A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography

IEEE SECON 2004

David J. Malan, Matt Welsh, Michael D. Smith
Division of Engineering and Applied Sciences
Harvard University

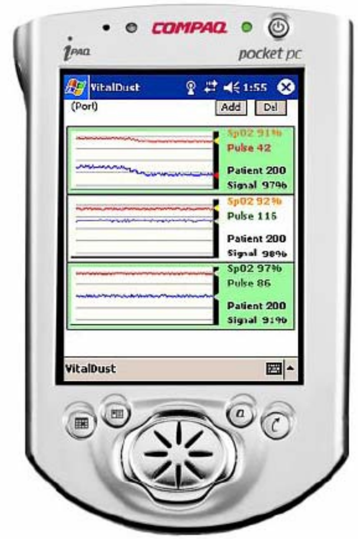
Contact
malan@eecs.harvard.edu

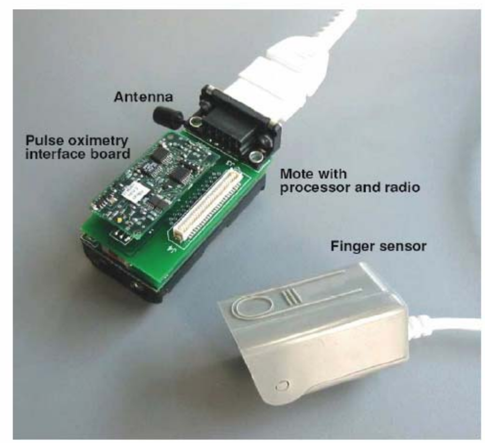
Source Code
http://www.eecs.harvard.edu/~malan/research/

CodeBlue

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

- Mass casualty incidents (MCIs) involve multiple patients
- Manual tracking of patient status is difficult
  - Current systems are paper-, phone-, and radio-based
- 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)





Ensuring Privacy and Security

Secret-Key Cryptography

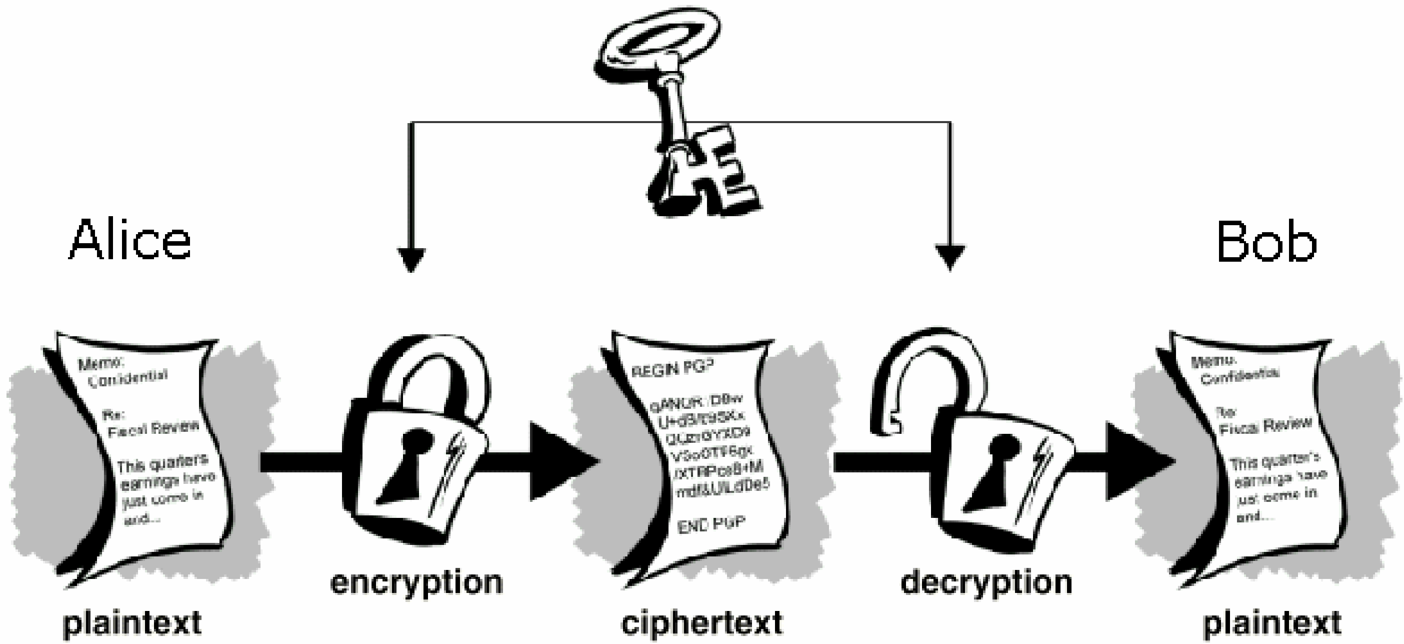
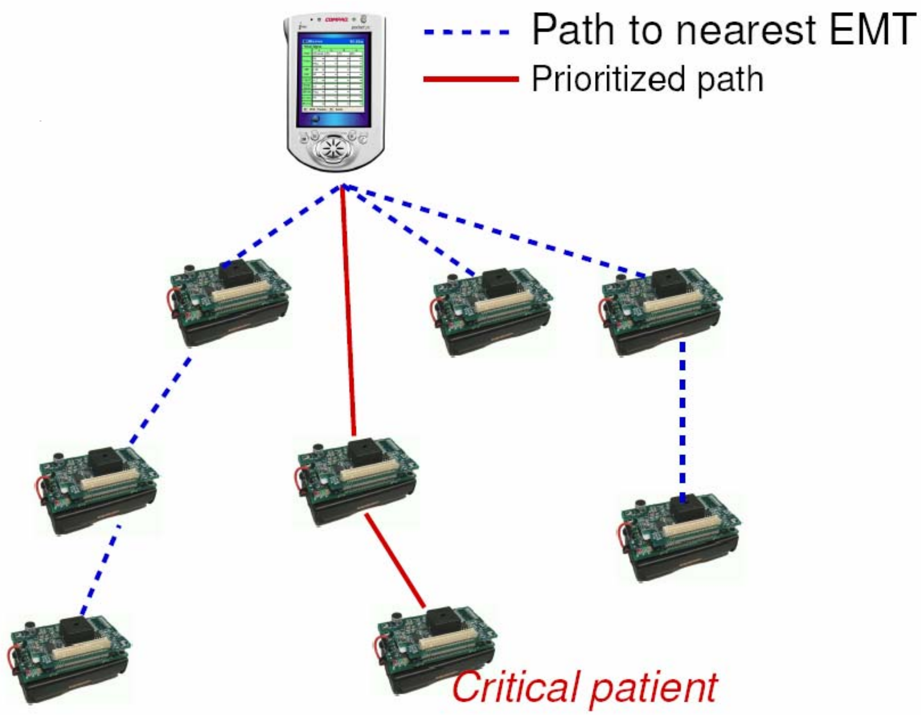


Image adapted from `http://www.nuitari.de/crypto.html`.

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)

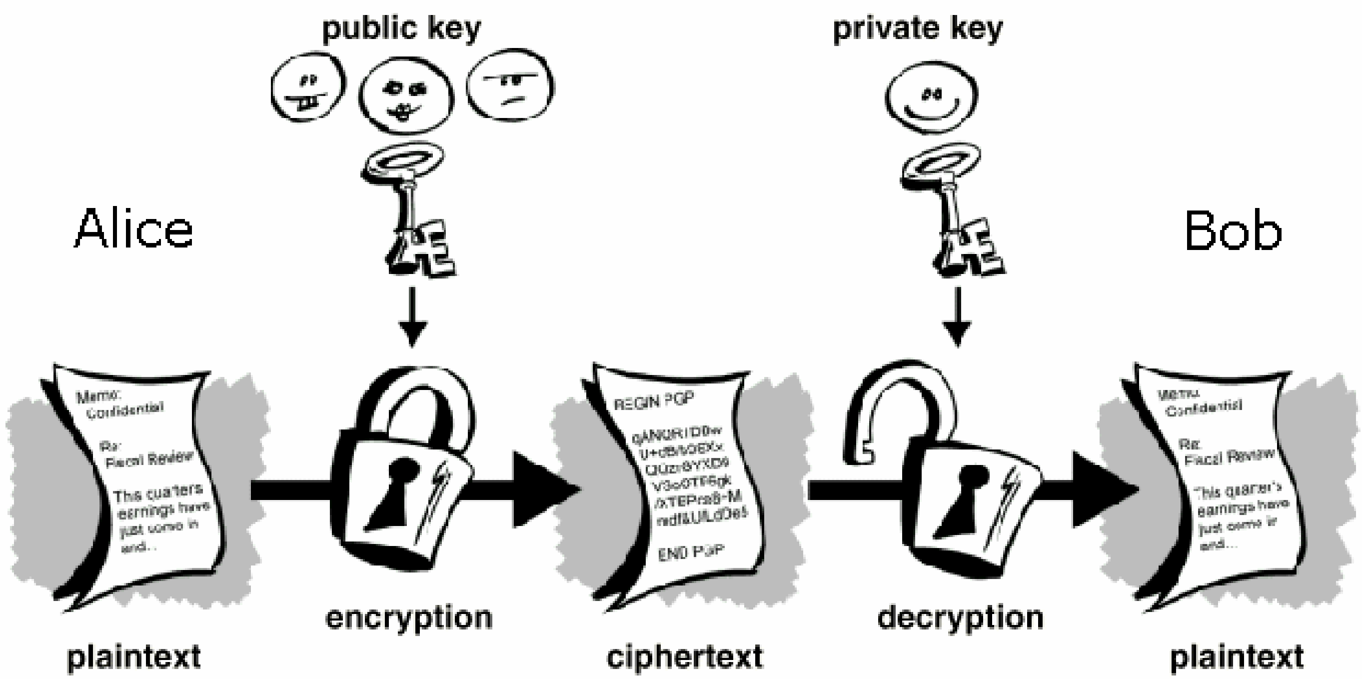


Image adapted from `http://www.nuitari.de/crypto.html`.

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' 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

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

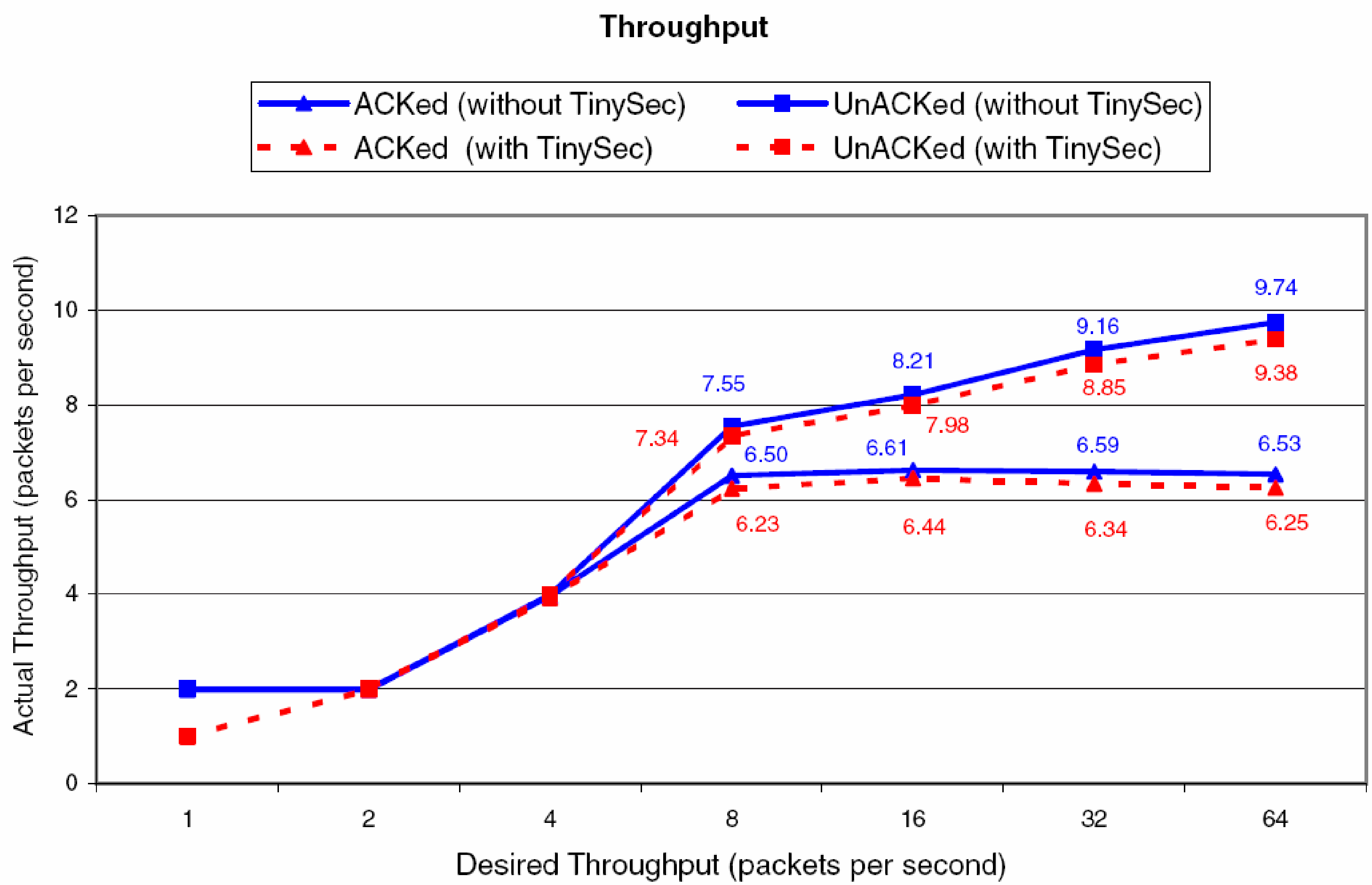
Costs are reasonable

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

See 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)

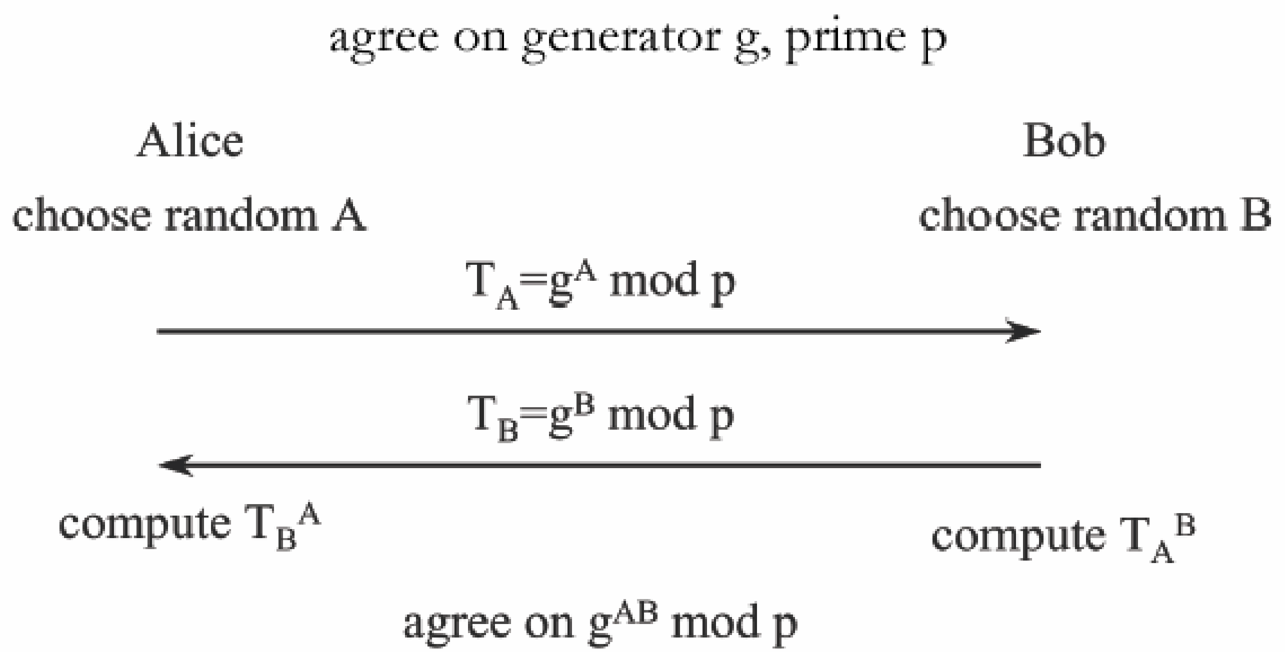
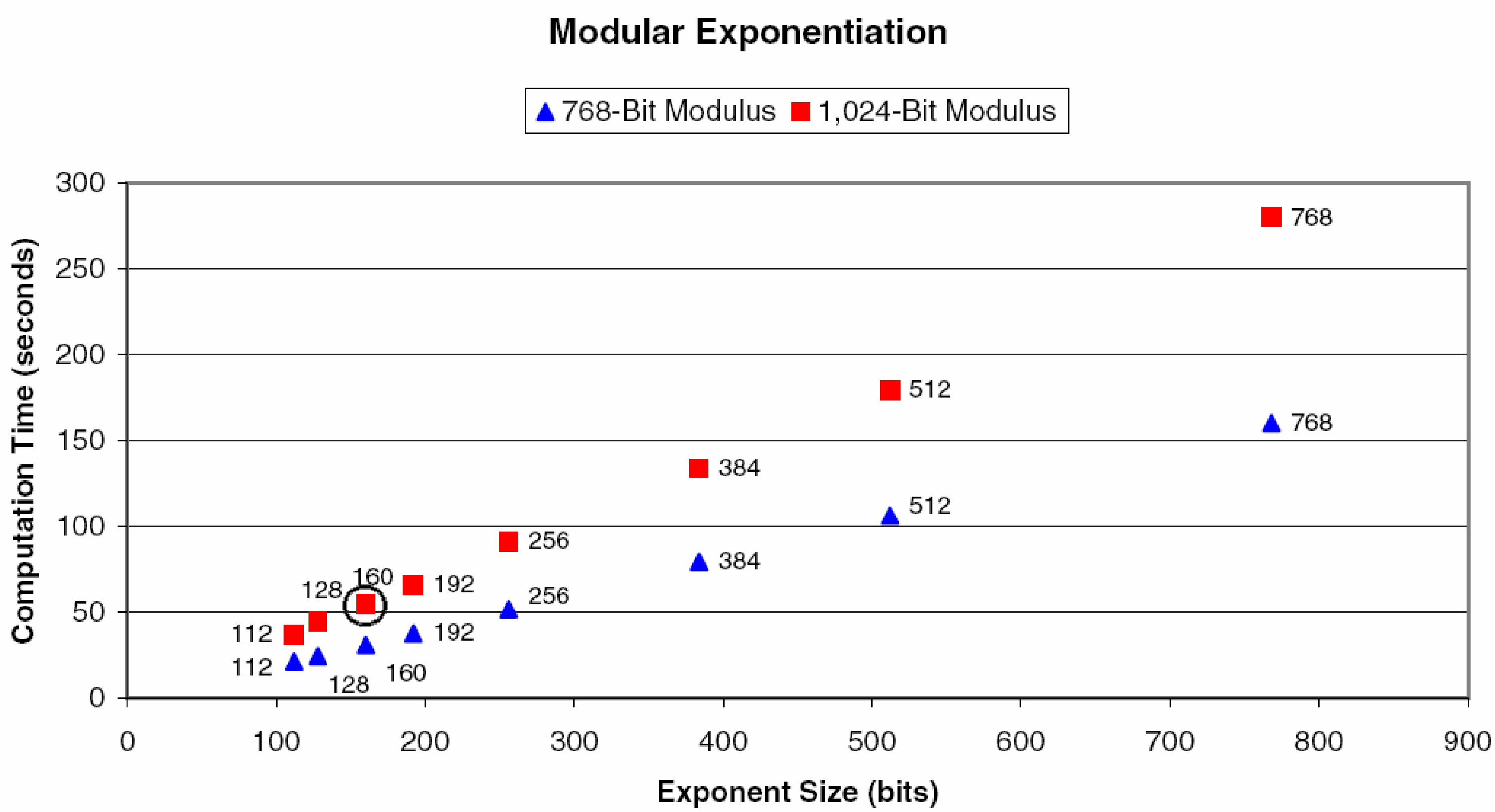


Image adapted from Radia Perlman's Computer Science 243.

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

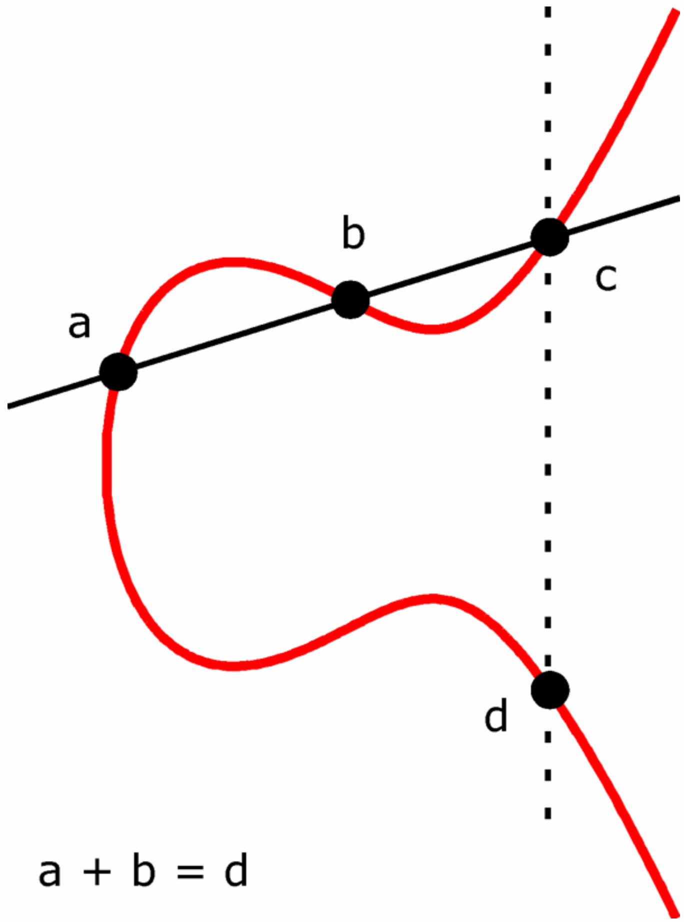
Costs of Generating a 1,024-Bit Public or Shared Key Are Significant

- 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 paper for breakdown of SRAM into `.bss`, `.data`, and stack requirements.

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
	$T_A = k_A * G$ →	
	← $T_B = k_B * G$	
compute $k_A * T_B$		compute $k_B * T_A$

agree on $k_A * k_B * G$

ECC and the MICA2

Our Elliptic Curve, over GF(2^{163}) Using a Polynomial Basis

curve

$$y^2 + xy \equiv x^3 + x^2 + 1$$

order

0x4000000000000000000020108a2e0cc0d99f8a5ef

point $G = (G_x, G_y)$

$$\begin{aligned} G_x &= \text{0x2fe13c0537bbc11acaa07d793de4e6d5e5c94eee8} \\ G_y &= \text{0x289070fb05d38ff58321f2e800536d538ccdaa3d9} \end{aligned}$$

reduction polynomial

$$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$$

ECC Offers Better Performance *and* Smaller Keys (163 v. 1,024 bits)

Costs of Generating a 163-Bit Public or Shared Key Are Lower

- Cost in time
  - 34.161 sec
- Costs in space
  - 1,140 B of SRAM
  - 34,342 B of ROM
- Cost in energy
  - 0.816 Joules (2.512 x 10^8 cycles)

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

Optimizations

Our Two Implementations of ECC (EccM 1.0 and EccM 2.0)

- Our first implementation of ECC, a straightforward port of Michael Rosing's code in *Implementing Elliptic Curve Cryptography*, proved a failure
- Our current implementation reflects the design of Dragongate Technologies' Java-based jBorZoi and employs various optimizations
  - Source-level, hand optimizations (*e.g.*, manually unrolled loops, special cases for common shifts)
  - Published algorithms from current literature (*e.g.*, J. López and R. Dahab, "High-Speed Software Multiplication in F_{2^m}," Institute of Computing, Sate University of Campinas, São Paulo, Brazil, Tech. Rep., May 2000)

Related Work

Current Literature

- 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.” 6th International Workshop on Cryptographic Hardware and Embedded Systems, August 2004
  - 0.81 sec for 160-bit point multiplication over GF(p)

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 x 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 x 10^8 cycles)

A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography

IEEE SECON 2004

David J. Malan, Matt Welsh, Michael D. Smith
Division of Engineering and Applied Sciences
Harvard University

Contact
malan@eecs.harvard.edu

Source Code
http://www.eecs.harvard.edu/~malan/research/