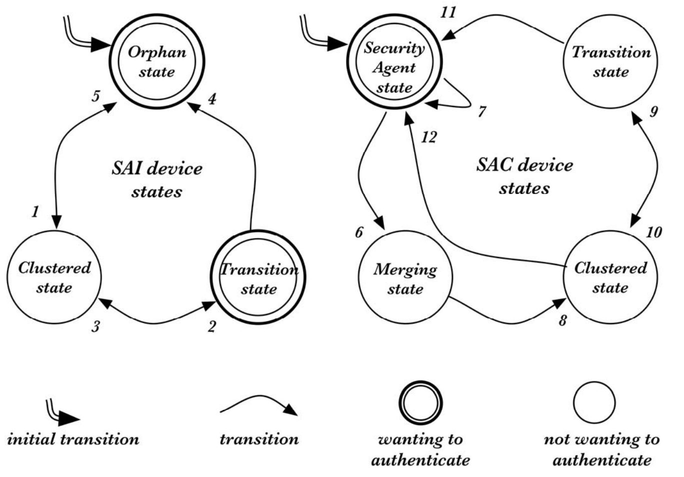

Figure 2. State transitions of PN devices

Figure 2 shows the state transitions of PN devices. We see that SAI devices start out in the orphan state, whereas SAC devices start out in the security agent state. After authenticating with an existing cluster, devices in the orphan state enter the clustered state `{1}` as members of the expanded cluster. When two clusters merge, one of the security agents will enter the merging state `{6}` where it will update the cluster key etc. of its own cluster to that of the cluster being merging with. The other security agent will then become the security agent of the two merged clusters `{7}`.

We define the redundancy factor **`'i'`** as the number of **sequential** cluster advertisements missed by a device before it believes it has lost connectivity with the security agent. Therefore if a clustered device misses `'i'` consecutive cluster advertisements, it enters the transition state `{2, 9}`. Without the security agent, the existing cluster key will eventually expire and devices can no longer communicate among themselves. SAI devices enter the orphan state `{4}` where they wait indefinitely to join another cluster while SAC devices form their own clusters `{11}`.

For SAI devices the duration of the transition state is limited by the time remaining till the existing cluster key expires. While this key is valid, SAI devices can continue taking part in intra-cluster communication. However if they hear an advertisement from another cluster belonging to their PN, they will attempt to join that cluster instead {3}. If they do not hear any cluster advertisements at all, they remain part of the existing zombie cluster till the current key timeouts and the cluster falls apart {4}.

SAC devices being less dependent can leave the transition state sooner and create their own clusters {11}, which other devices can then join. The transitions {5,12} show devices moving from the clustered state to the orphan and security agent states without going through the transition state. Such a device was in the transition state during the re-key or the hash disclosure phase, but later received one or more cluster advertisements so moved back to the clustered state before its cluster key expired. Finally, since devices only enter the transition state when they no longer hear cluster advertisements from their security agent they automatically stop advertising their cluster during this blackout period. In this way the system is quite self regulating.

VI. Overview of our Security Model

Reference [9] lists the traditional models used to offer security services to members of a long term ad-hoc community, such as a PN. Since no one model is suitable on its own, our security framework utilized three of them. In the **pre-shared common data model** each member of the community knows a shared secret. In our proposal, all personal devices belonging to a cluster use a symmetric group key, known as the cluster key, for securing intra-cluster communication. In the **pre-shared derived data model** each community member can be uniquely identified by a seed derived for it by another entity. We use tickets created by a trusted third party to uniquely authenticate personal devices to a cluster, before they are given the cluster key. Lastly, the **resurrecting duckling model** [7] ensures the security of a transient association between a master (mother) and a slave (duckling) device, in the absence of an online authentication server. This secure association it built when the master device **imprints** the slave by transferring a secret key using physical contact (to ensure the authenticity and confidentiality of information). Although an imprinted device is controlled by the mother, it is free to interact with other devices. A device in the imprinted state remains linked to its mother till the latter breaks the relation between them, at which point it enters the imprintable state again. Predictably, a device in the imprinted state cannot be re-imprinted. In the following segment we justify how these models fit in the context of PNs.

In general, use of the pre-shared common data model is often rejected (in spite of having the lowest storage cost and being very energy efficient) because compromise of a single device reveals the shared key. However, we believe that using a shared key for securing PN *clusters* is feasible because of:

- The transient nature of the cluster and its cluster key. Consequently the effects of key compromise only last as long as the compromised cluster is intact. Unlike proposals using the group key as a fixed globally shared key [18], the cluster key is time-limited and cluster-limited.
- The reduce risk of key compromise from cryptanalysis because of periodic cluster key updates.
- The reduced risk of key compromise from compromised devices because the loss or theft of a personal device is generally easier to detect when compared to large-scale unattended sensor networks.
- Alternatives to a shared cluster key would be pair-wise or sub-group keys, both of which increase the overhead of key management, storage and communication (because of repeated decryption and re-encryption for packets traveling multiple hops).

The use of the pre-shared derived data model is also often rejected because its online-verifiable mechanisms (like Kerberos) require an online entity issuing the ticket at the time of communication. The dynamic nature of PNs means that communication with such an entity cannot be guaranteed. On the other hand, the off-line verifiable mechanisms are traditionally based on using certificates signed by the public key of the central authority. We have already rejected using asymmetric cryptography due to the constrained nature of some personal devices. Our proposal is therefore to use an **offline verifiable mechanism based on symmetric key cryptography** i.e. something like an offline Kerberos. For most networks this would not be a feasible proposal because unlike certificates, a ticket is only valid for authenticating with a specific device. Meaning that in a network where a device may wish to authenticate with any other device, it will have to be loaded with as many tickets as there are devices in the network. Even though we do not envision a PN to contain more than a (few) hundred devices, clearly this is still not a feasible option for constrained devices. In Section VII.A.1 we look at why our architecture only requires constrained devices to load a small number of tickets to ensure secure connectivity. For clarification, a scheme based on tickets is quite different from pre-distributing pair-wise keys because the latter does not easily allow new devices to be added to a pre-existing network since existing devices will not have the new devices' keys.

We use the resurrecting duckling model and make one device (called the security manager) the mother to all other personal devices. Although personal devices typically have long term associations with their owners, this relationship on occasion needs to be severed e.g. when a device is sold or gets lost. In addition to this, the user centric concept of PNs suits having the security manager, as the cyber-representative of the user, in complete control of all personal devices. In [8] Stajano extends the original model by proposing the use of security policies to handle more than one-to-one relations. This allows the security manager to control the behavior of a device by uploading different policies to it, for example, specifying the list of devices allowed to access a certain service.

VII. Security mechanisms in detail

Being part of a PN allows devices seamless access to the common pool of PN resources. However before a new (i.e. imprintable) device can become part of a PN, a series of steps must be carried out to securely transfer the necessary credentials to access the PN. The remainder of this section looks at each of these steps in detail.

A. Imprinting

The first step in making a new device part of a user's PN is imprinting that device with the security manager of the PN. The major problem with symmetric key based schemes is to establish a secret key between two un-associated devices, without making use of asymmetric key cryptography such as Diffie-Hellman. Stajano and Anderson [7] propose a solution that uses physical contact to transfer the symmetric key in plain text. This is later formalized by Balfanz et al in [20] as a location-limited side channel that ensures the secrecy of communication. Location-limited side channels are separate from the main wireless link, and have the security properties of demonstrative identification and authenticity. This is by virtue of the medium over which the data travels, for example, directional channels such as infrared. Unfortunately all devices, especially the more constrained variety like Bluetooth headsets, do not generally come with suitable location-limited side channels over which the security manager can "push" the symmetric key in plain text.

Therefore we propose a new mechanism, in which the device to be imprinted pushes an initialization key to the security manager, which in turn uses it to securely transfer the long term key to the device being imprinted. To support our proposal we use as an example a sensor that comes from the factory pre-configured with an initialization key. The user would have access to this key in the form of a string of text, an RF ID or a bar code that comes in the box. During imprinting this initialization key is pushed to the security manager with user assistance, for example, the user takes a picture of the bar code with the built camera of his security manager. From this bar code the security manager can automatically extract the initialization key and the MAC address of the new device. Now, at any time in the future the security manager can authenticate itself to, and securely imprint the sensor over the insecure wireless channel. We do not use the actual pre-configured key of the sensor as the long term key with the security manager, because the user would like such keys to be secret even from the manufacturer of the device.

Such a mechanism, which does not require the user to position the security manager and the device to be imprinted according to the requirements of the location-limited side channel, is even more useful when imprinting a large amount of sensors. Imagine a scenario where a user buys a house, which has been pre-deployed with a large amount of sensors. He could be given a list of bar codes corresponding to the deployed sensors and only needs to scan the bar codes with his security manager. The remainder of the imprinting process, where the security manager needs to transfer the necessary credentials, will be done automatically over the insecure wireless link as the user walks around the house. The same benefit would apply if he were to buy a large quantity of sensors from the store. In Section VII.A.2 we give details on the protocol specification of our proposed imprinting mechanism.

Then again, the heterogeneity of devices in a PN means that other personal devices have multiple interfaces available for imprinting. We therefore generalize our example of the sensor to include more capable devices. If the device to be imprinted has an interface where a secret key can be pushed, then either the security manager pushes the long term key to the device or the device pushes an initialization key to the security manager. However if the device to be imprinted does not have a suitable interface to receive the long term key (our example of the sensor) then the only option is to push an initialization key to the security manager. Devices with suitable interfaces can generate the initialization keys dynamically; others will come with a factory configured initialization key. Dynamically generated initialization keys are discarded after the imprinting process completes. The static keys should be kept secure since they are used if the corresponding device ever needs to be imprinted again.

At the end of the imprinting phase we have a trust model shaped like a star, with the security manager at the center. However since personal devices also need to communicate with each other, this model as it stands, in not enough.

1) Tickets

Our selected approach for enabling mutual authentication between two previously un-associated personal devices is built on third party referral and uses as its basis the Needham-Schroeder protocol [21]. This protocol is based on symmetric cryptographic primitives and uses **tickets** to prove the identity of authenticating devices. The trusted third party responsible for issuing these tickets is the security manager. It is able to issue such tickets because it maintains a database of secret keys, one for each device belonging to the PN. Consistent with [22] we call such a paradigm a **Personal TGS** (i.e. a personal ticket granting server). In terms of characteristics, a Personal TGS is different from that proposed by Needham-Schroeder (Appendix 1) and extended in Kerberos.

Personal devices are pre-loaded with all necessary tickets during the imprinting phase and under normal circumstances will never contact the security manager for more tickets. Each ticket (valid indefinitely) is only used once, as the resulting key is considered long term and will be used for future authentications. In order to keep the storage overhead low, devices are only loaded with tickets needed to form clusters and the tunnels that interconnect different clusters. In other words, resource constrained SAI devices are only loaded with tickets needed to authenticate with SAC devices (for joining clusters). Since only a fraction of devices in a typical Personal Network will be SAC, this significantly reduces the storage overhead for constrained devices. This optimization is based on our assumption that SAI devices being less sophisticated (e.g. without suitable user interfaces) are not useful by themselves but only when networked with other smarter devices in a cluster. Once part of a cluster they can communicate securely with fellow cluster members using the cluster key and members of remote clusters through their cluster gateway.

SAC devices will of course need tickets for all other devices in the PN. This is because SAC devices, when functioning as security agents, should be able to authenticate all SAI devices joining their cluster and also other SAC devices during cluster merges. However we do not consider this a weakness because SAC devices like PDAs and cellular phones are not resource constrained and have sufficient storage capabilities. Finally, personal devices capable of functioning as gateways will need tickets corresponding to other PN gateways. This is because inter-cluster communication is secured by creating cryptographic tunnels between gateways.

Next we present the protocol specification for our proposed mechanism, divided into two phases. Phase 1 is pre-imprinting and requires user assistance. Phase 2 is the actual imprinting process during which all relevant information is transferred to the new device. Note that phase 2 can take place any time in the future after phase 1.

2) *Protocol Specification - Imprinting*

Device B is new SAI device being introduced into a Personal Network that contains a SAC device (A), and a security manager (M).

MAC_B	: MAC address of B
K_{init}	: Initialization key
ID_B	: Device ID of B
N_B	: Nonce generated by B
dec	: Decrement operation
hash	: One-way hash operation
C_B	: List of B's capabilities (I.e. SAC, SAI, GW)
P_B	: Policy to be used by B
K_{BM}	: Long term key between B and M
PN_{ID}	: Personal Network ID

Phase 1 (User assisted over a secure channel)

1. B -> M: MAC_B, K_{init}

Phase 2 (Insecure main wireless channel)

2. M->B: $\{ID_M, N_M\} K_{init}$, $hash(ID_M)$
3. B->M: $\{dec(N_M), N_B, C_B\} K_{init}$
4. M->B: $\{dec(N_B), ID_B, P_B, K_{BM}, PN_{ID}\} K_{init}$, $\{ID_B, K_{AB}\} K_{AM}$, $\{ID_A, K_{AB}\} K_{BM}$

The first step is performed by the user when he loads the security manager with the MAC address and initialization key of the new device. In step 2, the security manager sends its ID and a challenge, protected by the initialization key, to device B. In Section VIII we explain why sending a hash of already encrypted information protects device B from denial of service attacks. In step 3 device B responds to the challenge, and sends one of its own (for mutual authentication). It also sends a list of capabilities, based on which, in step 4 the security manager transfers relevant tickets. In the example given the security manager is only transferring one ticket, for device A. Besides tickets the security manager also imprints device B with its new ID, policy, long term pair-wise key and the PN ID.

B. *Authentication*

Earlier we said that access to the cluster, hence the cluster key, is only granted to devices after they successfully authenticate with the security agent. The very first authentication between a devices and a security agent utilizes the tickets transferred during the imprinting phase. A successful authentication will result in a security association based on the resultant long term key. Therefore once used, a ticket is destroyed since any future authentication will instead use the existing security association. Depending on the order that the two authenticating devices were added to the PN, either of them can be holding the relevant ticket. Next we present the protocol specification for both cases. Since cluster advertisements include the ID of the security agent, authenticating devices know which of the two mechanisms to use.

1) *Protocol Specification - Authentication*

Example 1: Device B begins authentication with device A after receiving its cluster advertisement i.e. the ticket is with the authenticating device.

1. B -> A: $\{ID_B, K_{AB}\} K_{AM}$, $\{N_B, hash(ID_B)\} K_{AB}$
2. A -> B: $\{dec(N_B), N_A\} K_{AB}$
3. B -> A: $\{dec(N_A)\} K_{AB}$
4. A -> B: $\{cluster\ key, \ldots\} K_{AB}$

Example 2: Later a new SAC device, C, is added to the Personal Network. Device B is un-clustered and begins authentication with device C after receiving its cluster advertisement i.e. the ticket is with the authenticator. Device B knows that it can authenticate with device C because the cluster advertisement has the same PN ID.

1. B -> C: ID_B
2. C -> B: $\{ID_C, K_{BC}\} K_{BM}$, $\{N_C\} K_{BC}$
3. B -> C: $\{dec(N_C), N_B\} K_{BC}$
4. C -> B: $\{dec(N_B)\} K_{AB}$, $\{cluster\ key, \ldots\} K_{AB}$

Depending on the final implementation the security agent may transfer a number of organizational parameters in step 4. These include but are not limited to the, cluster key, cluster key validity period, cluster policy and a commitment of the key chain of the security agent (Section D).

C. *Personal TGS*

For a more inclusive explanation of a Personal TGS, we compare it to the Needham Schroeder protocol and it's most well known and featured derivative, Kerberos. We assume that the reader is familiar with both of them. When comparing with the Needham Schroeder protocol, we do not use "N_A" in step 1 and 2 of Appendix 1. This is because the keys generated by the Needham Schroeder protocol are session keys that need to be periodically refreshed; in our case they are long term keys used for authentication. The nonce is meant to protect against replay attacks where an attacker replays step 2 with the old session key in order to increase the amount of cryptographic material available to him. We think it is appropriate to use tickets to generate long term keys because personal devices have a long term membership in the PN and the flexibility to authorize access to a service temporarily is not necessary. Also since the keys generated are long term, Dorothy Denning and Giovanni Sacco's recommendations [24] about time stamps in the Needham Schroeder protocol are not necessary.

In Appendix 1 we see that B is authenticating A, but not vice versa. Therefore we use an extra nonce when the ticket is first sent to the authenticator (step 3, Appendix 1), to ensure mutual authentication. When comparing with Kerberos which uses timestamps for authentication, our protocol uses tickets that are valid indefinitely and authenticates using a challenge response mechanism. This also has an advantage of not requiring PAN devices to be time synchronized with each other and the TGS.

The main disadvantage of our protocol is the difficulty of performing timely revocation in the absence of an online TGS. As such our protocol has something in common with public key cryptography and signed certificates and solutions proposed for them can be applied. If the security manager is not always online but is nevertheless online at frequent intervals, we can use routinely distributed revocation lists. In this approach the security manager distributes updated revocation lists at regular intervals to all personal devices. Due to the long term relationship of devices within a PN revocation will only be needed in cases where a personal device is lost, sold or compromised.

The proposed scheme based on establishing trust using tickets issued by the security manager is supported by all personal devices. Optionally, more powerful devices can also use asymmetric cryptography by asking the security manager to issue them with digital certificates. These certificates can be used for added security by protecting communication at higher layers. Similar to tickets, certificates should also be issued by the security manager during imprinting.

D. *Updating the cluster key*

Updating the cluster key (or **re-key**) is performed when the current cluster key is about to expire, when the user wants to expel a device from the cluster or when one cluster has to update its key to that of another cluster during cluster merging. Even though all cluster traffic is protected using the cluster key, we feel control messages such as re-key messages and cluster advertisements require an additional level of security. Otherwise misbehaving clustered devices or compromised cluster keys can be used to hijack the cluster. Consequently control messages broadcasted by the security agent are further protected using TESLA [10]. This way they cannot be forged by unauthorized devices. We employ TESLA for broadcast authentication because it is based on lightweight symmetric cryptography and is tolerant to packet loss.

In Section II.A we stated that TESLA requires each device to be time synchronized with the sender, perform key management functions and have sufficient buffer capacity. Since we are only securing control messages transmitted by the security agent we ensure loose time synchronization by having devices synchronize with the security agent during authentication. If there are issues of clock synchronization, we can utilize the periodic properties of cluster advertisements by using them as time synchronization beacons. Furthermore only securing control messages means that the number of messages being authenticated in each time period is very small and receivers do not need excessive buffer capacities. In order to reduce the energy overhead of cluster wide broadcasts we piggyback re-key messages, disclosed TESLA keys and other messages related to TESLA's key management on the periodically broadcasted cluster advertisements. Not only does this reduce the transmission overhead but it also reduces the number of times devices need to wake up from sleep state. Consequently we use a TESLA period that is equal to one cluster advertisement period. This also has the advantage that we only need to optimize for one parameter instead of two.

In order to piggyback messages on top of cluster advertisements we define certain extension headers [23]. The entire cluster advertisement including the extension headers is protected by a MAC generated using the cluster key and another (known as MAC-H) using TESLA. This is because MAC-H can only be verified in the next TESLA interval and the MAC derived from the cluster key ensures that attackers who do not know the cluster key cannot inject false cluster advertisements.

The unreliable nature of wireless transmission coupled with the fact that we employ broadcasts for efficiency reasons, means that we need a mechanism to ensure the reliable distribution of re-key messages. Figure 3 illustrates our proposed re-key mechanism where re-keying starts with the cluster advertisement (with the re-key extension header) of sequence number `x`. As per our earlier definition of the redundancy factor, any device that is part of the cluster should receive at least one of `i` sequential cluster advertisements. Consequently, in order to ensure the reliable delivery of re-key information we repeat it within `i` consecutive cluster advertisements. In [23] we use simulation results to justify a value for `i`. Once devices have verified that the new key is genuine, they are ready to switch over. However this should be done in a way that no ongoing communication is disrupted.

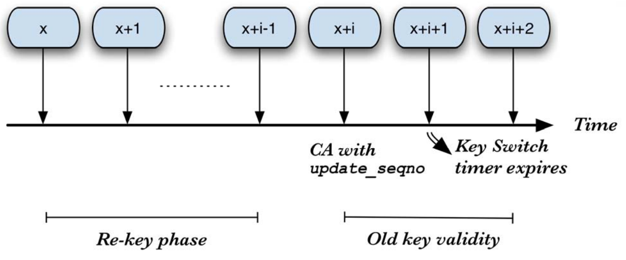

Figure 3. The re-key mechanism

Each re-key message contains the next cluster key (encrypted using the existing cluster key), the validity period of the new key and the **update sequence number**. The update sequence number field of the re-key message holds the sequence number of the cluster advertisement which signals the switch over to the new cluster key. This is the cluster advertisement enclosing the disclosed key corresponding to the last re-key message. Devices switch over to the verified cluster key immediately after forwarding the cluster advertisement with the update sequence number. However due to the unreliability of the medium there is a possibility that some devices will not receive this cluster advertisement. Consequently we have two backup mechanisms to switch over to the new cluster key. The first mechanism estimates the time in the future at which the cluster advertisement with the update sequence number *should* arrive. This is calculated based on the cluster advertisement period and the sequence number of the last verified re-key message. If devices exceed the estimated time plus one cluster advertisement period (to ensure that the message was indeed lost and not delayed), they update their cluster key automatically. This timer, called the **key switch timer** is automatically cancelled if the cluster key is updated using other mechanisms. Secondly, if a device waiting for the cluster advertisement with the update sequence number receives a message protected with the new cluster key it updates its cluster key as well. This is because it assumes that the sending device must have received the cluster advertisement with the update sequence number, otherwise it would not have shifted to the new key. This mechanism is not enabled when devices are re-keying as a result of cluster merging because other devices are already using the new cluster key. Lastly, since the system will not synchronize immediately there will be situations where some devices are using the old cluster key and others the new one. During the **old key validity phase** devices that have changed to the new key also accept packets that are signed with the old key. However any packets that they themselves transmit are signed with the new key. The old key valid phase lasts till the expected arrival of cluster advertisement with update sequence number + 2 cluster advertisement periods.

E. *Device eviction*

Devices are evicted from the cluster by excluding them from cluster key updates. Since foreign devices only have access to K_{mac} and not the cluster key from which it is derived, they are evicted by simply updating the cluster key. Personal devices are expected to have long term associations with the PN, so our proposed re-key mechanism does not have a way to evict personal devices besides blacklisting the excluded device and reforming the cluster. If evicting personal devices will be done frequently, we propose modifying our re-key mechanism to use GKMPAN [14]. GKMPAN is based on probabilistic key pre-deployment and also uses TESLA for secure broadcasting. However it allows us to evict selected personal devices during re-keying. Nevertheless, given that we only expect personal devices to be evicted rarely, we do not propose it as the default mechanism. This is due to increased overhead from storing the extra set of keys, increased transmission overhead and the fact that it *requires* a multicast delivery tree to distribute the new group key. GKMPAN was designed for multicast groups where such trees already exist.

VIII. ATTACKS

In general, attackers are classified into two main classes, passive and active. Passive attackers only eavesdrop on the communication thus they are a threat against the confidentiality and the anonymity of the communication. Confidentiality for PN traffic can be assured via optional link level encryption using K_{encr}. A compromised K_{encr} only compromises the confidentiality of the current session. On the other hand if the actual cluster key gets compromised, usually as a result of device compromise, then the confidentiality of all future cluster traffic is compromised. This will require revoking the compromised device and reforming the cluster. For clarification, we believe sensitive information such as credit card numbers are usually already protected at higher layers, e.g. using TLS.

Anonymity is assured if devices belonging to the user do not expose their identity or something that can be linked to their identity to un-trusted parties. Providing anonymity for the users of PNs requires:

- *Using encrypted cluster advertisements*: Broadcasting of clear text PN ID in cluster advertisement is no longer possible. The user can stay anonymous and undiscoverable by temporarily encrypting cluster advertisements. This has the automatic effect of stopping any authentication attempts in which devices transmit their IDs in clear text (Section VII.B.1 example 2).
- *Using dynamic MAC addresses*: Since authentication is based on device IDs, devices can change their MAC addresses if necessary. However this creates overhead in the system as it requires periodic updates of the ARP cache and checking for potential MAC address collision before each update.

Clearly, providing anonymity affects performance and possibly even usability. Consequently our proposed mechanisms do not provide anonymity as default.

An active attacker not only eavesdrops on the communication but also modifies and injects packets into the network. An active attack against the imprinting phase is not possible because the attacker does not have access to the initialization key (assuming phase 1 was performed securely). Similarly, since all link layer communication is protected with a MAC created using K_{mac}, attackers cannot become active unless they have compromised this key. For this reason K_{mac} is regularly updated and the system has forward security since a compromised K_{mac} does not compromise the next K_{mac}. Nevertheless if K_{mac} is compromised while it is still being used, the cluster becomes open to certain kinds of attacks, for example those against the ad hoc routing algorithm [25] and resource consumption attacks on TESLA [10].

Without compromised keys the attacker can only launch denial of service (DoS) attacks. We disregard physical layer attacks based on jamming because mechanisms such as spread spectrum have already been extensively studied to resist such attacks. DoS attacks against computational resources can be performed by fake authentication attempts as well as injecting messages into the cluster that will fail the integrity check. Our aim is to reduce harm from such DoS attacks by limiting their energy imprint. As transmission uses the lion's share of energy, we want to reduce wasteful transmissions triggered by attacks. Therefore in the imprinting process (step 2) we also transmit a hash of ID_M in clear text. This allows device B to confirm that the device M knows its initializing key before it transmits any replies (as ID_M is also sent encrypted with the initialization key). Similarly in Section VII.B.1 example 1, step 1, we also send a hash of ID_B encrypted using K_{AB}. We are unable to use this mechanism against DoS attacks against the security agent, Section VII.B.1 example 2, because the device initiating the message exchange does not hold the ticket. However the attacker will have to correctly guess the ID of a device that has a valid ticket at the security agent. In case the security agent has a time limit before a failed authentication attempt can be repeated, the attacker will need to guess multiple valid IDs to launch any meaningful attack. Finally, replay attacks are not possible because both imprinting and authentication between two given devices using tokens is only performed once.

IX. CONCLUSIONS

In this paper we have proposed a framework for secure clustering in Personal Networks. Our aim was to protect the Personal Network while maintaining its usability. Based on our constraints we have selected appropriate security models and presented details on our mechanism for key management and self organization. We proposed a novel way of imprinting devices and pre-deploying tickets which are later used for authentication. Consistent with our requirements all our proposed mechanisms are based on light symmetric cryptography and are applicable to a wide variety of devices. Furthermore our design minimizes the transmission overhead of adding security by reducing the number of messages that need to be exchanged. In [23] we have validated the transmission cost of our proposed mechanisms by simulations in NS-2 [19]. Although our work is done in the context of Personal Networks, our proposals are relevant to other models [27] [28] in which constrained personal devices are networked over unreliable wireless links.

REFERENCES

[1] I. G. Niemegeers and S. M. Heemstra de Groot, “Research issues in ad-hoc distributed personal networking”, Wireless Personal Communications, vol. 26, no. 2-3, pp. 149–167, August 2003.

[2] QoS for Personal Networks @ Home, http://qos4pn.irctr.tudelft.nl/

[3] A. Jehangir and S. M. Heemstra de Groot, “A Security Architecture for Personal Networks”, First International Workshop on Personalized Networks (PerNets), July 2006.

[4] IST MAGNET, http://www.ist-magnet.org/

[5] C. Karlof, N. Sastry and D. Wagner, “TinySec: A Link Layer Security Architecture for Wireless Sensor Networks”, The 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys), Nov 2004.

[6] A. Perrig, R. Szewczyk, V. Wen, D. Culler and J.D. Tygar. “SPINS: Security protocols for sensor networks”, 7th International Conference on Mobile Computing and Networking (MobiCom), July 2001.

[7] F. Stajano and R. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks", 3rd AT&T Software Symposium, Oct 1999.

[8] F. Stajano, “The resurrecting duckling—what next?”, 8th International Workshop Security Protocols, volume 2133 of Lecture Notes in Computer Science, pages 204–214. Springer-Verlag, 2001.

[9] N. Prigent, C. Bidan, J.-P. Andreaux and O. Heen, “Secure long term communities in ad hoc networks,” 1st ACM workshop on Security of Ad Hoc and Sensor Networks (SASN), Aug 2003.

[10] A. Perrig, R. Canetti, J.D. Tygar and D. Song, "The TESLA broadcast authentication protocol", RSA CryptoBytes, 5 (Summer), 2002.

[11] N. Gura, A. Patel, A. Wander, H. Eberle and S. C. Shantz, “Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs”, 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Aug 2004.

[12] G. Gaubatz, J. P. Kaps and B. Sunar, “Public key cryptography in sensor networks—revisited”, 1st European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS), Aug 2004.

[13] D. W. Carman, P. S. Kruus and B. J. Matt, “Constraints and approaches for distributed sensor network security”, Technical report, NAI Labs, Security Research Division, Glenwood, Maryland, USA.

[14] S. Zhu, S. Setia, S. Xu and S. Jajodia, “Gkmpan: An efficient group rekeying scheme for secure multicast in ad-hoc networks”, 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous), Aug 2004.

[15] Y. W. Law, J. Doumen and P. Hartel, “Survey and Benchmark of Block Ciphers for Wireless Sensor Networks”, 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), Oct 2004.

[16] S. Zhu, S. Setia and S. Jajodia, “LEAP: Efficient security mechanisms for large-scale distributed sensor networks,” 10th ACM Conference on Computer and Communications Security (CCS), Oct 2003.

[17] S.A. Camtepe and B. Yener, “Key Distribution Mechanisms for Wireless Sensor Networks: a Survey”, Technical Report TR-05-07, Rensselaer Polytechnic Institute.

[18] S. Basagni, K. Herrin, D. Bruschi and E. Rosti, “Secure pebblenets”, 2nd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), Oct 2001.

[19] Network Simulator 2 (NS-2), http://www.isi.edu/nsnam/ns

[20] D. Balfanz, D. K. Smetters, P. Stewart and H. C. Wong, “Talking to strangers: Authentication in ad-hoc wireless networks”, Network and Distributed Systems Security Symposium (NDSS), Feb 2002.

[21] R. Needham and M. Schroeder, “Using encryption for authentication in large networks of computers”, Communications of the ACM, 21(12):993–999, Dec 1978.

[22] C. Gehrmann, K. Nyberg and C.J. Mitchell, “The personal CA - PKI for a Personal Area Network”, IST Mobile & Wireless Communications Summit, pages 31-35, June 2002.

[23] A. Jehangir and S. M. Heemstra de Groot, “Evaluating Secure Cluster Formation in Personal Networks”, IEEE Wireless Communications and Networking Conference (WCNC), March 2007, in press.

[24] D. E. Denning and G. M. Sacco, “Timestamps in key distribution protocols”, Communications of the ACM, vol 24, issue 8, Aug 1981.

[25] Y. Hu, A. Perrig and D. Johnson, “Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks”, 8th ACM Conference on Mobile Computing and Networking (MobiCom), Sep 2002.

[26] H. Lim and C. Kim, “Multicast tree construction and flooding in wireless ad hoc networks”, 3rd ACM Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM), Aug 2000.

[27] D. Husemann, C. Narayanaswami and M. Nidd, “Personal Mobile Hub”, 8th Int. Symposium on Wearable Computers (ISWC), Nov 2004.

[28] IXI “Personal Mobile Gateway”, http://www.ixi.com

Appendix 1: Protocol specification Needham Schroeder Symmetric Key

Alice (A) authenticates herself to Bob (B) using a mutually trusted server M.

N_A, N_B	:	Nonce
K_{AM}, K_{BM}, K_{AB}	:	Session key
dec	:	Decrement operation

1. A -> M: A, B, N_A
2. M -> A: $\{N_A, B, K_{AB}, \{K_{AB}, A\}K_{BM}\}K_{AM}$
3. A -> B: $\{K_{AB}, A\}K_{BM}$
4. B -> A: $\{N_B\}K_{AB}$
5. A -> B: $\{dec(N_B)\}K_{AB}$