Introduction and Motivation

- Why do we need security in sensor networks?
- Sensor networks have security problems because the transmission medium is open so traditional security schemes do not go well for these sensor networks.
- Sensor devices have limited computational and communication capabilities (overhead, h/w resources, etc)
- We need the security protocols to use minimum amount of RAM and keep a check on the communication bandwidth.
- Should we use software or hardware solutions?

TinyOS – event driven operating system for networked applications in wireless embedded systems with a small memory footprint (400 bytes of data and instruction memory) which supports other hardware platforms as well.

TinySec – first fully implemented link layer security architecture for wireless sensor networks.

Implementation of TinySec has been incorporated in the official TinyOS release.

Why Link-Layer?

- Dominant traffic pattern in sensor networks is many-to-one
- To prune redundant messages, reduce traffic and save energy sensors in-network processing such as aggregation and duplication elimination is used.
- Check messages so that adversaries cannot inject garbage messages into the network and thus reduce the bandwidth and waste precious energy.
- Why end-to-end does not work? DoS attack!

Therefore link layer security mechanism is able to guarantee the authenticity, integrity and confidentiality of messages between nodes while permitting in-network processing.

Design Goals

- Security Goals
  - Access Control and Message Integrity
  - Confidentiality
  - Explicit Omission: Replay protection
  - Eavesdropping and replay the message at later time (replay attack)
  - Common solution – keep counter for every message from every node. (not good for us due to RAM constraints)
  - Application layer is better at handling these attacks as it may expect certain communication patterns and have some information about the network topology.
  - Authors believe this replay protection does not belong to the link layer but rather in higher layers of the protocol stack.

- Performance
  - System with cryptography will induce increased overhead causing decreased message throughput, increased latency and increased power consumption in wireless sensors.
  - Due to resource limitations important to carefully tune the strength of the security mechanism in a way that provides reasonable protection while limiting overhead.

- Ease of Use
  - Security Platform
    - Right set of interfaces for higher level protocols to rely on.
  - Transparency
  - It should be transparent to applications running on TinyOS.
  - Portability
    - TinyOS runs on different platforms and TinySec should fit in.
Security Primitives

- **Message Authentication Code (MAC)**
  - Cryptographically secure checksum of message
  - Sender computes the MAC over the message using the shared secret key, and includes the MAC with the packet.
  - Receiver computes the MAC over the message using the same shared secret key and compares it to the received MAC value.

- **Initialization Vectors (IV)**
  - Use unique initialization vector for achieving semantic security. It is a side input to the encryption process.
  - Main purpose of IV is to add variation of the encryption process when there is little variation in the set of messages.
  - Typically sent in the clear and are included in the same packet with the encrypted data.

Design of TinySec

- **Two different security options**
  - Authentication Encryption (TinySec-AE)
  - TinySec encrypts the data payload and authenticates the packet with a MAC.
  - MAC is computed over encrypted data and packet header.
  - Authentication only (TinySec-Auth)
  - TinySec authenticates the entire packet with a MAC but the data payload is not encrypted.

Design of TinySec -- Encryption

- **IV format**:
  - The structure of the 8 byte IV is dst||(AM)||L||(src||ctr), where dst is the destination address of the receiver, AM is the active message (AM) handler type, L is the length of the data payload, src is the source address of the sender, and ctr is a 16 bit counter.
  - Counter starts at 0 and sender increments it by 1 after a message is sent.

- **Encryption Schemes**:
  - TinySec makes use of the block cipher since this is the best solution in a repeating IV environment and also since the MAC process uses a block cipher.
  - The block cipher is used in CBC (cipher block chaining) mode since it degrades more gracefully in the presence of repeated IVs.
  - Make use of a technique known as ciphertext stealing to ensure the ciphertext has the same length as the underlying plaintext.
  - Default block cipher in TinySec is Skipjack.

Design of TinySec (cont)

- **Message Integrity**
  - To maintain message integrity TinySec makes use of a 4 byte MAC.
  - TinySec always authenticates messages but encryption is optional.
  - TinySec uses a cipher block chaining construction, CBC-MAC for computing and verifying MACs.
  - CBC-MAC is efficient and fast, and the fact that it relies on a block cipher as well minimizes the number of cryptographic primitives we must implement in the limited memory we have available.

Packet Format

- **Security Analysis**
  - **Security of CBC-MAC** is directly related to the length of the MAC.
  - Since the size of the TinySec Mac is 4 bytes adversaries will have to test the validity of the forgery by sending it to an authorized receiver implying about 2 raised to 31 packets before she can succeed at forging the MAC for a single malicious packet.
  - **Confidentiality**
    - Although TinySec uses a 8 byte IV only 4 bytes have been added as overhead. The other 4 bytes are already present in the header.
    - The last 4 bytes (src||ctr) guarantee that each node can send at least 2 raised to 16 packets before repetition of the IV.
    - Since IV reuse is imminent the authors propose a key update protocol.
Keying Mechanisms

- How cryptographic keys are distributed and shared throughout the network.
- TinySec not limited to any particular keying mechanism, any can be used in conjunction with TinySec.
- The different keying mechanisms are shown below:

<table>
<thead>
<tr>
<th>Keying Mechanism</th>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single secret key</td>
<td>Easy way to deploy supports passive participation and low bandwidth</td>
<td>Not robust to node compromise</td>
</tr>
<tr>
<td>Per-neighbor key</td>
<td>Support efficient key distribution protocol for large networks.</td>
<td>Requires key distribution protocol; possible security breaches</td>
</tr>
<tr>
<td>Using keys</td>
<td>Most keys are longer than the maximum packet size.</td>
<td>Requires key distribution protocol; possible security breaches</td>
</tr>
</tbody>
</table>

Implementation

- The authors implemented TinySec in 3000 lines of nesC code (the programming language used for TinyOS).
- The implementation of TinySec requires 728 bytes of RAM and 7146 bytes of program space.
- The default TinyOS radio stack has been modified to redirect byte level radio events to the TinySec module.
- Implemented a two priority scheduler where cryptographic operations run with high priority and all other tasks run at low priority.
- TinySec is cipher independent (both RC5 and Skipjack)
- Upper two bits of the length byte to indicate the protection used (since max payload in TinyOS is 29 bytes)

Evaluation

- Measurements
  - Larger packets cost us by reducing bandwidth, increasing latency, increasing energy consumption
  - TinySec-AE increases latencies by 8.0% while TinySec-Auth increases latencies by 1.5%

<table>
<thead>
<tr>
<th>Application</th>
<th>Packet Overhead (%)</th>
<th>Total Size (bit)</th>
<th>Time to Transmit (ms)</th>
<th>Increase Over Current TinyOS Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current TinyOS Stack</td>
<td>4.0</td>
<td>60</td>
<td>20.2</td>
<td>0.0%</td>
</tr>
<tr>
<td>TinySec-Auth</td>
<td>4.0</td>
<td>44</td>
<td>20.2</td>
<td>8.0%</td>
</tr>
<tr>
<td>TinySec-AE</td>
<td>4.0</td>
<td>60</td>
<td>20.2</td>
<td>7.5%</td>
</tr>
</tbody>
</table>

Evaluation (cont)

- Throughput
  - TinySec-Auth bandwidth characterization is nearly identical to those of the TinyOS stack while TinySec-AE achieved 6% lower throughput.
  - Throughput difference is only due to the differences in packet length and not the computational costs.

- Latency macro benchmark
  - Routing with TinySec-Auth takes longer than with the current TinyOS radio stack; routing with TinySec-AE takes longer than both.

Related Work

- GSM, 802.11 and Bluetooth
  - Gsm frame format was supposed to provide confidentiality (but not integrity) with little overhead
  - 802.11 specified WEP to use RC4 for encryption
  - Bluetooth specification also includes a cryptographic security mechanism which has been proved to be flawed.

- SNEP
  - Specifically target sensor networks but was never fully specified or implemented.

- IEEE 802.15.4
  - Specifies a physical and medium access control layer for low data rate wireless applications.
  - Includes provisions for link layer security
Conclusion

- TinySec addresses security in devices where energy and computation power present significant resource limitations.
- The TinySec implementation is in wide use throughout the sensor network community. Researchers building key exchange protocols on top of TinySec.
- TinySec energy consumption, even when used in the most resource-intensive and most secure mode, is a modest 10%. Using TinySec-Auth, the extra energy consumed is a scant 3%.
- The low impacts on bandwidth and latency prove that software based link layer security is a feasible reality for devices with extreme resource limitations.