Self-Certified Public Key Cryptography for Resource-Constrained Sensor Networks

May 10, 2006

Ortal Arazi, Hairong Qi,
Itamar Elhanany

ECE Department
The University of Tennessee
Knoxville, TN 37996-2100

Benjamin Arazi

CECS Department
University of Louisville
Louisville
KY 40292

Outline

- Background & motivation
- Overview of ECC key generation
- Prior work: security for WSN
- Self-certified public key generation for WSN
  - Two-node self-certified key generation
  - Group key generation
  - Implementation results on the Intel Mote 2
- Summary

Background & motivation

- Wireless sensor network applications are growing
  - Military and civilian
  - Supported by diverse research on entire WSN protocol stack
- Security is expected to play a key role …
  - Confidentiality – nodes need to be able to exchange data "securely"
  - Authentication – nodes should be able to prove their identity to other nodes
  - Message integrity – a node receiving a message should be able to prove it has not been altered

Background & motivation (cont.)

- Public key infrastructure (PKI) is a powerful and proven technology for addressing the three issues mentioned
- However, due to resource limitations in WSN, existing PKI solutions can not be directly applied
  - Low computational capabilities
  - Limited memory space
  - Energy constraints imposed on communications
- Moreover, traffic patterns and topology is unique

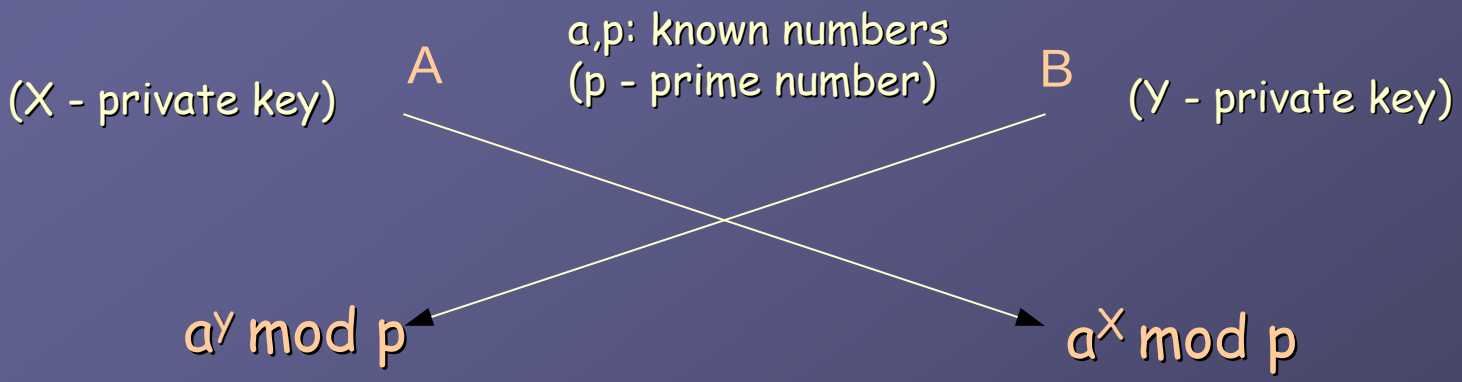
It would be highly desirable to have key generation methodologies that are specifically designed and optimized for ad-hoc clusters of wireless sensor nodes.

Outline

- Background & motivation
- Overview of ECC key generation
- Prior work: security for WSN
- Self-certified public key generation for WSN
  - Two-node self-certified key generation
  - Group key generation
  - Implementation results on the Intel Mote 2
- Summary

Diffie-Hellman Key Generation

- Employing DH for generating a symmetric key between a pair of nodes in the cluster



$$[a^Y \bmod p]^X \bmod p = a^{XY} \bmod p = [a^X \bmod p]^Y \bmod p$$

- x,y,a,p → typically 1024 bits long
- Relies on the *Discreet Log* problem: by knowing $a^x \bmod p$, a and p, one can not obtain x

What is an Elliptic Curve?

In $GF(2^m)$ an ordinary elliptic curve E suitable for elliptic curve cryptography is defined by the set of points (x, y) that satisfy the equation :

$$E : y^2 + x \cdot y = x^3 + ax^2 + b; \quad a, b \in GF(2^m)$$

Example:

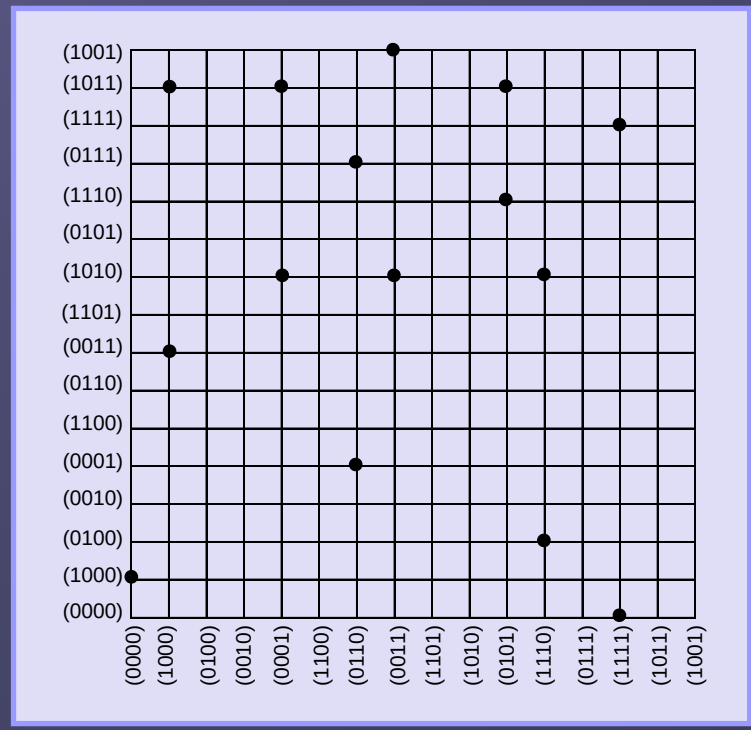
$$y^2 + xy = x^3 + (1100)x^2 + (1000)$$

$$x, y, a, b \in GF(2^4)$$

$$m = 4$$

$$a = 1100$$

$$b = 1000$$

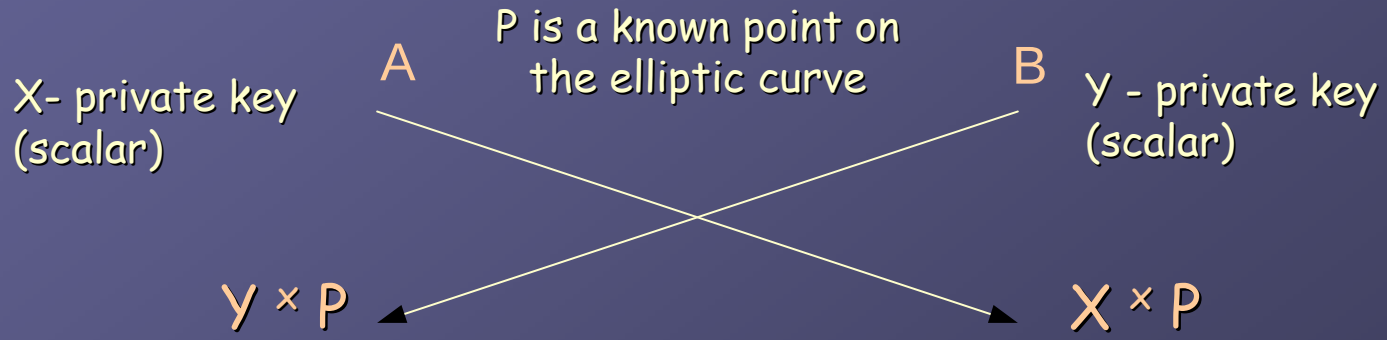


Diffie-Hellman Key Generation using ECC

Why use ECC?

- We use 160 bits (instead of 1024) and still retain the same "security strength"
- We use multiplications instead of exponentiation
- All mathematical calculations are without carry

Calculations take less time, less memory and less hardware



$$(Y \times P) \times X = \underline{XY \times P} = (X \times P) \times Y$$

The Discreet Log problem in ECC: by knowing $X \times P$ and P, one can not obtain x

Outline

- Background & motivation
- Overview of ECC key generation
- Prior work: security for WSN
- Self-certified public key generation for WSN
  - Two-node self-certified key generation
  - Group key generation
  - Implementation results on the Intel Mote 2
- Summary

Prior work: security for WSN

- Pre-distributed keys : scalability and security are compromised
- No self certified techniques for DH using ECC have been published
  - Using ECC: Authentication is always followed by key exchange (key distribution) – not suitable for WSN
  - Using regular mathematical basis self certified techniques for DH have been proposed – not suitable for WSN

Encouraging recent research results (Malan et. al / Harvard) Suggested that ECC scalar-point multiplication is feasible

Prior work: security for WSN

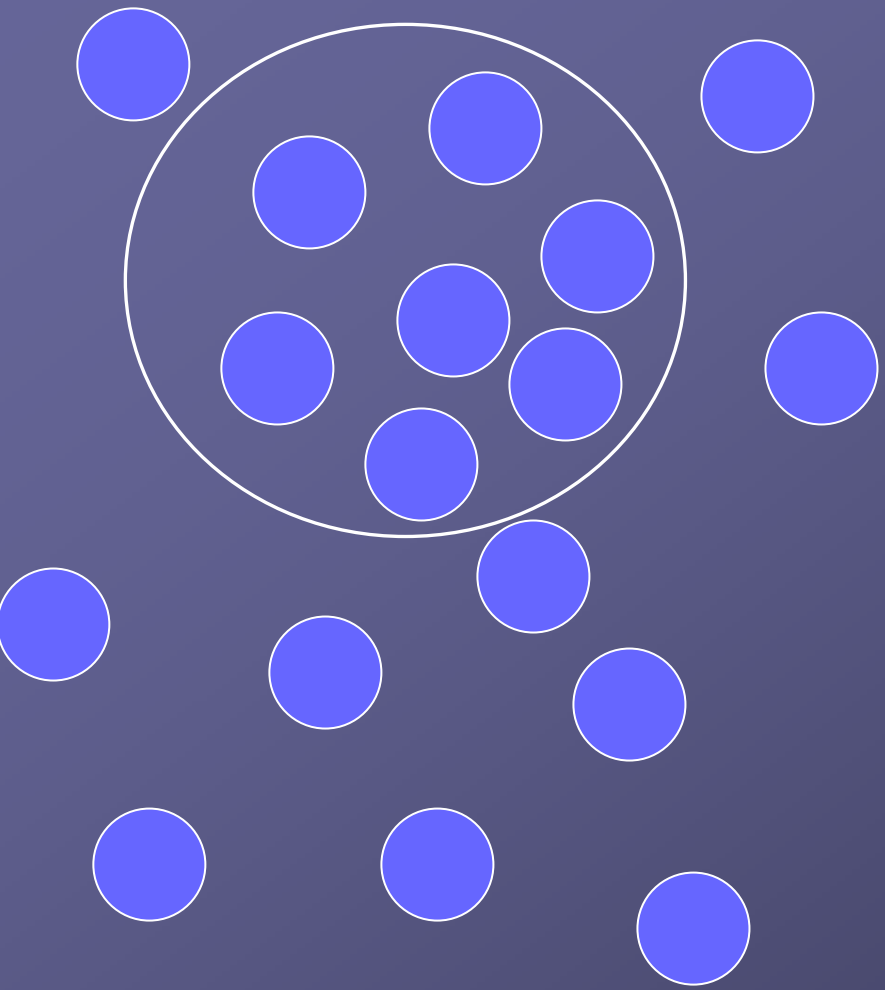
- Main drawbacks of current mechanisms proposed for WSN:
  - Not suitable for implementation of self-certification over Elliptic Curves
  - Treat only fixed-key generation (specific two parties always end up with the same generated secret key) without a clear extension to self-certified ephemeral key generation (more later…)
  - Do not treat self-certified group-key generation

There is a clear need for WSN *group key-generation method* with an efficient integration of *self-certified authentications*

Outline

- Background & motivation
- Overview of ECC key generation
- Prior work: security for WSN
- Self-certified public key generation for WSN
  - Two-node self-certified key generation
  - Group key generation
  - Implementation results on the Intel Mote 2
- Summary

Current research goal

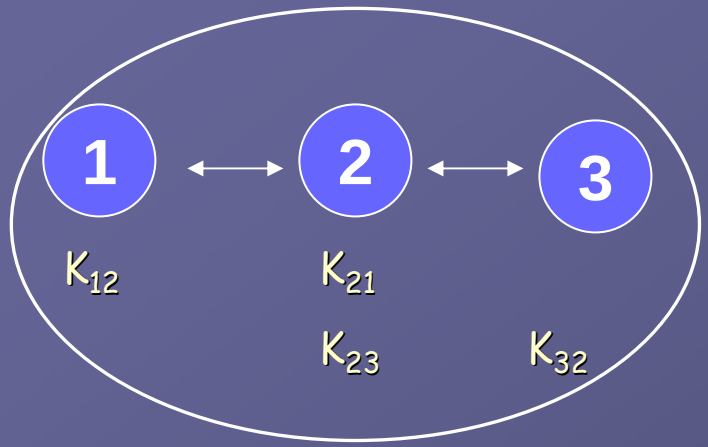


Goal: to establish a self certified group key within a node cluster

- First step
  Initializing symmetric keys for pairs of nodes (using self-certified DH)
- Second step
  Generating the group key

Note: Dynamic cluster, as target is moving

Current research goal (cont.)



First step:

(1,2)- shared self certified key

K_{12} , K_{21}

(2,3)- shared self certified key

K_{23} , K_{32}

Second step:

Generating the group key:

Node #1 generates the group key and via XOR it is transferred to nodes 2 and 3

Node 2 <- K_{Group} XOR K_{12}

K_{Group} XOR K_{12} XOR K_{21} = K_{group}

Node 3 <- K_{Group} XOR K_{23}

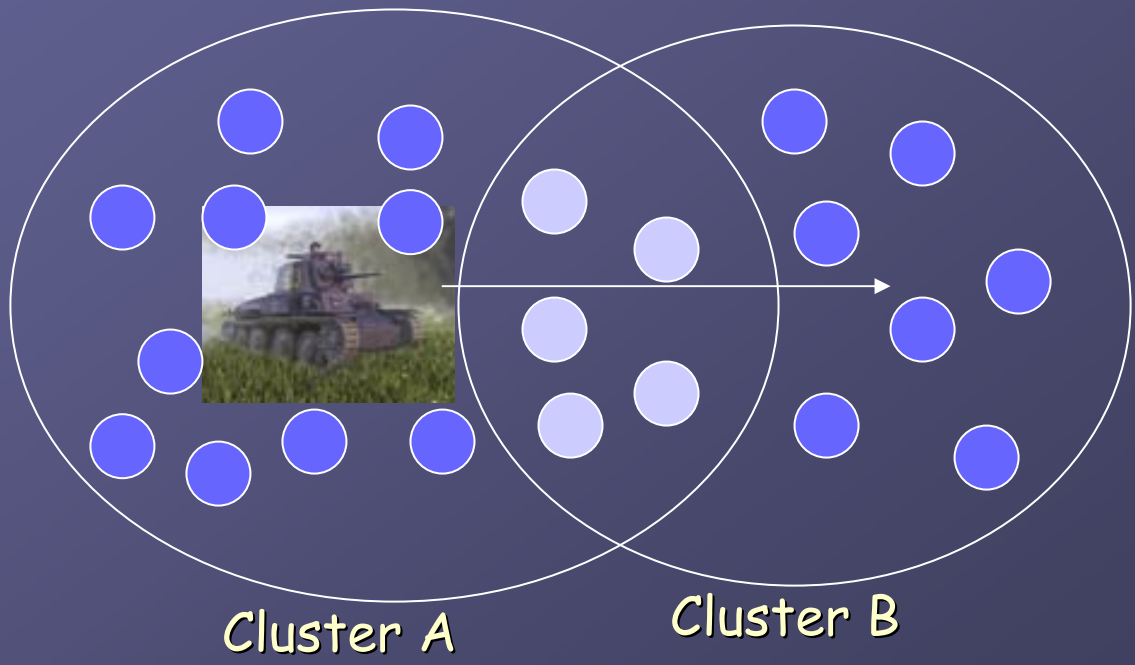
K_{Group} XOR K_{23} XOR K_{32} = K_{group}

Fixed key vs. Ephemeral key

- Fixed key

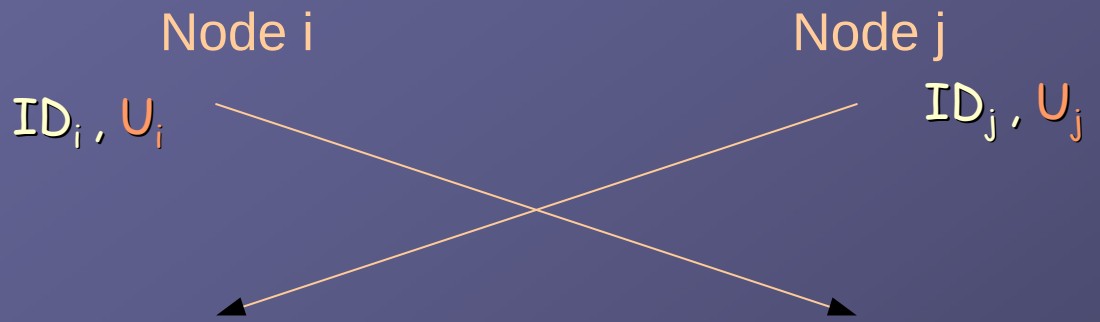
  The private key shared by a pair of nodes is constant
- Ephemeral key

  The private key shared by the same pair of nodes change



Self certified DH key generation: Fixed key

Each node is given by the CA (Certifying authority) a set of public and private keys: (U_v, X_v)



Node i calculates: $X_i[H(ID_j , U_j)_* U_j + R]$ = $X_j[H(ID_i , U_i)_* U_i + R]$:Node j calculates

ID_v: identification of node v - scalar
U_v : node v's public key, generated by the CA - a point on the curve
X_v : node v's private key, generated by the CA - scalar

Self certified DH key generation: Fixed key

mathematical assertions …

As given by the CA:

$U_i = h_i * G$

$x_i = [H(ID_i, U_i)_{,*} h_i + d] \bmod \text{org } G$

$U_j = h_j * G$

$x_j = [H(ID_j, U_j)_{,*} h_j + d] \bmod \text{org } G$

Node i calculates:

$x_i[H(ID_j, U_j)_{,}^{*} U_j + R]$

$= x_i[H(ID_j, U_j)_{,}^{*} h_j * G + d*G]$

$= x_i[H(ID_j, U_j)_{,}^{*} h_j + d] * G$

$= x_i^{*} x_j * G$

Node j calculates:

$x_j[H(ID_i, U_i)_{,}^{*} U_i + R]$

$= x_i[H(ID_i, U_i)_{,}^{*} h_i * G + d*G]$

$= x_i[H(ID_i, U_i)_{,}^{*} h_i + d] * G$

$= x_j^{*} x_i * G$

R : the CA's public key = $d*G$ - a point on the curve

d : the CA's private key - scalar

G : a generating group-point, used by all relevant nodes - a point on the curve

h_v : a random 160 bit number generated by the CA - scalar

Self certified DH key generation: Fixed key

Core contribution …

$$x_i[H(ID_j, U_j)^* U_j + R] = x_i H(ID_j, U_j)^* U_j + x_i R$$

scalar

Dynamic multiplication

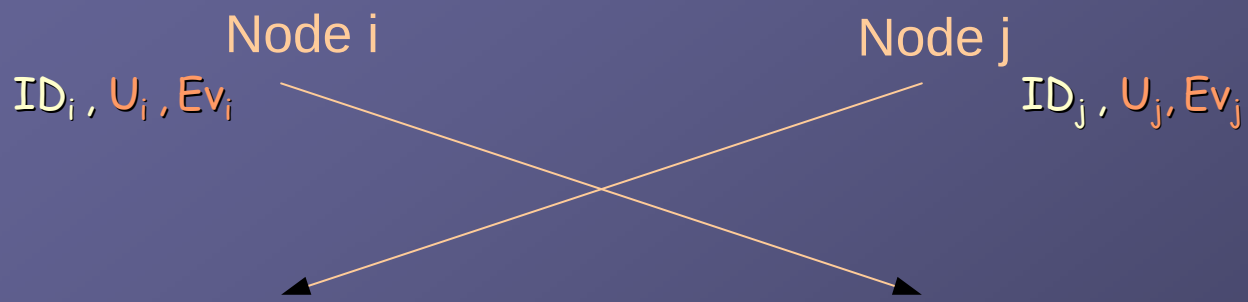
Off line multiplication

2 multiplications of a scalar by a point on the elliptic curve

Until now self-certified DH key generation was done with 3 dynamic multiplications

Self certified DH key generation: Ephemeral key

Each node is given by the CA (Certifying authority) a set of public and private keys: (U_v, X_v)



Node i calculates:

$$Pv_i[H(ID_j , U_j)_{,*} U_j + R] + (X_i + Pv_i) Ev_j$$

=

Node j calculates:

$$Pv_j[H(ID_i, U_i)_{,*} U_i + R] + (X_j + Pv_j) Ev_i$$

ID_v: identification of node v - scalar
U_v : node v's public key, generated by the CA - a point on the curve
X_v : node v's private key, generated by the CA - scalar
Pv_v : a random 160 bit number generated by node v - scalar
$Ev_v = Pv_v * G$

Self certified DH key generation: Ephemeral key

Mathematical assertions …

As given by the CA:

$U_i = Pv_i * G + h_i * G = (Pv_i + h_i) * G$

$x_i = [H(ID_i, U_i)_* h_i + d] \bmod \text{org } G$

$U_j = Pv_j * G + h_j * G = (Pv_j + h_j) * G$

$x_j = [H(ID_j, U_j)_* h_j + d] \bmod \text{org } G$

Node i calculates:

$Pv_i[H(ID_j , U_j)_* U_j + R]+ (x_i + Pv_i) Ev_j$

$x_j * G$

$= Pv_i * x_j * G + x_i * Pv_j * G + Pv_i * Pv_j * G$

Ev_j Ev_j

Node j calculates:

$Pv_j[H(ID_i , U_i)_* U_i + R]+ (x_j + Pv_j) Ev_i$

$x_i * G$

$= Pv_j * x_i * G + x_j * Pv_i * G + Pv_j * Pv_i * G$

Ev_i Ev_i

R : the CA's public key = d*G - a point on the curve
d : the CA's private key - scalar
G : a generating group-point, used by all relevant nodes - a point on the curve
h_v : a random 160 bit number generated by the CA - scalar

Self certified DH key generation: Ephemeral key

What is the main contribution?

$$Pv_i[H(ID_j, U_j)^* U_j + R] + (X_i + Pv_i)\, Ev_j = Pv_i^* H(ID_j, U_j)^* U_j + (X_i + Pv_i)\, Ev_j + Pv_i^* R$$

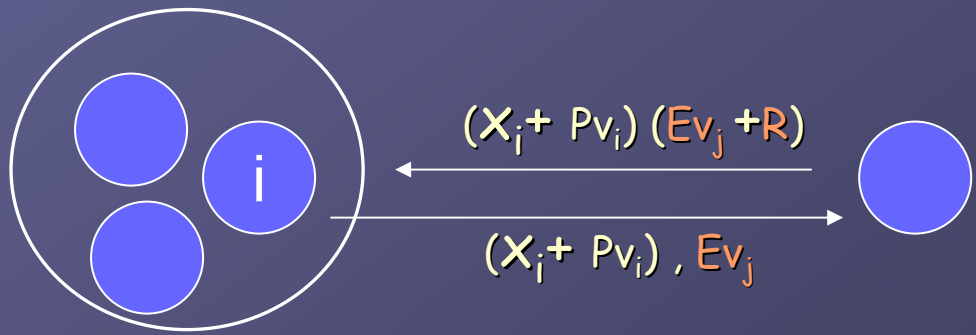
$$= Pv_i^* H(ID_j, U_j)^* U_j + (X_i + Pv_i)(Ev_j + R) - X_i^* R$$

Dynamic multiplication

Calculate d by A neighbor

Off line multiplication

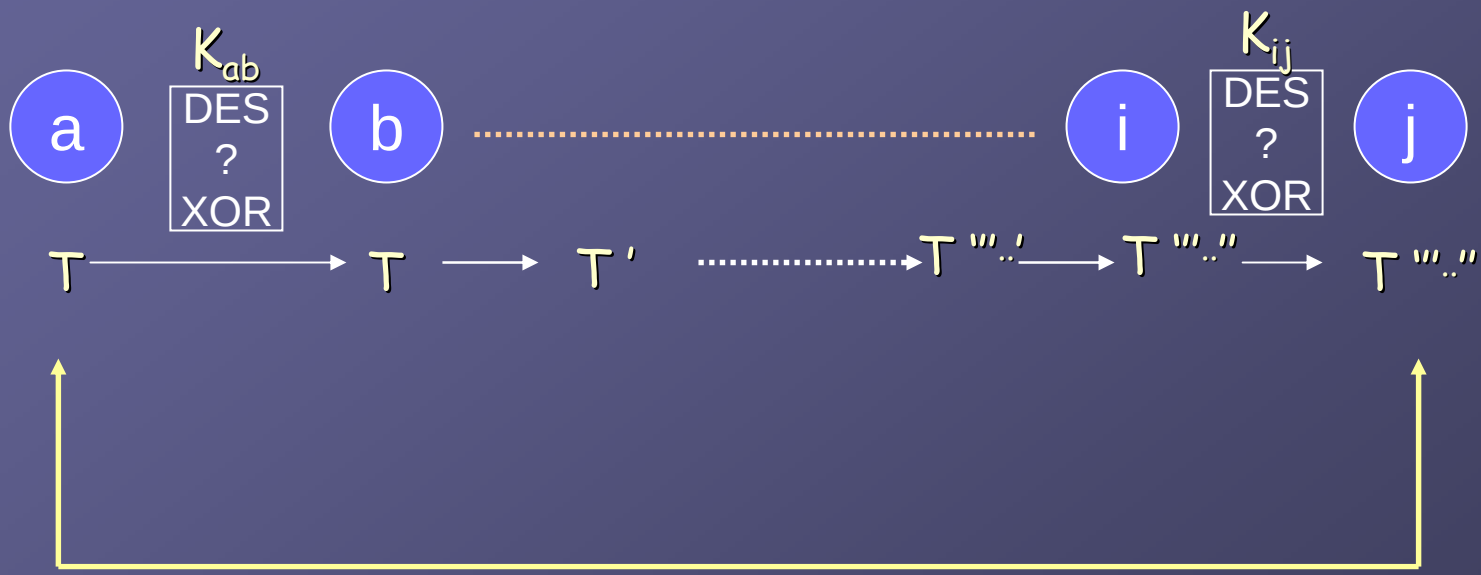
3 multiplications : one preformed dynamically by the node
the second preformed offline by the node
the third calculate by a neighbor node not in the cluster



Group key authentication and generation

K_{ij}- shared key of nodes i and j

T - Public group key needed to generate in the cluster



If T= T '''..'' then:

1) The system is Authenticated
2) we have a group key

Intel Mote 2 sensor network platform

- **Electronic**
  - 320/416/520MHz PXA271 XScale Processor (Dynamic voltage scaling)
  - Programming in NeSC
  - 32MB Flash on-board
  - 32MB SDRAM on-board
  - Mini-USB Client (slave), multiplexed with RS232 console over USB, power
  - I-Mote2 Basic Sensor connector (31+ 21 pin connector)
  - Zigbee [802.15.4] Radio (ChipCon CC2420)
  - Tri-color status LED; Power LED; battery charger LED, console LED
  - Switches: on/off slider, Hard reset, Soft reset, User programmable switch
- **Mechanical**
  - Size: 1.89inches x 1.42in. PCB Thickness 0.069in
  - Size: 48mm x 36mm. PCB Thickness 1.75mm

Intel Mote 2 (cont.)

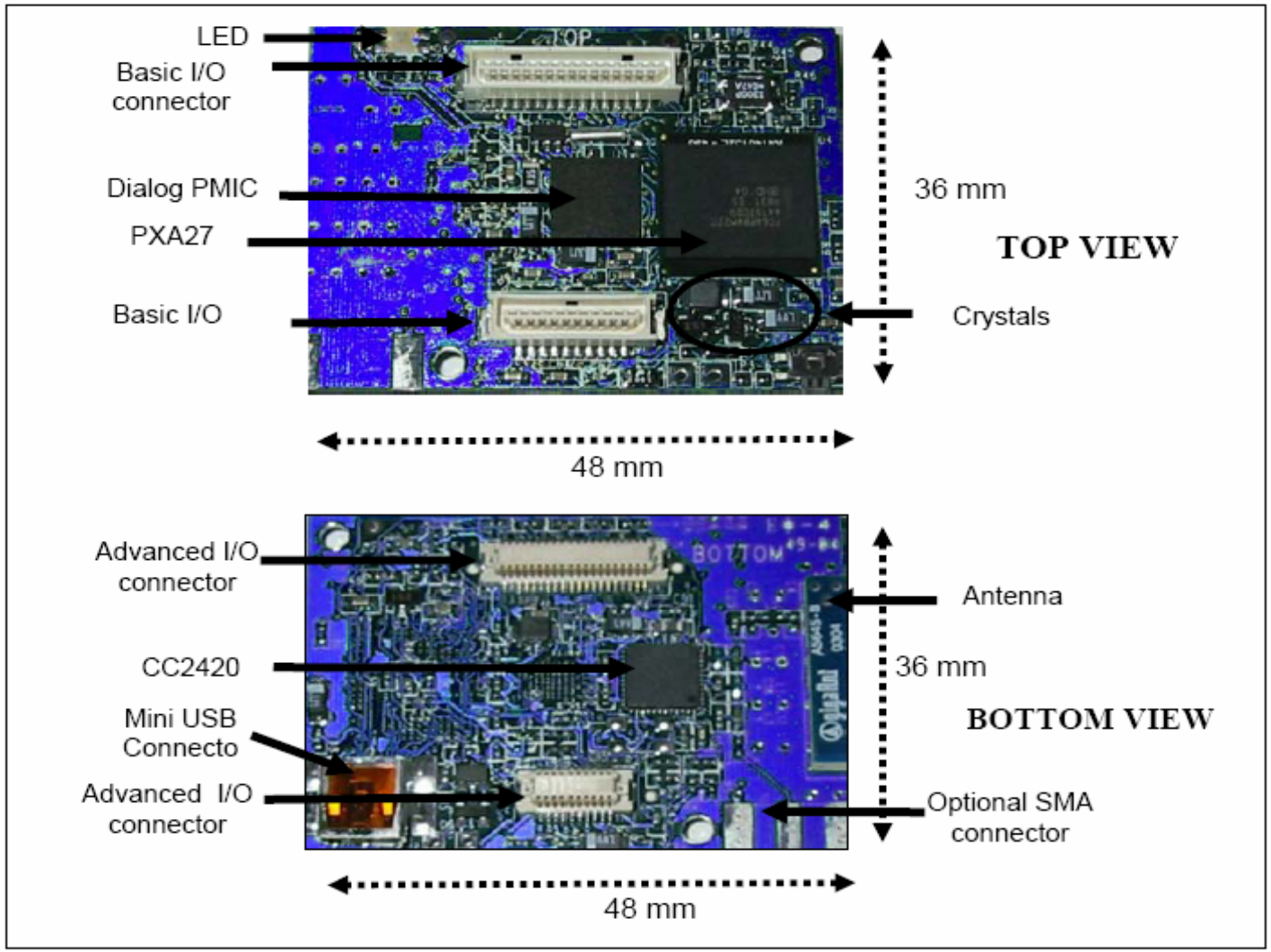


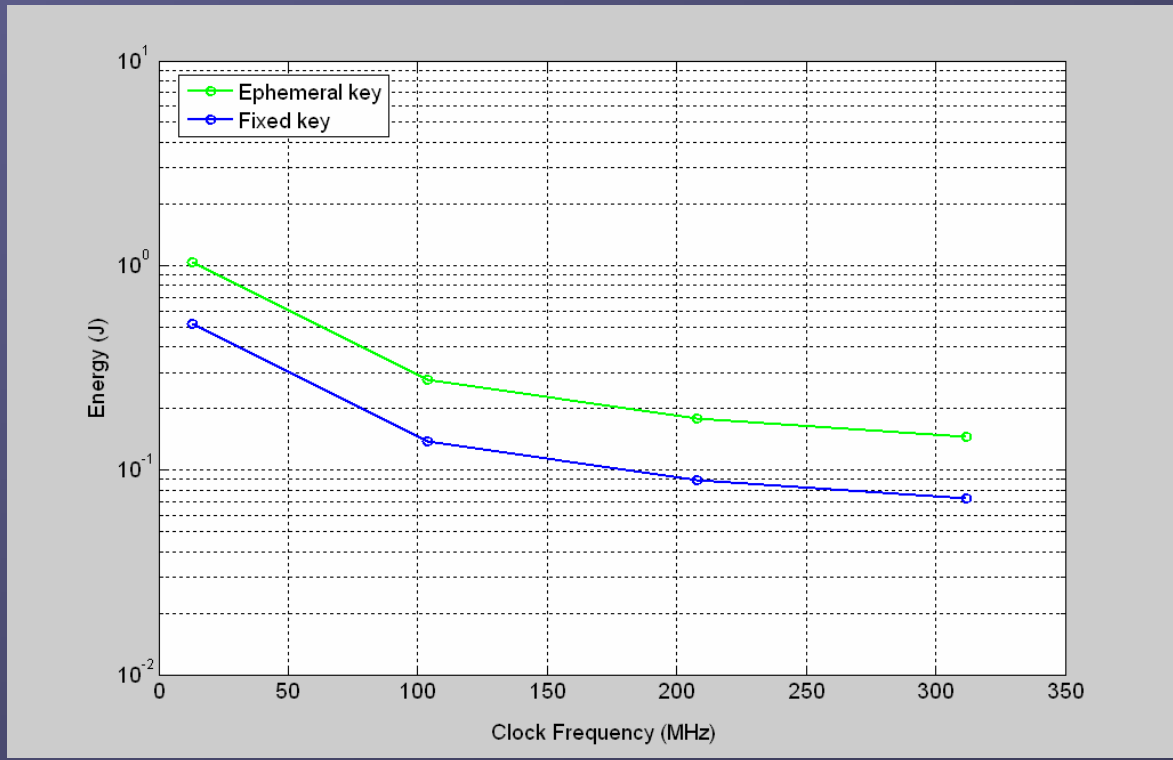
Figure 2.1: Intel® Mote 2 main board

Self-Certified Key Generation on Intel Mote^2

- Code for ephemeral-key generation written in NesC
  - 163-bit keys
  - 16-bit implementation
- Interoperability with TelosB platform (UC Berkeley)
- Dynamic reconfiguration of the CPU frequency
  - Higher frequency for core computations
  - Lower frequency for all other tasks

Voltage	Radio Off at 13MHz (mA)	Radio On at 13MHz
3.8	26.5	46.2
4.4	24.1	44.2
5	23.5	45.4
Datasheet	31	44

Clockrate (MHz)	Power (W)	Time PbS (sec)	Energy (J)
13	0.106	4.9	1.0388
104	0.231	0.6	0.2772
208	0.296	0.3	0.1776
312	0.363	0.2	0.1452



Outline

- Background & motivation
- Overview of ECC key generation
- Prior work: security for WSN
- Self-certified public key generation for WSN
  - Two-node self-certified key generation
  - Group key generation
  - Implementation results on the Intel Mote 2
- Summary

Conclusions

- Group key generation is possible despite the harsh resource constraints
  - Intel Mote 2 based results are encouraging
- Proposed scheme allows for additional nodes to be added ad-hoc
- Group key generation yields a self certified cluster
  - All of the nodes within a cluster are authenticated
- Mathematical computations can be significantly reduced
- Off loading the computations using neighbors not in a cluster offers additional improvement
- Self certified key distribution using DH in ECC is possible
  - While exchanging shared keys, the pair of nodes also authenticate themselves

Questions ?