PA197 Secure network design

Message security and key management

Petr Švenda svenda@fi.muni.cz
Faculty of Informatics, Masaryk University

CS
Centre for Research on
Cryptography and Security

Laboratory

- Protection of packets
  - Encrypt and MAC (AES), Speed? Latency? Replay?
- μTesla: Authenticated Broadcast
  - How it works
  - Implement verification on nodes
  - Try to attack protocol
  - Considerations

Message confidentiality

- Encryption/MAC available in radio module vs. software
  - E.g., TelosB' CC2420 radio module has native AES support
  - JeeNode's RF12 radio module has no native encryption support
    - packet needs to be encrypted before passed to rf12_sendNow
    - rf12_encrypt(key): XXTEA, 128bits code, sequence num. up to 30 bits
      - http://jeelabs.org/2010/02/23/secure-transmissions/
- Software encryption with AES algorithm (IS → encryptAES.zip)
  - Task: Measure approximate speed of encryption (ms/block)
  - Note: Implementation optimized for memory, not for speed
- Which mode to use?
  - CBC used often (be aware of trim attack)
  - CTR mode (possibility for precomputation => lower latency)
- Conclusion: even on simple nodes, encryption is possible
  - < 1ms / block

Integrity protection (MAC)

- What algorithm to use for integrity checksum?
  - HMAC-SHA3 is great (use it in ordinary programs), but:
    - Requires additional code (for SHA-3)
    - Requires relatively large internal state (RAM, 64B)
  - If AES is already used (encryption), CBC-MAC can be utilized
    - Use it properly! http://blog.cryptographyengineering.com/2013/02/why-i-hate-cbc-mac.html
- What is proper length of MAC tag?
  - Longer packets => more energy consumed & higher chance of collision
  - 16B? 4B? Birthday paradox? Number of messages?
  - How can attacker misuse two messages with same MAC?
- Task: Estimate latency introduced by encryption and MAC
  - Scenarios: node-to-node enc, end-to-end enc
  - MAC verification: every hop, end node only

Problem of packet replay

- How to deal with packet replay?
  - … in noisy environment with natural packet loss
- MAC-chaining
  - MAC value from previous packet used as IV for next one
  - lost packet => impossible verification of next one
- Incremental counter
  - Possibility to detect and recover even when packet lost
- Think about advantages and disadvantages

µTesla: Authenticated Broadcast

μTesla: Authenticated Broadcast

1. BS: generate new $MACKey_i$ for next $period_i$
2. BS: compute MAC on message M_i to be send
3. Send message M_i via flood (code from last week)
   - Or directly by single hop (strong transmission)
4. Node: receive and store message M_i on nodes
5. BS: After end of $period_i$, flood $MACKey_i$ to nodes
6. Node: Verify validity of $MACKey_i$ against stored *root*
7. Node: Verify MAC on message M_i

Lab work: µTesla (client-only)

1. BS: generate new $MACKey_i$ for next $period_i$
2. BS: compute MAC on message M_i to be send
3. Send message M_i via flood (code from last week)
4. Node: receive and store message M_i on nodes
5. BS: After end of $period_i$, flood $MACKey_i$ to nodes
6. Node: Verify validity of $MACKey_i$ against root
7. Node: Verify MAC on message M_i

- Implement node's reception and verification
  - Display received message and result of verification
  - Notify if something is wrong with MAC

µTesla details

- BS uses: rf12_initialize(1, RF12_868MHZ, 123);
- Format of authenticated broadcast message
  - “msg”[3B]epoch[2B]message[11B]MAC[4B]
- Root for hash chain is:
  - ‘99534f2ec8332239741261aaa803a2f73a018e76’
  - seed → root is 1001xSHA-1 => 1000 epochs possible
- MAC key is broadcasted after 15 seconds
  - Format: “key”[3B]epoch[2B]key[20B]
- MAC computed simply as AES-ECB(msg[16B])
  - First 4 bytes used as MAC value
- Hash chain computation function is pre-prepared
  - computeHashes()

µTesla client – how to start

- Capture message emitted by base station
- Parse content
  - Current (claimed) epoch (use memcpy(&epoch, packet+3, 2))
  - Data message
  - MAC value
  - MAC key (coming later)
- Verify MAC key against root
- Verify MAC (AES-ECB)
- Try to verify proper epochs etc… (considerations)

μTesla client – what to output

- Print on serial port received data message
  - Print parsed values (epoch, data, MAC)
- Print received key message
  - Print parsed values (epoch, MAC key)
- Print result of verification of MAC key against root
  - MAC key is genuine
- Print result of verification of MAC on data message
  - MAC is valid
- Print warning if older than expected epoch is received

Attacks against µTesla

- Try to attack µTesla protocol for others
  - Be BS, other students may have incomplete implementation
  - Change data message to 'evilYourName' [11B] when trying
  - You win if others will accept your messages

1. Replay older intercepted messages
   - When will nodes accept it? What is successful attack?
2. Wait for $MACKey_i$ and create/flood forged message
   - When will nodes accept it? How to prevent timing issues?
3. Impersonate BS and broadcast own messages including MACKey broadcast
   - How to prevent?
   - Pre-distributed root of hash chain inside every node

Considerations of µTesla

- How should base station keep stable epoch size?
  - Timers need to be used (otherwise epochs will drift)
  - (not covered in this labs)
- How should node behave after reboot?
  - How to know current time (=> period)?
    - Real clocks necessary (millis() just return time from start)
  - How to permanently store last epoch seen?
    - https://www.arduino.cc/en/Reference/EEPROM
- Alternative: broadcast warning on incorrect detected epoch
  - Rebooted node will just listen for several epochs
  - If no warning from other (still running) nodes received, then current epoch is synchronized

No homework ☺

- Homework 12 – Attack against routing still running
- Submit before: 29.5. 23:59am (full number of points)
  - Every additional started day (24h) means 1.5 points penalization