Toward PKI for Sensor Networks

8 November 2004

David J. Malan
Division of Engineering and Applied Sciences
Harvard University

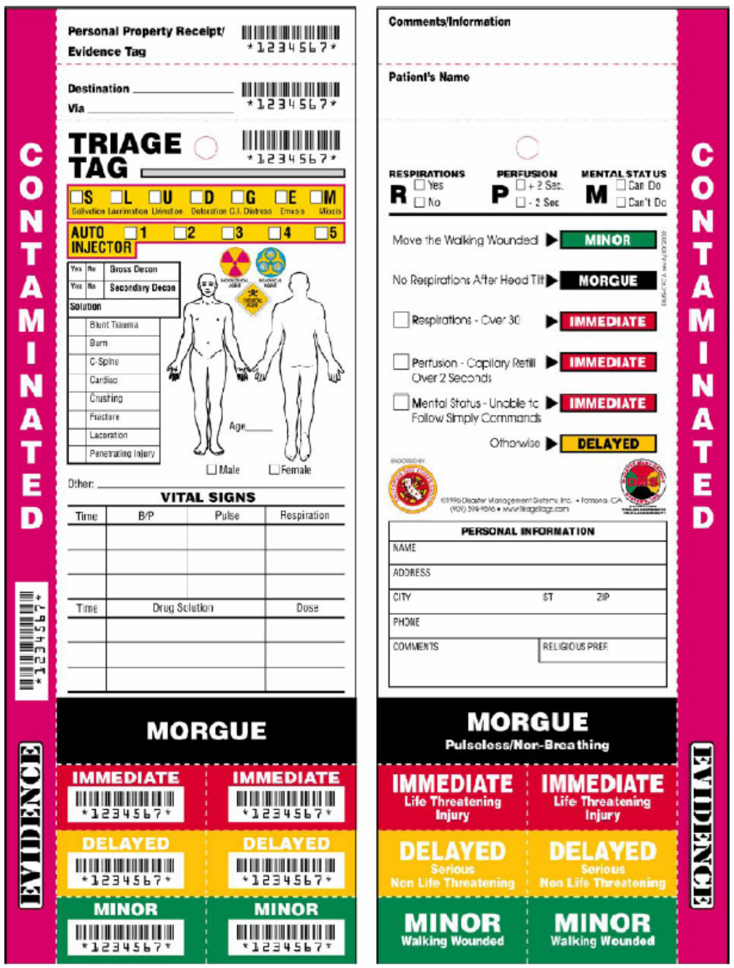
`malan@eecs.harvard.edu`
`http://www.eecs.harvard.edu/~malan/`
+1-617-523-0925

CodeBlue

Mass casualty incidents (MCIs) involve multiple patients

CodeBlue

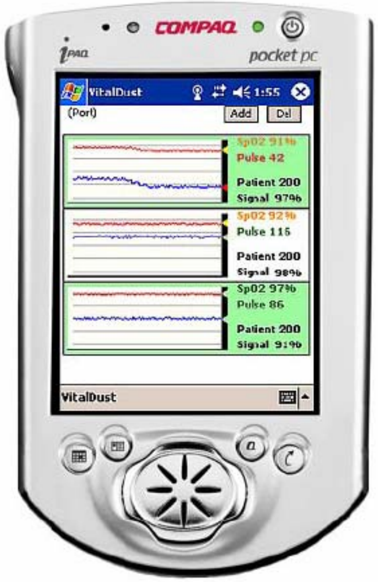
Current systems for monitoring patients are paper-, phone-, and radio-based

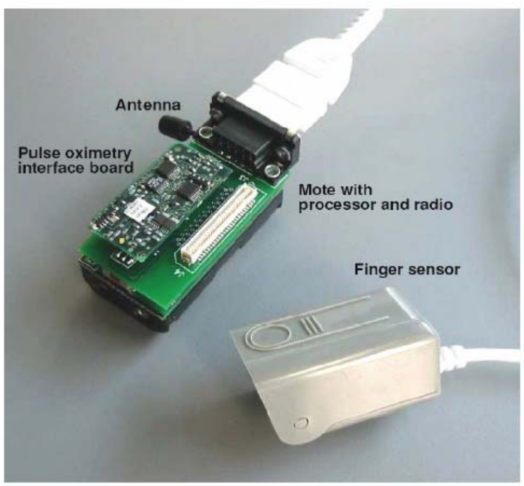


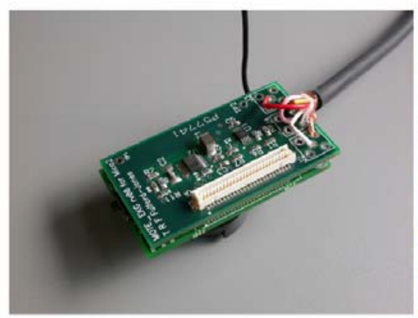
CodeBlue

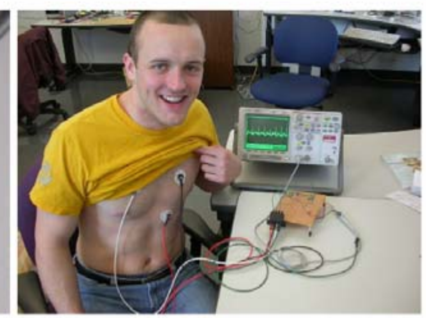
An *Ad Hoc* Sensor Network Infrastructure for Emergency Medical Care

- Sensor nets (*e.g.*, mote-based pulse oximeters, EKGs) have potential for large impact
  - Real-time, continuous monitoring
  - Immediate alerts of changes in patient status
  - Electronic triage tags to store patient data
  - Relay data to hospital, correlate with patient records
- Transmission of patient data must be secure (HIPAA)



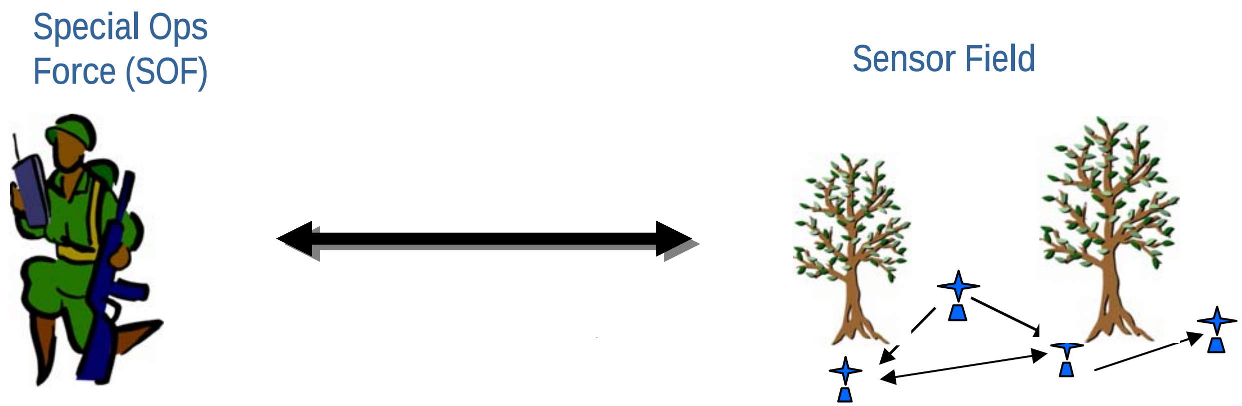






Secure Comm Problem

This slide should look familiar!



External parties need access to sensor data

- Assume a SOF needs access to a TinySec Sensor Field
- Not practical to preplace secret keys in all potential data users
- Need protocols that allow:
  - Motes to authenticate the SOF
  - Secure communication between SOF and Sensor Field
  - SOF to authenticate the Sensor Field

Slide adapted from `http://www.is.bbn.com/projects/lws-nest/bbn_nest_dec_03.ppt`.

BBN TECHNOLOGIES
A Verizon Company

Ensuring Privacy and Security

Secret-Key Cryptography

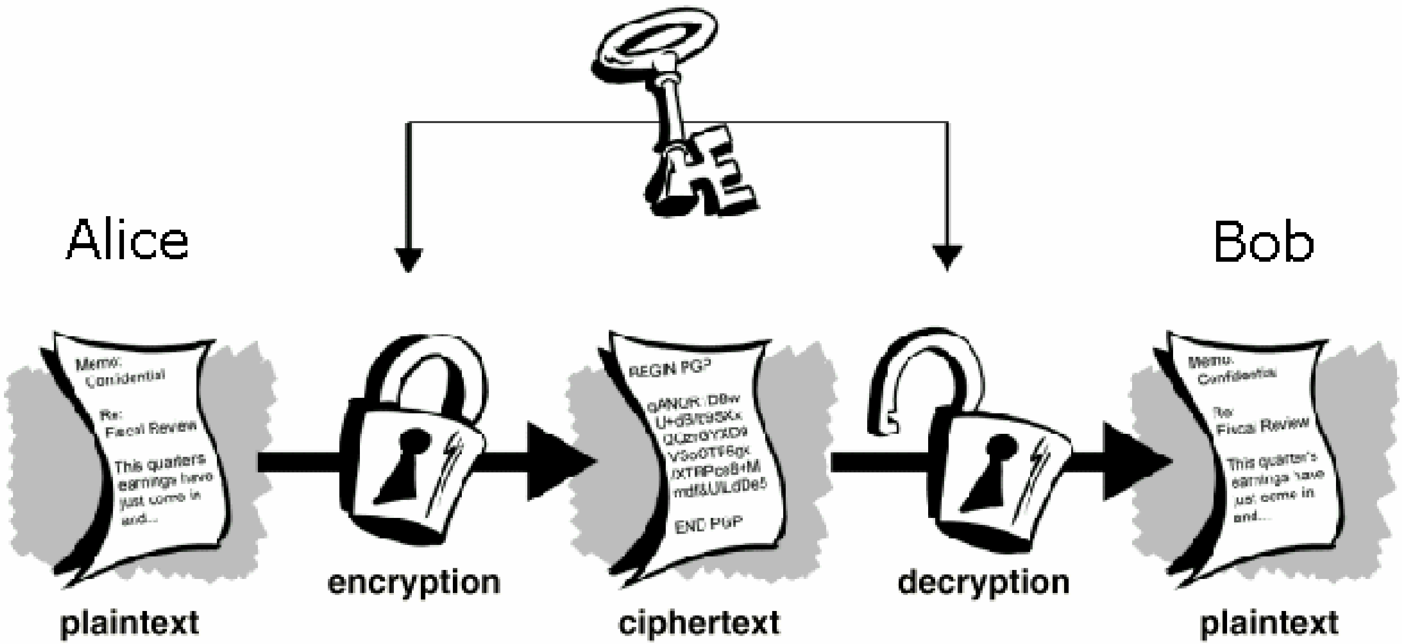
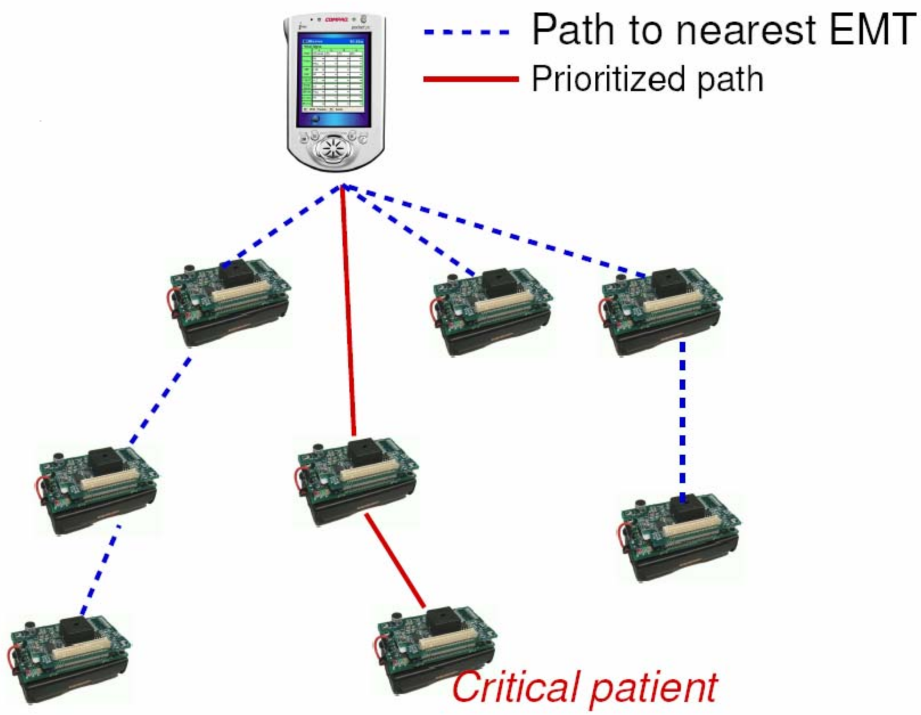


Image adapted from http://www.nuitari.de/crypto.html.

Ensuring Privacy and Security

The Problem: Key Distribution

- How to establish shared secrets among authorized nodes?
  - Network participants are coming and going
  - Motes themselves are vulnerable to theft



Ensuring Privacy and Security

A Solution: Public Key Infrastructure (PKI)

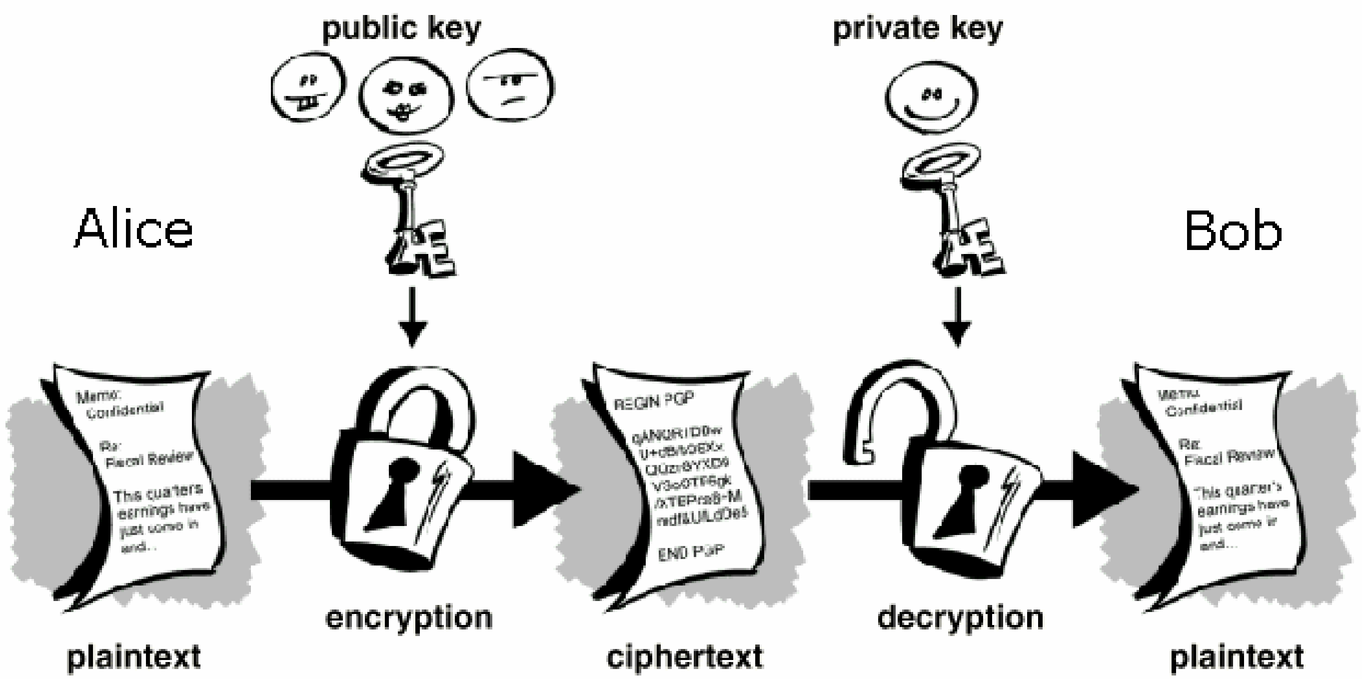


Image adapted from `http://www.nuitari.de/crypto.html`.

Ensuring Privacy and Security

The Real Problem: Implementation of PKI on Motes

- "Public key cryptography is prohibitively expensive for sensor networks in terms of computation and energy consumption."
  - C. Karlof, N. Sastry, and D. Wagner, "TinySec: Link Layer Security for Tiny Devices," `http://www.cs.berkeley.edu/~nks/tinysec/`
- "Many current sensor devices have limited computational power, making public-key cryptographic primitives too expensive in terms of system overhead."
  - A. Perrig, J. Stankovic, and D. Wagner, "Security in Wireless Sensor Networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53–57, June 2004

Ensuring Privacy and Security

Context for This Problem

- The MICA2 Mote
  - Developed at UC Berkeley
  - Fabricated by Crossbow, Inc.
  - Supported by TinyOS, UC Berkeley's open-source operating system for sensor networks
  - Limited Resources
    - 8-bit, 7.3828-MHz ATmega 128L processor
    - 4 KB of primary memory (SRAM)
    - 128 KB of program space (ROM)
    - 512 KB of EEPROM
    - 433-MHz radio: 38.4K baud rate, 29B per-packet payload

Image exerpted from `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-05_A_MICA2.pdf`.

Our Goals

Is PKI, in fact, viable for key distribution on the MICA2?

1. Analysis of UC Berkeley's TinySec, TinyOS's existing secret-key infrastructure for the MICA2 based on SKIPJACK with 80-bit keys
2. Evaluation of BBN Technologies' (*i.e.*, Jennifer Mulligan's) implementation of Diffie-Hellman with 1,024-bit keys (*i.e.*, 160-bit exponents and 1,024-bit moduli), per NIST's recommendation
3. Evaluation of our own implementations of elliptic curve cryptography (ECC) with 163-bit keys, per NIST's recommendation

TinyPK Results on Original MICA (slow!!)

This slide should look familiar too!

- Timing tests for RSA, public key operation with exponent 3 and varying modulus sizes:

512-bit	3.7 s
768-bit	8.0 s
1024-bit	14.5 s

- Private key timing not available: far too slow.
- Key-exchange: Exponentiation with a 512-bit modulus and 120 bit exponent with 50% one-bits: 315 sec!!

Slide adapted from `http://www.is.bbn.com/projects/lws-nest/bbn_nest_dec_03.ppt`.

BBN TECHNOLOGIES
A Verizon Company

Spoiler Ahead!

PKI is, in fact, viable for key distribution on the MICA2

1. SKIPJACK with 80-bit keys
   - Fast
   - Negligible impact on radio throughput
2. Diffie-Hellman with 1,024-bit keys
   - Relatively slow
   - Key sizes unappealing
3. ECC with 163-bit keys
   - Promising performance
   - Key sizes appealing

SKIPJACK and the MICA2

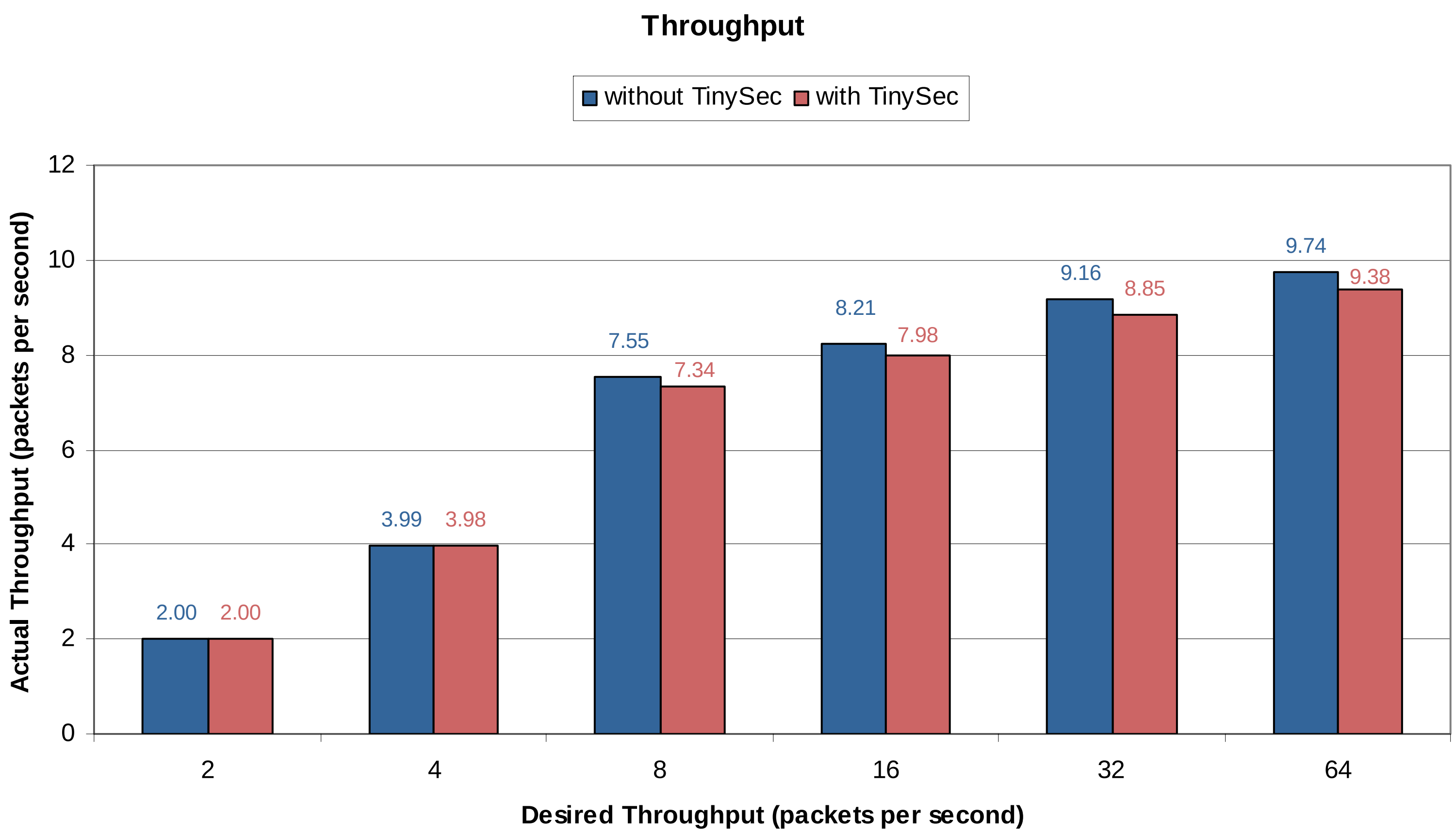
Costs are reasonable

- Costs in time
  - **`encrypt()`**
    - 2,190 μsec
  - **`computeMAC()`**
    - 3,049 μsec
- Costs in space
  - 822 B of SRAM
  - 7,076 B of ROM

See IEEE SECON 2004 paper for breakdown of SRAM into `.bss`, `.data`, and stack requirements.

SKIPJACK and the MICA2

Impact on Throughput Is Negligible



Diffie-Hellman and the MICA2

A Typical Exchange (determining x given g^x mod p is hard)

agree on generator g, prime p

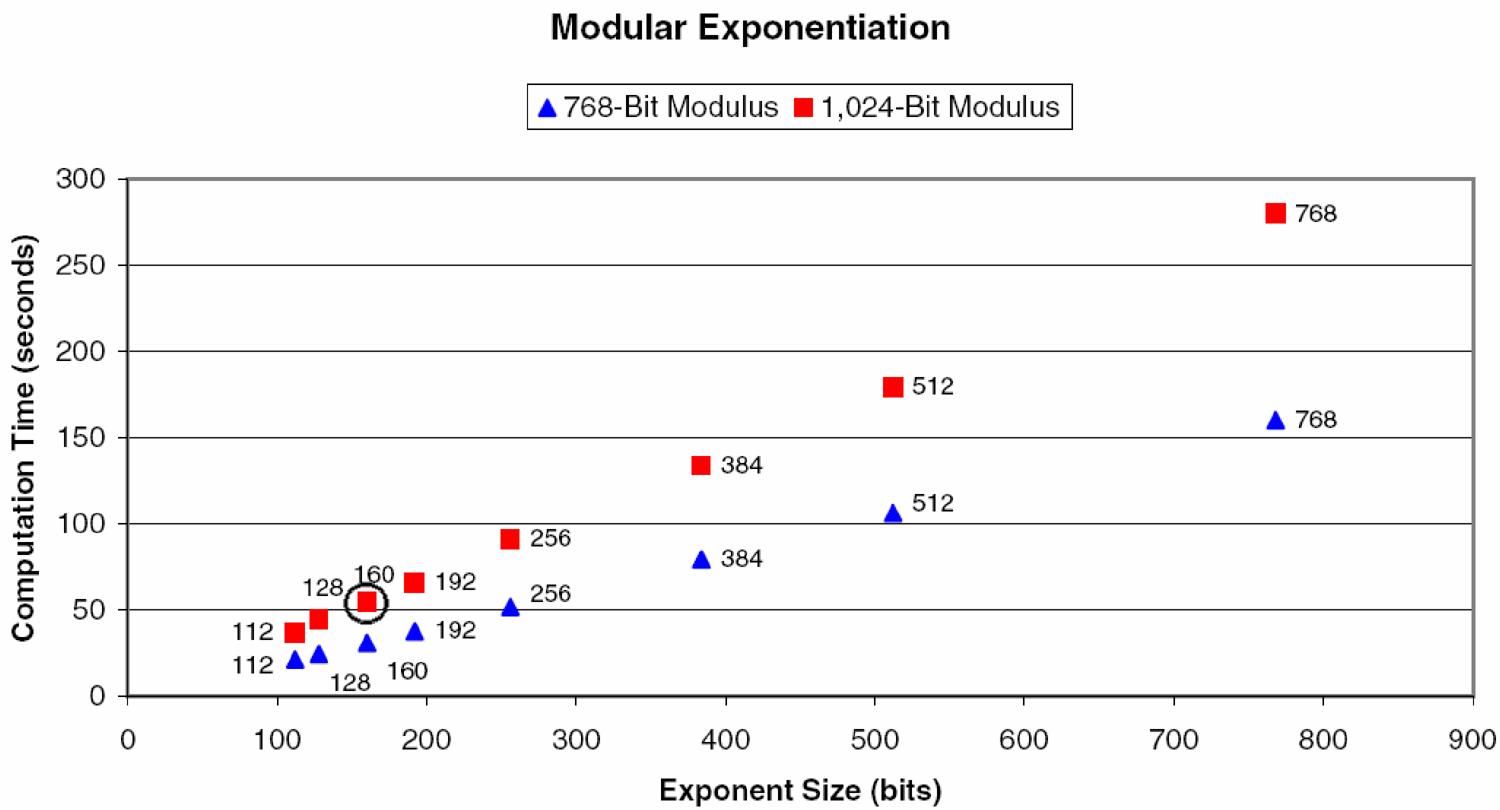
Alice		Bob
choose random A		choose random B
	$\xrightarrow{T_A=g^A \bmod p}$	
	$\xleftarrow{T_B=g^B \bmod p}$	
compute $T_B{}^A$		compute $T_A{}^B$

agree on g^{AB} mod p

Image adapted from Radia Perlman's Computer Science 243.

Diffie-Hellman and the MICA2

Performance Is Relative Slow



Time required to compute $2^x \pmod{p}$, where p is prime, on the MICA2.

Diffie-Hellman and the MICA2

Costs of Generating a 1,024-Bit Public or Shared Key Are Significant

- Cost in time
  - 54.1144 sec
- Costs in space
  - 1,250 B of SRAM
  - 11,350 B of ROM
- Cost in energy
  - 1.185 Joules (3.9897 x 10^8 cycles)

See IEEE SECON 2004 paper for breakdown of SRAM into `.bss`, `.data`, and stack requirements.

Diffie-Hellman and the MICA2

These results should look familiar!

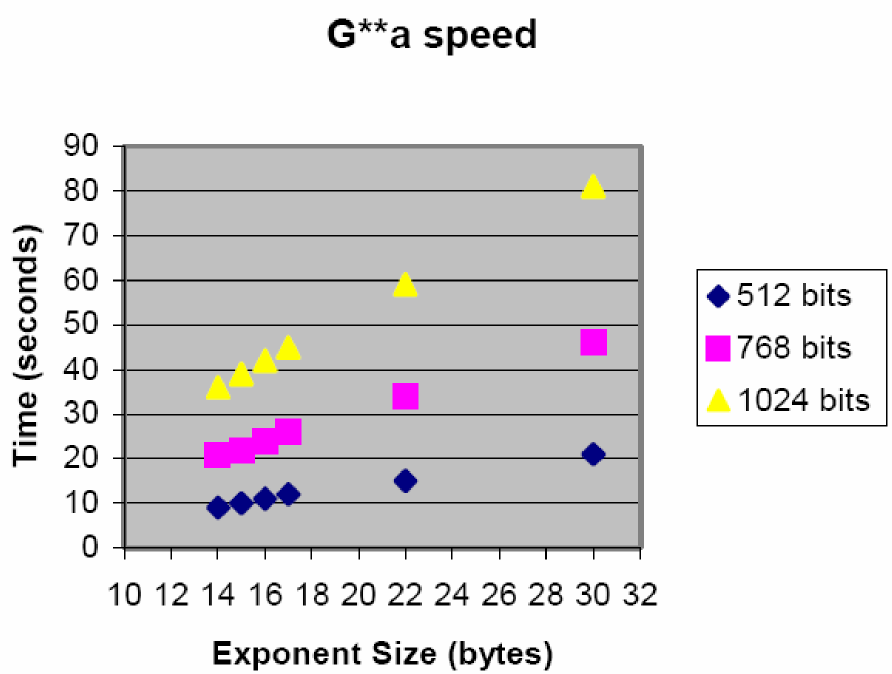
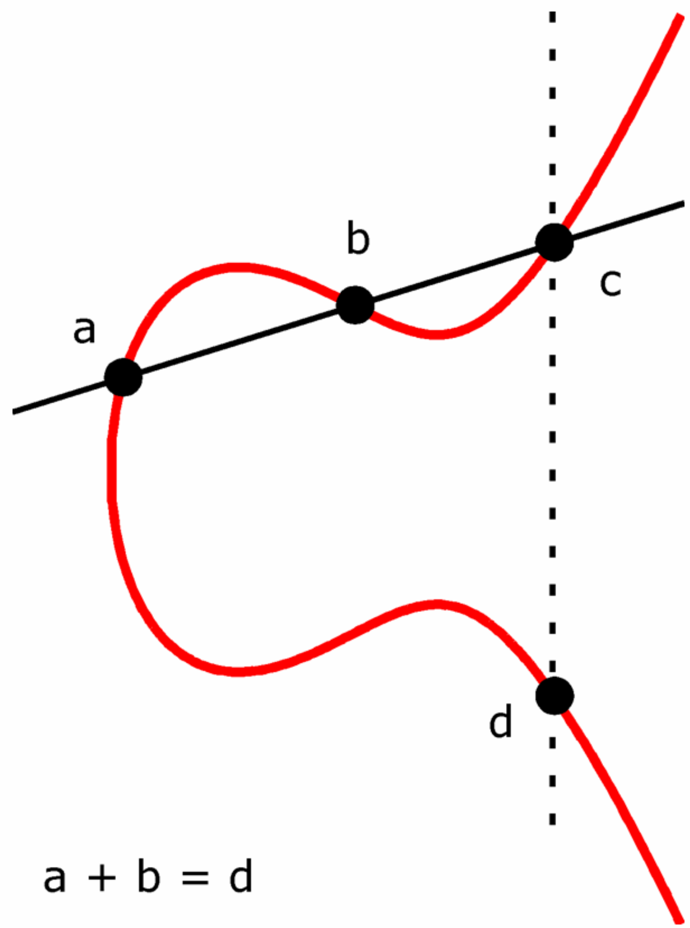


Figure excerpted from Watro *et al.*, TinyPK: Securing Sensor Networks with Public Key Technology, SASN 2004.

ECC and the MICA2

An Elliptic Curve, over Real Numbers



ECC and the MICA2

A Typical Exchange (determining k_x given $k_x * G$ is hard)

point G is publicly known

Alice | Bob
choose random k_A | choose random k_B

$$\xrightarrow{T_A = k_A * G}$$

$$\xleftarrow{T_B = k_B * G}$$

compute $k_A * T_B$ | compute $k_B * T_A$

agree on $k_A * k_B * G$

ECC and the MICA2

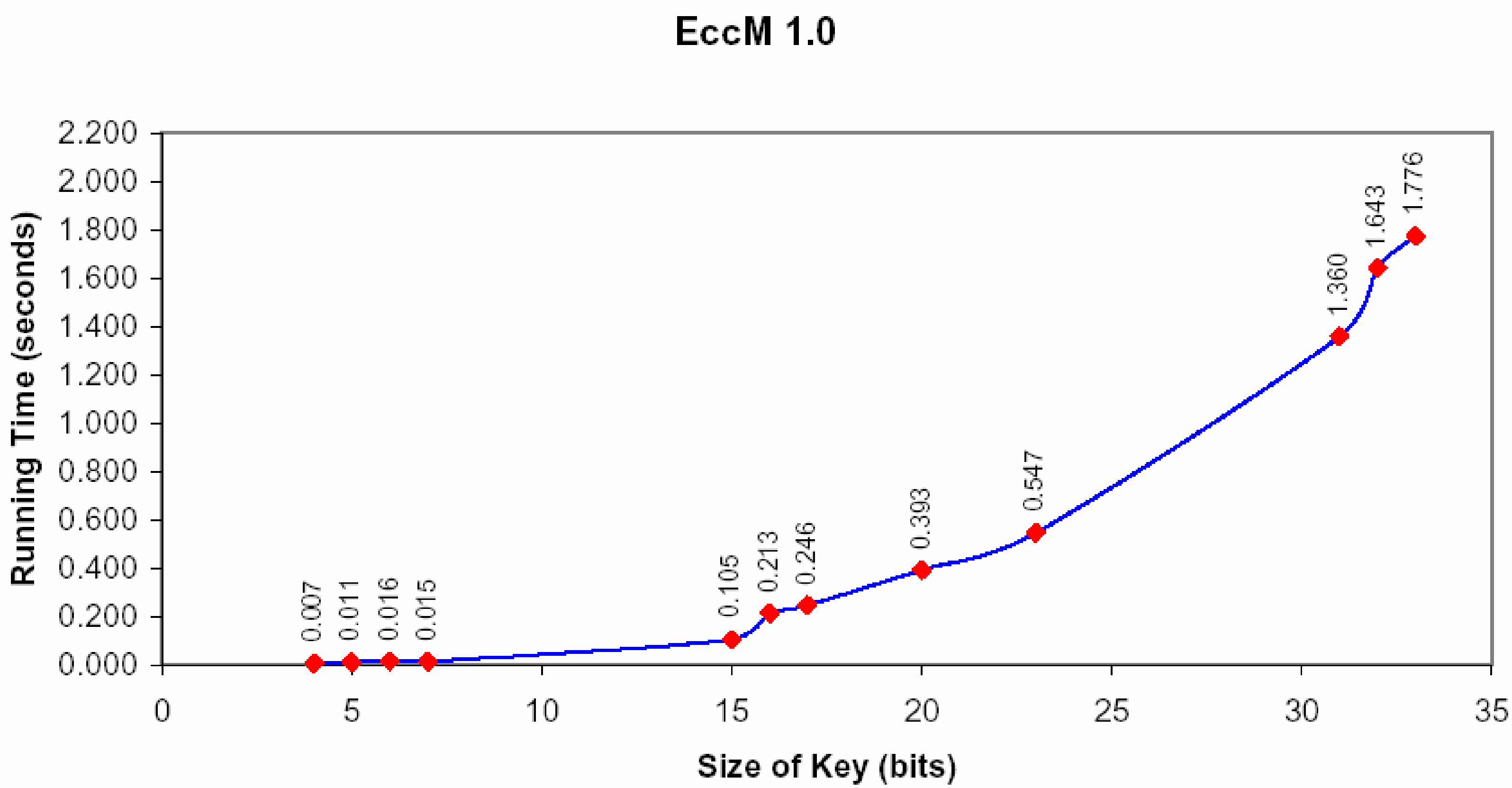
EccM 1.0: Our First Implementation

- A TinyOS module based on code ported from Michael Rosing's *Implementing Elliptic Curve Cryptography.*
- Operates over $GF(2^p)$, using a polynomial basis, modulo an irreducible polynomial.
- Features:
  - selects a random curve of the form $y^2 + xy \equiv x^3 + ax^2 + b$, where $a = 0$ and $b \in GF(2^p)$;
  - selects from that curve a random point, $G \in GF(2^p) \times GF(2^p)$;
  - selects randomly a private key, $k \in GF(2^p)$; and
  - computes $k \cdot G$, the corresponding public key.

See IEEE SECON 2004 paper for breakdown of SRAM into `.bss`, `.data`, and stack requirements.

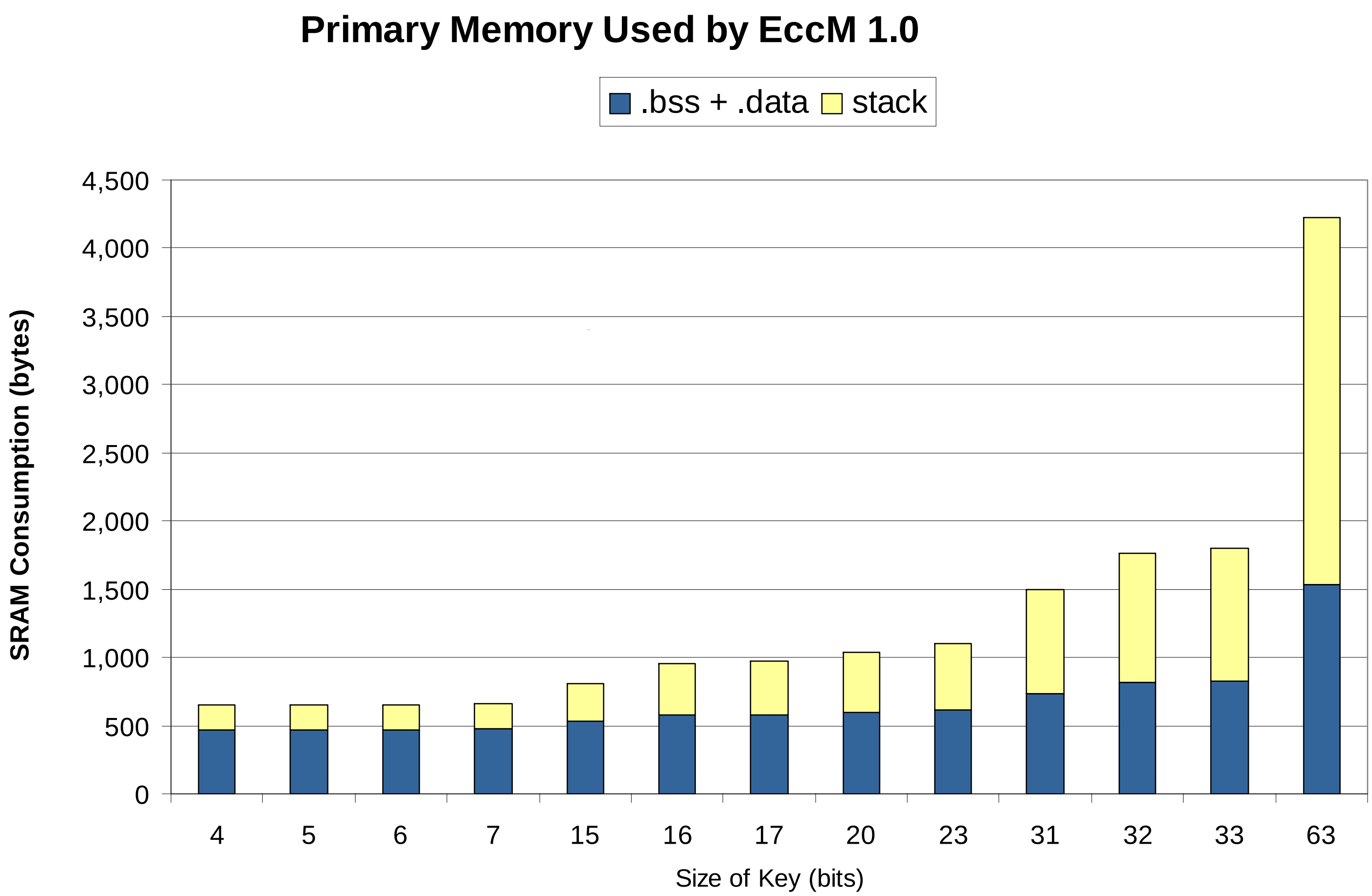
ECC and the MICA2

EccM 1.0: Cost in Time



ECC and the MICA2

EccM 1.0: Costs in Space



ECC and the MICA2

EccM 2.0: Our Second Implementation

- A TinyOS module inspired by Dragongate Technologies Limited's Java-based jBorZoi 0.9.
- Operates over $GF(2^{163})$, using a polynomial basis, modulo $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$.
- Features:
  - selects a Koblitz curve, $y^2 + xy \equiv x^3 + x^2 + 1$, the number of points on which is 0x4000000000000000000020108a2e0cc0d99f8a5ef;
  - selects a base point, $G = (G_x, G_y)$, where G_x = 0x2fe13c0537bbc11acaa07d793de4e6d5e5c94eee8 and G_y = 0x289070fb05d38ff58321f2e800536d538ccdaa3d9;
  - selects randomly for a node, Alice, a private key, $k_A \in GF(2^p)$;
  - computes Alice's public key, $T_A = k_A \cdot G$;
  - transmits T_A to a node, Bob, who similarly generates and transmits his own T_B;
  - computes for Alice $k_A \cdot T_B = k_A \cdot k_B \cdot G$ just as Bob computes $k_B \cdot T_A = k_A \cdot k_B \cdot G$, the same shared secret.

ECC and the MICA2

EccM 2.0 initially offered smaller keys but worse performance

- Cost in time
  - 495.92 sec
- Costs in space
  - 901 B of SRAM
  - 43,286 B of ROM
- Cost in energy
  - 12.65 Joules (3.65×10^9 cycles)

ECC and the MICA2

How We Optimized EccM 2.0

- Eliminated foolish code
  (*e.g.*, recomputing terminal conditions for loops)
- Optimized source by hand
  - manually unrolled loops
  - special-cased common shifts
  - re-ordered loops based on expected lengths of elements
- Implemented published algorithms from current literature
  (*e.g.*, J. López and R. Dahab, "High-Speed Software Multiplication in F_{2^m}," Institute of Computing, State University of Campinas, São Paulo, Brazil, Tech. Rep., May 2000)

ECC and the MICA2

EccM 2.0 ultimately offered smaller keys *and* better performance

- Cost in time
  - 34.161 sec [down from 495.92 sec]
- Costs in space
  - 1,140 B of SRAM [up from 901 B]
  - 34,342 B of ROM [down from 43,286 B]
- Cost in energy
  - 0.816 Joules (2.512×10^8 cycles)
    [down from 12.65 Joules (3.65×10^9 cycles)]

See IEEE SECON 2004 paper for breakdown of SRAM into `.bss`, `.data`, and stack requirements.

Related Work

Current Literature

- R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, P. Kruus, "TinyPK: Securing Sensor Networks with Public Key Technology," Second ACM Workshop on Security of Ad Hoc and Sensor Networks , October 2004
- C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," Second ACM Conference on Embedded Networked Sensor Systems, November 2004
- N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," Sixth International Workshop on Cryptographic Hardware and Embedded Systems, August 2004
  - 0.81 sec for 160-bit point multiplication over GF(p)

This Work

Additional detail can be found in these papers

- D. Malan, “Crypto for Tiny Objects,” Harvard University Technical Report TR-04-04, January 2004
- D. Malan, M. Welsh, and M. Smith. “A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography,” First IEEE International Conference on Sensor and Ad Hoc Communications and Networks, October 2004

Future Work

Considerations for EccM 3.0

- GF(p)
- Normal Basis
- AVR Assembly

Conclusion

PKI is, in fact, viable for key distribution on the MICA2

1. SKIPJACK with 80-bit keys
   - 2,190 μsec for `encrypt()`
   - 3,049 μsec for `computeMac()`
2. Diffie-Hellman with 1,024-bit keys
   - 54.1144 sec for key generation
   - 1,250 B of SRAM
   - 11,350 B of ROM
   - 1.185 Joules (3.9897×10^8 cycles)
3. ECC with 163-bit keys
   - 34.390 sec for key generation
   - 1,140 B of SRAM
   - 34,342 B of ROM
   - 0.82149 J (2.5289×10^8 cycles)

Toward PKI for Sensor Networks

8 November 2004

David J. Malan
Division of Engineering and Applied Sciences
Harvard University

`malan@eecs.harvard.edu`
`http://www.eecs.harvard.edu/~malan/`
+1-617-523-0925