Energy Analysis of Public-Key Cryptography on Small Wireless Devices

Arvinderpal S. Wander, Nils Gura, Hans Eberle, Vipul Gupta, Sheueling Chang Shantz
Sun Microsystems Laboratories
arv_w@yahoo.com {Nils.Gura, Hans.Eberle, Vipul.Gupta, Sheueling.Chang}@sun.com

Abstract

For many applications of small wireless devices, secure communication is an important requirement. In this paper, we quantify the energy costs of authentication and key exchange based on public-key cryptography on an 8-bit microcontroller platform. We present a comparison of two public-key algorithms, RSA and Elliptic Curve Cryptography (ECC), under two authentication scenarios. In the first scenario, an untrusted client authenticates itself to a trusted authority. This scenario applies, for example, to a contactless smart card authenticating itself to a trusted reader. In the second scenario, we consider mutual authentication and key exchange between two untrusted parties such as two nodes in a wireless sensor network. For authentication and key exchange, we use a simplified version of the Secure Sockets Layer (SSL) handshake tailored to the constraints of small devices.

Our measurements on an Atmel ATmega128L low-power microcontroller indicate that public-key cryptography is viable on 8-bit energy-constrained devices even if implemented in software. For single-party authentication, we found that challenge-response authentication based on ECC-160 used only 1/12 of the energy of its RSA-1024 counterpart. With a given amount of energy, we were able to perform 4.2 times the number of key exchange operations (including mutual authentication) with ECC-160 compared to RSA-1024. The benefits of ECC over RSA manifested not only in less computation, but also in the amount of data transmitted and stored. While the relative cost of public-key cryptography depends on the application, we show that for applications that require infrequent authentication and key exchanges, the energy cost of public-key cryptography is minimal, if not negligible.

1. Introduction

The availability of inexpensive radio transceivers has enabled new types of applications and created new research challenges. Applications range from wireless sensor networks (WSNs) that can be used for health monitoring, industrial control, and building automation to smart cards that enable seamless user authentication and signing of digital documents. Depending on the application, these devices may not only exchange information locally with peers, but also globally with entities on the Internet. They are often deployed in large quantities and in environments where they may be exposed to tampering, eavesdropping, and attempts to modify transmitted data and insert unauthorized messages into the network. To counter such threats, flexible and effective mechanisms for secure communication are essential.

For many applications and devices under consideration, energy is a critical and limited resource. To assess

the energy demands of public-key cryptography, we quantify the energy cost[1] of optimized software implementations of public-key algorithms. Since the true benefits and accurate analysis of any particular algorithm is closely tied to how it is used within a security protocol, we also consider protocols that provide single-party and mutual authentication. In addition, we consider the impact of public-key cryptography on battery life and compare public-key cryptography to other factors influencing energy consumption, such as idle listening, data reception and transmission, symmetric cryptography, etc. Our analysis can be generalized to estimate the cost of public-key cryptography in other security protocols and applications with varying characteristics.

This paper is structured as follows. Section 2 gives an overview of related work. Section 3 covers cryptographic primitives and the two authentication scenarios we analyzed. Section 4 presents our energy analysis of the authentication scenarios. Finally, we summarize our conclusions in Section 5.

2. Related Work

Earlier work by Gura et al. [4] showed public-key cryptography to be computationally feasible on 8-bit devices. However, the paper only considers processing time and memory requirements and does not analyze energy costs or the application of public-key operations in security protocols.

Carman et al. [8] estimate the energy usage of several public and private-key algorithms. Similarly, Yuan and Qu [9] estimated energy requirements for RSA, DSA and ElGamal on various processors and proposed a power control mechanism for processors with dynamic voltage scaling (DVS) support. Potlapally et al. [10] analyzed the energy consumption of SSL on the Compaq iPAQ running a StrongARM processor. With the exception of Potlapally, the aforementioned papers only use estimates rather than actual implementations and do not consider ECC in their analysis. Potlapally et al. ran their analysis on a powerful 32-bit processor exceeding our targeted range for unit cost, energy consumption, and memory size.

Goodman and Chandrakasan [11] implemented an energy-efficient public-key coprocessor for RSA and ECC. Hardware-based crypto solutions for embedded and smart card applications are also available from ARM [16], but they come at a cost of larger die size and monetary cost.

Security in WSNs has lately received increased attention. However, in almost all cases, non-public-key-based key distribution and authentication schemes have been presented [12, 31, 32, 33, 34, 35], with the underlying assumption that public-key cryptography is too costly or impossible to incorporate in WSNs. Unfortunately, none of the schemes are capable of providing the flexibility and security offered by public-key-based solutions.

Non-public-key solutions for authentication employ, for example, one-time passwords [17] or keyed hash schemes [18], which inhibit flexible key distribution. In addition, non-repudiation, which is essential for signing of digital documents, cannot be provided.

3. Public-key Authentication

1 In this paper, cost always refers to the energy cost, unless stated otherwise.

We first introduce cryptographic primitives used to secure communications and then consider two scenarios where public-key cryptography can be applied to small devices. As a first scenario, we examine an untrusted client that authenticates itself to a trusted gateway. Smart cards are a good example, where the card – representing its holder – is the untrusted entity and must authenticate itself to a trusted reader. Second, we consider a scenario where two parties perform mutual authentication and key exchange. A typical field of application are WSNs, where two nodes authenticate each other and exchange keys for data encryption and decryption.

Cryptographic Primitives

Secure communication can be achieved by employing strong cryptography to ensure confidentiality (non-disclosure of secret information), integrity (prevention of data alteration), authentication (proof of identity), and non-repudiation (unique, non-contestable message origin). Modern encryption algorithms can be divided into two primary categories, symmetric and asymmetric. Symmetric-key (aka. private-key) algorithms such as AES, DES, and RC4, rely on a secret key shared by the communicating parties to encrypt and decrypt the transmitted data. Symmetric-key cryptography is computationally inexpensive, and can provide confidentiality and authentication based on challenge-response protocols. In combination with symmetric algorithms, cryptographic hash functions such as MD5 or SHA can provide integrity by generating unique "fingerprints" of messages. However, symmetric-key approaches are inflexible with respect to key management as they require pre-distribution of keys. For example, in a network consisting of n nodes, if each node needed to communicate securely with any one of the other nodes, a pairwise key distribution scheme would require n choose 2 keys at each node. Another option is to use one global key. However, the loss of this key would compromise security on the entire network.

Asymmetric-key (aka. public-key) algorithms use a pair of keys to encrypt and decrypt. The two keys are related mathematically; a message encrypted using one key can be decrypted by the same algorithm using the other key. Most Internet security protocols (e.g. SSL [1], IPsec) employ a public-key cryptosystem for authentication and to derive shared secret keys. These keys are then used in fast symmetric-key and hashing algorithms to ensure confidentiality, integrity and source authentication of bulk data. While public-key cryptography allows for flexible key management and authentication, it requires a significant amount of computation. However, the capabilities of light-weight devices are often limited in terms of energy, clock frequency, and memory size. RSA [2] is by far the most widely used public-key algorithm on the Internet today. However, ECC [3] offers equivalent security at much smaller key sizes. ECC is especially attractive for constrained wireless devices because the smaller keys result in memory, bandwidth and computational savings.

Public-key algorithms provide primitive operations that make higher level key exchange and authentication protocols possible. The primitive operation for RSA is modular exponentiation. Using RSA, a ciphertext C can be generated from a message M by computing $C = M^x \bmod N$, where the exponent x is either the public or private key. Party A can encrypt a secret for party B by using B's public key (e_B) as the exponent of M. Only B can decrypt the secret by using its private key (d_B) as the exponent of C. To generate a signature, A uses its private key (d_A) as the exponent of the data to be signed. The signature can be verified by using A's public key (e_A) as the exponent. Unlike

the private key, an RSA public key need not be very long; a 17-bit value (typically $2^{16} + 1$) is sufficient. Thus, verify and encrypt operations take significantly less time than sign and decrypt operations. A detailed description of the RSA algorithm, its parameters, and a mathematical proof of correctness can be found, for example, in [2]. The elliptic curve equivalent to RSA modular exponentiation is scalar point multiplication $Q = k * P$, where $P=(x_P,y_P)$ is a point on an elliptic curve, k is a (large) integer, and $Q=(x_Q,y_Q)$ is another point on the elliptic curve resulting from the point multiplication of k and P. A well-known key exchange algorithm for ECC is the Elliptic Curve Diffie-Hellman (ECDH) algorithm [5]. Two communicating parties A and B first agree on an elliptic curve and a base point G. They generate private keys k_A and k_B and the corresponding public keys $Q_A = k_A * G$ and $Q_B = k_B * G$. To perform the key exchange, both parties first exchange their public keys Q_A and Q_B. A computes $k_A * Q_B$ and B computes $k_B * Q_A$ such that both arrive at a common shared secret $S=k_A * Q_B = k_B * Q_A = k_A * k_B * G = k_B * k_A * G$. ECC-based signatures can be generated and verified with the Elliptic Curve Digital Signature Algorithm (ECDSA). A description of ECDSA would exceed the scope of this paper, but can be found in [6].

Digital certificates are commonly used to verify the identity of a party sending a message, and to provide the recipient with the means to encode an encrypted reply. A digital certificate issued by a certificate authority (CA) typically contains the applicant's public key, the CA's digital signature, and other fields such as issuer name, applicant name, expiration date, etc. The CA makes its own public key readily available so that a recipient of a digital certificate can use the CA's public key to verify the CA's signature on the certificate and then use the applicant's public key and identification information held within the certificate. For example, for mutual authentication of arbitrary nodes in a WSN, each node only needs to store the CA's public key in addition to its own private key and certificate. The most widely used standard for digital certificates is X.509 [7].

Due to expected advances in cryptanalysis and increases in computing power available to an adversary, both symmetric and public-key sizes must grow over time to offer acceptable security for a fixed protection life span. RSA with 1024-bit keys (RSA-1024) is the currently accepted security level, and is equivalent in strength to ECC with 160-bit keys (ECC-160). To protect data beyond the year 2010, RSA Security recommends RSA-2048 (equivalent to ECC-224) as the new minimum key size [37]. Symmetric key sizes in use today are mostly 128 bits or greater.

Single-Party Authentication: Smart Cards

In this section we describe a protocol based on public-key cryptography, with which an untrusted party authenticates to a trusted party. While we talk about authentication in the context of smart cards, the underlying protocol for authentication can be used in other applications as well.

Modern smart cards are suitable for cryptographic implementations as they contain security features that enable the protection of sensitive cryptographic data and provide for a secure processing environment. Applications of smart cards range from authenticated access to facilities to signing of digital documents and national IDs [15]. For our analysis we assume contactless smart cards powered by internal batteries (aka. active cards), which are available from several vendors [13, 14].

We consider an authentication protocol using a challenge-response scheme for both ECC and RSA as shown in Figure 1. The reader sends a challenge to the nearby smart card, where the challenge is a random number or a combination of known data and a random number. The smart card signs the challenge with its private key and transmits its ID and the signature back to the reader. The reader can use this information along with the card's public key to verify the signature, which establishes the authenticity of the card, and thus its holder. Figure 1 assumes that the card's public key is available to the reader from a trusted directory containing a table of card IDs and their corresponding public keys. Alternatively, the card can transmit a certificate containing its public key (signed by a trusted CA) to the reader. In this case, the reader has to both verify the signature on the certificate and the signature on the challenge.

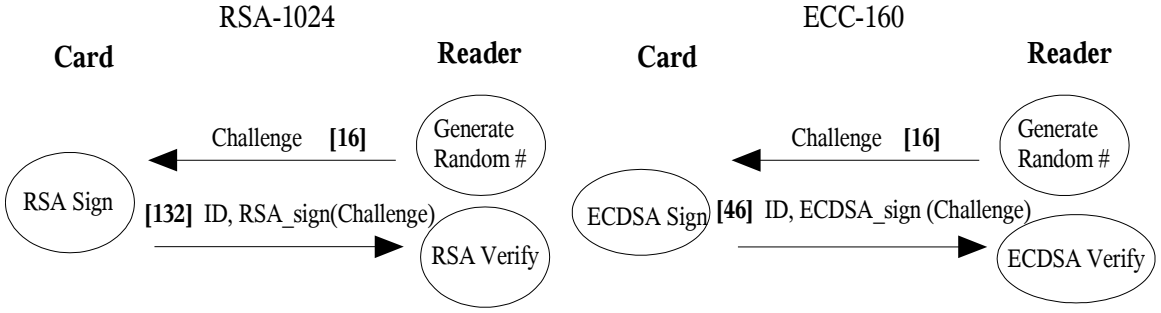

Figure 1: RSA and ECC-based authentication protocols for smart cards. Payload message sizes, in bytes, are shown in brackets.

Mutual Authentication and Key Exchange: WSNs

Two parties operating in an open environment that potentially includes adversaries seeking to impersonate or eavesdrop on communication, will want a mechanism to separate friends from foes and keep the content of their conversations secret. Mutual authentication and key exchange using public-key cryptography is a well-known mechanism that enables exactly that.

A pervasive network, such as a WSN, deployed in an untrusted environment is one examplary area for public-key authentication. Numerous applications of WSNs including health monitoring, military surveillance, industrial control, and building automation require some level of security. Since the number of nodes deployed can range from a handful to thousands, if not millions, flexible key distribution is essential, which is a hallmark of public-key cryptography.

The mutual authentication and key exchange protocol presented below is a simplified version of the SSL handshake [1]. Using SSL terminology, we refer to the party initiating the communication as the client and to the responding party as the server. Figure 2 shows the exchange of data between the client and the server during an SSL handshake process for both RSA-based and ECC-based algorithms.

In the first message (C.Rand), the client indicates the versions of SSL it implements and provides a random value later used for key derivation as well as an ordered list of supported asymmetric, symmetric, and hashing algorithms known as *cipher suites*. For constrained devices, a single cipher suite may be used such as ECDH-

ECDSA-AES-SHA1 or RSA-AES-SHA1. With the second message, the server selects one of the proposed cipher suites and responds to the client with a 32-byte random value and its ECC or RSA-based certificate. If RSA is used, the client must perform an RSA verify operation to verify the server's certificate, an RSA encrypt operation with the server's public key to encrypt a random 48-byte secret key, and an RSA sign operation on a derivate of the 32-byte random number proving possession of its private key. If ECC is used, the client performs an ECDSA verify operation to verify the server's certificate and an ECDH operation to calculate the shared secret. The third message, sent from the client to the server, includes the client's certificate and a finished message, which is the first symmetrically encrypted message based on the shared secret. If RSA is used, the third message also includes the encrypted secret key and the client's signature. In the RSA case, the server can now verify the client's identity by verifying the client's signature and prove its identity to the client by being able to decrypt the secret key with its private key. In the ECDH case, the server computes the shared secret using ECDH and performs an ECDSA operation to verify the client's certificate. Mutual authentication is achieved through mutual possession of the shared secret (client and server will compute a different ECDH secret if one party's public and private key do not match). For both RSA and ECC-handshakes, the server ends the handshake with a finished message containing a keyed hash of the data exchanged thus far.

To conserve energy, we reduced the amount of data exchanged in a typical Internet-based SSL handshake. We assume the WSN administered by a single organization such that many of the parameters can be fixed a priori and need not be negotiated in the handshake (e.g. cipher suite and protocol version).

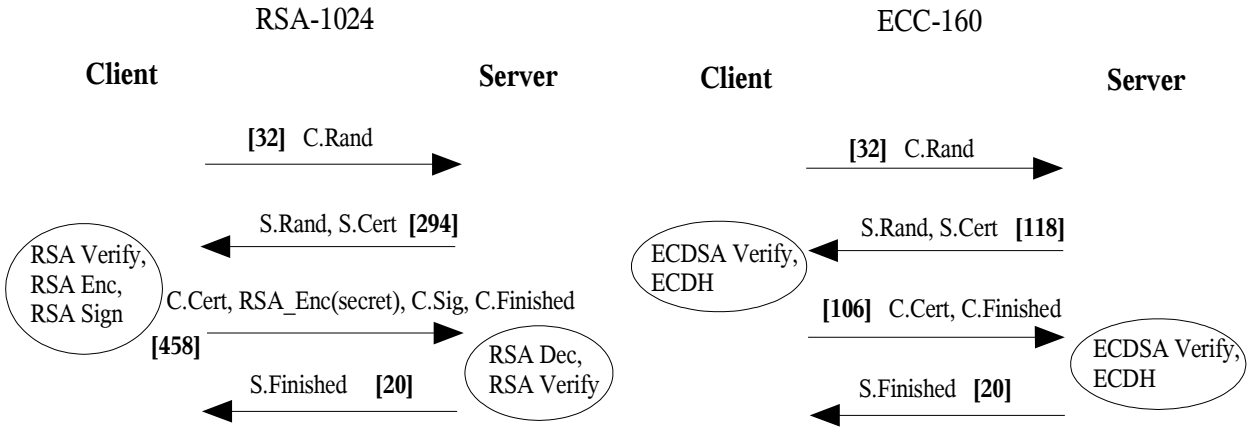

Figure 2: Simplified RSA and ECC-based SSL handshake with mutual authentication. Payload message sizes, in bytes, are shown in brackets.

SSL specifies the use of X.509 certificates, which contain fields for validity period, holder's public key, CA signature, etc. Since we assume that all certificates are issued by the same authority for a common validity period, which may even be the life time of the WSN, a simplified certificate representation can be used. Without compromising the security of the handshake, certificates can be shortened to only contain a unique device ID, a public key and a signature (see Appendix A). Depending on the individual fields, a traditional X.509 RSA-1024

certificate is on the order of 700 bytes long, while the simplified certificate is only 262 bytes long. The difference is even more significant for ECC-160, for which a certificate is reduced from approximately 530 bytes to 86 bytes.

4. Energy Analysis

We conducted our experiments on the Berkeley/Crossbow motes platform, specifically on the Mica2dots [19], which are a popular platform for WSN research. The major energy consumers on these sensor devices are the Atmel ATmega128L [20] 8-bit microcontroller and the Chipcon CC1000 [21] low-power wireless transceiver. The Atmega128L contains 128kB of FLASH program memory, 4 kB of data memory and runs at a clock frequency of 4 MHz. We supplied the Mica2dot with an operating voltage of 3 Volts and configured the wireless transceiver to use a carrier frequency of 915 MHz and the maximum transmission power of 3 mW (5 dBM). Our program code was written in a combination of nesC [22], C and assembly language, where optimized assembly code was used to implement RSA and standardized ECC over integer fields GF(p), as presented in [4].

We found that while the ATmega128L is designed for low power applications, it does not employ techniques such as gated clocks to shut down processor modules that are temporarily unused. Therefore, we approximated the energy consumption for individual cryptographic algorithms and other activities such as data transmission by measuring the current drawn from the power supply. An earlier, more accurate approach using an oscilloscope and a sense resistor showed the error to be less than 5%.

Table 1 presents characteristic data for the Mica2dot platform that we measured and calculated. It is interesting to note that the power required to transmit 1 bit is equivalent to roughly 2090 clock cycles of execution on the microcontroller alone. This confirmed our assumption that the energy cost of computation is cheap compared to data transmission. The cost of receiving one byte (28.6µJ) is roughly half of that required to transmit a byte (59.2 µJ). During transmit and receive, the microcontroller is powered on along with the wireless transceiver. We used a packet size of 41 bytes, 32 for the payload and 9 bytes for the header. The header, ensuing a 8-byte preamble, consists of source, destination, length, packet ID, CRC, and a control byte. Receiving one 41-byte packet (including 8-byte preamble) costs 49*28.6uJ=1.40mJ and transmitting one 41-byte packet costs 49*59.2uJ=2.90mJ.

Field	Value
Effective data rate	12.4 kbps
Energy to transmit	59.2 µJ/byte
Energy to receive	28.6 µJ/byte
ATmega128L active mode	13.8 mW
ATmega128L power down mode	.0075 mW
ATmega128L MIPS/Watt	289 MIPS/W

Table 1: Characteristic data for the Mica2dot sensor platform.

Energy Cost of Primitive RSA and ECC Operations

Table 2 compares the energy consumed by RSA and ECC for generating and verifying signatures. While the cost of an RSA verify is small, it is overshadowed by the more expensive sign operation, both of which are required

for authentication. In comparison, ECDSA[2] signatures are significantly cheaper than RSA signatures and ECDSA verifications are within reasonable range of RSA verification. Note that when transitioning from RSA-1024 to RSA-2048 the energy cost of signing increases by a factor of more than seven, while ECDSA-224 signing is less than three times as expensive as ECDSA-160 signing. To put these numbers into perspective, one RSA-1024 sign operation is equivalent to transmitting 5,132 bytes, compared to 385 bytes for an ECDSA-160 sign operation.

Algorithm	Sign [mJ]	Verify[mJ]
RSA-1024	304	11.9
ECDSA-160	22.82	45.09[3]
RSA-2048	2302.7	53.7
ECDSA-224	61.54	121.98[3]

Table 2: Energy cost of digital signature computations.

Table 3 compares the energy cost of key exchanges not including authentication and certificate verification. The RSA-based key exchange protocol relies on party A to encrypt a randomly generated secret key with party B's public key, and party B decrypting the key using its private key. With ECC, both parties perform a single ECDH operation to derive the secret key.

Algorithm	Client [mJ]	Server [mJ]
RSA-1024	15.4	304
ECC-160 ECDH	22.3	22.3
RSA-2048	57.2	2302.7
ECC-224 ECDH	60.4	60.4

Table 3: Energy cost of key exchange computations.

We do not present the cost of key generation. However, we note that for ECC, key generation only involves generating a random number, which becomes the user's private key, and executing an ECDH operation to compute the corresponding public key. RSA key generation is much more time consuming as it requires the generation of large prime numbers. For details, we refer the reader to [23].

Energy Cost of Symmetric-Key and Hash Algorithms

For our analysis we chose to focus on AES with 128-bit keys for data encryption/decryption and SHA-1 for hashing. We used an assembly-optimized AES implementation [24] and a C-implementation of SHA-1 [25]. Table 4 contains data for both implementations. The cost of block encryption and hashing is cheap compared to public-key operations and wireless communication. For example, the energy required for an RSA-1024 and ECC-160 signature operation is equivalent to encrypting 191kB and 14kB of data with AES-128, respectively. Similarly, the cost to encrypt one byte is only 2.7% to that of transmitting one byte. For a more comprehensive discussion of energy

2 We estimated the energy numbers for ECDSA based on our measurements of ECC point multiplication and integer inversion. In addition, ECDSA requires hashing, integer addition and multiplication operations, all of which we expect to consume only a minimal amount of energy.

3 Our ECDSA numbers assume that a signature verification requires two point multiplications. Known optimizations can reduce this to roughly 1.2 point multiplications leading to further reduction in energy usage.

consumption of symmetric ciphers and hash algorithms see [8, 10].

Algorithm	Energy
SHA-1	5.9 µJ/byte
AES-128 Enc/Dec	1.62/2.49 µJ/byte

Table 4: Energy numbers for AES with 128-bit keys and SHA-1. AES-128 numbers include key-setup. The numbers were averaged over inputs ranging from 64 to 1024 bytes.

Authentication Scenarios

In this section we present a detailed analysis of the cost of security in the context of the two authentication scenarios described in Section 3. Assuming unique device IDs of 4 bytes and a 16-byte challenge as shown in Figure 1, the smart card requires 25.5mJ for authentication with ECDSA-160 and 311.8mJ with RSA-1024[4]. Figure 3(a) illustrates the energy consumption for a varying number of authentications. Figure 3(b) plots the operational lifetime of a smart card based on the number of daily authentication operations. If on average one authentication is performed each day, a battery with a minute 30 mAh capacity could theoretically[5] last almost 3 years with RSA-1024, but over 34 years with ECDSA-160. The 30 mAh capacity number was taken from one of the smallest commercially available lithium coin batteries, CR1025 [26].

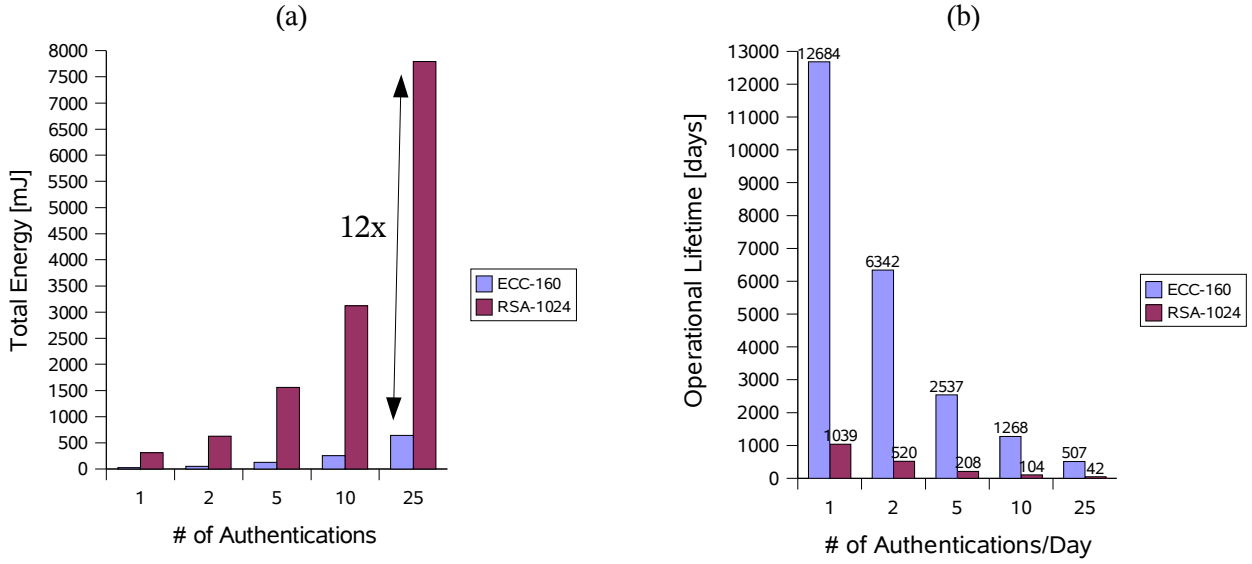

Figure 3: Energy consumption for single-party authentication: (a) Total energy consumed by public-key operations and data communication. (b) Operational lifetime versus number of daily authentications.

Single-party authentication favors the use of ECC over RSA because the primary purpose of the smart card is to generate signatures, for which ECC is far more efficient. Since the amount of exchanged data is small, the cost

4 We only account for the payload data sent and received and assume that the smart card is active (powered on) only when it comes into close proximity to a reader.

5 There are several other factors that influence battery life. However, for simplicity we assume the battery will provide the specified capacity.

of communication has little impact on overall energy consumption; public-key computations account for 89.3% of the consumed energy for ECDSA-160 and for 97.5% of the energy for RSA-1024.

Next, we consider the mutual authentication and key exchange handshake for nodes operating in a WSN. In order to realistically quantify the impact of public-key cryptography on battery life, we first start by determining the factors that influence energy consumption and then present our analysis.

For most applications of WSNs, energy is a scarce resource that is inconvenient or impossible to replenish. For applications that do not require secure communication, three factors dominate the overall energy consumed; idle listening, application-specific operations and communication. Secure communication through public-key-based authentication and key exchange introduces two new factors: bulk data encryption/decryption including hashing and public-key operations including communication. We briefly discuss these factors.

Considerable research has gone into designing low-power media access control (MAC) protocols [27, 28], specifically towards duty cycle mechanisms seeking to eliminate idle listening. Duty cycles (i.e. receive time / sleep time) as low as .1% are often used [29, 30], while still being flexible enough to efficiently transfer different workloads and adapt to changing network conditions.

Application-specific computation and communication in most WSNs today ranges from polling sensors to data aggregation and message forwarding. The amount of energy spent on communication is likely to vary based on the functionality and placement of the node within a network. For example, a node serving as an intermediate hop between a base station is likely to experience much more traffic than a node at the edge. Similarly, the amount of energy spent on symmetric encryption/decryption will vary based on traffic characteristics.

The cost of a public-key security protocol can be broken down into computational cost (e.g. an ECDH operation) and communication cost (e.g. sending/receiving a certificate). The amount of energy spent can be minimized by choosing the best algorithm(s), reducing the communication overhead of the security protocol, and limiting the number of authentication and key exchange handshakes. A network consisting of stationary nodes will likely require handshakes to exchange secret keys at deployment time; symmetric encryption can be used thereafter. Handshakes will be more frequent when nodes are mobile such as in ad-hoc networks and dynamic WSNs. In these cases, a demand-based authentication scheme may be more appropriate. For example, a node may discover its neighbors, but perform a handshake only when it has confidential data to send/receive or otherwise remain dormant.

We analyze the energy usage of the simplified SSL handshake based on RSA-1024 and ECC-160. The amount of combined energy spent by client and server is determined by public-key computation, transmitting and receiving handshake messages, hash computation, and random number generation[6]. The first three factors depend on the public-key algorithm used. Table 5 presents the energy consumed by the entire handshake for RSA-1024 and ECC-160. Figure 4(a) shows the percentage of energy spent on each major part of the handshake process. For both RSA-1024 and ECC-160, the public-key computation dominates, consuming 82% and 72% of the energy, respectively. Communication costs are second, while random number generation and hashing costs are negligible.

6 We did not implement a random number generator. Based on commonly known algorithms we assume that a number that meets the randomness criteria can be generated in a quarter of a second.

Figure 4(b) shows the ratio of the energy consumed using ECC versus RSA for each part of the handshake process. The computational benefits of ECC-160 over RSA-1024 are apparent, where the RSA-1024 computations consume 4.9 times the energy of ECC-160 computations. Compared to ECC-160, an RSA-1024 handshake also consumes 2.7 times the energy in transmitting and receiving data. Communication costs for RSA-1024 are higher because of the longer key sizes, thus making the certificates larger as well. With RSA-1024, the entire handshake requires the client to transmit 490 bytes of payload and the server to transmit 314 bytes of payload. With ECC-160, both parties transmit the same amount of payload data, 138 bytes.

Algorithm	Client [mJ]	Server [mJ]
RSA-1024	397.7	390.3
ECC-160	93.7	93.9

Table 5: Energy consumption of handshake protocol based on RSA-1024 and ECC-160.

(a) (b)

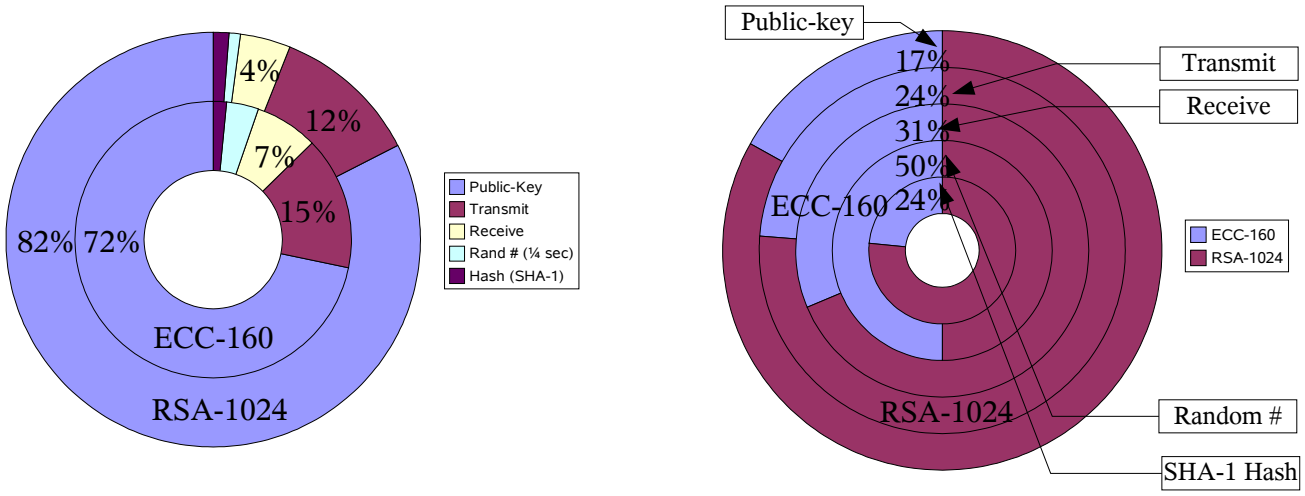

Figure 4: (a) Decomposition of energy for mutual authentication and key exchange on the client side.
(b) Relative energy consumption on the client side. Percentages represent energy ratio E(ECC)/(E(ECC)+E(RSA)).

Figure 5 shows the number of handshakes possible as we vary the battery capacity from 30mAh to 1000 mAh and only a certain fraction, 5% or 10%, of the overall capacity is available for handshakes. The capacities reflect those of common coin-based watch and general purpose batteries [26]. From the figure we can see that even with 5% of the energy available from a miniature 30 mAh battery, a node can perform 173 ECC-160 and 41 RSA-1024 handshakes.

Next, we examine the cost of the handshake in a likely usage scenario. Consider a newly deployed WSN consisting of an arbitrary number of nodes. Each node has been assigned a public and private key pair along with a digital certificate issued by a trusted certificate authority (CA) and the CA's public key. After deployment, nodes authenticate themselves to their immediate neighbors and exchange secret keys for data encryption/decryption, for

example, using the simplified SSL handshake described earlier. The next logical step would be to form a mesh network, where all nodes, if possible, can communicate with one or more base stations. Having formed a mesh network, the nodes can focus on application-specific tasks.

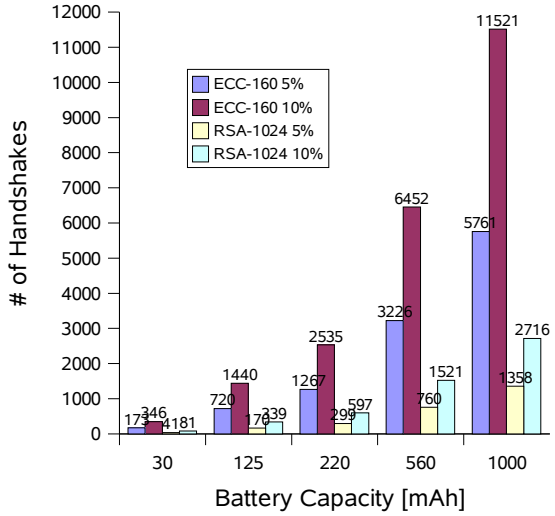

Figure 5: Number of handshakes when only a 5% or 10% fraction of the battery capacity is available.

After completing the handshakes, nodes use the agreed-upon secret keys for encrypting/decrypting all or parts of the application data. The secret keys may be used for the lifetime of the nodes or while the nodes remain in communication range. Figure 6(a) plots the percentage of energy consumed by a single handshake versus the number of bytes that have been transmitted after being encrypted by the secret key. The initial cost of the ECC-160 handshake falls below 10% after transmitting 16kB and is almost negligible (~2%) after 64kB have been transmitted. As expected, the relative cost of an RSA-1024 handshake takes longer to fall. While the number of bytes encrypted and transmitted per handshake will vary based on the application, nodes near the base station will generally experience more traffic, thus the cost of handshakes at these nodes will likely be small.

As was mentioned earlier, the fraction of energy spent on listening over an idle channel can be significant. Ignoring other factors such as application-specific computation and communication, Figure 6(b) considers three nodes with varying duty cycles of .1%, .5% and 1%, where each node performs five handshakes over a one day period. In all three cases, the five ECC-160 based handshakes represent less then 11% of the energy consumed per day.

The above results indicate that for applications where handshakes are relatively infrequent the cost of the handshakes is negligible. The fraction of energy spent on idle listening, application-specific computation and communication will likely dominate over the initial cost of the handshake. When considering which public-key algorithm to use, ECC is a better fit for this class of devices. In addition to consuming significantly less energy, the execution time and memory requirements for ECC are also much lower compared to RSA. For example, an ECC-160 point multiplication takes just 1.61 seconds and 282 bytes of data memory, while an RSA-1024 private-key modular exponentiation takes nearly 22 seconds and 930 bytes of data memory [4].

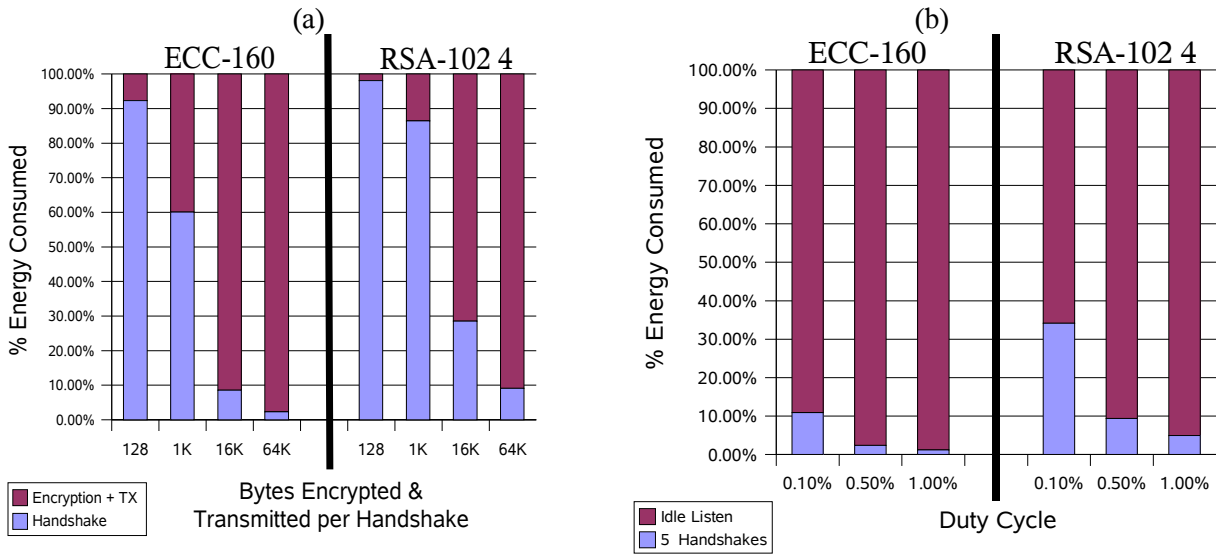

Figure 6: (a) Comparison of energy for handshake and subsequent bulk data encryption and transmission.
(b) Comparison of energy spent on idle listening and energy spent on five handshakes over a one day period.

5. Conclusions

We analyzed the energy consumption of a single-party authentication protocol and a mutual authentication and key exchange protocol in the context of two likely applications. Contrary to widely held beliefs [31, 32, 33, 34, 35], our results indicate that authentication and key exchange protocols using optimized software implementations of public-key cryptography are very viable on small wireless devices. Depending on the frequency of public-key computations, its relative energy cost may even be negligible. Furthermore, our analysis suggests that the use of ECC over RSA can lead to significant energy savings. In addition to the computational benefits of ECC, its smaller keys and certificates lead to significant savings in public-key communication costs.

For single-party authentication, we found that challenge-response authentication based on ECC-160 used only 1/12 of the energy of its RSA-1024 counterpart. With a given amount of energy, we were able to perform 4.2 times the number of key exchange operations (including mutual authentication) with ECC-160 compared to RSA-1024. While such absolute numbers are platform-specific, we expect the computational cost to fall faster than the cost to transmit and receive. For example, ultra-low-power microcontrollers such as the 16-bit Texas Instruments MSP430 [36] can execute the same number of instructions at less than half the power required by the 8-bit ATmega128L. A similar analysis conducted on such platforms is likely to show communication costs representing a larger fraction of the overall energy spent on authentication and key exchange protocols. The benefits of transmitting smaller ECC keys and certificates will in turn be more significant.

In addition to flexible key exchange and peer authentication, public-key cryptography can be the enabling technology for numerous other WSN applications, including securely connecting pervasive devices to the Internet and distributing signed software patches.

Acknowledgments

We thank Matt Millard for setting up the Mica2 motes and Arun Patel for providing AES and SHA-1 numbers.

References

[1] A. Freier, P. Karlton and P. Kocher, "*The SSL Protocol Version 3.0*", see http://home.netscape.com/eng/ssl3/

[2] C. K. Koc, "*High-speed RSA implementation*", Tech. Rep. TR 201, RSA Laboratories, November 1994.

[3] D. Hankerson, A. Menezes, S. Vanstone, "*Guide to Elliptic Curve Cryptography*", Springer-Verlag New York, Inc. 2004. ISBN 0-387-95273-X.

[4] N. Gura, A. Patel, A. Wander, H. Eberle, S. Chang Shantz, "*Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs*", Cryptographic Hardware and Embedded Systems (CHES), August 2004.

[5] ANSI X9.63, "*The Elliptic Curve Key Agreement and Key Transport Protocols*", American Bankers Association, 1999.

[6] ANSI X9.62, "*The Elliptic Curve Digital Signature Algorithm (ECDSA)*", American Bankers Association, 1999.

[7] Public-key Infrastructure (X.509), see http://www.ietf.org/html.charters/pkix-charter.html

[8] D. W. Carman, P. S. Kruus, and B. J. Matt, "*Constraints and Approaches for Distributed Sensor Security*", Network Associates Labs Tech. Rep. 00-010, 2000.

[9] L. Yuan and G. Qu, "*Design Space Exploration for Energy-Efficient Secure Sensor Network*", Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'02), 2002.

[10] N. Potlapally, S. Ravi, A. Raghunathan and N. K. Jha, "*Analyzing the Energy Consumption of Security Protocols*", International Symposium on Low Power Electronics and Design (ISLPED), August 2003.

[11] J. Goodman and A. Chandrakasan, "*An Energy-Efficient Reconfigurable Public-Key Cryptography Processor*", IEEE Journal of Solid-State Circuits, Vol. 36 No. 11, November 2001.

[12] A. Perrig, J. Stankovic and D. Wagner, "*Security in Wireless Sensor Networks*", Communications of the ACM, Vol. 47 No. 6, June 2004.

[13] Philips Semiconductors, 8-bit MIFARE® PROX Dual Interface Smart Card Controllers, http://www.semiconductors.philips.com/markets/identification/products/prox/

[14] Alien Technology Corp., "*Battery Assisted Passive Hardware",* see http://www.alientechnology.com/products/rfid-battery/index.php

[15] H. Leitold, A. Hollosi and R. Posch, "*Security Architecture of the Austrian Citizen Card Concept*", 18th Annual Computer Security Applications Conference, December 2002.

[16] ARM SecureCore Family, see http://www.arm.com/products/CPUs/families/SecurCoreFamily.html

[17] N. Haller, C. Metz, P. Nesser, M. Straw, "*A One-Time Password System*", IETF Request for Comments 2289. February 1998.

[18] H. Krawczyk, M. Bellare, R. Canetti, "*HMAC: Keyed-Hashing for Message Authentication*", IETF Request for Comments 2104, February 1997.

[19] Crossbow Technology Inc., *Processor/Radio Modules*, see http://www.xbow.com/Products/productsdetails.aspx?sid=62

[20] Atmel Corporation. http://www.atmel.com

[21] Chipcon AS. http://www.chipcon.com

[22] E. Brewer, D. Gay, P. Levis, R. von Behren, and M. Walsh, "*NesC: A Programming Language for Deeply Networked Systems*", see http://nescc.sourceforge.net/

[23] A. Juels and J. Guajardo, "*RSA Key Generation with Verifiable Randomness*", RSA Laboratories, see http://www.rsasecurity.com/rsalabs/node.asp?id=2041

[24] C. Röpke, W. Urowski and K. Tellman, "*AES assembly implementation for the AVR instruction set"*, see http://www.christianroepke.de/praktikum_b.html

[25] D. Eastlake, P. Jones, "*US Secure Hash Algorithm 1 (SHA1)*", IETF Request for Comments 3174, September 2001.

[26] Matsushita Electric Industrial Co., Ltd. "*Manganese dioxide lithium batteries(CR series)",* see http://industrial.panasonic.com/www-cgi/jvcr21pz.cgi?E+BA+3+AAA4003+4

[27] W. Ye, J. Heidemann, and D. Estrin, "*An Energy-efficient MAC protocol for wireless sensor networks", I*n Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM 2002), New York, NY, June 2002.

[28] J. Polastre, J. Hill, D. Culler, "*Versatile Low Power Media Access for Wireless Sensor Networks*", ACM SenSys, 2004.

[29] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, "*Habitat Monitoring: Application Driver for Wireless Communications Technology",* ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, San Jose, Costa Rica, April 2001.

[30] M. Hamilton, M. Allen, D. Estrin, J. Rottenberry, P. Rundel, M. Srivastava, and S. Soatto. "*Extensible Sensing System: An advanced Network Design for Microclimate Sensing*", June 2003, see http://www.cens.ucla.edu

[31] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. "*SPINS: Security protocols for sensor networks",* Seventh Annual ACM International Conference on Mobile Computing and Networks (MobiCom 2001). July 2001.

[32] T. Park, K. G. Shin, *"LiSP: A lightweight security protocol for wireless sensor networks",* ACM Transactions on Embedded Computing Systems (TECS), August 2004.

[33] D. Liu, P. Ning. "*Location-based pairwise key establishments for static sensor networks",* Proceedings of the 1st ACM workshop on Security of Ad-hoc and Sensor Networks, 2003.

[34] L. Eschenauer and V. Gligor, "*A key-management scheme for distributed sensor networks",* Proceedings of

the 9th ACM Conference on Computer and Communications Security, 2002.

[35] C. Karlof, N. Sastry, D. Wagner, "*TinySec: A Link Layer Security Architecture for Wireless Sensor Networks*", ACM Sensys, November 2004

[36] Texas Instruments Inc., "*MSP430 Family of Ultra-low-power 16-bit RISC Processors"*, see http://www.ti.com

[37] B. Kaliski, *"TWIRL and RSA Key Size"*, RSA Laboratories Technical Note, May 2003, see http://www.rsasecurity.com/rsalabs/node.asp?id=2004

Appendix A

Table 6 shows the required fields of a X.509 certificate. The three fields we kept for the simplified SSL handshake are underlined. We assume that a single authority is responsible for managing the network and that all certificates have a fixed validity period. This applies, for example, to a WSN deployed by a single organization where certificates are valid for the lifetime of the network. This simplification enables us to remove the following fields: issuer name, issuer's ID, and validity period. Given the limited memory on constrained devices, we assume that they implement one specific cipher suite and signature algorithm, for example, ECC-160 with AES-128 and SHA-1. That is, the signature algorithm field can also be removed. Serial numbers are generally used for certificate revocation, which we do not consider. Finally, we assume that a subject's ID is sufficient for identification and that all entities in the network implement the same certificate version. Table 7 shows the fields and their sizes for the RSA-1024 and ECC-160 certificates we used in our analysis.

Version Number	Serial Number	Signature Algo.	Issuer Name	Validity Period
Subject Name	Subject's Public Key	Subject ID	Issuer ID	Signature

Table 6: Fields required by a X.509-compliant (v. 3) certificate, optional extension fields are ignored.

Fields RSA-1024	***Size [Bytes]***	***Description***
Unique ID	4	ID of the node to which the certificate was issued.
Modulus	128	RSA modulus.
Exponent	3	RSA public-key exponent.
Signature	128	RSA signature from a certificate authority (CA).
Fields ECC-160	***Size [Bytes]***	***Description***
Unique ID	4	ID of the node to which the certificate was issued.
EC Public Key	40	Elliptic curve point serving as the public key of the subject.
ECDSA-Signature	42	ECDSA signature from a CA.

Table 7: Certificate formats for simplified RSA-1024 and ECC-160 certificates.