October 2004
Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks SASN '04
Publisher: ACM Press

TinyPK: Securing Sensor Networks with Public Key Technology

Ronald Watro, Derrick Kong, Sue-fen Cuti, Charles Gardiner, Charles Lynn[1] and Peter Kruus
BBN Technologies
10 Moulton St
Cambridge MA 02138
617-873-3200
{rwatro, dkong, sfcuti, gardiner, pkruus}@bbn.com

ABSTRACT
Wireless networks of miniaturized, low-power sensor/actuator devices are poised to become widely used in commercial and military environments. The communication security problems for these networks are exacerbated by the limited power and energy of the sensor devices. In this paper, we describe the design and implementation of public-key-(PK)-based protocols that allow authentication and key agreement between a sensor network and a third party as well as between two sensor networks. Our work is novel in that PK technology was commonly believed to be too inefficient for use on low-power devices. As part of our solution, we exploit the efficiency of public operations in the RSA cryptosystem and design protocols that place the computationally expensive operations on the parties external to the sensor network, when possible. Our protocols have been implemented on UC Berkeley MICA2 motes using the TinyOS development environment.

Categories and Subject Descriptors
C.2.0 [**Computer-Communication Networks**]: General – *security and protection.*

General Terms
Security, Algorithms, Performance.

Keywords
Authentication, Cryptography, Diffie-Hellman, Encryption, Key Management, Public Key (PK), Rivest Shamir Adelman (RSA), sensor networks, TinyOS.

1. INTRODUCTION
Recent advances in the fields of computation, communication, and sensor technology have combined to produce the potential for a new generation of wireless sensor networks. These networks will consist of miniaturized sensor/actuator devices communicating over an ad hoc wireless network. The new generation of sensor networks will deploy large numbers of low-cost, low-power nodes, using an approach that leverages hardware research at the University of California called Smart Dust [1]. Companies such as Crossbow, Dust, Ember, Millennial Net, Moteiv, Sensicast, and Sensoria, to name just a few, are currently marketing sensor network node hardware. The work of this paper was carried out using MICA Mote networks [11] but could equally be applied to any of the other sensor network hardware options. Many applications of sensor networks have been envisioned, from smart buildings to smart battlefields to smart human beings. These applications envision sensor networks that unobtrusively monitor building perimeters, battlefield situations data, or human health characteristics, reporting data to end users to correct malfunctions or to repel adversaries.

In many sensor network applications, security and privacy of the data collected will be a critical concern. Providing security services for sensor networks is a technical challenge. The low power design objective for the sensor nodes forces security mechanisms to fit under very limiting processing and bandwidth constraints. For example, the MICA2 mote carries an 8-bit Atmega 128L microcontroller running at about 7 MHz, and the communication bandwidth of the radio signal is just 40 kilobits/second.

This paper focuses on supporting confidentiality and source authentication for sensor network traffic. Other researchers have previously achieved a strong step towards these goals in that symmetric encryption algorithms have been implement on several sensor networks. For MICA mote hardware, efficient software implementations of symmetric (i.e., secret key) encryption and corresponding message authentication codes have been developed by UC Berkeley [TinySec, 12] and in unpublished work by SRI. Hardware implementations of AES are now included on devices using 802.15.4 radios (such as the Ember sensor nodes). Our expectation is that secure symmetric encryption will be widely available on the sensor networks of the future. The critical problem is making effective use of that secure symmetric encryption capability. As is always the case with symmetric encryption, proper key management is a fundamental concern.

Public key (PK) technology is a widely used tool to support symmetric key management in the realm of Internet hosts and high-bandwidth interconnections. It is the thesis of this paper that

SASN'04, October 25, 2004, Washington, DC, USA.

[1] Charles Lynn recently passed away and this paper is dedicated to his memory.

public key technology can also be very selectively deployed in the realm of sensor networks. In the past, the constraints of sensor networks have fostered a belief in some researchers that many Internet-level security techniques are too heavyweight for sensor networks and that new alternatives must be developed. This opinion has been very valuable in that it has lead to interesting new research, but in this paper, we demonstrate that with careful design, the widely used RSA public key cryptosystem [17] and Diffie-Hellman key agreement techniques [8] can be deployed on even the most constrained of the current sensor network devices.

There are several security problems that are beyond the scope of our current work. For example, we do not handle the problem of revocation of compromised private keys. We also have designed only limited protection against denial of service attacks [18]. Improved radio hardware including spread spectrum capability with low probability of detection and interception may someday help resolve the denial of service problems. Unfortunately, the currently proposed spread spectrum capability defined in IEEE 802.15.4 is designed to protect against only unintentional interference rather than denial of service attacks.

In section 2 below we describe the initial concepts of TinyPK. Section 3 describes an initial implementation on the MICA motes for authenticating an external party and securely transferring a common session key to that party. Section 4 describes Diffie-Hellman key agreement implemented on the MICA2 mote. Section 5 covers authentication of sensor network nodes to external parties. Section 6 reports on AuthXNP, our authenticated over-the-network programming capability for motes. Section 7 covers related work. Section 8 provides a summary and conclusions.

2. DATA ACCESS DESIGN

A simple security model for a sensor network employees a single secret key that is known by all nodes in the sensor network. There are many potential problems with this simple model and it clearly fails to scale to large systems. However, in a small local region of the sensor network, this simple model is likely to hold. The single shared secret key creates a communication cryptonet: possession of the key authenticates the holder as part of the secure group. Traffic sent in the sensor network is encrypted and integrity protected using the common key. Since traffic is sometimes broadcast and sometimes routed hop-to-hop across the sensor network, the single encryption key provides a very user-friendly security architecture, albeit with the already mentioned limitations.

Whenever a session key is used to create a secure group, maintenance of group membership must be considered. For sensor networks, high compromise rates are possible, making group membership issues more complex. In this section, we consider an external party that wishes to establish secure communications with a sensor network. It is impractical in many settings to directly provide the session key to the external party.

Our TinyPK security scheme is a mechanism for providing authentication and key exchange between an external party and a sensor network. TinyPK is based on the well-known RSA cryptosystem, using e=3 as the public exponent. The security properties of the Low Exponent variant of RSA have been extensively studied [3,4]. As long as proper precautions are taken (e.g., appropriate use of random padding when encrypting text), the Low Exponent RSA system is free from known attacks. To make TinyPK practical for low power sensor devices, we design our protocols to require only the public key operations (data encryption and signature verification) on the sensor network. RSA has the pleasant property that its public operations are very fast compared to other public key technology computations. All that is required to perform a TinyPK 1024-bit basic public operation is to cube a 1024-bit number and to take its residue modulo a large prime.

TinyPK requires a modest amount of public-key infrastructure that we now describe. The first element of the infrastructure is a Certification Authority (CA), which is an entity with a private and public key pair that is trusted (or can establish an authenticated chain to a trusted entity) by all friendly units. Any third party that wishes to interact with the motes also requires its own public/private key pair and must have its public key signed (not on a hash of the data, but by transforming the data directly) by the CA's private key, thus establishing its identity. Finally, as each mote is loaded with software before being deployed to the field, it must have the CA's public key installed. Traditionally, a public key is made part of a certificate (e.g. an X.509 certificate) but TinyPK eliminates certificates as sensor networks are assumed to not have the processing power or the data context to make use of certificates, e.g., no real-time access to the CA infrastructure. Without a certificate structure, there is no direct way to deal with compromise of an external party private key. Our protocols try to minimize the damage by such a compromise but as a concession to efficiency we have no recovery mechanisms for compromise of private keys.

We provide here an overview of the TinyPK challenge-response protocol that authenticates the external party to the sensor network and securely transfers a session key from the sensor network to the third party. To perform authentication, the external party submits its signed public key and some text signed with its private key. Protocol operation starts when the third party provides a challenge to the sensor network. This challenge consists of two parts: The first is its own public key, signed by the CA private key; the second is a compound object consisting of a nonce (a timestamp) and a message checksum, signed with the third party's own private key. This information is not encrypted. The nonce serves to detect replay attacks, wherein a malicious party records previous valid messages and rebroadcasts them in order to provide false identification or otherwise attack a system. The checksum is used to insure message integrity.

Upon receipt of the message, a sensor node uses the preloaded CA public key to verify the first part of the challenge and extract the third party's public key. It then uses this public key to verify the second part of the message and extract the nonce and checksum. The nonce and checksum are validated. If they pass validation, the third party has successfully authenticated to the sensor network and is considered to be an authorized entity for sensor data.

The sensor node now encrypts the session key plus the received nonce using the third party's public key. This combination is sent back to the third party, which decrypts it using its private key, checks that the nonce is the same as the one it sent, and if so, can record the session key for future use.

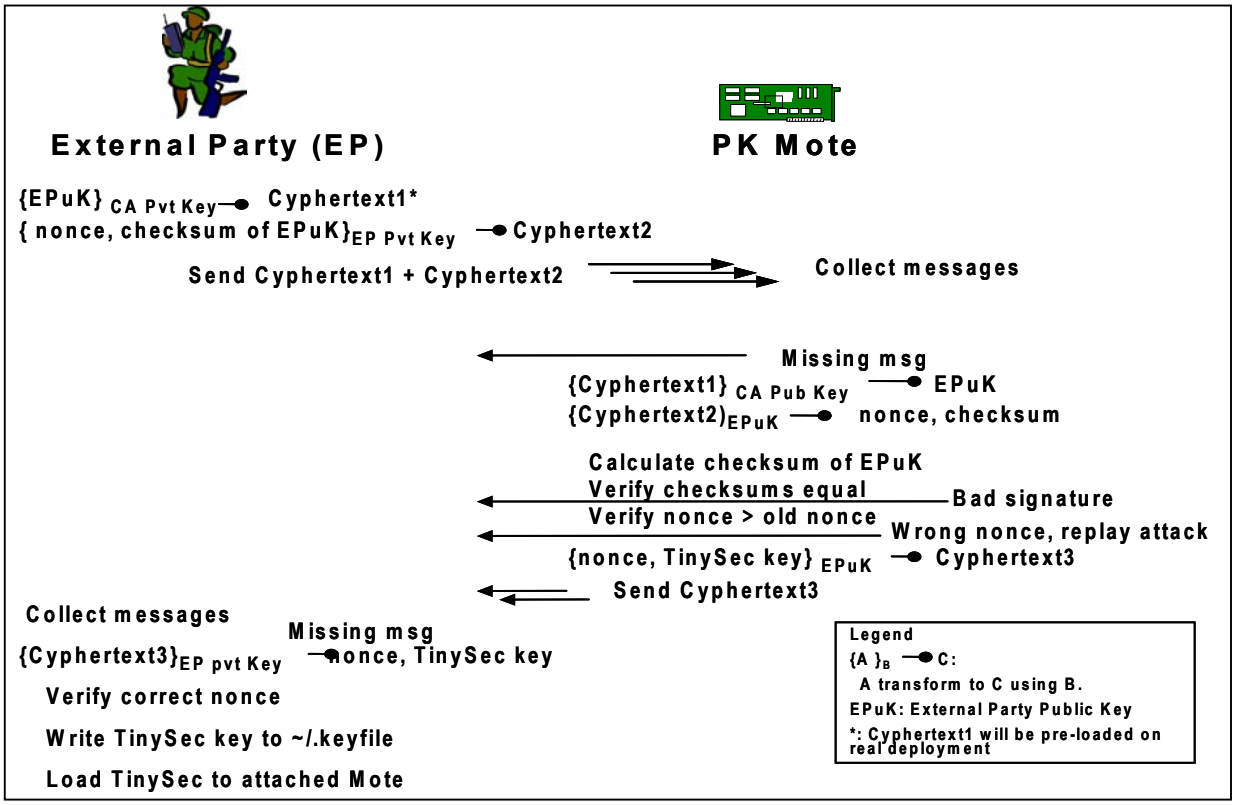

Figure 1 TinyPK EP Protocol Exchange Diagram

Two natural questions arise this point. First, while the external party has authenticated itself to the sensor network, the reverse authentication has not yet been discussed. We cover that issue in section 4 below. Second, one might wonder why a dedicated session key isn't created for the interaction with the external party. This would have some advantages, and can be done using the Diffie-Hellman methods we discuss in section 3. Such a dedicated key will need to be held by all parties associated with a specific section of the sensor network.

3. DATA ACCESS IMPLEMENTATION

The protocol described in section 2 was implemented on MICA1 Motes [11]. These devices carry an Atmel ATmega 128L microcontroller running at 4 MHz with 4KB of RAM and 128KB of flash memory. Our implementation work was done using UC Berkeley's TinyOS development environment [10] and the NesC programming language. A laptop computer with an attached mote acted as an external party. The laptop communicates with the mote over a serial cable while the mote provides the radio interface to the laptop. A small amount of java code was written to implement the external party actions of the protocol. Freely available code was used to run the RSA private operations on the PC.

NesC code was written for the motes to implement the RSA public operations and the sensor network protocol steps. The modular exponentiation was implemented in general for any exponent but is used only for e=3. The exponentiation is done by the conventional method of squaring and multiplying. The multiplication is performed from the most significant end down to minimize the need for extra storage. Squaring is similar but takes advantage of the symmetry of squaring so that the procedure takes half as long as multiplication. Re-modularization is done by a method analogous to decimal long division, except that the calculations are base 256, so the trial divisor is just one byte. No quotient is calculated, just the remainder. All the arithmetic is done in 8-bit words, except that 16-bit partial products are formed and then stored as pairs of 8-bit words with appropriate propagation of carries.

Table 1: RSA Small Exponent Operation Times

RSA Key Size	Time (sec)
512	3.8
768	8.0
1024	14.5

The performance of one RSA exponentiation is shown in Table 1. These times seem long given the current widely available experience with high-speed computing but are perhaps reasonable in a physical setting where TinyPK is used to perform infrequent authentications on a human-user time scale. Our execution times for 1024-bit RSA operations are close to what have been previously reported for other C-based implementations [7] but much slower than recently reported results on assembly language implementations of RSA on motes [9].

The details of our implementation of the challenge response protocol are shown in Figure 1. The left side of the figure shows operations of the external party, represented by a soldier. The PKMote on the right side of the figure is simply a MICA Mote loaded with our TinyPK software. As shown in figure 1, the

protocol operation starts with two "cyphertexts" being transmitted from the external party (EP) to the PKMote. The first cyphertext is the signature data for the EP's public key, while the second cyphertext is the signature data for the EP's public key checksum and a nonce, transformed using the EP's private key. The first cyphertext is loaded and stored in the external party at initialization time, while the second is generated locally at each execution of the protocol. The nonce is a relative timestamp used to detect replay attacks. While a reply of the EP challenge would not lead to compromise of the session key, replay protection is still useful to allow the sensor network to defer responses to unauthorized challenges. The goal is to allow the sensor network to authenticate the EP before it sends a response message. Our implementation breaks the two cyphertexts into parts small enough to fit within TinyOS active messages (about 25 bytes of data per message) and transmits the messages to the PKMote.

At the PKMote, the challenge messages are received and processed. A few error cases, such as bad checksums or missing messages, can be detected by our implementation and are shown in the 2nd column of Figure 1. If the PKMote successfully receives the messages, it must verify that it has received a properly signed public key and that the EP appears to hold the corresponding private key. If the verification is successful, the session key and nonce are encrypted using the EP's public key and transmitted in TinyOS messages back to the EP. The EP then decrypts the material it receives, validates the nonce as matching the original, and stores the session key for future use.

4. KEY EXCHANGE

The TinyPK implementation uses the RSA algorithm to exploit its speed in public key operations. Currently, our implementation of RSA private operations is too slow to be used on motes, as extrapolations indicated that execution times would be in the tens of minutes.

After some validation tests which demonstrated a noticeable speed improvement, we implemented Diffie-Hellman key exchange on the MICA2 platform. The goal of Diffie-Hellman is to provide a shared secret between two parties that can then be use to create a cryptographic key. We use Diffie-Hellman to generate a secret suitable for use in creating a new or replacement TinySec key. Such a key would allow two disjoint sensor networks to communicate and allow the deployment of replacement motes into an existing sensor field without having to look up and preload the TinySec key in use by the field.

The protocol operates in the standard manner: The mote initiates the exchange by generating a random number R1, performing the blinding function (i.e. calculating $g^{R1} \bmod p$) and sending that quantity to another mote. The second mote generates its own random number, R2, and performs the blinding function in parallel with the first mote. Upon receipt of the blinded quantity, it responds by sending its blinded quantity to the first mote. Each mote then calculates $(g^{R1} \bmod p)^{R2} \bmod p = (g^{R2} \bmod p)^{R1} \bmod p$ $= g^{R!*R2} \bmod p$, which is the shared secret.

In our current implementation of modular arithmetic, we use a generator, g=2, and have the capability of using exponents and moduli of differing sizes. Size and performance characteristics of the modular exponentiation code for various parameter values are summarized below. Code sizes are summarized in Table 2. Timing results for the modular arithmetic operations are summarized in Figures 2 and 3.

The nature of Diffie-Hellman is such that the communications between parties can take place in an unsecured manner. This mode of operation is needed between two motes that may not yet share a common TinySec key. In the next section, we will use a semi-static Diffie-Hellman key exchange to support mote authentication.

Table 2 Memory Utilization on Motes

	Modulus Size		
	512	768	1024
ROM (bytes)	12340	12376	12408
RAM (bytes)	847	1007	1167

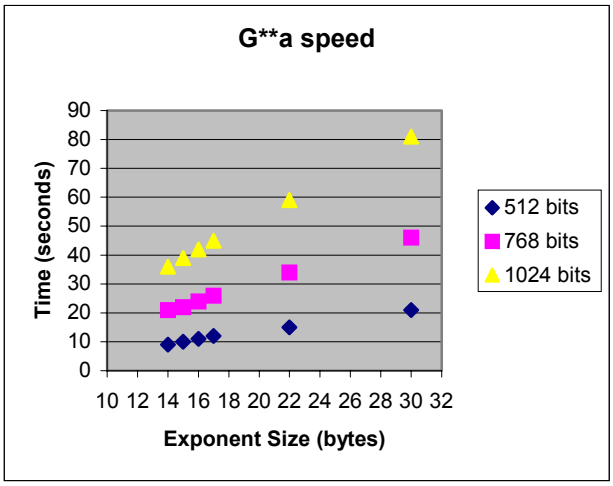

Figure 2 Execution time for first exponentiation

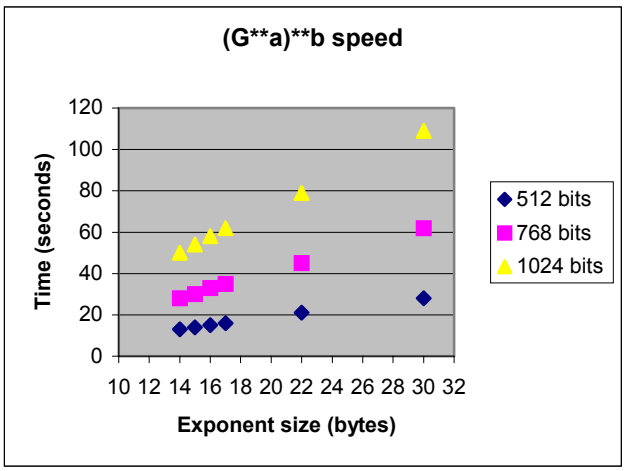

Figure 3 Execution time for second exponentiation

5. AUTHENTICATION OF NODES

Sections 2 and 3 have established that the EP can authenticate itself to the sensor network and securely obtain the sensor network's session key. This integrates the EP into the sensor cryptonet. The missing step is that the EP has no reason to believe in the validity of individual sensors in the field. This section provides some support for this missing authentication.

Our design requires that each sensor be preloaded at initialization with a credential that can be presented to establish identity data about the mote. Because the motes cannot perform full RSA private key operations, we cannot require the motes to contain RSA key pairs and to generate signatures in the field. Instead, we propose that a mote use a static Diffie-Hellman key pair along with a text string processed by the CA's private key as a credential. The text string will specify some identity properties of the mode, for example, its serial number, date of construction, and initial assignment information, in a fixed format. This information is also stored in plain text in the mote. Any party that holds the CA's public key can receive a copy of a mote's credential and verify the mote's identify information. To associate the text credential to a mote, the external party does a semi-static key exchange with the mote. The mote then generates a message that allows the external party to verify that the mote holds the private half of the target Diffie-Hellman key pair.

To perform mote authentication, we assume that an EP has completed the protocol of section 2 and is communicating with the sensor network using TinySec encrypted traffic. The EP now repeats the challenge of section 2 over the secure link. The EP also transmits an ephemeral Diffie-Hellman public value. A mote that receives and verifies the challenge will respond by transmitting its credentials (text string and CA-signed Diffie-Hellman public value) and the challenge nonce (HMAC'ed using the Diffie-Hellman resultant key and encrypted in the EP's public key).

The external party decrypts and verifies the messages from the mote, examines the mote serial number and stores it in a database along with the session key and the mote ID number. The nonce is checked against the version that was sent out. Future communications from valid sensor nodes must have a validated serial number along with a message authentication code using the appropriate session key. Invalid nodes will either fail to have a properly registered serial number or fail the message authentication check.

6. AUTH-XNP

This section is a high-level summary of an application of TinyPK to over-the-network programming (XNP) in TinyOS. XNP [6] allows motes to receive and load a new program via radio transmissions instead of over a wired link. The XNP software module must be loaded on a mote via a wired link (or in the factory), but after that the mote can receive new code over the network. XNP is primarily used as a single-hop programming process. An updated version of XNP based on flooding transmission of the code is under development at UCB to better support multi-hop programming.

To reprogram a mote using XNP, a central broadcast node starts by sending out a number of initialization pings that cause motes to stop operations and prepare for code reception. The broadcaster then sends out the new code in a series of numbered packets. After the last packet is transmitted, the broadcaster queries motes to identify any missing packets, which are then retransmitted. Finally, after receiving no more requests for missing packets, the signal is given to reload the mote code.

There are no security mechanisms present in the above protocol; so any mote will accept reprogramming from any broadcaster. It is clear that to provide secure operations, a mote must be able to perform some sort of identity verification of the broadcaster before accepting new code; this type of functionality is similar to what is provided by TinyPK.

In AuthXNP, the verification steps of TinyPK (as shown in Figure 1) are inserted after the XNP initialization pings. As with TinyPK, all motes are loaded with the CA public key, allowing them to verify the signature on the broadcaster's private key. Once the verification step is complete (and optionally, the TinySec key loaded by the broadcaster), the rest of XNP proceeds.

7. RELATED WORK

TinyPK currently relies on conventional modular arithmetic cryptosystems. There are several options for more energy efficient cryptosystems, including Efficient and Compact Subgroup Trace Representation (XTR) [14], NTRU [2], and Elliptic Curve Cryptography (ECC) [9,15]. Currently, ECC shows the most promise. There are also some concerns with using the newer cryptosystems related to patents and, in a few cases, some lingering questions about the inherent security of the new approaches.

The University of California at Berkeley has developed two security protocols, Secure Network Encryption Protocol (SNEP) and Micro Timed Efficient Stream Loss-tolerant Authentication (μTESLA) as part of the SPINS project [16]. The SPINS protocols provide point-to-point confidentiality/integrity protection and broadcast message authentication for MICA mote networks. Since these protocols are specially designed for mote networks, they are tailored to meet the resource constraints and performance requirements of hardware/software components. SNEP is one of several available symmetric encryption routines on MICA platforms. μTESLA has the drawback of requiring a shared symmetric key between a base station and each of the sensor nodes. This configuration is difficult to guarantee in an ad hoc network.

Kong et.al [13] define an RSA-based security architecture for ad hoc networks. Their model requires individual nodes that are powerful enough to perform conventional RSA private operations. They employ a secret sharing routine that requires critical operations to be approved by a group of nodes rather than just one. This provides a form of fault tolerance but at a high static cost. It would be more desirable to have a lower-cost, gracefully degrading system of protection, rather than a high-cost system that perfectly tolerates K faults and provides no service guarantees at all when K+1 faults occur.

Under the DARPA Sensor Information Technology (SensIT) program, NAI Laboratories (now McAfee Laboratories) developed a sensor network communication security architecture and a suite of nano-cryptographic mechanisms [5]. The NAI report contains many useful ideas but the SensIT program was based on more fully functional sensor nodes than the MICA

motes. Many of the functions developed by NAI require advanced functionality on the nodes and thus are not directly comparable to our results.

8. CONCLUSIONS

The TinyPK system demonstrates that a public-key based protocol is feasible for an extremely lightweight sensor network. Incorporating the use of TinySec or any other symmetric encryption service for mote networks, TinyPK provides the functionality needed for a mote and a third-party to mutually authenticate to each other and to communicate securely. Our mote authentication can be used either to collect evidence that a mote field is valid or to validate a specific mote and to link future traffic to that mote. Our notion of a mote credential makes the task of a spoofing a mote network much more difficult than in an environment without credentials. In our protected environment, a single stolen and reverse engineered mote cannot be used to impersonate other motes with different credentials. This level of protection is achieved with very little overhead and has been shown to operate on the most limiting of sensor network platforms.

Our future work will study the problem of supporting mote networks that employ multiple session keys. As mote networks scale to larger sizes, the use of multiple session keys will be inevitable. Mote networks will need to internally generate new keys and deploy them as the communication patterns in the network change.

9. ACKNOWLEDGMENTS

This research is part of the DARPA Network Embedded Software Technology (NEST) Program. Dr Vijay Raghavan is the DARPA program manager and Mr. Roger Dziegiel Jr. is the Air Force Research Laboratory (AFRL) technical point of contact.

10. REFERENCES

[1] B. Atwood, B. Warneke, K.S.J. Pister, “Preliminary Circuits for Smart Dust,” Proceedings of the 2000 Southwest Symposium on Mixed-Signal Design, San Diego, California, February 27-29, 2000, pp. 87-92.

[2] D.V. Bailey, D. Coffin, A. Elbirt, J. H. Silverman and A.D. Woodbury, “NTRU in Constrained Devices,” in Proceedings of 2001 Conference of Cryptographic Hardware and Embedded Systems (CHES), Lecture Notes in Computer Science #2152, pp. 262–272.

[3] D. Boneh, “Twenty years of attacks on the RSA cryptosystem,” Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203-213, 1999.

[4] D. Boneh and H. Shacham, “Fast variants of RSA,” in RSA Laboratories’ Cryptobytes, vol 5 no. 1, pages 1-8, Winter/Spring 2002.

[5] D. Carman, P. Kruus, and B. Matt, “Constraints and Approaches for Distributed Sensor Network Security,” NAI Labs, NAI Labs Technical Report #00-010, 1 September 2001.

[6] Crossbow Technology, Inc., “Mote In-Network Programming User Reference,” http://www.xbow.com/Support/Support_pdf_files/Xnp.pdf.

[7] J. Deng, R. Han, and S. Mishra, “A Performance Evaluation of Intrusion-Tolerant Routing in Wireless Sensor Networks,” in F. Zhao and L. Guibas (Eds.), IPSN 2003, LNCS 2634, Springer-Verlag, pp.349 –364, 2003.

[8] W. Diffie and M.E. Hellman, “New Directions in Cryptography,” IEEE Transactions on Information Theory, vol. IT-22, no. 6, pp. 644-654", 1976.

[9] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, “Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs,” Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004), Boston, August 2004.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K.S.J. Pister, “System Architecture Directions for Networked Sensors,” in Architectural Support for Programming Languages and Operating Systems, pp. 93-104, 2000.

[11] M. Horton, D. Culler, K.S.J. Pister, J. Hill, R. Szewczyk, and A. Woo, “MICA: The Commercialization of Microsensor Motes,” *Sensor*, April 2002.

[12] C. Karlof, N. Sastry, and D. Wagner, “TinySec: A Link Layer Security Architecture for Wireless Sensor Networks,” to appear, Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004), Baltimore, MD, November 2004.

[13] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang, “Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks,” 9th International Conference on Network Protocols, Nov. 2001.

[14] A.K. Lenstra and E.R. Verheul, The XTR public key system, Proceedings Crypto 2000, LNCS 1880, Springer-Verlag, 2000.

[15] D. Malan, Crypto for Tiny Objects, TR-04-04, Computer Science Group, Harvard University, 2004.

[16] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J. D. Tygar, “SPINS: Security Protocols for Sensor Networks,” Proceedings of Seventh Annual International Conference on Mobile Computing and Networks (MOBICOM 2001), July 2001.

[17] R.L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” Communications of the ACM, vol. 21, no. 2, pp 120-126, February 1978.

[18] A. Wood and J. A. Stankovic, "Denial of Service in Sensor Networks," IEEE Computer, vol. 35, no. 10, pp 54-62, October 2002.