Design of a Secure Distributed Service Directory for Wireless Sensornetworks

Hans-Joachim Hof, Erik-Oliver Blaß, Thomas Fuhrmann, and Martina Zitterbart

Institut für Telematik
Universität Karlsruhe, Germany
`{hof,blass,fuhrmann,zit}@tm.uka.de`

Abstract. Sensor networks consist of a potentially huge number of very small and resource limited self-organizing devices. This paper presents the design of a general distributed service directory architecture for sensor networks which especially focuses on the security issues in sensor networks. It ensures secure construction and maintenance of the underlying storage structure, a Content Addressable Network. It also considers integrity of the distributed service directory and secures communication between service provider and inquirer using self-certifying path names. Key area of application of this architecture are gradually extendable sensor networks where sensors and actuators jointly perform various user defined tasks, e.g., in the field of an office environment.

1 Introduction

As computer miniaturisation continues and small devices become cheap, more and more of these little helpers are embedded into the user's environments to offer pervasive services. Sensor networks are special occurrences of networks build by such small devices. They are about to change the paradigm of sensors being passive devices that are connected to a more or less centralized computing engine that processes the so collected data. With the new paradigm of sensor networks, measurement and control tasks are performed inside a *sensor network* which is formed by a vast number of sensor nodes. Those nodes have low-power, are self -organising and have little computation capabilities. Such devices are called "peanut CPUs" in [9]. In the long run, sensor networks can be expected to create synergy out of the huge number and high density of devices.

In order to fully benefit from this new paradigm, sensor networks need to employ several new mechanisms that account for the special properties of their nodes: Low energy consumption, little CPU power, limited memory, regular node failures, and maintenance-free operation in heterogeneous and maybe even hostile environments. Given these limited abilities and other characteristics, it stands to reason that traditional security methods do not work satisfyingly in the context of sensor networks. Especially asymmetric cryptography is computationally too complex for these nodes, even with elliptic curve algorithms [3]. Therefore, security architectures for sensor networks benefit of focusing on symmetric cryptography which can be used efficiently even on peanut CPUs.

This paper describes the construction of a secure distributed service directory which can be used as a general lookup service for sensor networks. The service directory is based on an overlay network, the Secure Content Addressable Network (S-CAN). It draws from the idea of a Content Addressable Network [7] that is extended with security features that specifically take into account the requirements of sensor networks. A typical scenario where this architecture can be advantageously deployed is an intelligent office environment with dozens or even hundreds of sensors and actuators that act jointly within a sensor network to accomplish various tasks. In order to find its respective peers for a given task, a sensor needs to discover nodes that offer the required sensor data or are able to perform specific tasks. Typically such lookups are rare as compared to the overall operation time of the network. They occur only upon addition or removal of nodes or the creation of new tasks by the user. Both can be assumed to occur only seldom.

The key idea behind the use of a CAN is to create a simple distributed system that easily scales with its growth, is considerably failsafe, and does not require any centralised component during normal operation. Hence, a user can sequentially add nodes to the network without bothering to provide enough directory capacity. Moreover, by having such a flexible lookup service, nodes can immediately benefit from other nodes already present in the network without the need for any manual configuration.

This paper is structured as follows. Chapter 2 states some goals for a distributed service directory. Chapter 3 describes Content Addressable Networks, the basis for the design of a distributed service directory. Chapter 4 presents an architecture of a sensor network platform. Chapter 5 describes the design of the proposed distributed service directory. Chapter 6 characterises a hardware prototype for further work. Chapter 7 gives a summary of this paper and an outlook on further work.

2 Goals for a Distributed Service Directory

The usual desired task of a service directory is insertion of service names and service descriptions as well as lookup of these names and descriptions. A distributed service directory spreads the load of insert and query operations on a number of nodes. A distributed solution avoids a single point of failure compared to a central implementation. Security for a distributed service directory implies:

- Secure Insertion of data
- Integrity of stored data
- Correctness of lookup answers
- Availability
- Robustness

Securing our distributed service directory starts with securing the CAN which is the basis of our directory design. See Chapter 3 for a brief description of CAN. CAN is used because it has a solid structure with non-overlapping zones each owned by exactly one node. Every zone has specific neighbours. If we could ensure that communication between neighbors in the CAN space is secure all the time, i.e. the CAN neighbors are authenticated and use encryption for communication, we can secure the structure of the CAN space. CAN uses periodic update messages to maintain neighbours states. These messages are critical for the structure of CAN and need to be secured. The best time to build trust is the joining of a device thus new devices have to be added in a secure way. In general, a secure join operation of a new device implies three steps:

1. A new device gets authenticated. Otherwise, all security is obsolete, as you can not build up trust when you do not know with whom you interact.

2. A new device joins CAN space at a random point and could not determine this point a priori. A fraudulent device *A* should not be able to absorb a specified part of the distributed hash table realized by CAN during joining by making a proper device *B* to split its zone and hand over *A* half of the zone.

3. No attacker should be able to claim to be owner of a CAN zone it does not own. If we can assure this, takeover of one or more devices has only a local effect and ensures at least functionality of those parts of the hash table not affected. It also protects joining nodes against being drawn into a fake CAN by a fraudulent node which claims to own the whole CAN space.

3 Content Addressable Network

The Secure Content Addressable Network (S-CAN) is an integral part of the architecture presented in chapter 4. S-CAN basically extends the original CAN (Content Addressable Network) [7] by security considerations. This chapter gives a short overview of CAN.

CAN utilizes a d-dimensional Cartesian coordinate space on a d-torus. We will call this virtual space CAN-space. The coordinate space is completely logical and has no relation to any physical coordinate system. CAN forms an overlay network which lies above a network layer and utilizes the communication abilities offered by it. At any time the entire CAN-space is divided into zones which are administrated by the nodes participation in the CAN. Every node "owns" a distinct zone within the overall space. The virtual coordinate space is used to store *(key,value)* pairs as follows: *key* is mapped on a point *P* in CAN-space using a hash function. The *(key,value)* pair is then stored on the node which owns the zone within which *P* lies. To retrieve an entry corresponding to a key *K*, any node can apply the same deterministic hash function on *K* to map *K* on a point *P* and then retrieve the corresponding value from the zone in which *P* lies. The request is routed through the CAN-space until it reaches the node which owns the zone including *P*. Routing is done by greedily forwarding a message to the neighbor which coordinates are closest to the destination. Each node keeps a list of neighbors that abut their own zone. This list is used for forwarding and acts as a routing table. Periodic update messages between neighbours inform about the neighbor's state and help to recognize failed devices and abandoned zones. Please note that many different paths exist between two points in the CAN-space. This means, that even if one or more of a node's neighbors crash, a node would automatically route along the next best available path. If a node loses all its neighbors, it can do an expanding ring search to get connected to the CAN again. CAN describes some repair mechanisms for restoring a consistent state. Those are of no interest for this paper and will get careful attention in further work.

As mentioned earlier CAN-space is partitioned among a number of nodes. If a new node decides to participate in CAN an existing zone is split into half and the joining device gets one of the two resulting zones. A uniform distribution of join points is eligible to maintain zones of equal size which is important because effective routing depends on a similar zone size of all zones. CAN offers some optimizations of routing and robustness. Routing can be improved using multiple CAN-spaces (called multiple realities) with different hash functions on each node. Reaching a point in CAN then translates to reaching this point in any reality. Robustness can be improved by zone overloading. This means that multiple nodes own one zone and all owners are permitted to issue answers for requests on this zone. If one owner fails, there are still other zone owners which may answer requests.

4 Architecture of Sensor Network Platform

The distributed service directory described in this paper is embedded in a general architecture for sensor networks which aims to provide a flexible, scalable and secure platform for sensor-based services. An overview of the architecture is depicted in figure 1. A detailed description of the modules of the architecture follows:

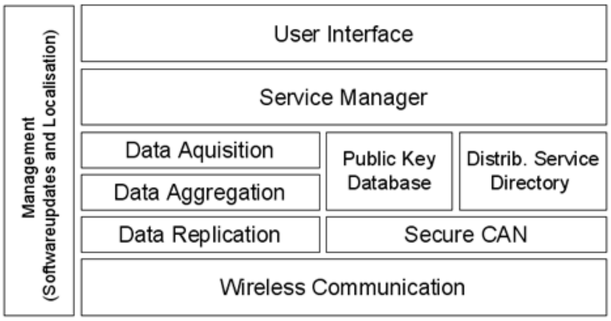

Fig. 1. Architecture of sensor network platform

4.1 Distributed Service Directory Module

The Service Directory module realises a distributed and robust service directory. There are two ways to implement a distributed service directory which are interesting for sensor networks as they take into consideration the properties of sensor networks: cluster-based or fully decentralised. We chose to implement the latter using a distributed hash-table. This table is realised by the Secure Content Addressable Network Module. This reduces the administrative overhead in contrast to a cluster-based solution. The service directory is used to store attributes about services which are available in the sensor network. Attributes include, but are not limited to, address of service provider, address of data replication point, physical location of service, validity of service, quality category, needed input to provide service, output format etc.

This paper is about the design of the service directory and S-CAN (See 4.2) on which it is based.

4.2 Secure Content Addressable Network Module

The S-CAN module realises a secure distributed hash table. A Content Addressable Network (CAN) is used as base of the secure hash table. S-CAN extends CAN by security features. CAN is the first choice, because it has a simple solid yet verifiable structure. Other overlay networks which realise a distributed hash table (like Pastry [8], Tapestry [12] or Chord [11]) miss this solid structure and are therefore not that easy to secure. Another interesting property of CAN is the low memory usage to hold neighbour information in a CAN with well distributed zones. See chapter 3 for more information on CAN. We focus on a secure construction of the CAN and on secure maintenance of its structure. S-CAN is a central service of the proposed sensor network platform and serves as the basis of the Service Directory Module. Its robustness is vital for the overall architecture.

It is not necessary that all devices of the sensor network participate in the S-CAN. Some devices could offer to issue queries to the S-CAN for devices which can not participate in the S-CAN. This is a step towards a cluster based solutions, but gives us more flexibility considering energy constraints on devices, especially in heterogeneous sensor networks. Another way to save energy is to rotate the job of S-CAN member and query server among a group of trusted devices. This involves some extra overhead but may still save energy. These considerations are out of scope of this paper and will be included in further work.

4.3 Service Manager Module

The Service Manager Module pairs actuators and sensors to execute a user task and supervises service execution. Sensors are nodes generating data. Actuators offer services other than data gathering. They usually control external devices. The pairing of actuators and sensors could be temporary or permanent, access could be simultaneous, competitive or exclusive. The Service Manager Module uses the Service Directory Module to bring actuators and sensors together. As services may require results of other services as input, the Service Manager Module may need to do cascading lookups in the service directory. Please note that the Service Manager Module has only to do lookups at the time of pairing. Later on, the paired actuators and sensors do not need any more service lookups as they know each other. The Service Manager Module also interacts with the User Interface Module and offers a simple language which can be used by the user to issue commands for actuators and sensors. User Interface Module and Service Manager Module are responsible for service provision.

4.4 Management Module

The Management Module realises some vital functions for sensor networks. This module has access to all layers as functions and informations provided by it are needed on all layers. The Management Module includes software updates and localisation information.

Software updates are vital for unattended sensor networks. They allow error correction, adaptation to unforeseen environmental conditions and recalibration. Software updates also guarantee a long lifetime of a sensor network as it is possible to redesign applications to latest needs not known when the network was deployed. Finally, software updates makes a sensor network reusable when its original aim has been achieved. It stands to reason that security is a very important matter with software updates. The goal is to ensure integrity and authenticity of updates. Another goal is to get back to a consistent state of the network as fast as possible, and, under any circumstances, must the sensors be able to recover from a failed software update. Software updates should even be possible in heterogeneous networks.

Localisation information is needed to build content aware applications. Localisation information can be acquired on different layers, starting on MAC layer with delay measurement and can go up to application layer where GPS data provides the global position on earth.

4.5 Data Replication Module

The Data Replication module is responsible for replication of sensor data. The goal is to bring frequently used data pieces as close as possibly to their inquirers to avoid large pieces of data to be multiply transported over great distances. The challenge is a timely delivery of the data and to keep several data replication points consistent. Security considerations on the integrity of replicated data are another open issue. The Data Replication Module firmly interacts with the Data Aggregation Module to reduce the overall energy usage on transmission in the whole sensor network.

4.6 Data Aggregation Module

The Data Aggregation Module is responsible for aggregation of sensor data. Transmitting messages is a very energy consuming operation in sensor networks. A considerable percentage of energy is spent on relaying messages in the sensor network thus any computation operation which reduces the data set is worth the effort as less data may be transmitted in a shorter time and therefore needs less energy. The idea is to reduce data by aggregation of data from various sensors before transmission to either get “the big picture” (which could be expressed by less data) or to eliminate redundant data. Sensor Data Aggregation is application centric as aggregation is only possible with a certain knowledge of data semantics. The Data Aggregation Module is also responsible for reducing accuracy of data. Less exactness usually means less data.

4.7 Data Acquisition Module

All data of the sensor network is acquired in the Data Acquisition Module. Data acquisition is initiated by the Service Manager Module. Data Acquisition Module also provides a part of the information needed by the Data Aggregation Module to reduce the amount of data. This information includes sensor accuracy, kind of data etc.

4.8 User Interface Module

The User Interface module enables the user to communicate with the sensor network. It gives her the possibility to issue commands to the sensor network and it provides a nice way to get results back from the sensor network. The User Interface is accountable for execution of commands and returns (aggregated) overall results of quests given to the sensor network. It represents a gateway between a the user's network (like the Internet) and an off-site sensor network.

4.9 Public Key Database Module

The Public Key Database Module realises a robust distributed public key database based on the Secure Content Addressable Network. Public keys may be necessary for the Service Manager Module to get access to confidential services etc. However, public key operations are very energy- and time-consuming so they should be kept to a minimum. The Public Key Database Module mainly uses functions of the Secure Content Addressable Network and can therefore be easily implemented.

4.10 Wireless Communication Module

The proposed architecture requires some underlying wireless communication ability. Our testbed (see chapter 6) uses Bluetooth as communication layer. However, the architecture itself is independent of the actual used communication technology and can be applied on most of the recently available sensor network platforms.

5 Design of a Distributed Service Directory

5.1 Assumptions

The design of the proposed distributed service directory is based on the following assumptions:

– Attackers are able to take over one or more nodes because nodes of our sensor network are unattended and spread in public or semi-public places. This assumption is called Big Stick principle in [9]. It implies to use tamper prove hardware. However tamper proveness and tamper resistance are way out of scope of this paper. Refer to [9] for a discussion of tamper proveness and resistance. A takeover of multiple nodes is considered possible for our proposed design of a secure distributed service directory.

- Collaboration of overtaken nodes is considered possible
- Takeover of a node enables an attacker to respond on data request with fake data for the zone the overtaken node owns. Furthermore, any data which is routed through the overtaken node can be faked by the attacker.
- The service directory is used only casually and insertion of data is rare. Most communication in the sensor network is not done in CAN space but on network layer. Therefor, it is acceptable for our service directory to use overlay routing (which is not as efficient as routing on network layer) as long as it does not paralyze our sensor network for too long.
- The hardware is severely constraint, meaning that computational power, energy reserves and memory are low. This brings us in a situation where avoidance of communication, only casual use of symmetric and almost no use of asymmetric cryptography is a must.

5.2 Principles

One of the main ideas of the Secure Content Addressable Network is the use of a so called Master Device (MD). A MD is a hardware gimmick (like a ring or a key fob [15]) which is needed to bring new devices into the S-CAN. This typically involves a human interaction. It stands to reason that a part of this interaction should be authentication as this difficult action for computers is much easier for a human ("This is the device I took out of that fancy box I bought at Radio Shack"). As stated earlier it is not necessary for all devices to participate in the S-CAN. Therefor it is possible that a device joins the sensor network even if the Master Device is not available. We want the MD to be stateless so it is no Single-Point-of-Failure as the user can always keep a device constructed the same way in reserve. A stateless MD also potentiates the simultaneous use of multiple MDs. We further assume that the MD is able to perform "costly" operations like signature checking. As the MD is possessed by the user (or users) and as it is only used from time to time to bring new devices into the CAN, energy preservation is not a concern on the Master Device. E.g. it could be rechargeable. The MD is not part of our sensor network meaning it is not used as a online server of any kind. However for simplification, it has the ability to communication with the sensor network during the join of a new device. Please note that this is just for simplification and that the MD could otherwise use the communication abilities of the trusted joining device.

Another important idea is the use of zone certificates to prove the ownership of a CAN zone. Certificates are stored on every CAN node, but only the MD checks certificates when a new device is joining the CAN. This ensures, that no node of the CAN can claim possession of a CAN zone larger than the one it got assigned at the time of joining. If we combine this with the strict order for all good-natured nodes to immediately remove old certificates from their memory when they got a new one during the joining of a device, this means that a mischievous node can only claim possession of a zone of the size it had, when the attacker took over the node. Therefore it is not possible for the attacker to draw other joining nodes into a fake CAN (see protocol description in chapter 5.3 for details). This limits the impact of hostile node takeovers.

We do not want to use asymmetric cryptography, we want our MD to be stateless but we also want encrypted communication with distinguished nodes of the CAN. Our protocol is designed in a way that every communication during a join involves the Master Device. So it is not necessary for two arbitrary nodes to have a common key with each other. Our protocol presents a simple but efficient solution which only uses symmetric encryption between the MD and CAN nodes and enables the MD to calculate the needed keys on-the-fly so we do not need to store any keys on the MD (thus, it is stateless). When joining, a device gets a random join point which presents the point where the device will join into the CAN space. This point is encrypted with the private key of the MD and used as a symmetric key between the joining device and the MD. The joining device stores the key it has in common with the MD. The MD has not to keep track of all the keys it has in common with any CAN node, as it can derive the key from a join point by simply encrypting it with its private key. The join point of a device must always lie in the CAN zone the device owns. It is included in the node's zone certificate.

A Location Limited Side Channel [1] is used for the communication between the MD and the joining device. Location Limited Side Channels can be used as a secure channel as described in the paper mentioned above. Infrared or sound are examples for Location Limited Side Channels. For our prove of concept, we use simple physical contact which is another Location Limited Side Channel. This is not convenient for the user as it may take some time for the device to join the CAN and the user does definitely not want to keep physical contact between i.e. the ring on her hand and the new toy for minutes. To circumvent this, further design will exchange a short cryptographic information by physical contact or infrared and all other communication is done in the main communication channel, secured by the exchanged cryptographic information. Physical contact is preferred, because it is an intuitive way for the user to authenticate a device as stated earlier.

5.3 Protocol Description

5.3.1 Secure Join

As stated earlier, every join of a device starts with physical contact between the joining device and the Master Device. This activates any further action. The description of the proposed protocol uses the following syntax:

Channel | A -> B: Message

Channel denotes the channel a message is send in. In the present case, this can be the channel established by physical contact (*PHY*), the main communication channel like Bluetooth Scatternets in our case (*MC*) or the CAN overlay (*CAN*). *A* denotes the sender and *B* denotes the receiver of *Message*. In the case under consideration, sender and receiver can be Master Device (*MD*), joining device (*JD*), owner of the zone *JD* wants to join into (*ZO*) and neighbours of *ZO* (N_i).

Our Protocol consists of ten steps:

1.) PHY | MD -> JD: JP, E_{SK}(JP, "temp")

First, we send *JD* the join point (*JP*) at which it will join the CAN over the Location Limited Side Channel (physical contact in this case). This point is selected randomly by the Master Device. Random selection is very important as we want to achieve a equal distribution of nodes in CAN to ensure an equal zone size so that routing in the CAN is ideal. The Master Device also sends the join point encrypted with its super key (SK), which is the private key of the Master Device. The encrypted join point is used as a symmetric key by the node and will be denoted with *k* in the further description of our protocol. The string "temp" should provide an indication of the temporary character of the key. As it is important to always ensure that the join point of a device lies in its owned zone, there are cases where it is necessary for a joining device to get another join point. However it is not necessary to derive *k* from the join point. In fact, it could be any given ID as the ID is included in a certificate, issued in step 5.

2.) CAN | MD -> ZO: "who is responsible for JP", Address

Next, the Master Device sends a message into the CAN with address *JP* to find out, who owns the zone *JP* lies in. The message also includes the network layer address of the Master Device to enable the receiver to communicate with the Master Device outside the CAN. As we stated earlier, it is not necessary for the Master Device to have the ability to communication with the CAN because it could otherwise use the communication abilities of the trusted joining device. For simplification however, we assume the Master Device to be able to communication with the CAN in our further protocol description.

3.) MC | ZO -> MD: JP_{ZO},E_k(certificate, time)

The zone owner answers the Master Device with its join point and the certificate of its zone, encrypted with the common key between the Master Device and the zone owner. The encrypted message also contains a timestamp to prevent reply attacks. The Master Device derives *k* using its private key as described earlier. It checks the zone owners certificate, compares the join point, included in the certificate, with JP_{ZO} and tests if the join point of the joining device really lies in the owned zone.

4.) PHY | MD -> JD: JP, E_{SK}(JP, "perm")

The Master Device evaluates if the joining node needs a new join point taking in consideration the join point of the zone owner and the future split line in CAN. In all cases, the joining device gets a new common symmetric key with the Master Device, which is constructed in the same way described earlier. This time, key and join point are marked as permanent. For any further interaction of the joining node with the Master Device, only the permanent key and therefore only the permanent join point is valid.

5.) PHY | MD -> JD:$key_{JD,ZO}$, E_{ZO}*(“JP,JD”,* $key_{JD,ZO}$, $certificate_{ZO}$,E_{JD}($certificate_{JD}$))

In the next step, the Master Device hands out a “ticket” to *JD* which enables it to get its new zone from *ZO*. The ticket is encrypted with *ZO*’s symmetric key and includes address (*JD*) and join point of *JD*, a symmetric key for communication between *JD* and *ZO*, a certificate for *ZO*’s zone after the split operation ($certificate_{ZO}$) and a certificate for *JD*’s new zone ($certificate_{JD}$). The certificate for *JD* is encrypted with *JD*’s symmetric key with the Master Device. If needed, step 5 is the right moment to hand *JD* a certificate for its private key and the public key of the Master Device. However, the proposed design does not (yet) make any use of asymmetric cryptography, so this step is obsolete at the moment.

6.) MC | JD -> ZO: E_{ZO}*(“JP,JD”,* $key_{JD,ZO}$, $certificate_{ZO}$,E_{JD}($certificate_{JD}$))

JD hands the ticket to *ZO*. *ZO* starts the zone split.

7.) MC | ZO -> JD: E_{key}*(data of hash table)*

ZO starts the zone split with transmission of data stored in the part of the distributed hash table, which formerly belonged to *ZO*’s zone but now belongs to *JD*. Transfer of data is encrypted with $key_{JD,ZO}$ which *JD* and *ZO* got handed out by the Master Device. This ensures, that no intermediate node is able to alter the data stored in *ZO*’s former hash table. Encryption is important to ensure integrity of the distributed hash table. All received data is acknowledged by *JD*. Those messages are not included in our protocol discussion for simplification.

8.) MC | ZO -> JD: E_{key}(E_{JD}($certificate_{JD}$))

When data transfer successfully ended, *ZO* hands out *JD*’s zone certificate. It is important that the certificate is handed over at the end of the data transfer. If the certificate would be delivered to the joining device before the successful data transmission, the joining device could simply drop any data but can prove that it is zone owner. With the certificate being handed over at the end of the data transfer, *ZO* could test some values it formerly stored. This does not prevent the joining device to drop all data anyway, but it makes cheating more difficult. From the moment of certificate handover forth, *ZO* is no longer responsible for the half of its former zone. *ZO* immediately and thoroughly destroys its old zone certificate. Extra care should be paid on this as an attacker who would be able to recover the old certificate would be capable of boarding *JD*’s zone.

9.) MC | ZO -> JD: E_{key}*(TempKeys, Neighbours)*
MC | ZO -> N_i *:* E_{Ni}*(“New Neighbour”, JD, TempKey)*

As the zone split is now official in effect, *ZO* notifies all affected neighbours about the split and assigns them a temporary key for secure communication with *JD*. This messages are encrypted with the symmetric key *ZO* has in common with each of its neighbours. *ZO* sends *JD* an encrypted message containing all *JD*´s new neighbours and the temporary key which *ZO* assigned them.

10.) MC | JD -> N_i: Key construction

As the last step of our protocol, *JD* establishes a symmetric key with each of its new neighbours. This key is used for important communication between the neighbours, like periodic update messages etc.

At the end of step 10, the CAN has reached a consistent state again. *JD* joined securely the CAN and has established secrets with all of his neighbours. Every former neighbour of *ZO* has updated its routing information and *ZO* handed over its part of the distributed hash table to *JD*. This data transfer is encrypted and acknowledged. Therefore, integrity of the distributed hash table is ensured. Figure 2 summarises the protocol.

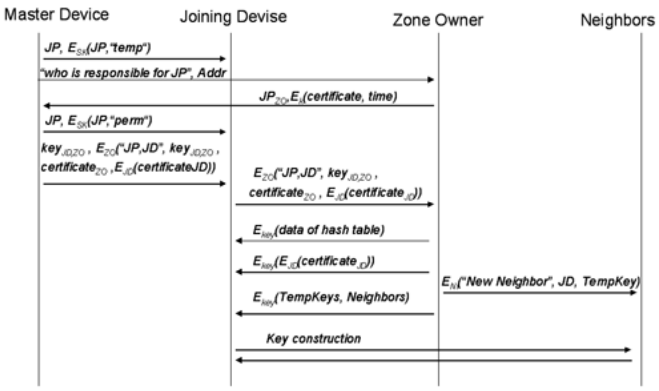
Master Device
Joining Devise
Zone Owner
Neighbors
JP, E_SK(JP,"temp")
"who is responsible for JP", Addr
JP_ZO, E_N(certificate, time)
JP, E_SK(JP,"perm")
key_JD,ZO , E_ZO("JP,JD", key_JD,ZO , certificate_ZO , E_JD(certificateJD))
E_ZO("JP,JD", key_JD,ZO , certificate_ZO , E_JD(certificate_JD))
E_key(data of hash table)
E_key(E_JD(certificate_JD))
E_N("New Neighbor", JD, TempKey)
E_key(TempKeys, Neighbors)
Key construction

Fig. 2. Protocol overview

There are some cases where a device does not hold a valid certificate for its owned zone but is a valid owner. E.g. a node who gratefully took over an abandoned zone of a failed device has no valid certificate. Node failure can have different causes: battery outage, damage of hardware, signal jamming and environmental influences. Also, certificates of zones which are larger than a given boundary are time constraint to avoid that a hostile node takes over a large zone in the beginning of the sensor networks lifetime and is able to infiltrate a major part of the sensor network. However, the absence of a valid certificate is only grave in the case a device wants to join that zone. In this situation, the Master Device is to be considered online and can be used to issue the missing certificate. But the Master Device needs to be sure that the claiming zone owner is really the in possession of that CAN zone. To be sure, the Master Device asks the proposed zone owner to get guaranties for its ownership from its neighbours. Those guaranties include the zone limits of the neighbours, their certificates and a timestamp. The guaranties are encrypted with the neighbours' keys they have in common with the Master Device and can therefore be used to validate the zone owners claim on the zone. Protection of reply attacks is done by the timestamp. Figure 3 illustrates, how the Master Device verifies the zone claim of a device: The gray boxes illustrate zone certificates of neighbours, the dotted lines are used to boarder the zone of the claimant. Those lines are located at the edge which abuts the claimed zone. Nevertheless, the best way to avoid the loss of zone certificate is to make applications of the sensor network energy aware. An application which recognizes a power outage in the near future could hand over at least its zone certificate to a trusted device. If enough energy is available, data could be replicated, too.

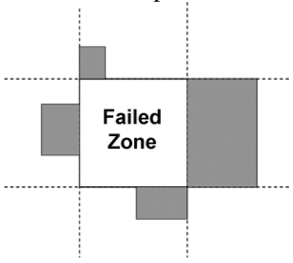

Fig. 3. Verification of zone claim after node failure

5.3.2 Self Certifying Path-Names

When registering a service in the secure distributed service directory, a node provides amongst other things its network layer address. If integrity of the proposed secure distributed service directory is ensured, how could a inquirer be sure, that she will contact the host at this address and not a malicious node? One way are Self-certifying Pathnames [6]. The idea is to store not only a network layer address in the service directory but also a *HostID* where *HostID* is a commitment to the host's public key. An entry would look like this: *Address:HostID*. Mazières [6] uses SHA-1 [4], a collision-resistant hash function, to calculate *HostID*. When the inquirer requests a service, it also asks the host to send its public key which the inquirer can verify with the corresponding *HostID*. The inquirer now holds the verified public key of the host. Inquirer and host can now establish a common symmetric key for all further communication. However, this hybrid method still involves asymmetric cryptographic operations which are very costly on peanut CPUs. As the use of public key operation is not enforced, an application can autonomously decide, if it needs that extra security for a given service. Please note that Self Certifying Path-names allow for secure communication both on network layer and in CAN.

5.3.3 Integrity

To ensure integrity of data, we use a redundancy approach like in [5] and [2]. The nodes of the CAN only store parts of the service information and artificial redundancy is added to the data, therefore not all parts of the data are needed to restore service information. Data parts are stored on different nodes. This ensures, that node failure does not affect the function of the distributed service directory, as long as there are enough nodes to restore the data. It also enables us to identify malicious or erroneous nodes, as they contribute inconsistent parts of the service information.

Another way to ensure integrity is self-certifying data: Zone owners in the CAN-space need not know the services offered by a sensor network. If a zone owner is not aware of the service names which are mapped to points in her CAN zone, we can establish data security using only hash functions. Integrity of entries is added by simply adding one new field to the entry which holds a hash of the value of the entry together with the function name. As the function name is never published into the CAN-space (hashing a service name to a point in CAN is the entry to CAN), a node not knowing available service names is not able to alter content in its hash table without the inquirer (which knows the service name) noticing. However, this approach works only if service names are long enough. Otherwise, a malicious node could use a dictionary attack to find out available service names.

6 Prototype

Further evaluation of the architecture will depend on actual prototypes of hardware like those presented by BeeCon GmbH [13]. Our prototype uses an Atmel ATMega 128 AVR Risc processor [14]. This processor comes with 128 kB of In-System reprogrammable Flash Memory, 4 kB of EEPROM and 4 kB of internal SRAM. Clock rate is 1 MHz internal and can be optionally boosted to 16 MHz using an extern clock generator. These features are responsible for severe restrictions on all operations in the sensor network. Connectivity of our testbed is realized by a robust and efficient Bluetooth stack which was developed for Ericsson Bluetooth modules in use on our current sensors and actuators. Bluetooth is used because actual hardware is available which allows us to build sensors with off-the-shelf components. There are also already some devices, which use Bluetooth (like a mobile phone). Those can be easily integrated in our sensor network. However, Bluetooth is not mandatory for the proposed architecture itself.

7 Summary and Outlook

This paper presented a work-in-progress status of an architecture for sensor networks. It showed the design of a secure distributed service directory. The design is based on an improvement of CAN, the Secure Content Addressable Network (S-CAN). Main focus of the paper is a secure join of new devices into the S-CAN. The paper showed how common secrets are constructed between S-CAN neighbours during the join of a new device without the use of public key cryptography or an infrastructure of any kind. The join process includes an intuitive and secure way to authenticate devices. The paper also presented a way to securely request services by the use of Self-Certifying Pathnames and showed how integrity protection could be done by Self-Certifying Data.

CAN in its basic implementation does not take into consideration the location of devices. This means, that eventually, communication of two physical neighbours takes plenty of hops in CAN. There are some enhancements of the basic proposal which we plan to integrate into our protocol.

With a distributed design, it is a goal to balance load to a huge number of nodes. However, hashing a service name to get a point in CAN-Space decides where a service description is stored. If there are a few services available, the load of storing service descriptions will be distributed to only a few nodes which own the zones corresponding to these points. One idea to circumvent this problem is to "spread" service names meaning to add additional information to extend the service name. Some attribute of the service description, like location, can be used. If those attributes are usually part of search queries, this procedure has an advantage as fewer entries will be returned. Another possibility is to use a hierarchy of services and to code this hierarchy in the service name string. Further research is needed in this direction.

As membership in our S-CAN is kind of energy costing, it is possible to rotate the task of S-CAN member in a closed group of trusted devices to save energy. We plan to present a robust and secure protocol for efficient rotation.

References

1. Dirk Balfanz, D. K. Smetters, Paul Stewart and H. Chi Wong: „Talking To Strangers: Authentication in Ad-Hoc Wireless Networks“, Symposium on Network and Distributed Systems Security (NDSS'02), Xerox Palo Alto Research Center, Palo Alto, USA, 2002
2. Ian Clarke, Oskar Sandberg, Brandon Wiley and Theodore W. Hong: “Freenet: A Distributed Anonymous Information Storage and Retrieval System”, in Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, USA, 2000
3. Neal Koblitz: “A course in number theory and cryptography, 2nd edition”, Springer Verlag, Berlin, 1994
4. FIPS 180-1: “Secure Hash Standard”, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, 1994
5. J. Kubiatowicz, et al: “OceanStore: An Architecture for Global-Scale Persistent Storage”, Proceedings of ACM ASPLOS, December 2000
6. David Mazières, Michael Kaminsky, M. Frans Kaashoek and Emmett Witchel: “Separating key management from file system security”, in 17th Symposium on Operating Systems Principles (SOSP'99), Kiawah Island, SC, 1999
7. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Shenker, „A Scalable Content-Addressable Network“, In Proceedings of ACM SIGCOMM 2001, August 2001
8. Antony Rowstron and Peter Druschel: “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems”, 18 Conference on Distributed Systems Platforms, Heidelberg, Germany, 2001
9. Frank Stajano, „Security for ubiquitous computing“, John Wiley & Sons, West Sussex, England, 2002
10. Emil Sit and Robert Morris, ”Security Considerations for Peer-to-Peer Distributed Hash Tables”, MIT Laboratory for Computer Science, Boston
11. Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan: ”Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, Technical Report TR-819, Massachusetts Institute of Technology, Cambridge, USA, March 2001
12. B.Y. Zhao, K.D. Kubiatowicz and A.D. Joseph: „Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing”, Technical Report UCB//CSD-01-1141, Computer Science Division, U. C. Berkeley, Berkeley, USA, April 2001
13. beeCon GmbH, http://www.beecon.de/ , access on July, 12th, 2003
14. Atmel Corporation, http://www.atmel.com/dyn/resources/prod_documents/2467s.pdf, access on July, 12th, 2003
15. Dallas Semiconductor Corp, http://www.ibutton.com, access on July, 12th, 2003