Efficient Cooperative Signatures: A Novel Authentication Scheme for Sensor Networks*

Stefaan Seys** and Bart Preneel

K.U.Leuven, Department Electrical Engineering-ESAT, SCD/COSIC,
Kasteelpark Arenberg 10, B-3001 Leuven, Belgium
{stefaan.seys,bart.preneel}@esat.kuleuven.ac.be

Abstract. This paper describes an efficient and strong authentication mechanism for ad hoc sensor networks. Our protocol focuses on providing strong authentication and privacy for requests from query nodes to the network and for the corresponding responses. Our scheme uses the asymmetrical energy consumption of the well known public key cryptosystems RSA and Rabin. As the sensor nodes are assumed to be power-restrained, we only employ efficient public key operations at their side of the protocol, this leaves us only with the public operations encryption and signature verification. We have extended this set with a novel building block that allows nodes to sign messages cooperatively. We show that our protocol is robust against attacks from both outsiders and insiders.

1 Introduction

As technology advances and integration of low-power radio, computing and sensor technology becomes reality, the road is paved for distributed sensor networks (DSNs). These networks will typically consist of 1000's of ultra-low power nodes, with limited communication means and CPU power [1, 5, 12, 10, 14].

Distributed sensor networks can be used in a wide range of applications, including military sensing, environment monitoring, collecting vital signs of patients, smart houses, etc. As sensor networks will be deployed and possibly left unattended in hostile environments, security is very important.

In this paper we focus on the key operation of a sensor network: pulling data from it. We propose a novel scheme that allows low-power devices to cooperatively send an authenticated answer to the requests from query nodes.

1.1 Network Operation

Our security architecture is designed with the following network infrastructure in mind. The majority of the nodes in the sensor network, *sensor nodes*, mea-

* This work was supported by the Concerted Research Action (GOA) Mefisto-2000/06 of the Flemish Government.

** Research financed by a Ph.D. grant of the the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

D. Hutter and M. Ullmann (Eds.): SPC 2005, LNCS 3450, pp. 86–100, 2005.

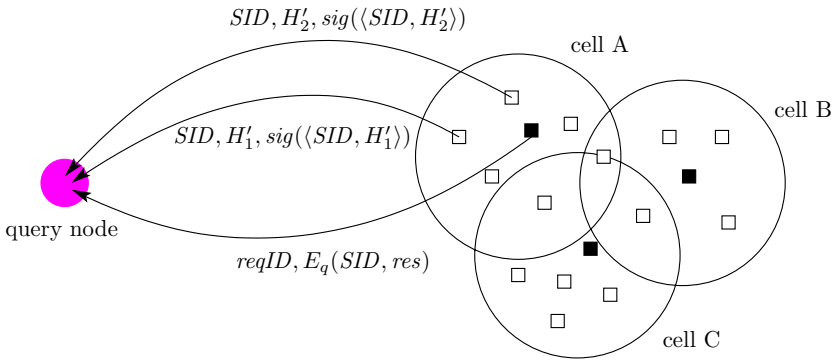

Fig. 1. Example network with three cells. Every cell has one manager node (black square)

sure whatever property they are designed to measure, for example, temperature, pressure, light intensity, etc. These sensor nodes are organized in *cells* (sometimes referred to as clusters). One node in each cell will act as a *cell manager*. The cell manager is responsible for collecting information from the sensor nodes in its cell and forwarding it to a *query node* or *sink node*. A query node requests (pulls) a specific cell manager for an update, while a sink node is used when an event is triggered by a sensor node and the update information is pushed to the sink node. Obviously a single node can act as both a query and a sink node. Figure 1 shows an example network topology with three cells. When a query node sends a request to the cell manager to pull data from the sensors, the cell manager broadcasts the request to the rest of its cell. Next to requests from query nodes, an update can be triggered by any sensor and will be forwarded to the cell manager. Sensors within a cell collect data, and locally process it resulting in a single response or update that is transmitted to a query or sink node respectively. The response/update is transmitted to the query/sink node by the cell manager. The cell manager also ensures that every node in its cell gets a copy of the final result, as this is required for our authentication scheme.

We propose an energy-efficient security architecture for sensor networks that provides the following security properties:

1. Query nodes can authenticate their requests.
2. The confidentiality of the response/update data can be guaranteed (only the query/sink node can read it).
3. Sensors in a cell *have* to cooperate in order to authenticate the response/update. This prevents that a single malicious node in the network can provide the query or sink nodes with incorrect information.

1.2 Assumptions

This paper is focused on providing strong authentication for the messages transmitted between a cell and a query or sink node. A number of additional measures needs to be taken to make the complete network operation secure.

– *Secure intra-cell communications.* Our scheme depends on nodes to be able to securely communicate with each other within a cell. It has little use to protect the confidentiality of the response to some query *only* between the cell manager and the query node – it also has to be protected while the cell is negotiating on the response.
– *Robust routing scheme.* Query nodes need to be able to contact the sensor nodes and vice versa. To make our scheme robust, it should be possible to adapt the configuration of the cells as sensor nodes can stop functioning or become corrupted.

1.3 Our Contributions

We present an authentication protocol that forces multiple nodes to cooperate in order to be able to authenticate a message. This prevents a single compromised or malicious node (or even a small subset of nodes) from sending authenticated messages. Moreover, our scheme is designed to work in the setting of power-constrained devices such as sensor nodes: the low-power devices only use the efficient *public* operations of RSA or of the Rabin public key cryptosystem, or symmetric building blocks. To the best of our knowledge no design has been proposed in the literature that can offer similar properties.

1.4 Notation

We will use the following notations:

– N_x: nonce generated by X,
– $Sig_x(m)$: signature on message m using X's private key,
– $E_x(m)$: public key encryption of m using X's public key,
– $E_K[m]$: symmetric encryption of m using symmetric key K,
– $\text{MAC}_K[m]$: Message Authentication Code of m using symmetric key K,
– $\langle a, b \rangle$: concatenation of a and b.

2 Efficient Encryption and Signature Verification

We use the asymmetric computational cost of the RSA and Rabin public key cryptosystems [7]. The textbook version of the RSA public key encryption scheme works as follows:

– Each user generates two large primes p and q.[1]
– Each user picks a public exponent e and computes the inverse $d = e^{-1} \mod \phi(pq)$, with $\phi()$ indicating the Euler function.
– The public key for a user is the pair $(n = pq, e)$; the private key consists of the prime factors p and q, or the pair (n, d).

[1] "*large*" in this context means 512 or more bits.

- The encryption c of a message m is equal to $c = m^e \bmod n$.
- The decryption m of a ciphertext c is equal to $m = c^d \bmod n$.

In order to make RSA more efficient, popular choices for the RSA public exponent e are either 3 (not recommended) or 65535 ($= 2^{16} - 1$), while the value d has about the same bit length as the modulus n. This means that the computational effort of encrypting (public operation) is much less than decrypting (private operation).

The textbook version of the Rabin public key encryption scheme is very similar to RSA, but it uses the even public exponent $e = 2$.[2] This is not a special case of RSA as this function is not 1-to-1: every ciphertext $c = m^2 \bmod n$ results in four possible plaintexts. Redundancy in the plaintext is required to ensure that only one square root is a legitimate message. Rabin encryption (public operation) is extremely efficient as it only involves a single modular squaring. By comparison, RSA with $e = 3$ requires an additional modular multiplication. Rabin decryption (private operation) is comparable in efficiency to RSA decryption.

The same efficiency difference holds for the RSA and Rabin signature schemes, where signing is equivalent to "decrypting" (private operation) and verifying is equivalent to "encrypting" (public operation). In all cases the public operation is very efficient, while the private operation is rather inefficient and requires a large computational effort [16]. While public key operations are sometimes considered too expensive for ultra low-power devices such as sensor nodes, we argue that RSA with a small public exponent or the Rabin public key cryptosystem allow the use of the *public* operations in these power-restrained devices.

3 Lamport One-Time Digital Signatures

Lamport proposed a so-called one-time signature scheme based on a general one-way function (OWF) F [6]. Lamport's scheme can be used to sign a single bit in the following way: the secret key consists of two random values x_0 and x_1, while the public key is the pair $\{F(x_0), F(x_1)\}$. The signature for bit b is x_b. For signing longer messages, several instances of this scheme are used. Lamport's scheme was further generalized in [2, 3, 8, 15]. There are other approaches like [9, 11] but these are not suitable for our purposes.

3.1 Lamport Scheme Using the Winternitz Improvement

One generalization of the Lamport scheme attributed by Merkle to Winternitz [8] is to apply the OWF F to the secret key iteratively a fixed number of times, resulting in the public key. Briefly the scheme works as follows. Suppose we wish to sign a m-bit message M. First the message M is split in m/t blocks of size t bits. Let these parts be $M_1, \ldots, M_{m/t}$. The secret key

[2] Note that all possible RSA public exponents e are odd.

is $sk = \{x_0, \ldots, x_{m/t}\}$ where x_i is a l-bit value . The public key is $pk = \{F^{(2^t-1)m/t}(x_0), F^{2^t-1}(x_1), \ldots F^{2^t-1}(x_{m/t})\}$[3]. The signature of a message M is computed by considering the integer value of the blocks $\mathrm{Int}(M_i) = I_i$. The signature $Sig(M)$ is composed of $m/t+1$ values $\{s_0, \ldots, s_{m/t}\}$ where, for $i \geq 1$, $s_i = F^{2^t-1-I_i}(x_i) = F^{-I_i}(y_i)$, while $s_0 = F^{\sum_i I_i}(x_0)$ for $1 \leq i \leq m/t$. The signature length is $l(m/t+1)$. On average, computing a signature requires $2\frac{2^t m}{t}$ evaluations of F. To verify a signature, one splits the message M in m/t blocks of size t bits. Let these parts be $M_1, \ldots, M_{m/t}$. One then verifies that pk equals $\{F^{2^t-1-\sum_i I_i}(s_0), F^{I_1}(s_1), \ldots, F^{I_{m/t}}(s_{m/t})\}$ for $1 \leq i \leq m/t$. It is possible to prove that forging a signature of a message M' given a message M, a valid signature $Sig(M)$ and the public key requires inversion of the function F.

In practice we assume that F maps 64 bits to 64 bits. Since collision resistance is not required from F we believe that this parameter is sufficient. In order to prevent attackers from building a large table of evaluations of F, F can be made different for each signature by defining $F(x)$ to be $G(\mathrm{Salt}||x)$, where G is a one-way 128 bits to 64 bits function and Salt is generated at random by the signer and transmitted to the verifier. Suitable F's can be constructed from efficient block ciphers such as AES or from fast hash functions such as SHA-1.

Further we assume that the message M that needs to be signed is hashed with a cryptographic hash function such as SHA-1 before it is fed to the signing algorithm. If the length of the message is smaller than the output of the hash function, then the hash function is not applied. This ensures that the input length of the signing algorithm is at most the output length of the hash algorithm being used.

Note that the secret key $sk = \{x_0, \ldots, x_{m/t}\}$ can be generated with a good pseudo-random generator using a single seed x. This means that storing the secret key only requires l bits instead of $(m/t)l$ bits. Obviously this is not true for the public key.

3.2 Merkle Trees

One disadvantage of the Lamport scheme is the size of the public key. All verifiers need an authenticated copy of this public key in order to verify the validity of a signature. Merkle proposed the use of binary trees to authenticate a large number of public keys with a single value, i.e., the root of the tree [8]. A Merkle tree is a complete binary tree with a n-bit value associated to each node such that each interior node value is a OWF of the node values of its children (Fig. 2):

$$P[i,j] = F\big(\langle P[i,(i+j-1)/2], P[(i+j+1)/2,j]\rangle\big)\,.$$

The N values that need to be authenticated are placed at the N leaves of the tree. Although the leaf value may be chosen arbitrarily, usually it is a cryptographic hash of the values that need to be authenticated. In this case these values are

[3] Note that $F^2() = F(F())$ is applying the OWF F twice iteratively.

called *leaf-preimages*. A leaf can be verified with respect to a publicly known root value and the *authentication path* of the leaf. We assume that the public keys of the Lamport one-time signature scheme (Sect. 3.1) are stored at the leaf-preimages of the tree (one public key per leaf-preimage).

Authentication Paths. Let sib_i be the value of the sibling of the node on height i on the path from the leaf to the root. A leaf has height 0, the OWF of two leaves has height 1, etc., and the root has height H if the tree has 2^H leaves. The authentication path is then the set $\{sib_i \mid 0 \leq i \leq H\}$. For example, the gray nodes in Fig. 2 are the authentication path for leaf y_3.

A leaf may be authenticated as follows: First apply the OWF to the leaf and its sibling sib_0, then apply the OWF to the result and sib_1, etc., all the way up to the root. If the calculated root value is equal to the published root value, then the leaf value is accepted as authentic. This operation requires $\log_2(N)$ invocations of the OWF.

Authentication Path Generation. The goal of *Merkle tree traversal* is the sequential output of the leaf values and their authentication paths. In [8], Merkle presents a straightforward technique that requires a maximum of $2\log_2(N)$ invocations of the OWF per round, and requires a maximum storage of $\log_2^2(N)/2$ outputs of the OWF. In [4], Jakobsson et al. present an algorithm which allows a time-space trade-off. When storage is minimized, the algorithm requires about $2\log_2(N)/\log_2(\log_2(N))$ invocations of the OWF, and a maximum storage of $1.5\log_2^2(N)/\log_2(\log_2(N))$ outputs of the OWF. Finally in [13], Szydlo presents an algorithm that requires $2\log_2(N)$ time and a maximum storage of $3\log_2(N)$. All three Merkle tree traversal algorithms described here start with the calculation of the tree root. During this root calculation, the initial internal state of the algorithms are also calculated and stored in memory. This initialization requires $N-1$ invocations of the OWF.[4] The initial state storage requirements are maximized as stated before.

3.3 Public Key Chaining

Another means for producing multiple one-time signatures associated to a single public key is the use of public key chaining. In this technique the public keys are still computed by applying a OWF F multiple times to the private key, but now the OWF is applied s times more than in the simple case of Sect. 3.1). This enables us to use the same private key s times. Figure 3 shows an example of this process. The columns in Fig. 3 represent OWF-chains starting from the top, going downwards. The public key of the first signature to be generated with the private key sk is depicted by the bottom row pk. Recall that $s_i = F^{-I_i}(y_i)$, this means that computing a signature is equivalent with going up the chain

[4] The cost of this initial setup is not included in the time and storage requirements stated previously.

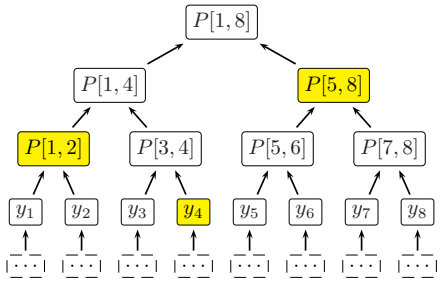

Fig. 2. Merkle tree with 8 leaves. The root $P[1,8]$ can be used to authenticate the complete tree

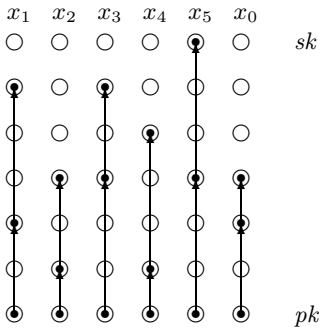

Fig. 3. Public key chaining: the previous signature becomes the public key for the following signature. This process continues until the secret part of one of the chains becomes to short

I_i times; this is depicted by the arrows pointing up.[5] For the next signature, the public key becomes the *previous* signature, and the signature is computed by going up the chains I_i times starting from that point, etc. The disadvantage of this technique is that the computational effort for the signatures is larger as the chains are longer. This technique provides a means to exchange storage requirements for computation time.

4 One-Time Cooperative Signature Scheme

4.1 High Level Overview of the Protocol

Figure 4 depicts the preparation phase of the signature scheme. First an error-correcting code is applied to strengthen the scheme. Let $H' = ECC(H)$ be the result of applying the error-correcting code ECC to the cryptographic hash H of the original message M. Now we split H' in k parts $H'_1, \ldots, H'_k$. Every node is assigned a subset of these parts to sign using the scheme explained in Sect. 3.1. Let k be the number of users in a group, and b the maximum allowed number of non-cooperative users in this group.

4.2 Error-Correcting Codes

The use of an error-correcting code that can recover the original message from a fraction $\frac{k-b}{k}$ of the code words provides the following properties:

[5] In practice the signer has to start from the private key and compute downwards since the function F is irreversible.

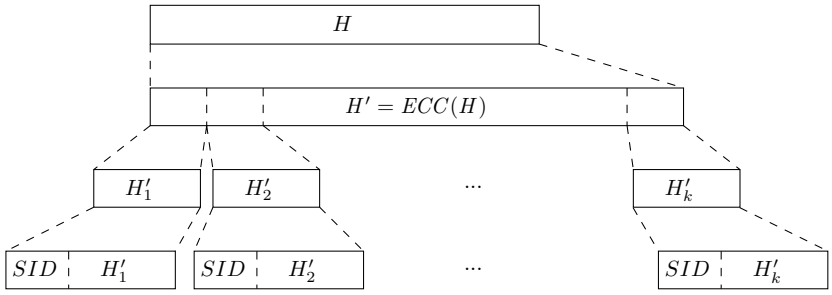

Fig. 4. Preparation phase of the signing process

- *Robustness.* If at least $k - b$ parts of the signature arrive unaltered, a valid signature can be recovered from them.
- *Honest insider detection.* If no more than b out of k users misbehave (i.e., by signing a different message or by creating an invalid signature), the honest users can be identified using their valid partial signatures.

We use a concrete example to further clarify this. Suppose we have a group of $k = 15$ users and use a cryptographic hash function with a 160-bit output. Every user computes the hash value of the original message M resulting in the 160-bit message hash H. Further suppose that we want to be able to reconstruct a valid signature even if $b = 3$ out of the 15 users refuse to cooperate. One error-correcting code that can achieve this property is a (45,27) Reed-Solomon code over $GF(2^6)$ [17]. This code operates in the q-ary alphabet ($q = 2^6$) and encodes 27 information symbols into 45 code symbols, having a fractional redundancy of 40%, and guarantees a 9 symbol-error-correcting capability. As each user is supposed to sign its 3 designated code words, 3 malicious users cannot corrupt more than 9 code words and hence the signature can be recovered from the remaining 36 code words. Adapting the scheme to the group size k and threshold b is simply a matter of selecting a suitable error-correcting code.

Note that our scheme requires that more than half of the users behave correctly. If more than half of the users in a cell behave incorrectly, they can cooperate and jointly sign some altered message $\widehat{M}$, regardless of the error-correcting code that is used.

4.3 Partial Signatures

After applying the error-correcting code, user i uses the following scheme to sign its designated part H'_i of H'. First i increments the Signature Identifier (*SId*) and concatenates it with its identity ID_i resulting in SID_i. This SID_i together with H'_i are signed using the scheme explained in Sect. 3.1. The *SId* is used to link the different partial signatures with each other. Without this link an adversary could collect partial signatures on *different* messages and try to combine them to create

a signature on a new message[6]. The identity ID_i is included in the signature in order to allow a user to *prove* that he created a valid partial signature on the message M. Finally the user transmits $\langle M, SID_i, H'_i, Sig_i(\langle SID_i, H'_i\rangle)\rangle$ to the verifier.

Note that in many cases, depending on the k and b parameters, the length of H'_i together with SID_i is less than 160 bit. This means that the computational cost of a partial signature is less than the cost of a normal individual signature.

4.4 Verification

Upon receiving the message M, its multiple parts H'_i of H', and their corresponding partial signatures $Sig_i(\langle SID_i, H'_i\rangle)$ from the different cooperating signing users in a group, the verifier uses the following protocol to verify the correctness of the *complete* signature on the message M.

First the verifier checks if $Sig_i(\langle SID_i, H'_i\rangle)$ is a valid signature on $\langle SID_i, H'_i\rangle$ for all the individual partial signatures using the protocol described in Sect. 3.1. Next the verifier checks whether there are enough valid signatures, if so, the verifier recombines the different H'_i (replacing missing or invalid parts with 0's) into $\bar{H}'$. Enough here means at least $k - b$ valid parts. The verifier now decodes this $\bar{H}'$ into $\bar{H}$ using the error-correcting code. Finally the verifier checks whether $\bar{H}$ equals the cryptographic hash of the received message M. Depending on the result of this verification process, the verifier can conclude the following:

1. If there are sufficient valid partial signatures and $\bar{H}$ equals the computed H, then the combination of the different partial signatures is accepted as a valid signature on the message M, and the users that *did* generate a valid partial signature can be identified;
2. if there are sufficient valid partial signatures but $\bar{H}$ is not equal to the computed H, then the cooperating users signed different messages (indicating an attack by users inside the group), or the message was altered (indicating a Denial of Service (DoS) attack by insiders or outsiders);
3. if there are insufficient valid partial signatures, then this indicates a DoS attack by insiders or outsiders.

4.5 Informal Security Analysis

The one-time signatures on the H'_i's protect them from being altered by an adversary. This means that the verifier can be assured of the validity of the received H'_i and the identity ID_i of the user that signed it. The use of unique identifiers that link the partial signatures make it impossible to combine partial signatures on different messages in order to create a signature on some new

[6] Note that if a hash function is applied to the message before it is signed, the adversary will only obtain a valid signature on a message hash. She still needs to invert the hash function in order to get a signature on a message itself. This is referred to in the literature as existential forgery.

message. These measures prevents *outsiders* from altering the signed messages or creating a valid signature on a new message.

The scheme also protects against a limited number of maliciously *collaborating insiders*. If no more than b malicious insiders do not follow the correct signing process, then they cannot prevent the rest of the users to create a valid signature. The valid partial signatures can be used to identify the honest users. If more than b malicious insiders do not follow the correct signing process, then they can prevent the others from creating a valid signature (DoS attack). If more than $k-b-1$ malicious insiders collaborate to sign an altered message $\widehat{M}$ (while the remaining honest users faithfully sign M), then the attackers will succeed and can produce a valid signature on $\widehat{M}$. Note that in this case the verifier will incorrectly conclude that the honest users are trying to disrupt the signing process. It is easy to see that our scheme can only support thresholds b smaller than $k/2$.

5 Security Architecture

In the previous sections we have showed how we can achieve the following *efficient* public key operations: (1) encryption, (2) signature verification and (3) signature generation (cooperative or individual). In this section we propose a scheme that uses these building blocks to provide strong authentication for query-response conversations between query (sink) nodes and cells in the sensor network. Note that in our setting the low-power devices do not possess an asymmetric decryption (private) key since we assume that the decryption operation is too power consuming.

Our scheme requires the following Public Key Infrastructure (PKI) to be in place:

1. Every query and sink node has a private/public key pair for signing and another pair for encryption, both accompanied by a certificate signed by some third party.
2. Every sensor node has an authenticated copy of this third party's public key in order to be able to verify the certificates of the query or sink nodes. Note that signature verification is an efficient operation.
3. Every sensor node has a number of private/public key pairs to be used with the one-time signature scheme explained in Sect. 4. In Sect. 5.2 we show how these key pairs can be renewed.

5.1 Strong Authentication Between Query Nodes and Cells

Using the proposed building blocks, implementing the authentication scheme itself is straightforward.

Authenticated Requests. When a query node Q wishes to send an authenticated request *req* to a manager node M, it uses the following protocol:

$$Q \longrightarrow M: \quad reqID, req, Sig_q(\langle reqID, req\rangle) \ .$$

The *reqID* is incremented for every request and stored in memory by both the query node and the manager nodes. Only requests with an *reqID* larger than the one in memory are accepted. The signature in combination with the *reqID* ensure the manager node that the request is not a replay and that it originated from a valid query node. If freshness of the request must be guaranteed, then a three-message challenge/response can be used:

$$\begin{array}{lll} Q \longrightarrow M: & \text{notify} & (1) \\ Q \longleftarrow M: & N_m & (2) \\ Q \longrightarrow M: & req, Sig_q(req, N_m) & (3) \end{array}$$

Here the first message is only necessary to notify the manager node that the query nodes wishes to send an authenticated request. In the push model towards a sink node this message is not necessary.

Authenticated Replies or Updates. For this purpose we developed the Cooperative Signature Scheme explained in Sect. 4. Obviously our scheme can be used in any low-power setting were a single device is not trusted to sign a message individually. As we explained, we assume that upon arrival of a valid request, the manager node broadcasts the request to the cell, and the cell locally computes the best result from the collective data. The manager ensure that all nodes in its cell know this final result *res*. Once the final result is established, the manager node replies to the request with the following message: $\langle reqID, E_q(\langle SID, res\rangle)\rangle$. This message contains the identity of the corresponding request and the encryption (with the query nodes public key) of the final result and the *SID* that will be used for the cooperative signatures.

All nodes in the cell employ the cooperative signature scheme in order to create the partial signatures on the final result *res*. These partial signatures $\langle H'_i, Sig_i(\langle SID, H'_i\rangle)\rangle$ are transmitted by every node in the cell to the query node (see Fig. 1). The query node collects all partial signatures and verifies the correctness of the complete signature on the result *res* it received from the manager node. Note that the result of this verification process might be used to distinguish between honest nodes and possibly uncooperative sensor nodes.

5.2 One-Time Secret Key Updates

Two important aspects in the use of one-time signature schemes is (1) generating public keys, and (2) providing the verifier with an authenticated copy of these public keys [8]. In our architecture we efficiently solve this problem by reversing it: we let the verifier (query nodes) generate the public key and transmit an authenticated and encrypted version of the corresponding private key to the signers (sensor nodes). This has multiple advantages:

1. The computational burden of generating the random private keys and computing the corresponding public keys is off-loaded from the low-power sensor

nodes. When Merkle trees are used, computing the root node and the initial internal state of the tree traversal algorithm is also off-loaded from the sensor nodes.
2. The verifier automatically obtains an authenticated copy of the public key.
3. The private key sk can be generated from an l-bit seed $\underline{sk}$. This means that transmitting the private key to the signer is more efficient than transmitting the public key to the verifier. This is true particularly in this case where there is only one dedicated verifier.

The disadvantage is that the secret key is known by two parties, but in this scenario that is not an issue, as the query nodes are assumed to be trusted.

Public Key Authentication Using Public Key Chaining. The query node Q first generates n private keys sk_i from the seeds $\underline{sk}_i$ and computes the public keys pk_i. Protocol (1) shows the scheme we propose in order to install these new key pairs when authenticating public keys using public key chaining. First a symmetric session key K is established between the sensor node S and the query node Q. The signature in message (2) is required to provide the query node with prove that this session key K is really generated by sensor node S. In the last message, the query node transmits an encrypted set of new private keys, and authenticates them with a MAC. Both the encryption key and authentication key are derived from the session key K.

Note that this protocol is only efficient if multiple secret keys are transferred using the session key K since one signature is required in message (2). Even the small sensor nodes should be able to store multiple private keys simultaneously since only a single l-bit seed has to be stored per private key (the sensor nodes do not need to store or compute public keys in this case). The query node has to store the bottom rows of all n key chains, i.e., all the public keys (see Fig. 3).

Protocol (1): One-time Secret Key Update Protocol when Using Key Chaining

Pre-protocol setup: The query node Q prepares n fresh private/public key pairs (sk_i, pk_i) that are to be used by sensor node S. The private keys sk_i are generated by the seed values $\underline{sk}_i$.
Conventions: K_1 and K_2 are two distinct keys derived from the session key K.
Protocol messages:

$$Q \longrightarrow S: \quad N_q \tag{1}$$
$$Q \longleftarrow S: \quad N_q, N_s, E_q(K), Sig_s(N_q, N_s, K) \tag{2}$$
$$Q \longrightarrow S: \quad E_{K_1}[\underline{sk}_1, \ldots, \underline{sk}_n], \mathrm{MAC}_{K_2}[\underline{sk}_1, \ldots, \underline{sk}_n, N_s] \tag{3}$$

Result: Node S can now use the new secret keys to sign messages.

Public Key Authentication Using Merkle Trees. When using Merkle trees to authenticate the public keys, the query node Q first generates n private keys sk_i from the seeds $\underline{sk}_i$ and computes the public keys pk_i as before. The hashes of these public keys $h(pk_i)$ are then placed at the leaves of a Merkle tree. Finally the

query node calculates the root of the tree and the initial internal state *Init* of the tree traversal algorithm. The sensor node requires the following information in order to sign messages and compute authentication paths: the $\underline{sk}_i$'s, the $h(pk_i)$'s and *Init*. Note that both $\underline{sk}_i$ and $h(pk_i)$ are short bit-strings (compared to a complete public/private key) and that the size of *Init* is maximized by $3\log_2(n)$ l-bit values when using Szydlo's tree traversal algorithm.

The protocol messages of Protocol (1) can be reused when using Merkle trees when replacing message (3) by:

$$Q \longrightarrow S: \quad E_{K_1}[m], \mathrm{MAC}_{K_2}[m] \text{ with } m = \langle \{\underline{sk}_i, h(pk_i)\}_{1 \leq i \leq n}, Init \rangle \ .$$

After successful completion of the private key update protocol the query node only has to store the root of the Merkle tree in order to be able to verify signatures. When using Merkle trees, the sensor node has to regenerate the public key when signing a message and include it in the signature as the verifier (i.e., the query node) needs the public key to verify the validity of the signature.

Comparison. When using public key chaining, generating signatures requires multiple evaluations of the OWF F as the signer has to work his way down the chains starting from the top (Fig. 3). Next to this the verifier needs to store the current public key in memory.

The use of Merkle trees requires that the signer computes the public key and the authentication path and includes both in the signature. On the other hand, the verifier only needs to store the root of the tree. The size of message (3) of Protocol 1 when using Merkle trees will be about double the size of this message when using public key chaining.

The optimal choice depends on multiple factors such as the number of verifiers, relative cost of communications and computations, specific scenario in which the protocol is used, etc.

6 Related Work

Zhou and Haas present a distributed key management service based on threshold cryptography [18]. In particular the functionality of the Certification Authority (CA) is distributed among multiple nodes in the network (servers). A node has to collect and combine partial signatures on its certificate from a subset of these servers. Distributing the secret key of the CA prevents an attacker from compromising the whole PKI by capturing a single node. This scheme relies heavily on demanding public key operations, while our scheme only uses efficient operations. Moreover, in the scheme of Zhou and Haas, the workload for every partial signer is exactly as large as in the case when he would sign the message individually. Hence the cost of a cooperative signature is k times larger than the cost of a normal signature. In our scheme the cost of a partial signature will normally be smaller than the cost of a normal signature, so the nodes actually distribute the workload amongst each other.

7 Conclusions

In this paper we have described an efficient and strong authentication mechanism that enables query nodes and cells to securely exchange request/response conversations. As the sensor nodes are assumed to be power-restrained, we only employ efficient public key operations at their side of the protocol. We have developed and presented a new building block that allows nodes to sign messages cooperatively and have shown that our protocol is robust both against attacks from outsiders as from insiders.

References

[1] F. Bennett, D. Clarke, J. Evans, A. Hopper, A. Jones, and D. Leask, "Piconet: embedded mobile networking," *IEEE Personal Communications*, vol. 4, pp. 8–15, Oct. 1997.

[2] D. Bleichenbacher and U. Maurer, "Directed acyclic graphs, one-way functions and digital signatures," in *Advances in Cryptology – CRYPTO '94* (Y. Desmedt, ed.), vol. 839 of *Lecture Notes in Computer Science*, pp. 75–82, Springer-Verlag, 1994.

[3] S. Even, O. Goldreich, and S. Micali, "On-line/off-line digital signatures," in *Advances in Cryptology – CRYPTO '89* (G. Brassard, ed.), vol. 435 of *Lecture Notes in Computer Science*, pp. 263–275, Springer-Verlag, 1990.

[4] M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo, "Fractal Merkle tree representation and traversal," in *Topics in Cryptology – RSA Conference Cryptographers' Track (RSA-CT '03)*, vol. 2612 of *Lecture Notes in Computer Science*, Springer, 2003.

[5] J. Kahn, R. Katz, and K. Pister, "Next century challenges: Mobile networking for "smart dust"," in *Proceedings of the 5th International Conference on Mobile Computing and Networking (MobiCom '99)*, pp. 483–492, ACM Press, Aug. 1999.

[6] L. Lamport, "Constructing digital signatures from a one way function," Technical Report CSL-98, SRI International, Oct. 1979.

[7] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[8] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology – CRYPTO '89* (G. Brassard, ed.), vol. 435 of *Lecture Notes in Computer Science*, pp. 218–238, Springer-Verlag, 1990.

[9] A. Perrig, "The BiBa one-time signature and broadcast authentication protocol," in *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS '01)*, ACM Press, New York, NY, USA, 2001.

[10] J. Rabaey, J. Ammer, J. da Silva, D. Patel, and S. Roundy, "Picoradio supports ad hoc ultra-low power wireless networking," *IEEE Computer Magazine*, July 2000.

[11] L. Reyzin and N. Reyzin, "Better than BiBa: Short one-time signatures with fast signing and verifying," in *Proceedings of the 7th Australian Conference on Information Security and Privacy* (J. Seberry, ed.), vol. 2384 of *Lecture Notes in Computer Science*, pp. 144–153, Springer-Verlag, 2002.

[12] R. Szewczyk and A. Ferencz, "Power evaluation of smartdust remote sensors," CS252 project reports (final), Berkeley Wireless Research Center, 2000.

[13] M. Szydlo, "Merkle tree traversal in log space and time," in *Advances in Cryptology – EUROCRYPT '04* (C. Cachin and J. Camenisch, eds.), vol. 3027 of *Lecture Notes in Computer Science*, pp. 541–554, Springer, May 2004.
[14] University of California, "Wireless integrated network sensors (WINS)." ⟨`http://www.janet.ucla.edu/WINS/`⟩.
[15] S. Vaudenay, "One-time identification with low memory," in *Proceedings of EUROCODE '92* (P.Camion, P.Chappin, and S.Harari, eds.), no. 339 in CISM Courses and lectures, pp. 217–228, Springer-Verlag, 1992.
[16] M. J. Wiener, "Performance comparison of public-key cryptosystems," *RSA Laboratories' CryptoBytes*, vol. 4, no. 1, pp. 1+3–5, 1998.
[17] S. G. Wilson, *Digital Modulation and Coding*. Prentice Hall, 1996.
[18] L. Zhou and Z. Haas, "Securing ad hoc networks," *IEEE Network Magazine Special Issue on Network Security*, vol. 13, no.6, 1999.