

Key establishment in sensor networks

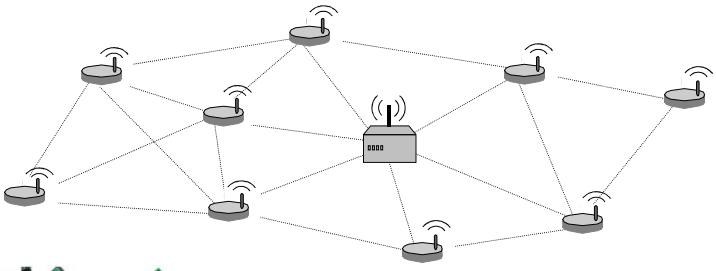
Security Protocols (bmevihim132)

Dr. Levente Buttyán
associate professor
BME Hálózati Rendszerek és Szolgáltatások Tanszék
Lab of Cryptography and System Security (CrySyS)
buttyan@hit.bme.hu, buttyan@crysys.hu

Outline

- introduction to wireless sensor networks
- needed key types
- the LEAP protocol
- random key pre-distribution
- polynomial based random key pre-distribution

Wireless sensor networks (WSNs)



some applications:

- environmental monitoring (for ecological and/or agricultural purposes)
- monitoring the state of structures (e.g., bridges, tunnels, railway tracks)
- remote patient monitoring (elderly and chronically ill people)
- industrial process automation, control systems
- building automation
- military applications

Sensor hardware

characteristics of the Berkeley MICA mote

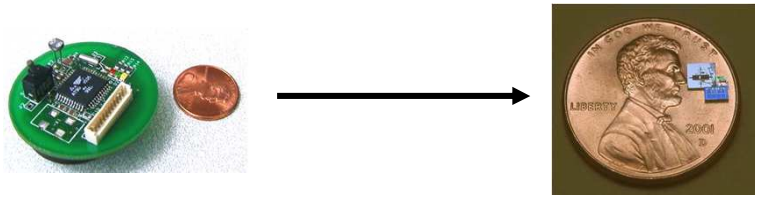
CPU	8-bit, 4 MHz
storage	8KB instruction flash
	512 bytes RAM
	512 bytes EEPROM
communication	916 MHz radio
bandwidth	10 Kbit/sec
OS	TinyOS (3.5 KB)
space left	4.5 KB

Severe resource constraints

- memory constraints
  - memory is not enough to store even the variables of standard asymmetric key crypto systems (e.g., RSA)
  - standard implementations of symmetric key primitives (ciphers and hash functions) need to be optimized in order to fit in the memory

  but:
  - available memory may increase in the future (price is still an issue)
  - some asymmetric crypto systems may require less resources (e.g., ECC)
- processor
  - 4 MHz, 8 bit RISC processor, with 32 general purpose registers
  - limited instruction set
    - good support for bit- and byte-level I/O operations
    - lack of arithmetic and logic operations
  - existing crypto libraries must be re-written for this special platform
- battery power
  - will remain a crucial limitation for some time
  - communications consume much more energy than computation
  - crypto algorithms and PROTOCOLS must be designed and optimized to reduce energy consumption

Resource constraints remain with us…

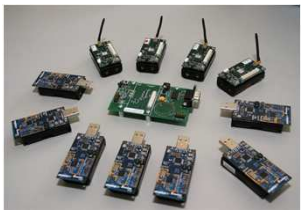


Security challenges in WSNs

- no physical protection of sensor nodes
  - unattended operation, tamper resistance is expensive
  - some nodes may be compromised (secrets leaked, Byzantine behavior)
- wireless operation
  - eavesdropping, jamming, spoofing, replay, …
- resource constraints
  - battery powered operation (energy efficiency is key to increase network lifetime)
  - limited computing and storage capability
  - limited radio range (communication is very energy consuming)
- ad hoc topology
  - possibly random deployment, node failures, battery exhaustion, replenishment
- special mechanisms
  - one-to-many (broadcast, geo-cast) and many-to-one (convergecast) communication
  - in-network processing, aggregation
  - localization, scheduling, clustering, …

Our work on WSN security

- CrySyS WSN test bed:
  - 4 Crossbow MicaZ motes + programming board
  - 20 MoteIV TmoteSky motes
  - ZigBee compatible (2.4 GHz)



- research topics (~WSAN4CIP project):
  - secure and resilient routing protocols (RPL implementation and security extensions)
  - resilient data aggregation algorithms
  - secure and reliable cluster head election protocols
  - dependable transport protocols
  - secure distributed data storage schemes (also for forensics purposes)
  - prevention of traffic analysis (identification of special nodes)

Key establishment in WSNs

- due to resource constraints, asymmetric key cryptography should be avoided in sensor networks
- we aim at setting up symmetric keys

- requirements for key establishment depend on
  - communication patterns to be supported
    - many-to-one (convergecast)
    - one-to-many (local and global broadcast)
    - one-to-one (unicast)
  - need for supporting in-network processing
  - need to allow passive participation

- useful key types
  - node keys – shared by a node and the base station
  - link keys – pairwise keys shared by neighbors
  - cluster keys – shared by a node and all its neighbors
  - network key – a key shared by all nodes and the base station

Node, cluster, and network keys

- node key
  - can be preloaded into the node before deployment

- cluster key
  - can be generated by the node and sent to each neighbor individually protected by the link key shared with that neighbor

- network key
  - can also be preloaded in the nodes before deployment
  - needs to be refreshed from time to time (due to the possibility of node compromise)
    - neighbors of compromised nodes generate new cluster keys
    - the new cluster keys are distributed to the non-compromised neighbors
    - the base station generates a new network key
    - the new network key is distributed in a hop-by-hop manner protected with the cluster keys

Constraints for link key establishment

- no a priori knowledge of post-deployment topology
  - it is not known a priori who will be neighbors
- gradual deployment
  - need to add new sensors after deployment

Traditional approaches

- use of public key crypto (e.g., Diffie-Hellman)
  - limited computational and energy resources of sensors
- use of a trusted key distribution server (Kerberos-like)
  - base station could play the role of the server
  - requires routing of key establishment messages
    - but routing may already need link keys
  - base station becomes single point of failure
- pre-loaded link keys in sensors
  - post-deployment topology is unknown
  - single “mission key” approach
    - vulnerable to single node compromise
  - *n* -1 keys in each of the *n* sensors
    - scalability issues
    - excessive memory requirements
    - gradual deployment is difficult

The LEAP protocol

- LEAP – Localized Encryption and Authentication Protocol
- main assumptions:
  - any sensor node will not be compromised within T_{min} time after its deployment
  - any node can discover its neighbors and set up neighbor relationships within $T_{est} < T_{min}$ time
    - typically, T_{est} is a few seconds, so these assumptions make sense in practice
- protocol phases:
  - key pre-distribution phase
  - neighbor discovery phase
  - link key establishment phase
  - key erasure phase

LEAP operation

- key pre-distribution phase
  - before deployment, each node is loaded with a master key K_I
  - each node u derives a node key K_u as $K_u = f(K_I, u)$, where f is a one-way function
- neighbor discovery phase
  - when a node is deployed, it tries to discover its neighbors by broadcasting a HELLO message

    $u \rightarrow *: u, N_u$

    where N_u is a random nonce
  - each neighbor v replies with

    $v \rightarrow u: v, mac(K_v, v|N_u)$

  - u can compute $f(K_I, v) = K_v$, and verify the authenticity of the reply

LEAP operation

- link key establishment phase
  - u computes the link key $K_{uv} = f(K_v, u)$
  - v computes the same key
  - no messages are exchanged
  - note:

    u does not authenticate itself to v, but …
    - only a node that knows K_I can compute K_{uv}
    - a compromised node that tries to impersonate u cannot know K_I (see below)

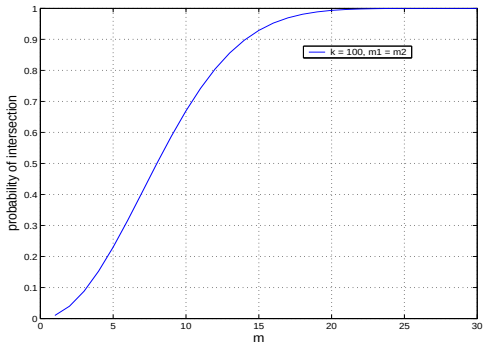
- key erasure phase
  - T_{min} time after its deployment, each node deletes K_I and all node keys it computed in the neighbor discovery phase

Random key pre-distribution

Given a set S of k elements, we randomly choose two subsets S_1 and S_2 of m_1 and m_2 elements, respectively, from S.

What is the probability of $S_1 \cap S_2 \neq \emptyset$?

$$\Pr\{S_1 \cap S_2 \neq \emptyset\} = 1 - \frac{(k-m_1)!(k-m_2)!}{k!(k-m_1-m_2)!}$$



The basic random key pre-distr. scheme

- initialization phase
  - a large pool S of unique keys are picked at random
  - for each node, m keys are selected randomly from S and pre-loaded in the node (key ring)
- direct key establishment phase
  - after deployment, each node finds out with which of its neighbors it shares a key (e.g., each node may broadcast the list of its key IDs)
  - two nodes that discover that they share a key verify that they both actually posses the key (e.g., execute a challenge-response protocol)
- path key establishment phase
  - neighboring nodes that do not have a common key in their key rings establish a shared key through a path of intermediaries
  - each link of the path is secured in the direct key establishment phase

Setting the parameters

- connectivity of the graph resulting after the direct key establishment phase is crucial
- a result from random graph theory [Erdős-Rényi]:
  in order for a random graph to be connected with probability c (e.g., c = 0.9999), the expected degree d of the vertices should be:

$$d = \frac{n-1}{n}(\ln(n) - \ln(-\ln(c))) \qquad (1)$$

- in our case, $d = pn'$ (2), where p is the probability that two nodes have a common key in their key rings, and n' is the expected number of neighbors (for a given deployment density)
- p depends on the size k of the pool and the size m of the key ring

$$p = 1 - \frac{((k-m)!)^2}{k!(k-2m)!} \qquad (3)$$

- $c \xrightarrow{(1)} d \xrightarrow{(2)} p \xrightarrow{(3)} k, m$

Setting the parameters (example)

- number of nodes: n = 10000
- expected number of neighbors: n' = 40
- required probability of connectivity after direct key establishment: c = 0.9999

- using (1):
  required node degree after direct key establishment: d = 18.42
- using (2):
  required probability of sharing a key: p = 0.46
- using (3):
  appropriate key pool and key ring sizes:
  k = 100000, m = 250
  k = 10000, m = 75
  …

Qualitative analysis

- advantages:
  - parameters can be adopted to special requirements
  - no need for intensive computation
  - path key establishment have some overhead …
    - decryption and re-encryption at intermediate nodes
    - communication overhead
  - but simulation results show that paths are not very long (2-3 hops)
  - no assumption on topology
  - easy addition of new nodes

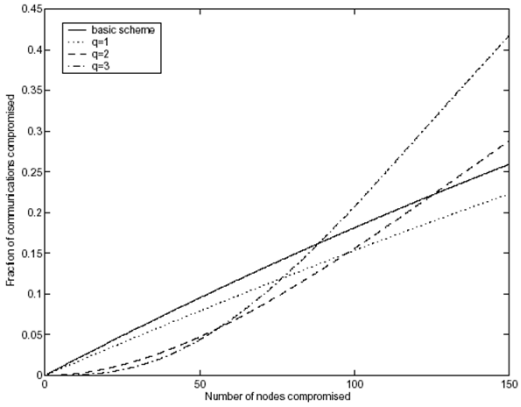
- disadvantages:
  - node capture affects the security of non-captured nodes too
    - if a node is captured, then its keys are compromised
    - these keys may be used by other nodes too
  - if a path key is established through captured nodes, then the path key is compromised
  - no authentication is provided

q-composite rand key pre-distribution

- basic idea:
  - two nodes can set up a shared key if they have at least *q* common keys in their key rings
  - the pairwise key is computed as the hash of *all* common keys
- advantage:
  - in order to compromise a link key, *all* keys that have been hashed together must be compromised
- disadvantage:
  - probability of being able to establish a shared key directly is smaller (it is less likely to have *q* common keys, than to have one)
  - key ring size should be increased (but: memory constraints) or key pool size should be decreased (but: effect of captured nodes)

q-composite scheme: Simulation results

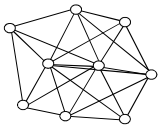
$m = 200$, $p = 0.33$



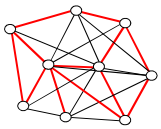
taken from: H. Chan and A. Perrig and D. Song, "Random key predistribution schemes for sensor networks", IEEE Security and Privacy Symp. (Oakland), 2003

Multipath key reinforcement

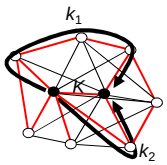
- basic idea:
  - establish link keys through multiple disjoint paths
  - assume two nodes have a common key K in their key rings
  - one of the nodes sends key shares k_1, …, k_j to the other through j disjoint paths
  - the key shares are protected during transit by keys that have been discovered in the direct key establishment phase
  - the link key is computed as $K + k_1 + \ldots + k_j$



radio connectivity



shared key connectivity



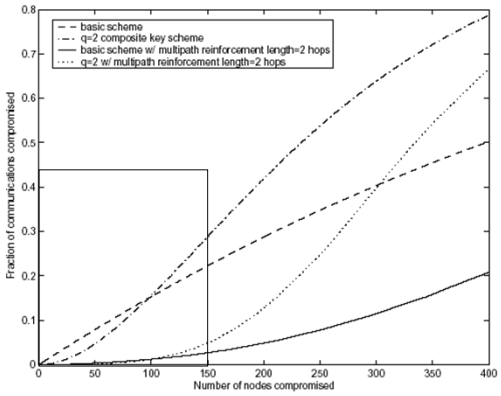
multipath key reinforcement

Multipath key reinforcement

- advantages:
  - in order to compromise a link key, at least one link on each path must be compromised → increased resilience to node capture
- disadvantages:
  - increased overhead
- note:
  - multipath key reinforcement can be used for path key setup too

Multipath scheme: Simulation results

m = 200, p = 0.33



taken from: H. Chan and A. Perrig and D. Song, "Random key predistribution schemes for sensor networks", IEEE Security and Privacy Symp. (Oakland), 2003

Polynomial based key pre-distribution

- let f be a bivariate t-degree polynomial over a finite field GF(q), where q is a large prime number, such that f(x, y) = f(y, x)

$$f(x,y) = \sum_{i,j=0}^{t} a_{ij} x^i y^j$$

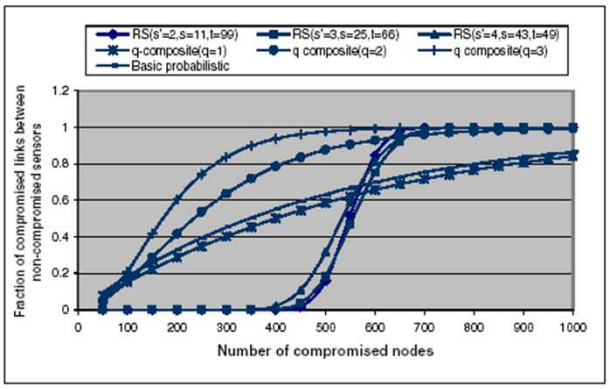
- each node is pre-loaded with a polynomial share f(i, y), where i is the ID of the node
- any two nodes i and j can compute a shared key by
  - i evaluating f(i, y) at point j and obtaining f(i, j), and
  - j evaluating f(j, y) at point i and obtaining f(j, i) = f(i, j)
- this scheme is unconditionally secure and t-collusion resistant
  - any coalition of at most t compromised nodes knows nothing about the shared keys computed by any pair of non-compromised nodes
- any pair of nodes can establish a shared key without communication overhead (if they know each other's ID)
- memory requirement of the nodes is (t +1) log(q)
  → memory limits the level of security achievable

Poly. based random key pre-distribution

- operation:
  - let S be a pool of *s* bivariate *t*-degree polynomials
  - for each node i, we pick a subset of *s'* polynomials from the pool
  - we pre-load into node i the polynomial shares of these *s'* polynomials computed at point i
  - two nodes that have polynomial shares of the same polynomial *f* can establish a shared key *f*(i, j)
  - if two nodes have no common polynomials, they can establish a shared key through a path of intermediaries
- advantage:
  - can tolerate the capture of much more than *t* nodes
    - in order to compromise a polynomial, the adversary needs to obtain *t* + 1 shares of that polynomial
    - it is very unlikely that *t* + 1 randomly captured nodes have all selected the same polynomial from the pool
    - *t* can be smaller, but each node needs to store *s'* polynomials

Comparison with previous schemes

- *m* = s'*(t+1) = 200, *p* = 0.33



taken from D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks", ACM CCS, 2003.

Matrix based key pre-distribution

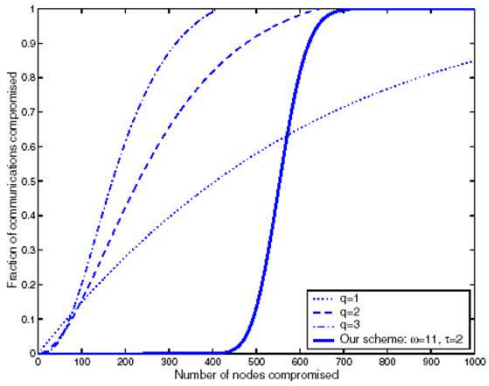
- let G be a (t + 1)×n matrix over a finite field GF(q) (where n is the size of the network)
- let D be a random (t +1)×(t +1) symmetric matrix over GF(q)
- G is public, D is secret
- let $A = (DG)^T$ and $K = AG$
  - K is a symmetric matrix, because
    $K = AG = (DG)^T G = G^T D^T G = G^T D G = G^T A^T = (AG)^T = K^T$
- each node i stores the i-th row of A
- any two nodes i and j can compute a shared key K_{ij}
  - i computes $A(i,.)G(.,j) = K_{ij}$
  - j computes $A(j,.)G(.,i) = K_{ji} = K_{ij}$

Matrix based random key pre-distr.

- G is as before
- $D_1, \ldots, D_k$ are random (t +1)×(t +1) symmetric matrices
- $A_v = (D_v G)^T$ and $\{A_v\}$ is the pool (of spaces)
- for each node i, we pick a random subset of the pool and pre-load in the node the i-th row of the selected matrices (i.e., $A_v(i,.)$ for each selected v)
- if two nodes i and j both selected a common matrix A_v, then they can compute a shared key
- if two nodes don't have a common space, they can setup a key through intermediaries

Simulation results

m = 200, p = 0.33



taken from W. Du and J. Deng and Y. S. Han and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks", ACM CCS, 2003

Summary

- in sensor networks, we need different types of keys
- node keys, cluster keys, and network keys can be established relatively easily using the technique of key pre-loading and using already established link keys
- link keys can be established using a short-term master key or with the techniques of random key pre-distribution