Strategies for Secure Data Dissemination in Wireless Sensor Networks

CenSCIR Graduate Student Collaboration Series
01/31/2007
Patrick E. Lanigan
Wireless Sensor Networks

- **Large scale**
  - 100’s - 1000’s of nodes
  - Multi-hop network

- **Resource constrained**
  - Limited energy
  - Limited computational power

- **Deeply embedded**
  - Environmental (low-impact monitoring)
  - Infrastructural (poured into concrete)
  - Industrial (enclosed within pipelines)
Maintenance

- **Software deployment**
  - Fix bugs, add new functionality
  - Update must be deployed to all of the nodes

- **Traditional method: manual, in-situ updating**
  - Code is flashed over a hardwired back-channel
  - Requires a physical connection to the node

- **Often infeasible following deployment!**
  - Scale - Who wants to collect 1000’s of nodes?
  - Access - Can you even get to the nodes?
Network Reprogramming

- Update an entire network of nodes en masse
  - Send application updates over-the-air
  - Epidemic propagation across multi-hop networks

- Existing protocols have mainly focused on...
  - Efficiency
    - Reduce propagation time
    - Limit resource usage
  - Reliability
    - Robust against packet loss
    - Tolerate nodes coming and going
Deluge [1]

- The most widely available network-reprogramming service
  - Distributed with TinyOS
- Full network-reprogramming capabilities
  - Data preparation
  - Multi-hop data dissemination
  - Code installation / execution
- Provides eventual consistency
  - Eventually all nodes will receive every byte of the update
  - Places no bounds on completion time

Deluge: Data Preparation

code update

padding

pages

packets

...
Security Concerns

- **Lack of authentication**
  - No restrictions on the source of an update!
  - Assumes correct operation from all nodes

- **If you can inject packets, you can inject code**

- **Local weaknesses can affect entire network**
  - Data dissemination protocols are epidemic
  - An attacker needs to subvert only a single node
  - Node compromise is a serious concern!
Threat Model

- **Insecure wireless medium**
  - Packets can be injected, modified, corrupted, captured, etc.

- **Base station is trusted**
  - Physically hardened against compromise
  - Associated public / private key-pair

- **Sensor nodes are untrusted**
  - Might exhibit arbitrary behavior
  - Might become compromised
  - Pre-configured with base station’s public key
Sluice
Sluice: Goals

- **Authenticity**
  - Updates can be verified as originating from an authorized source

- **Integrity**
  - Updates cannot be undetectably altered

- **Progressiveness**
  - Data can be verified on-the-fly, as it is received

- **Compatibility**
  - Existing services should not require significant modification

- **Efficiency**
  - Minimize overhead
  - Respect resource constraints
Sluice: Hash-Chain

- Construct a hash-chain over the pages of an update
- Digitally sign the head of the chain
- Embed the hashes and the signature in the pages
Sluice: Informal Evaluation

- **Authenticity & Integrity**
  - Pre-image & 2nd pre-image resistance of the hash function
  - Unforgeability of digital signature

- **Progressiveness**
  - Hash-chain construction allows pages to be verified as they are received

- **Compatibility**
  - Uses default page and packet sizes
  - No need to modify the dissemination protocol

- **Efficiency**
  - Cost of a hash amortized over an entire page
  - Cost of a single signature amortized over an entire update
Sluice: Implementation

- **Cryptographic functions**
  - All crypto libraries are off-the-shelf and unoptimized
    - TinyECC provides ECDSA for sensor nodes [1]
    - BouncyCastle JCE provides ECDSA for the base station [2]
  - Hash chain uses 160-bit SHA1 digests

- ** Modifications to Deluge**
  - Reserved space in pages
    - 20 byte hash in each page (except the last)
    - 40 byte signature in the first page
  - Buffer pages in RAM until verified
  - Host-based Java tools construct hash-chain

Sluice: Spatial Overhead

- **Memory:** ~9.5 Kb ROM, 2 Kb RAM
  - SurgeTelos: 25,640 ROM / 956 RAM
  - SurgeTelos w/ Sluice: 35,224 ROM / 2697 RAM

- **Transmission**
  - Theoretically, Sluice adds $S_o + n \times S_h$ bytes to the update
  - In practice, a multiple the page size
  - Overhead can be masked by using existing padding bytes
  - The *actual* transmission overhead (e.g., message cost) was *not* determined experimentally
Sluice: Evaluation

- **Simulation**
  - TinyOS simulator (TOSSIM)
    - Modified to simulate long-running operations
    - \( N \times N \) grid topologies

- **Testbed**
  - 12 Tmote Sky nodes (TinyOS)
  - Pentium 4 base-station (Linux)
  - USB back-channel (debugging, data collection)
Sluice: Completion Time

- Significant overhead relative to Deluge
  - Relative overhead drops as update size increases
  - Absolute overhead shows effects of pipelining

- Unoptimized cryptographic functions take a long time!
  - 30 - 35 seconds for signature verification
  - 200 ms for hash computation

- Reduce overhead by optimizing cryptographic routines

![Completion time in a 12-node testbed](image)
Alternative Strategies

- **Drawbacks of page-level verification**
  - Can’t distinguish good packets from bad
  - Gives attacker a potential asymmetric advantage

- **Others have proposed packet-level verification [1,2]**
  - Verify authenticity and integrity of *individual packets*
  - Prevents bad packets from consuming node resources

- **Three different proposals**
  - Chain-based [1,2]
    - Similar to Sluice, except chained over data packets
    - Sensitive to out-of-order packet arrival
  - Tree-based [1]
  - Hybrid [1]

Deng et al.: Tree-based

- Data packets placed in leaves
- “Index packets” make up interior nodes
  - Contain hashes of packets in higher levels
- Allows out-of-order packets to be verified
- Has some disadvantages...
  - Memory requirement for large updates
  - Does not account for pages
Deng et al.: Hybrid Scheme

- Construct a hash-tree over individual pages
- Chain root packets of each page
- **Combines advantages of tree-based and chain-based schemes**
  - Single digital signature per-update
  - Tolerant to out-of-order packet arrivals within pages
Comparison: Sluice vs. Deng et al.

- **Difficult to directly compare chain schemes**
  - Packets vs. pages
  - Page chaining should not be affected by out-of-order packets

- **Better to compare Sluice to hybrid scheme**
  - Identical page-chaining mechanisms
  - Hybrid has additional overhead for per-packet hashes
  - Sluice should perform slightly better than hybrid scheme

- **In an attack scenario…**
  - Both schemes limit the scope of damage
  - Deng et al. could allow attacked nodes to function longer
  - Overhead still affects nodes outside of attack range
  - Open question: Cost / benefit tradeoffs?
Moving forward…

- **Explore the tradeoff space**
  - Packet-level vs. page-level verification
  - Digital signatures vs. alternatives (e.g., TESLA)
  - Realistic attack scenarios?
  - Appropriate metrics?

- **Develop more flexible approach**
  - Support different chaining granularities
  - Support different hash sizes
  - Allow tuning according to network characteristics
    - Expected threat model
    - Network topology
  - Provide a model for determining optimal settings
Questions?
Additional Slides
Epidemic Protocols

- Used to disseminate information in distributed systems
  - Each process forwards messages directly to randomly chosen peers [1]
    - Buffer \( b \) messages, forward \( t \) times, to \( f \) recipients.
  - Design constraints: membership, network awareness, buffer management, message filtering

- In wireless sensor networks...
  - “Network protocols that allow rapid dissemination of information from a source, through purely local interactions.” [2]
  - Nodes re-broadcast messages to neighboring nodes on a hop-by-hop basis
  - Most simple form is flooding
    - Nodes always re-broadcast a message the first time it is received
  - More sophisticated techniques use various heuristics to determine when to re-broadcast

Deluge: Multi-Hop Dissemination

- **Three-state protocol**
  - Maintain (MAINT)
    - Nodes advertise version numbers
    - Allows nodes to learn of inconsistencies
  - Request (RX)
    - Nodes request data packets from missing/old pages
  - Transmit (TX)
    - Broadcast requested packets from a given page
    - Data packets are sent in round-robin order

- **Pipelining reduces completion time**
  - Nodes can forward pages before the entire update has been received
  - Pages must be received sequentially
Deluge: Efficiency Mechanisms

- **Message suppression reduces control-traffic overhead**
  - Local rules for identifying redundant messages
  - Takes advantage of broadcast channel

- **Pipelining reduces completion time**
  - Nodes can forward pages before the entire update has been received
  - Pages must be received sequentially

- **Dynamic advertisement-period further reduces unneeded control traffic**
  - Longer period during steady-state operation
  - Shorter period during upgrade
Deng et al.: Simulation

**Message cost...**
- Tree & hybrid schemes improve significantly over chain scheme
- Presumably due to out-of-order message reception / retransmissions / ACKs

**Completion time...**
- Hybrid performs the best
- Chain scheme beats tree scheme
- Chaining has more messages, but *fewer digital signatures* to verify

Graph source: Deng et al. [IPSN 2006]
**Merkle Tree vs. Deng et al.**

**Merkle Tree:** Each node contains a single hash value.

**Deng et al.:** Each node contains multiple hash values.
TinyECC

- Developed at NC State University
- Implements elliptic curve operations over $F_p$
  - Uses many known optimizations
  - Includes inline AVR assembly code
- Supports domain parameters recommended by SECG
- Includes ECDSA functionality
BouncyCastle JCE Provider

- Provides concrete implementations under the JCE / JCA framework
- Includes support for elliptic curve operations (Sun JCE provider does not)
- Available free from www.bouncycastle.org
ECC vs. RSA

- The underlying problem...
  - ECC - discrete logarithms
  - RSA - integer factorization
  - ECDLP is “harder than” integer factorization

- For a given key size, ECC is stronger but slower [2]
  - ECC gives comparable security with smaller keys
  - Smaller keys can lead to a performance advantage
  - Also, reduced memory & transmission overhead

- RSA public key operations can be made very fast
  - For small processors, ECC is still comparable [1]

[1] Gura et al., “Comparing the Performance of ECC and RSA on 8-bit CPUs”