TinySec: A Link Layer Security Architecture for Wireless Sensor Networks

Chris Karlof, Naveen Sastry, David Wagner

Presented by Paul Ruggieri
Introduction

What is TinySec?
- Link-layer security architecture for wireless sensor networks

Why do we need TinySec?
- Sensor Networks need a way to communicate securely
- Wireless inherently insecure due to it’s broadcast nature
- Existing secure protocols are too bloated for wireless sensor networks
  - Sensor networks have limited computational resources, battery life and communication capabilities
Contributions

TinySec is the first fully-implemented link-layer security protocol for wireless sensor networks
- TinySec is implemented in official TinyOS release

Tradeoffs between performance, transparency and security are discussed
- The authors try to balance this tradeoff for the application (wireless sensor networks)

Bandwidth, latency and power consumption are analyzed for TinySec
- It is feasible to implement this in software

TinySec is a basis for higher level security protocols
Sensor Networks

- Heterogeneous system of sensor with general-purpose computing elements
- Most networks will consist of hundreds or thousands of sensors
- Generally used to collect some information about an environment
Representative Hardware

Mica2

- Several cubic inches
- 8 MHz 8-bit Atmel CPU
- 128 kB instruction memory
- 4 kB RAM (data)
- 512 kB flash memory
- 19.2 kbps radio with a range of ~100 meters
- Operates for ~ 2 weeks at full power
- Run TinyOS
Mica2 Hardware
Security Risks & Threat Models

Broadcast medium
- Adversaries can listen to data, intercept data, inject data and alter transmitted data

What TinySec does
- Guarantee message authenticity, integrity and confidentiality

What TinySec doesn’t protect against
- Resource consumption attacks
- Physical tamper resistance
- Node capture attacks
Link-Layer vs End-to-End

- **End-to-end security**
  - Typical approach in wired networks
  - Packets are encrypted by the sender and decrypted by the receiver
  - Nodes relaying the message don’t decrypt the message, relay as-is
  - Transport layer

- **Link-layer security**
  - Each physical transmission of the packet gets encoded and decoded
  - Data link layer
Why Link-Layer Security?

Sensor networks typically have a many-to-one architecture:
- All sensors transmit their readings to the base station.
- Ideally, duplicate messages (from different sensors) will be dropped.
- Link-layer architecture needed.

Link-layer architecture detects “bad” packets immediately:
- Saves resources.
Design Goals: Security

**Access Control**
- Unauthorized parties should not be able to participate
- Solution: MAC code

**Message Integrity**
- If a message is modified in transit, it needs to be detected
- Solution: MAC code

**Message Confidentiality**
- Information needs to be kept private from unauthorized parties
- Solution: Encryption
Design Goals: Security (Omission)

Replay Protection

- An unauthorized party resends a legitimate packet which it overheard at a later time
- Typical defense: associate counter with each message
- Problem: state needs to be kept for this and we don’t have the resources for this
- Solution: Let a higher level protocol deal with this if it is a problem
Design Goals: Performance

- **Overhead**
  - Increase in message length
    - Decrease throughput
    - Increase latency
    - Increase power consumption
  - Increase in computation (encryption)
    - Increase power consumption

- **8 bytes is ~25% of packet size**
  - Traditional security protocols use 8-16 bytes at least
Design Goals: Ease of Use

- Higher level security protocols will rely on TinySec
- Transparency
  - TinySec should be transparent to the application developer when in use
- Portability
  - TinySec should support different CPU and radio hardware
  - Any necessary porting should be as painless as possible
Security Primitives: MAC

Message Authentication Code (MACs)

- Solution to message authenticity and integrity
- Cryptographically secure CRC
- Sender and Receiver share a private key
- Sender computes MAC over message using private key and includes it in the packet
- Receiver does the same, if MAC computed is different from MAC in the message, receiver rejects the message
Security Primitives: IV

Initialization Vectors (IVs)

- Encryption mechanism
- Side input to encryption algorithm
- Helps to achieve Semantic Security
  - Adversaries should have no better than a 50% chance at guessing any yes/no question about a message
- IV adds variation to Encryption
  - Important when encrypted messages vary little
- IV is publicly included as part of message
- Tradeoff on IV length of overhead vs resource usage
TinySec Design

TinySec-AE
- Authentication & Encryption
- MAC computed over encrypted data and the packet header
- Ensures data received is from a trusted node
- Prevents adversaries from seeing data

TinySec-Auth
- Authentication Only
- Only ensure data received is from a trusted node
- Good when data does not need to be private
TinySec Encryption

Encryption Scheme
- Cipher block chaining (CBC)

IV format
- 8 byte IV
- Want to minimize overhead while getting enough security
- Part of IV is a counter
- More on this later…
Encryption Algorithm Options

Stream ciphers

- Faster than block ciphers (good!)
- If we ever use the same IV, it is highly likely both messages can be decrypted (bad!)
  - We have limited resources to vary the IV
  - Must use a block cipher algorithm

Block ciphers

- Keyed pseudorandom permutation over bit strings
- Operates on blocks of data (message broken up into blocks)
More on Block Ciphers

Good MAC algorithms use block cipher-s

- Two bird with one stone (save code space)

Mode of operation

- Counter (CTR)
  - Similar to stream ciphers – reject
- Cipher block chaining (CBC)
  - Can be made to work with IVs that may repeat
    - XOR encryption of message length with first plaintext block

Examples include:

- DES, AES, RC5, Skipjack
- Skipjack chosen due to licensing issues and practicality of software implementation
Packet Format

**TinyOS Packet Format**
- Dest (2)
- AM (1)
- Len (1)
- Grp (1)
- Data (0..29)
- CRC (2)
- 36 Bytes

**TinySec-Auth Packet Format**
- Dest (2)
- AM (1)
- Len (1)
- Data (0..29)
- MAC (4)
- 37 Bytes

**TinySec-AE Packet Format**
- Dest (2)
- AM (1)
- Len (1)
- Src (2)
- Ctr (2)
- Data (0..29)
- MAC (4)
- 41 Bytes

IV
Packet Format Explained

- Destination, AM and length sent unencrypted
  - Used for early rejection of messages
  - Only data is encrypted (TinySec-AE)
- Take 2 bytes for CRC and put them toward 4 bytes used for MAC (+2 bytes)
  - MAC computed over entire packet (data + header)
- Group field dropped (-1 byte)
  - Differentiates between multiple sensor networks
  - MAC does this for us
- TinySec-AE additional fields (+4 bytes)
  - src – source address
  - ctr – counter
  - These add variability to the IV
Security Analysis

Message Integrity and Authenticity

- Based on MAC length (4 bytes for TinySec)
- 1 in $2^{32}$ chance to guess it
- Adversary must send $2^{32}$ packets to correctly fake a message
- This is not OK for regular networks, given our data rate, this is ok
  - It would take 20 months to send this many packets at 19.2kb/s
  - (What if hardware improves significantly?)
  - (How will TinySec keep up?)
  - (Authors argue that the trend is not in this direction)
Security Analysis

Message Confidentiality

- Security based on IV length, assuming no reuse
  - 8 byte counter or 16 byte random would be sufficient
- However, we have an 8 byte total IV
  - 2 Destination, 1 AM, 1 Length, 2 Source and 2 Counter
  - Try to maximize packets each node can send before global reuse of an IV
- Each node can send $2^{16}$ packets before IV reuse
  - Assume same destination, AM and length
  - At 1 packet per minute -> reuse will not occur for 45 days
    - (Again, what if this changes?)
- IV reuse only problem when using same private key
How do we distribute private keys to trusted nodes?

- Keys preconfigured
- Network-wide
  - 1 key for all nodes in the network
- Per-link
  - Each pair of nodes that communicate share a key
- Per-group
  - Each set of nodes that communicate share a key

(Slightly off topic, but relevant to making the system work)
Implementation

TinySec implemented for TinyOS
- Runs on Mica, Mica2, and Mica2Dot
- 3000 lines of nesC code
- TinySec uses 728 bytes of RAM and 7146 bytes of program space
  - (Good!)
- Needed to modify the TinyOS scheduler to be a priority FIFO queue (2 priorities)
  - Cryptographic operations highest priority
Implementation

- TinySec is independent of the encryption algorithm used
  - Both RC5 and SkipJack implemented
- Upper two bits of length field hijacked to determine mode of operation
- Enabling TinySec is as simple as defining an environment variable in your makefile
- For testing, network-wide key used
Evaluation

Cost of TinySec?

- Larger packets – more overhead
  - Decrease throughput
  - Increase Latency
  - Increase power consumption
  - This occurs regardless of implementation

- Encryption algorithm (MAC and encryption)
  - Increase power consumption
  - This can be vary depending on implementation
    - Hardware support could remove this cost
Evaluation – Cipher Costs

Cipher algorithm

- Affects computation time
- Thus affects power consumption overhead
- Must complete in time for data to be ready for the radio

<table>
<thead>
<tr>
<th>Cipher &amp; Impl.</th>
<th>Time (ms)</th>
<th>Time (byte times)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC5 (C)</td>
<td>0.90</td>
<td>2.2</td>
</tr>
<tr>
<td>Skipjack (C)</td>
<td>0.38</td>
<td>0.9</td>
</tr>
<tr>
<td>RC5 (C, assembly)</td>
<td>0.26</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 3: Time to execute cipher operations on the Mica2 sensor nodes. We display the time both in milliseconds and in byte times.
Figure 2: The power consumption for sending a packet. All packets contained 24 byte payloads. The top graph shows the power consumption when sending the packet with the current TinyOS stack (no security). In the middle graph, we use TinySec-Auth, while the bottom graph uses TinySec-AE. Notice the large power draw at the beginning of sending as the encryption and MAC computation is overlapped with the sending of the start symbol. Additionally, note that when sending with TinySec, the packets are larger in length.
Evaluation – Energy Cost

- Sampled current drawn for sending 24 bytes of data
- Power draw at the beginning is cryptographic operations + sending start symbol
  - TinyOS – 0.00016 mAH
  - TinySec-Auth – 0.000165 mAH (+3%)
  - TinySec-AE – 0.000176 mAH (+10%)
- Most of excess power draw comes from radio use of additional overhead (not CPU draw)
  - Helps support argument that we can do this in software
Figure 3: Bandwidth, plotted as a function of the number of send-receive pairs. We compare TinySec-Auth and TinySec-AE to the bandwidth without using TinySec.
Evaluation - Throughput

- Computed how many packets could be sent in 30 seconds
- Varied the number of senders
  - 24 bytes of data sent
- TinySec-Auth almost identical to TinyOS
- TinySec-AE is ~6% lower with more than 5 senders
- Payloads later varied to equate packet size
  - Shows that added throughput comes totally from additional packet size (Obvious, but good to show)
Figure 4: End-to-end latency in a large system using TinySec. We measured the time to route a message over a number of hops. We used 37 nodes in this experiment. Note that TinySec increases end-to-end latency. As shown in Figure 5, the extra latency can be fully explained by TinySec’s impact on packet sizes: longer packets take longer to transmit and hence increase latency.
Evaluation - Latency

- 36 nodes setup in 6 x 7.75 ft grid
- Network diameter is 10 hops
- Message path:
  - Base station, node, landmark, node, base
  - Minimum hop count = 4
- For each route length, route 200 packets
- Average transmission times per hop count
- Additional latency attributed to additional overhead
Conclusions

TinySec addresses an area which was previously unimplemented
  - Link-layer security for wireless sensor networks

Very easy for developers to take advantage of

Performance and resource hit is not bad
  - TinySec-Auth very close to TinyOS
  - TinySec-AE performs well
Comments

(Good explanation)
(Authors thoroughly evaluated the system)
(System proved to satisfy the authors’ goals)
(Covers a gap in wireless networking)
(Many other projects seem to be using TinySec)
Acknowledgements

www.xbow.com
  - Slide 6 - mica2 picture

http://camars.kaist.ac.kr/~hyoon/courses/cs710_2004_fall/rhoyo.ppt
  - Slide 6 - mica2 picture
  - Slide 20 – Packet format pictures

Original Paper
  - Slide 28 – Table 3
  - Slide 29 – Figure 2
  - Slide 31 – Figure 3
  - Slide 33 – Figure 4