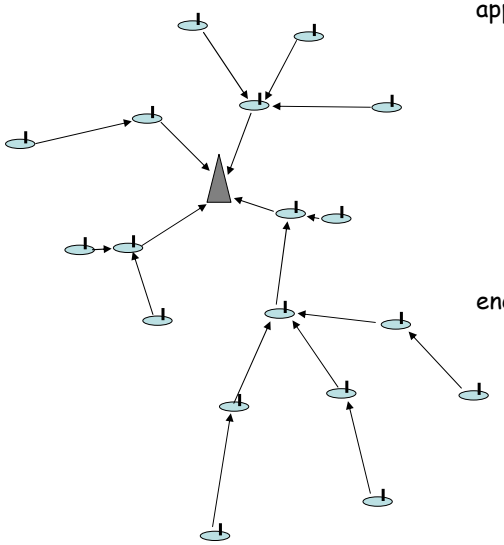
# SPINS:
# Security Protocols for Sensor Networks

a paper by:
A. Perrig, R. Szewczyk, V. Wen, D. Culler, J. D. Tygar
(UC Berkeley)

## Sensor networks



applications:

- environmental monitoring and disaster prevention
- building monitoring and automation
- monitoring the physical conditions of patients (e.g., elderly people)
- battlefield

energy consumption is an issue:

- multi-hop communications
- in-network processing
- specially designed protocols
- sleep mode
- energy harvesting
- …

## 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 |





## Is security on sensors possible at all?

- 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

## Communication architecture

- RF communications → broadcast
  - easy to eavesdrop messages
  - easy to inject fake messages
  - easy to delete messages (jamming)
  - modification of messages on-the-fly is hard
  - but: delete – modify - re-inject may work
- typical communication patterns:
  - many-to-one (nodes to base station) (measurement)
  - one-to-many (base station to all nodes) (control information)
- periodic beacons for neighbor discovery and establishing the routing topology (base station rooted tree)
- nodes can
  - recognize packets addressed to them (addressing)
  - handle broadcast messages
  - forward packets toward the base station (using the routing topology)
- the base station can access individual nodes using source routing, if needed




## Trust setup

- the base station is trusted by all nodes
- sensor nodes are untrusted
  - they are unattended
  - they are not tamper resistant
  - they can be captured and compromised
- RF communication channels are untrusted
- initial keys
  - each node has a unique key that it shares with the base station
  - compromise of this key affects only a single sensor
- time synchronization
  - upper bound on the nodes' clock drift

## Some design guidelines

- base the security architecture on symmetric key cryptography
- reduce memory usage by re-using code
  - all crypto primitives are built out of from a single block cipher
- reduce communication overhead by maintaining and exploiting a common state between communicating parties




## Security requirements

- data confidentiality
  - sensor readings might be sensitive, some control data (e.g. keys) must be kept secret
  - and eavesdropping is easy
- data authentication
  - origin of sensor reading might be important, origin of control data is very important
  - and it is easy to inject fake packets into the network
  - special requirements of broadcast authentication


- symmetric MAC cannot be used
- asymmetric digital signatures are not feasible

## Security requirements (cont'd)

- data integrity
  - integrity of sensor readings and control data is important
  - forwarding sensors may be compromised and modify data
- data freshness
  - freshness of sensor readings is usually important
  - and replay of old packets is easy
  - weak freshness
    - provides partial message ordering, but no delay information
    - useful for sensor readings
  - strong freshness
    - allows delay estimation
    - required by time synchronization




## Building blocks: SNEP

- Sensor Network Encryption Protocol (SNEP):

$$A \rightarrow B : enc_{K_{enc},C}(data) \mid mac_{K_{mac}}(C|enc_{K_{enc},C}(data))$$

where

- $enc_{K_{enc},C}$ is encryption in CTR mode with key $K_{enc}$ and counter $C$
- $mac_{K_{mac}}$ is CBC-MAC computation with key $K_{mac}$
- MAC is computed over the encrypted data and counter $C$
- MAC length is 64 bits
- $K_{enc}$ and $K_{mac}$ is derived from the master key K (shared by the node and the base station) through a one way function:

$$K_{enc} = mac_K(1)$$
$$K_{mac} = mac_K(2)$$

## Properties of SNEP

- semantic security
  - same messages are encrypted differently each time due to the different counter value
- data authentication and integrity by using MAC
- weak freshness and replay protection
  - counter is part of the MAC
  - it ensures message ordering
- low communication overhead
  - counter is not sent, it is maintained locally by both parties
  - using the block cipher in CTR mode results in a stream cipher → encrypted messages has the same length as plain messages
  - MAC adds only 8 bytes per message
- reduced computational overhead
  - MAC verification doesn't need decryption




## SNEP with strong freshness

$A \rightarrow B : N_A$, request

$B \rightarrow A : enc_{K_{enc,C}}(response) \mid mac_{K_{mac}}(N_A|C|enc_{K_{enc,C}}(response))$

where

- the request can use plain SNEP for confidentiality and authentication
- $N_A$ is an unpredictable random number computed as

$$N_A = mac_{K_{rnd}}(S)$$

- after generating a random number, S is incremented by one
- $K_{rnd}$ is a key derived from the master key K (shared by the node and the base station) through a one way function:

$$K_{rnd} = mac_K(3)$$

and regenerated from time to time:

$$K_{rnd}' = mac_K(K_{rnd})$$

## Code re-use in SNEP

- only encryption part of RC5 is implemented
- this is used
  - to encrypt and to decrypt (due to CTR mode) data
  - to implement the MAC function
  - to generate encryption and MAC keys from the master key
  - to generate random numbers

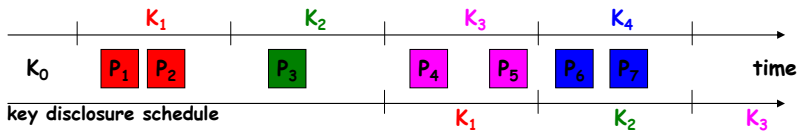



## Building block: micro-TESLA

- main idea: asymmetry through delayed disclosure of authentication keys
  - base station computes a MAC with a key unknown to the sensors
  - base station sends and sensors receive the message with the MAC
  - later, the base station discloses the key used to compute the MAC
  - every sensor can verify the MAC, if it is correct the sensor knows that the message was sent by the base station, because at the time of reception nobody else knew the key
- assumptions:
  - loose time synchronization between the base station and the sensors
  - each sensor knows an upper bound on the maximum synchronization error
  - initial secret between the base station and each sensor to bootstrap the whole mechanism

## Brief overview of the micro-TESLA protocol

- MAC keys are consecutive elements in a one-way key chain:
  - $K_n \rightarrow K_{n-1} \rightarrow \ldots \rightarrow K_0$
  - $K_i = F(K_{i+1})$
- micro-TESLA protocol:
  - setup: $K_0$ is sent to each sensor using the initial secret between the base station and the sensor
  - time is divided into epochs
  - each message sent in epoch i is authenticated with key $K_i$
  - $K_i$ is disclosed in epoch i+d, where d is a system parameter
  - $K_i$ is verified by checking $F(K_i) = K_{i-1}$
- example:



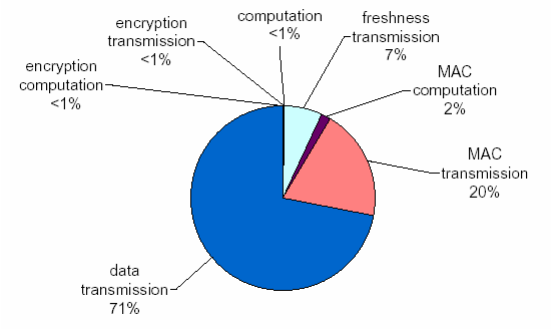




## Implementation (UC Berkeley)

- code size
  - smallest
    - MAC: 480 bytes
    - encryption: 392 bytes
    - key setup: 622 bytes
    - total: 1594 bytes
  - fastest
    - MAC: 596 bytes
    - encryption: 508 bytes
    - key setup: 622 bytes
    - total: 1826 bytes
  - \+ micro-TESLA: 574 bytes
- performance
  - max throughput is 20 packets (30 bytes) per second
  - with 50% idle time

## Energy costs







## Conclusions

- implementing security on sensors is feasible …
- … but needs special attention during design
  - limited CPU → symmetric key primitives
  - limited memory → code re-use
  - limited energy → special protocols to reduce communication overhead (exploit common state rather then send messages)
  - special communication patterns → need for authenticated broadcast
- further investigations
  - other symmetric ciphers (e.g., TEA – Tiny Encryption Algorithm) ?
  - public key crypto ?
  - how to support in-network processing ?
  - …