Collaborative Processing in Sensor Network

Lecture 7 - Light-weight Security Solutions

Hairong Qi, Associate Professor
Electrical Engineering and Computer Science
University of Tennessee, Knoxville
http://www.eecs.utk.edu/faculty/qi
Email: hqi@utk.edu

Lecture Series at ZheJiang University, Summer 2008
Research Focus - Recap

• Develop **energy-efficient collaborative processing algorithms** with **fault tolerance** in sensor networks
  – Where to perform collaboration **securely**?
    – Computing paradigms
  – Who should participate in the collaboration **securely**?
    – Reactive clustering protocols
    – Sensor selection protocols
  – How to conduct collaboration **securely**?
    – In-network processing
    – Self deployment <---> Coverage
What is Network Security?

- Confidentiality: only sender, intended receiver should “understand” message contents
  - sender encrypts message
  - receiver decrypts message
- Authentication: sender, receiver want to confirm identity of each other
- Message Integrity: sender, receiver want to ensure message not altered (in transit, or afterwards)
- Non-repudiation
- Access Control and Availability: services must be accessible and available to legitimate users (no DoS attacks)
Friends and Foes: Alice, Bob, Trudy

- Well-known fixtures in network security world
- Bob, Alice want to communicate “securely”
- Trudy (intruder) may intercept, delete, add message
What Can the “Enemy” Do?

- A lot!
  - Eavesdrop: intercept messages
  - Actively insert messages into connection
  - Impersonation: can fake (spoof) source address in packet (or any field in packet)
  - Hijacking: “take over” ongoing connection by removing sender or receiver, inserting himself in place
  - Denial of service: prevent service from being used by others (e.g., by overloading resources)
The Language of Cryptography

- Symmetric key crypto: sender, receiver keys identical
- Public-key crypto: encryption key public, decryption key secret (private)
Symmetric Key Cryptography

- All users (e.g., Bob and Alice) share and know the same (symmetric) key: K (e.g., DES)
- Encryption and decryption algorithms are identical

Problem: How can Bob and Alice share the same key in the first place?
Public Key Cryptography

• Radically different approach [Diffie-Hellman76, RSA78]
  – Uncovered an entire new approach to cryptography

• Sender, receiver do not share secret key
• Public encryption key known to all
• Private decryption key known only to receiver
Diffie-Hellman Key Generation

\((X - \text{private key})\)  \quad A  \quad a,p: \text{known numbers}  
\(p - \text{prime number}\)  \quad B  \quad (Y - \text{private key})

\(a^y \mod p\)  \quad a^x \mod p

\([a^y \mod p]^x \mod p = a^{xy} \mod p = [a^x \mod p]^y \mod p\)

- \(x,y,a,p \rightarrow \text{typically 1024 bits long}\)
- The **Discrete Log** problem: by knowing \(a^x \mod p\), \(a\) and \(p\), one cannot obtain \(x\)
Public Key Cryptography

plaintext message, $m$ → encryption algorithm → ciphertext $K_B^+(m)$ → decryption algorithm → plaintext message $m = K_B^-(K_B^+(m))$

Requirements:

1. $K_B^-(K_B^+(m)) = m$

2. Given a public key it should be impossible to compute the private key.
RSA (Rivest-Shamir-Adelman): Choosing Keys

1. Choose two large prime numbers $p$, $q$.
   (e.g., 1024 bits each)

2. Compute $n = pq$, $z = (p-1)(q-1)$

3. Choose $e$ (with $e < n$) that has no common factors with $z$. ($e$, $z$ are “relatively prime”).

4. Choose $d$ such that $ed - 1$ is exactly divisible by $z$.
   (in other words: $ed \mod z = 1$).

5. Public key is $(n, e)$. Private key is $(n, d)$. 

RSA: Encryption, Decryption

Given \((n,e)\) and \((n,d)\) as computed above:

1. To encrypt bit pattern, \(m\) (\(m<n\)), compute
   \[ c = m^e \mod n \] (i.e., remainder when \(m^e\) is divided by \(n\))

2. To decrypt received bit pattern, \(c\), compute
   \[ m = c^d \mod n \] (i.e., remainder when \(c^d\) is divided by \(n\))

\[ m = (m^e \mod n)^d \mod n \]
RSA: Why is That?

Useful number theory result: If $p, q$ prime and $n = pq$, then:

$$x^y \mod n = x \mod (p-1)(q-1) \mod n$$

(Fermat's Little Theorem)

$$(m^e \mod n)^d \mod n = m^{ed} \mod n$$

$C$ - the encrypted message

= $m^{ed} \mod (p-1)(q-1) \mod n$

(using number theory result above)

= $m^1 \mod n$

(since we chose $ed$ to be divisible by $(p-1)(q-1)$ with remainder 1)

= $m$ (since $m < n$)
Authentication

- There is a clear need to “prove” the identity of a sender
- Insufficient options:
  - ID by IP #?
  - Send secret password along with message?
  - Choose a random number, R …

```
“I am Alice”

Bob computes

\[ K_A^+ (K_A^- (R)) = R \]

and knows only Alice could have the private key, that encrypted R such that

\[ K_A^+ (K_A^- (R)) = R \]
```
Man-in-the-middle Attack

• Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)
Certification Authorities

• Question: How do you “prove” that a key is really your key?
• Solutions: Certification authority (CA) - binds public key to particular entity (for example: Bob).
• Bob registers its public key with CA.
  – Bob provides “proof of identity” to CA.
  – CA creates certificate binding Bob to its public key.
  – Certificate containing Bob’s public key digitally signed by CA – CA says “this is Bob’s public key”
Certification Authorities (cont.)

- When Alice wants Bob’s public key:
  - gets Bob’s certificate (Bob or elsewhere).
  - apply CA’s public key to Bob’s certificate, get Bob’s public key
What is an Elliptic Curve?

In $GF(p)$ an ordinary elliptic curve $E$ suitable for elliptic curve cryptography is defined by the set of points $(x; y)$ that satisfy the equation:

\[ y^2 = (x^3 + ax + b) \mod p \]

Using ECC (Elliptic Curve Cryptography), the Discrete-log problem takes the following form:

For a given $P$ and $Q$, where $P = Y \times Q$, there is no available algorithm to recover $y$. 

<table>
<thead>
<tr>
<th>$P$, $Q$: Points on the curve</th>
<th>$Y$: Large scalar (e.g 160)</th>
</tr>
</thead>
</table>
Diffie-Hellman Public Key Distribution Using ECC

- Why use Elliptic Curve Cryptography (ECC)?
- Calculations take less time, less memory and less hardware
- We use 160 bits (instead of 1024 bits used not in EC modulus exponentiation, e.g. DH over a prime) and still retain the same "security strength"

Point-by-scalar multiplication is the core!
Prior Work: Key Pre-distribution Schemes

• Loading keys into sensor nodes prior to deployment
• Two nodes find a common key between them after deployment (a.k.a. “key discovery” phase)
• Possible solutions:
  – **Master key** – one key to all networks
    – (+) Minimal communications (low power consumption)
    – (+) Memory efficient, Key discovery is not really needed
    – (-) However, once key is compromised entire network is compromised
  – **N-1 keys to each node**
    – (+) Key discovery is not really needed
    – (-) Cannot add new nodes!
    – (-) Memory requirements are not practical (non-scalable)
Prior Work: Key Pre-distribution Schemes (cont.)

- Random Key Predistribution
  Each node is provided with a subset of a large key pool
  - (+) Ability to add nodes after deployment
  - (+) Lower network compromise with captured nodes
  - (-) Key discovery is needed

- Fundamental limitations to random key pre-distribution schemes:
  - **Scalability** – the memory, network size
  - **Communication framework** - finding nodes sharing keys
  - **Cryptographic robustness** – inherently offer “statistical” security, which always questionable
Potential Solutions

- How to reduce the amount of point-by-scalar multiplication?
  - Self-certified key generation
  - Fixed key generation (1 multiplication)
  - Ephemeral key generation (2 multiplication)
    - Off-loading 1 multiplication to neighbors
  - Group key generation

- How to mitigate denial-of-service attack?

- How to reduce the complexity of point-by-scalar multiplication?
Reference