Link layer security protocols for sensor networks

SPINS (SNEP & micro-TESLA)
TinySec
MiniSec

Motivation for link layer security

- in traditional networks, routers need to see only headers in order to be able to route packets → end-to-end security makes sense
- in sensor networks, it is useful if intermediate nodes are able to access the contents of the packets destined to the sink
  - in-network processing
  - duplicate elimination

  → energy saving
- in addition, if integrity and authenticity can only be verified at the sink, then bogus packets can be injected in the network, which are forwarded uselessly consuming precious energy (and bandwidth) → sort of DoS attack

Security objectives

- message authentication and integrity protection
  - legitimate nodes should be able to detect messages from unauthorized nodes or modified messages, and reject them
- replay protection
  - it should not be possible to replay a legitimate messages sent by node A to node B in the past such that B accepts it again
- confidentiality
  - in some applications, confidentiality may be needed
  - in this case, *semantic security* is preferable
    - encrypted messages do not leak even partial information about the plaintext
    - e.g., sensor readings may take only a few values, and it should not be possible for the attacker to learn which ciphertext belongs to which value (in other words, two encryptions of the same value should be different and indistinguishable from any other encryptions)

Other design requirements

- low energy consumption
  - increase in message length (due to crypto stuff) should be minimized
    - in traditional networks, the difference between 8 and 16 bytes of overhead is negligible
    - in sensor networks, 8 bytes is nearly 25% of the total packet size
    - some level of security may even be sacrificed for the sake of efficiency
  - keeping state may help reducing the communication overhead
    - e.g., instead of sending packet sequence numbers explicitly, one should rely on synchronized counters at both ends of the link
  - symmetric key crypto consumes much less energy than asymmetric key crypto
- resiliency to lost messages
  - relatively high message loss rate should not hinder reliable operation

Trust assumptions

- the base station is usually 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 may have a unique key that it shares with the base station → compromise of this key affects only a single sensor
  - nodes can set up link keys and cluster keys as needed
- time synchronization
  - some solutions requires an upper bound on the nodes' clock drift

SPINS: Sensor Network Encryption Protocol (SNEP)

- high level message format:

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

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 link key K (shared by A and B) through a one way function:

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

Properties of SNEP

- semantic security
  - same message is encrypted differently each time due to the different counter value
- data authentication and integrity by using MAC
- replay protection
  - counter is part of the MAC
  - it ensures message ordering (weak freshness)
- 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 overhead per message
- reduced computational overhead
  - MAC verification doesn't need decryption

Low memory consumption in SNEP

- SNEP uses the RC5 cipher
  - RC5 is a strong cipher
  - it does not use large tables (like DES or AES) → small memory consumption
  - it can be implemented efficiently in software
- only the encryption part of RC5 is implemented, and this is used
  - to encrypt and to decrypt (due to CTR mode) data
  - to implement the MAC function (CBC-MAC)
  - to generate encryption and MAC keys from the master key
  - to generate random numbers (see next slide)

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 link key K through a one way function:

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

and regenerated from time to time:

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

SNEP's problems

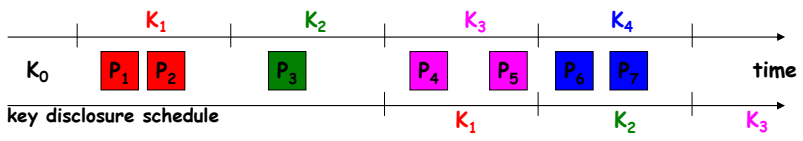
- 8 byte MAC is too large overhead
- counters used for replay protection may desynchronize when a message is lost
  - needs counter re-synchronization which may be expensive
  - easy target for a DoS attacker

SPINS: micro-TESLA

- problem: broadcast authentication (e.g., base station to sensor, sensor to neighbors)
- main idea: asymmetry through delayed disclosure of auth keys
  - the base station computes a MAC with a key unknown to the sensors
  - the base station sends and the sensors receive the message with the MAC
  - the sensors check that the key has not been disclosed
  - later, the base station discloses the key used to compute the MAC
  - every sensor can now 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, and is aware of the key disclosure schedule
  - initial secret between the base station and each sensor to bootstrap the whole mechanism (node key)

micro-TESLA key generation and protocol

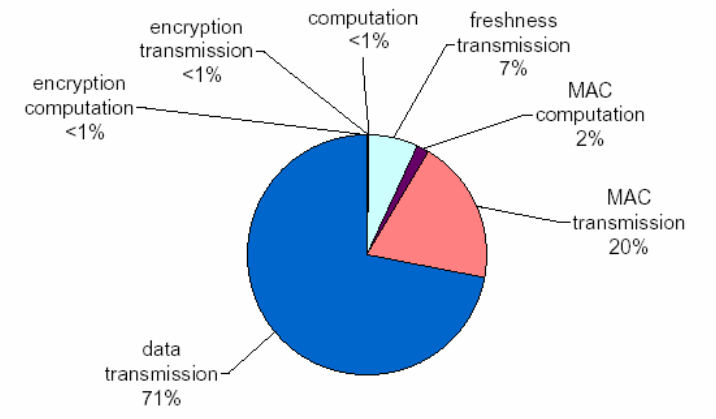
- TESLA keys are consecutive elements in a one-way key chain:
  - $K_n \rightarrow K_{n-1} \rightarrow \ldots \rightarrow K_0$
  - $K_i = F(K_{i+1})$
- protocol:
  - setup: K_0 is sent to each sensor using the initial secret between the base station and the sensor
  - time is divided into epochs
  - 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 $F(K_i) = K_{i-1}$ (or $F^j(K_i) = K_{i-j}$ if some keys were missed)
- example:



Implementation (UC Berkeley)

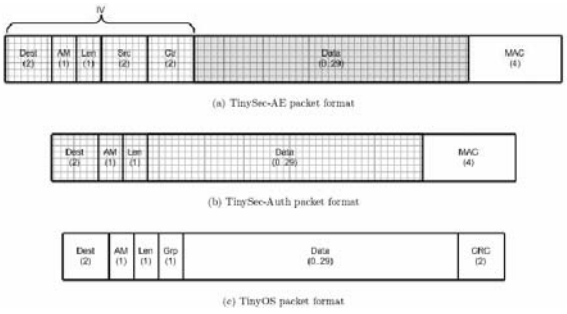
- code size
  - smallest
    - MAC: 480 bytes
    - encryption: 392 bytes
    - key setup: 622 bytes
    - total: 1594 bytes
  - fastest
    - MAC: 596 bytes
    - encryption: 508 bytes
    - key setup: 622 bytes
    - total: 1826 bytes
  + micro-TESLA: 574 bytes
- performance
  - max throughput is 20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

Measured energy costs



TinySec

- services
  - authentication and encryption (TinySec-AE)
  - authentication only (TinySec-Auth)
  - no replay protection
    - maintaining synchronized counters needs memory
    - list of neighbors is not available at the link layer → how much memory should we reserve for the table?
    - isn't the same problem arises for keys? (unless we use a network wide shared key, which is supported by TinySec)
- packet formats

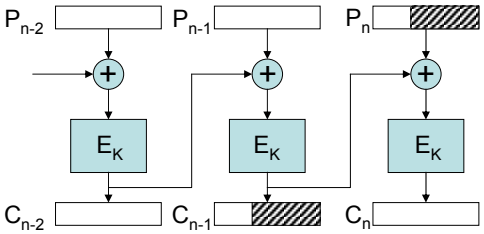


Encryption in TinySec

- CBC encryption
  - stream ciphers have advantages, but they are vulnerable to repeating IVs → long IVs are needed → large overhead
  - CBC is more robust in the presence of repeated IVs → IV can be shorter
    - if two messages P, P' are encrypted with the same IV, the ciphertexts will leak the length of the longest common prefix of P and P', nothing more
  - however, CBC may still leak some information
    - if $C_1 = C_1'$, then the attacker knows that $P_1+IV = P_1' + IV'$, and hence $P_1+P_1' = IV + IV'$ (note that IVs are sent in clear)
    - if IV and IV' are random, then this has a very low probability
    - however, if the IV is a counter this may be a problem
      - IV as a counter is desirable to avoid repeating due to the birthday paradox
    - to prevent this problem, the IV is pre-encrypted (this randomizes the IV before use, even if the IV is a counter)
  - standard CBC increases the message size (due to padding)
    - ciphertext stealing is used

Ciphertext stealing (CTS) in CBC mode

- encryption:
  - $C_i = E_K(P_i + C_{i-1})$ for i = 1..n-1
  - $C_n = E_K(P_n|0^* + C_{n-1})$
  - ciphertext: $C_1 \mid C_2 \mid \ldots \mid C_{n-2} \mid C_n \mid C_{n-1}^{trunc(|Pn|)}$
- decryption:
  - $P_i = D_K(C_i) + C_{i-1}$ for i = 1..n-2
  - $P_n = D_K(C_n)^{trunc(|Pn|)} + C_{n-1}^{trunc(|Pn|)}$
  - $C_{n-1} = D_K(C_n) + P_n|0^*$
  - $P_{n-1} = D_K(C_{n-1}) + C_{n-2}$



Encryption in TinySec (cont'd)

- IV format
  - IV = dst | AM | len | src | ctr
    where
    dst – destination id
    src – source id
    len – length of the payload
    AM – active message handler (sort of application id)
    ctr – 2 byte counter, starting at 0 and incremented after each sent message
- block cipher is Skipjack
  - has similar properties to RC5
  - but RC5 is patented and Skipjack is not

Message authentication

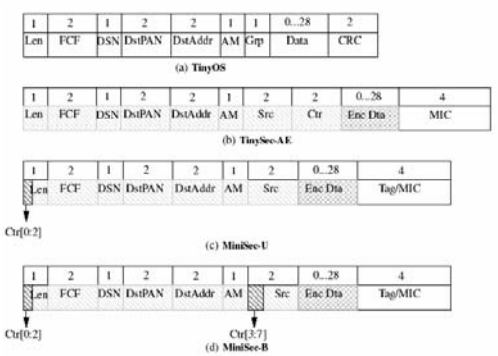
- CBC MAC
  - computed over the (encrypted) payload (if encryption is used) + the header fields
  - simple and efficient
  - uses the same block cipher as encryption (code reuse) but with a different key
  - however, it has known vulnerabilities for variable length messages
    - to address this, encryption of the message length is used as IV
  - MAC is truncated to 4 bytes to reduce overhead

Brief analysis

- MAC
  - blind forgery needs 2^{31} trials on average
  - success of a forgery cannot be determined off-line by the adversary → she needs to interact with the victim
  - given that the channel supports only ~20kb/s, sending 2^{31} packets needs ~20 months
  - constantly receiving messages will deplete the battery of the victim much earlier (in a few weeks)
  - in addition, the saturation of the channel for 20 months can be easily detected
- IV reuse
  - last 4 bytes of the IV: src | ctr
  - src ensures that IVs of different nodes do not collide
  - 2 byte ctr ensures that exactly 2^{16} packets can be sent by each node before IV reuse happens
  - sending rate is very low in sensor networks (e.g., 1 message per hour)
  - one can estimate how long the network can work before IVs are reused and change keys frequently enough
    - IV reuse is a problem only if the same key is used!

MiniSec

- services
  - message authentication and encryption
  - replay protection
- two modes
  - MiniSec-U
    - uses synchronized counters
    - used mainly between communicating pairs (unicast)
  - MiniSec-B
    - uses Bloom filters to keep track of already seen packets
    - used mainly for broadcast communications
- packet formats



MiniSec-U counters

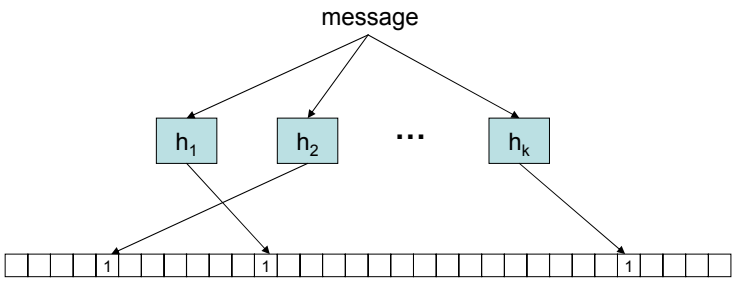
- tries to combine the advantages of SNEP and TinySec
  - SNEP:
    - uses long counters → practically no IV reuse
    - counters are not sent explicitly → low overhead but de-synchronization is possible when a message is lost
  - TinySec:
    - sends counters explicitly
    - in order to minimize overhead, counters are short
  - MiniSec-U:
    - uses long counters → practically no IV reuse
    - hint for currently used counter (last few bits) is sent explicitly

    → low overhead, helps re-synchronization if not too many consecutive packets are lost (and even beyond)
    - example: let's assume that the last x bits of the counter are sent
      - if less than 2^x packets are lost, then receiver can decrypt successfully
      - even if more packets than that are lost, the receiver can continue to increment the counter by 2^x, and reattempt decryption
- implementation:
  - counter length is 32 bits (or longer)
  - hints are 3 bits long
  - they are sent in the MSB part of the length field (as packet length cannot exceed 29 bytes anyway, so 5 bits are enough for the length value)

MiniSec-B design

- replay protection
  - if many nodes are potential senders of broadcast messages, then long implicit counters become inefficient
    - lot of counters to maintain in synchrony and lot of memory
  - short counters can still be used
    - they can be sent explicitly
    - they can be combined with time epochs
      - if counter doesn't repeat in an epoch, then (epoch number, counter) pair is unique (non-repeating)
      - epoch number is known by every node (loose time synch)
  - Bloom filters are used to keep track of already seen packets
    - one filter for the current epoch, and another one for the previous epoch
    - reduced memory requirement
    - false positives (previously unseen packet may be rejected as replay)

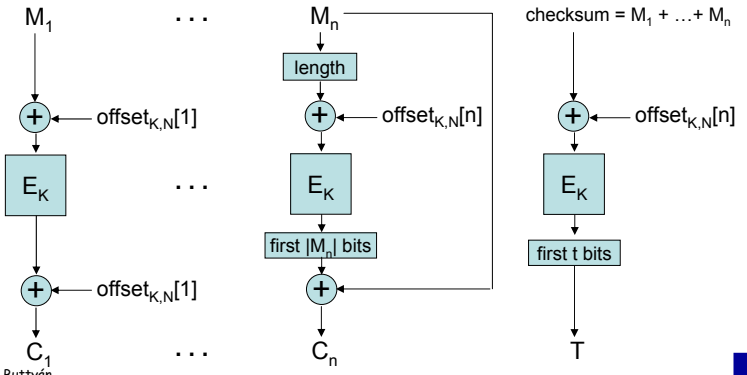
Bloom filter operation principle



- store:
  - message is hashed with k hash functions
  - each hash value is a position in the bit map
  - selected positions turned into 1
- verification:
  - message is hashed with the k hash functions
  - selected positions are checked; if they are all 1s, then replay is signaled

Message authentication and encryption

- OCB (Offset Code Book) mode
  - provides authenticated encryption (in one pass)
  - provably secure
  - very efficient
  - patented but available for free under GPL
- encryption: input: M, K, N (nonce, sort of IV); output: C, T (authentication tag)
- decryption: input: C, T, K, N; output: M or "invalid"



Conclusions

- implementing security on sensors is feasible …
- … but needs special attention during design
  - limited CPU
    - → symmetric key primitives
  - limited memory
    - → code re-use, Bloom filters
  - limited energy
    - → reduce byte overhead (ciphertext stealing in CBC, short MACs, exploit common state)
    - → reduce computation (encryption and authentication in one pass with OCB mode)
  - special communication patterns
    - → need for authenticated broadcast (micro-TESLA)