A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols

Nachiketh R. Potlapally, *Student Member*, *IEEE*, Srivaths Ravi, *Member*, *IEEE*, Anand Raghunathan, *Senior Member*, *IEEE*, and Niraj K. Jha, *Fellow*, *IEEE*

Abstract—Security is becoming an everyday concern for a wide range of electronic systems that manipulate, communicate, and store sensitive data. An important and emerging category of such electronic systems are battery-powered mobile appliances, such as personal digital assistants (PDAs) and cell phones, which are severely constrained in the resources they possess, namely, processor, battery, and memory. This work focuses on one important constraint of such devices—battery life—and examines how it is impacted by the use of various security mechanisms. In this paper, we first present a comprehensive analysis of the energy requirements of a wide range of cryptographic algorithms that form the building blocks of security mechanisms such as security protocols. We then study the energy consumption requirements of the most popular transport-layer security protocol: Secure Sockets Layer (SSL). We investigate the impact of various parameters at the protocol level (such as cipher suites, authentication mechanisms, and transaction sizes, etc.) and the cryptographic algorithm level (cipher modes, strength) on the overall energy consumption for secure data transactions. To our knowledge, this is the first comprehensive analysis of the energy requirements of SSL. For our studies, we have developed a measurement-based experimental testbed that consists of an iPAQ PDA connected to a wireless local area network (LAN) and running Linux, a PC-based data acquisition system for real-time current measurement, the OpenSSL implementation of the SSL protocol, and parameterizable SSL client and server test programs. Based on our results, we also discuss various opportunities for realizing energy-efficient implementations of security protocols. We believe such investigations to be an important first step toward addressing the challenges of energy-efficient security for battery-constrained systems.

Index Terms—3DES, AES, cryptographic algorithms, DES, Diffie-Hellman, DSA, ECC, embedded system, energy analysis, handheld, low-power, RSA, security, security protocols, SSL.

1 Introduction

Today, an increasing number of battery-powered embedded systems—PDAs, cell phones, networked sensors, and smart cards, to name a few—are used to store, access, manipulate, or communicate sensitive data, making security an important issue. Security concerns in such systems range from user identification to secure information storage, secure software execution, and secure communications. Most battery-powered systems contain wireless communication capabilities for untethered operation, introducing new security concerns due to the public nature of the physical communication medium or channel.

With the evolution of the Internet, network and communications security has gained significant attention [1], [2], [3], [4]. Secure communication across wired and wireless networks is typically achieved by employing security protocols at various layers of the network protocol stack, e.g., WEP [5] at the link layer, IPSec [6] at the network layer, TLS/SSL [7] and WTLS [8] at the transport layer, SET at the application layer, etc.). The building blocks of a security protocol are cryptographic algorithms, which are selected based on the security objectives that are to be achieved by the protocol. They include asymmetric and symmetric encryption algorithms, which are used to provide authentication and privacy, as well as hash or message digest algorithms that are used to provide message integrity.

While security protocols and the cryptographic algorithms they contain address security considerations from a functional perspective, many embedded systems are constrained by the environments they operate in and the resources they possess. For such systems, there are several challenges that need to be addressed in order to enable secure computing and communications. For battery-powered embedded systems, perhaps one of the foremost challenges is the mismatch between the energy and performance requirements of security processing,[1] and the available battery and processor capabilities. Rapid increases in communication data rates and security levels required, together with slow increases in battery capacities, threaten to widen this "battery gap" to a point where it will impede the adoption of applications and services that require security.

In this work, we demonstrate that security processing can have a significant impact on battery life. *Addressing the battery gap in secure communications requires that we first analyze and understand the energy consumption characteristics of security*

- *N.R. Potlapally and N.K. Jha are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544. E-mail: {npotlapa, jha}@ee.princeton.edu.*
- *S. Ravi and A. Raghunathan are with NEC Laboratories America, Princeton, NJ 08540. E-mail: {sravi, anand}@nec-labs.com.*

Manuscript received 20 Sept. 2003; revised 28 Apr. 2004; accepted 9 Sept. 2004; published online 15 Dec. 2005.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0153-0903.

1. We use the term *security processing* to refer to any computations performed for the sake of security, including the execution of security protocols and cryptographic algorithms.

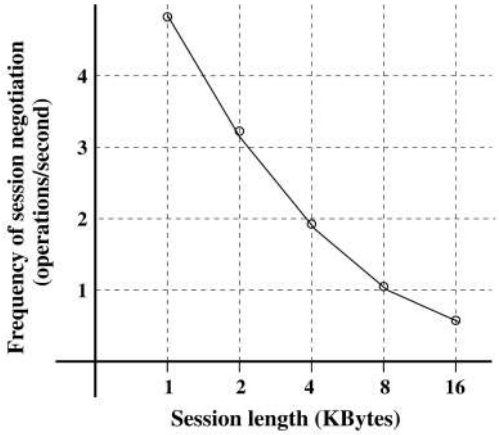


Fig. 1. Frequency of session set-up as a function of session length.

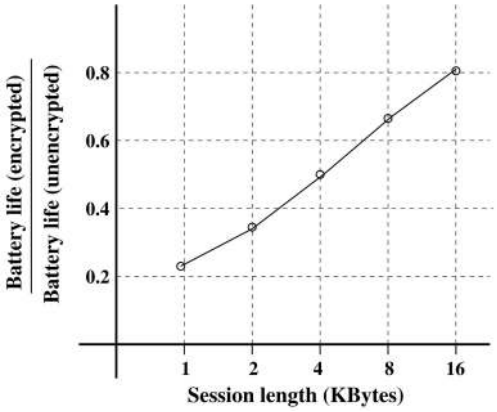


Fig. 2. Effect of encryption on battery life.

protocols and cryptographic algorithms. This paper presents a comprehensive energy measurement and analysis of the most popular transport-layer security protocol used in the Internet, the Secure Sockets Layer (SSL), or Transport Layer Security (TLS) protocol. To our knowledge, this is the first comprehensive energy analysis of the energy requirements of SSL/TLS. The energy analysis in this study is performed by executing secure data transactions on a battery-powered system (a Compaq iPAQ PDA [9]), measuring the current drawn from the power supply and calculating the energy consumed during the time intervals in which the security protocol or its constituent cryptographic algorithms are executed. Our results can be used to explore the impact of various parameters, at the protocol and cryptographic algorithm levels, on overall energy consumption for secure data transactions. Based on our analysis, we discuss various opportunities for energy-efficient implementations of security protocols.

The rest of this paper is organized as follows: Section 2 motivates the need for addressing energy consumption issues in security protocols. Section 3 introduces the reader to pertinent security terms and concepts. Section 4 describes the experimental testbed used in our work to execute and analyze secure wireless transactions and provides details of the energy measurement setup. Section 5 presents the results of our energy measurements, applies this information to analyze the SSL protocol, and suggests ways of optimizing the energy requirements of SSL. Section 6 summarizes the insights gathered in this work and enumerates future avenues of research.

2 Motivation

In this section, we provide an example to motivate the need for studying the energy consumption of security protocols. Then, we discuss the possible ways in which the energy consumption of security protocols can be optimized. Finally, we conclude by surveying related work on performance/energy analysis and optimization of security protocols.

2.1 The Impact of Security on Battery Life: An Illustration

In order to illustrate the burden imposed by security processing, we consider an example involving the operation of a sensor node with and without the need for security. We show that computations resulting from the use of security algorithms significantly reduce the amount of energy available for normal operations of the node.

Sensor nodes are normally used for aggregating specific information about their surroundings and transmitting it to a centralized location. Our example sensor node performs a similar functionality. It uses a Motorola "DragonBall" MC68328 processor, operates at a data rate of 10Kbps, and has a battery capacity of $26KJ$ (typical). Studies [10] have shown that the node consumes $13.9\mu J$ and $21.0\mu J$ for receiving and transmitting a bit, respectively. When encryption is used, the node consumes $41.0\mu J$ per bit during asymmetric algorithm operation and expends $7.9\mu J$ per bit for symmetric algorithm operation. In the absence of encryption, the node transmits the collected data as is. However, when encryption is used, the data transfer is broken up into *sessions*. Each session comprises two stages: authentication and key-establishment by using an asymmetric algorithm, followed by transmission of data after encrypting them using a symmetric algorithm with the key established (in the first stage of the session). The amount of data transmitted in a session is referred to as *session length*. The maximum amount of data which can be transferred in each session, i.e., upper bound on the session length, is specified by the security policy and is determined by the sensitivity of the data: The greater the sensitivity of the data, the shorter the session length, and vice versa. Thus, in the case of sensitive data, session lengths are short and the frequency of setting up new sessions is high. This is illustrated in Fig. 1, where the X-axis gives the session length and the Y-axis gives the frequency of session set-up (until the battery runs out). For example, when the session length of 1KB is used, the node has to negotiate nearly five new sessions per second. For highly sensitive data, session lengths are made less than or equal to 4KB and, consequently, greater than or equal to two sessions have to be set up every second.

Fig. 2 shows the effect of encryption on battery life as the session length is varied. "Battery life (encrypted)" refers to the number of sessions which can be transacted before the battery runs out. Similarly, "Battery life (unencrypted)" is the multiple of session lengths of data that can be sent without encryption until the battery drains out. We assume the session length to be the same in both the cases. The Y-axis in Fig. 2 is the ratio of "Battery life (encrypted)" to "Battery life (unencrypted)" and is an indicator of how fast the battery gets drained in the presence of encryption. The

figure shows that security processing causes an appreciable reduction in battery life. For example, when security requirements are high (and sessions are made less than or equal to 4KB), we see that the battery runs out more than twice as fast as when there is no encryption. Thus, there is a strong motivation to investigate techniques which lead to energy-efficient execution of security protocols.

2.2 Energy-Efficient Security Protocols

The objective of energy-efficient security protocol execution can be achieved in multiple ways, which can be divided into two broad classes listed below:

- By making the execution of constituent cryptographic algorithms (also referred to as cryptographic primitives) efficient through a combination of hardware and software techniques [11], [12], [13], [14], [15], we can improve the performance and energy requirements of security protocols. Usually, in these techniques, there is an overhead in the form of an increase in silicon area or more complex software.
- We can make the security protocols energy-cognizant by allowing them to alter their operation depending on the operating environment. This adaptation of behavior is guided by rules, which determine the best possible alternative with respect to energy efficiency under any given input conditions. These changes may involve a conscious and conservative tradeoff between the level of security and energy.

In either scenario, the challenges of energy-efficient secure communications can be better addressed if energy requirements and bottlenecks of the underlying security protocols are better understood. Commonly used security protocols, like SSL/TLS, IPSec, etc., have the freedom of realizing the desired security objectives by choosing specific cryptographic algorithms from a predefined set. In addition, the communicating parties can also decide upon parameters which influence the mode of operation of the chosen cryptographic algorithm. These provisions are made in security protocols primarily to lend flexibility to interactions between parties having diverse capabilities in terms of number of cryptographic algorithms supported by each of them (usually, resulting from the usage of different versions of security protocol software).

In this work, we perform a detailed analysis of the energy requirements of various cryptographic primitives with the intention of using this data as a foundation for devising energy-efficient security protocols. We performed several experiments where we varied several protocol and cryptographic algorithm-level parameters and observed the impact on energy. We use the results of our experiments to suggest ways for making the execution of the SSL protocol energy-efficient.

2.3 Related Work

Security protocols and cryptographic algorithms are known to have significant computational requirements, and studies have indicated that they stretch the processor capabilities available in many embedded systems [16], [17], [18], [19], [20], [21]. While researchers have quantified and addressed the performance overhead of security, the energy implications are relatively less understood. Nevertheless, researchers have recently proposed interesting approaches to the design of lightweight security protocols. Low-power key management protocols have been devised for sensor nodes by analyzing the impact of security algorithms on the energy consumption of sensor nodes [10]. The work in [22] evaluated the energy consumption of selected key-exchange protocols on a WINS sensor node and proposed energy-efficient ways for exchanging cryptographic keys, while custom protocols for low-power mutual authentication were proposed in [23], [24]. Energy tradeoffs in the network protocol and key management design space of sensor nodes were explored in [25]. Techniques to minimize the energy consumed by secure wireless sessions have also been proposed in [26]. We believe that comprehensive energy analyses of security protocols, such as the one performed in our work, will facilitate identification of energy bottlenecks and development of energy-efficient security mechanisms.

3 Preliminaries

In this section, we provide a brief overview of commonly employed security concepts and terminology [3], [4]. We begin by defining the widely used terms in the fields of cryptography and network security, and follow it by describing different kinds of protection measures, referred to as *security objectives*, desired in practical applications with a need for security. The concern for security in practice is addressed by choosing a security protocol, which achieves all the required security objectives. Security protocols realize the security objectives through the use of appropriate cryptographic algorithms. In the latter part of the section, we define the three classes into which all the cryptographic algorithms can be categorized based on their characteristics, and conclude the section by illustrating the working of a widely used security protocol, SSL.

3.1 Basic Security Terminology

A message present in a clear form, which can be understood by any casual observer, is known as the *plaintext*. The *encryption* process converts the plaintext to a form that hides the meaning of the message from everyone except the valid communicating parties, and the result is known as the *ciphertext*. *Decryption* is the inverse of encryption, i.e., the ciphertext is mapped back to its corresponding plaintext. The processes of encryption and decryption are parameterized on a quantity known as the *key*, which is ideally known only to the legitimate communicating parties. Since the strength of a security scheme depends on the secrecy of the key(s) used, it is highly imperative that the communicating parties take utmost precaution to safeguard the keys belonging to them. A *security protocol* formally specifies a set of steps to be followed by two or more communicating parties, so that the mutually desired security objectives are satisfied. It is assumed that the parties involved have the means to execute the various steps of the security protocol.

The term *security objectives* is often used to denote the security services or functionality required in a system or network to protect sensitive data and/or identity. The four main security objectives include:

- **Confidentiality**. This is the most popular requirement of security protocols, and it means that the secrecy of the data being exchanged by the communicating parties is maintained, i.e., no one other than

the legitimate parties should know the content of the data being exchanged.

- **Authentication**. It should be possible for the receiver of a message to ascertain its origin, i.e., to ensure that the sender of the message is who he claims to be, and the message was sent by him. This prevents a malicious entity from masquerading as someone else.
- **Integrity**. It provides a means for the receiver of a message to verify that the message was not altered in transit. This is necessary to prevent a malicious entity from substituting a false message in the place of a legitimate one or to tamper with the original message.
- **Nonrepudiation**. The sender of a message should not be able to falsely deny later that he sent the message, and this fact should be verifiable independently by an independent third-party without knowing too much about the content of the disputed message(s). This feature has important applications in the E-commerce domain, where it is common for users to send online messages authorizing the intended recipients of the messages to perform important actions on their behalf.

Security objectives thus provide trust, analogous to that present in face-to-face meetings, to the "faceless" interactions on the Web (or any data network). They are realized through the use of cryptographic algorithms (also referred to as cryptographic primitives), which are divided into three categories depending on their characteristics. These categories are:

- **Symmetric algorithms**. These algorithms use the *same* key for encryption and decryption. They rely on the concepts of "confusion and diffusion" [27] to realize their cryptographic properties and are used mainly for confidentiality purposes.
- **Asymmetric algorithms**. These algorithms use different keys, known as the *public key* and the *private key*, for encryption and decryption, respectively. They are constructed from the mathematical abstractions known as "trapdoor one-way functions," which are based on computationally intractable number-theoretic problems like integer factorization, discrete logarithm, etc. [28]. They are primarily used for authentication and nonrepudiation.
- **Hash algorithms**. These algorithms take a message of arbitrary length and output a fixed-length number (hash) representative of the message. Even a minor change in the original message can result in the computation of a different hash value. The algorithms can be made parameterizable on a key, in which case, they are referred to as "keyed hash algorithms." They are used for verifying the integrity of the messages exchanged.

Depending on the security objectives needed by a transaction among various parties and the constraints imposed by them, a security protocol is devised by composing a formal sequence of steps and deciding which algorithms should be used for carrying out each step.

3.2 An Example Security Protocol: Secure Sockets Layer (SSL) Protocol

SSL is one of the most widely used security protocols on the Internet. It is implemented at the transport layer of the protocol stack. SSL offers the basic security services of encryption, source authentication, and integrity protection for data exchanged over underlying unprotected networks. The SSL protocol is typically layered on top of TCP/IP layers of the protocol stack and is either embedded in the protocol suite or is integrated with applications such as browsers. The SSL protocol consists of two main layers, as shown in Fig. 3. The SSL record protocol provides the basic services of privacy and integrity to the higher-layer protocols: SSL handshake, SSL change cipher, and SSL alert. Let us now examine how the SSL record protocol is used to encrypt application data. The first step involves breaking the application data into smaller fragments. Each fragment is then compressed, if compression options are enabled. The next step involves computing a message authentication code (MAC), which facilitates message integrity. The compressed message plus MAC is then encrypted using a symmetric cipher. If the symmetric cipher is a block cipher, then a few padding bytes may be added. Finally, an SSL header is attached to complete the assembly of the SSL record. The header contains various fields, including the higher-layer protocol used to process the attached fragment.

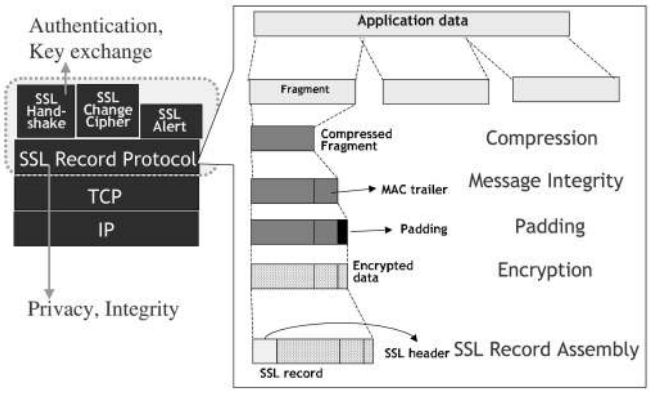


Fig. 3. The SSL protocol, with an expanded view of the SSL record protocol.

Of the three higher-layer protocols, SSL handshake is the most complex and consists of a sequence of steps that allows a server and client to authenticate each other and negotiate the various cipher parameters needed to initiate a session. For example, the SSL handshake is responsible for negotiating a common suite of cryptographic algorithms (cipher-suite), which can then be used for session key exchange, authentication, bulk encryption, and hashing. The cipher-suite RSA-3DES-SHA1, for example, indicates that RSA can be used for key agreement (and authentication), while 3DES and SHA1 can be used for bulk encryption and integrity computations, respectively. More than 30 such cipher suite choices exist in the OpenSSL implementation [29] of the SSL protocol, resulting from combinations of various cipher alternatives for implementing the individual security services.

Finally, the SSL change cipher protocol allows for dynamic updates of cipher suites used in a connection, while the SSL alert protocol can be used to send alert messages to a peer. Further details of the SSL protocol can be found in [3].

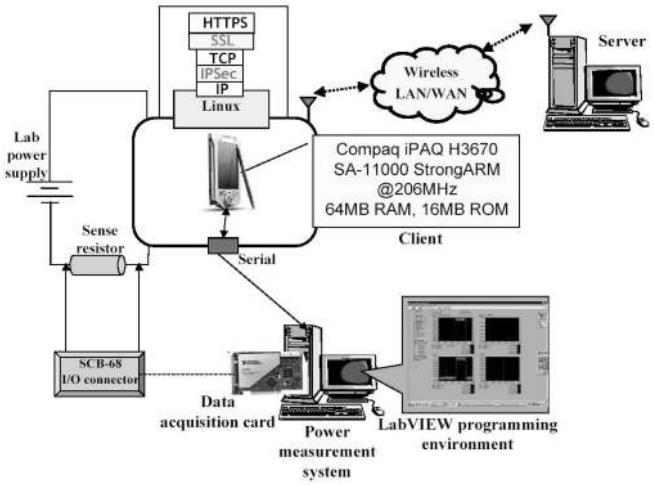


Fig. 4. Secure client-server configuration and the energy measurement testbed.

4 Experimental Setup

Fig. 4 describes the experimental setup used to execute secure client-server interactions, and the testbed developed to quantify the energy consumption of the various constituent security protocols.

The experimental setup for secure client-server communication consists of a client that connects to a LAN through a wireless access point, while the server is a PC that is wired to the LAN. The handheld used in the experiment is a Compaq iPAQ H3670, which contains an Intel SA-1110 StrongARM processor clocked at 206MHz. It is provided with 64MB of RAM and 16MB of FlashROM, and has an expansion sleeve which allows for memory expansion using compact flash cards. It connects to the wireless access point using a Cisco Aironet 350 series WLAN card. The handheld also supports additional communication capability through a serial port, a USB port and IrDA at 115.2 Kbps. It is powered by a Li-Polymer battery with a 950 mAh rating. The handheld uses the Familiar distribution [30] of Linux as its OS. The server is a PC equipped with a 700MHz Intel Pentium III having 256MB of RAM and running the RedHat Linux OS. The security of client and server interactions is provided by the SSL software from the OpenSSL [29] open-source project.

The energy consumption values for individual cryptographic algorithms are obtained by running their implementations on the client and measuring the current drawn from the power supply. Fig. 4 also shows the arrangement used for measuring the energy consumption of the cryptographic algorithms. The energy measurement is done using LabVIEW [31], a GUI-based data acquisition, measurement analysis, and presentation software. The data acquisition software runs on a PC (called a power measurement system), which is also directly connected to the handheld through its serial port. This enables the handheld to send synchronization signals to the data acquisition unit to start and stop the energy measurements. This signaling mechanism allows us to precisely measure the energy dissipated by the chosen software kernels. The current drawn by the client is measured by connecting a sense resistor in a series between the handheld and the energy source, i.e., the battery. The voltage drop across the sense resistor is measured using an SCB-68 I/O connector block [31]. This block interfaces to the data acquisition software, LabVIEW, through a data acquisition (DA) card in the PC running the LabVIEW software. LabVIEW is used to calculate the energy supplied to the handheld by integrating power over the time interval between the start and stop synchronizing signals.

5 Experimental Results

In this section, we present a comprehensive empirical analysis of the energy consumption characteristics of cryptographic algorithms (Section 5.1) using the experimental set-up described in Section 4. We also present a comprehensive energy analysis for various stages of the SSL protocol (Section 5.2).

5.1 Energy Analysis of Cryptographic Algorithms

We analyze variations in the energy consumption of various asymmetric, hash, and symmetric algorithms used for the purposes of authentication, integrity, and secrecy of data transactions, respectively (Sections 5.1.1 - 5.1.3). In Section 5.1.4, we investigate the influence of commonly used software implementation techniques of cryptographic algorithms on their energy consumption. We conclude this section by illustrating energy consumption versus security-level trade-offs realized by varying the operational parameters of cryptographic algorithms (Section 5.1.5). The implementations of the cryptographic algorithms were obtained from a standard cryptographic library used in the widely deployed OpenSSL package [29] and run on an SA-1110-based energy testbed described in the previous section. Since all the implementations are derived from the same standardized library, we presume that more or less similar software engineering techniques were uniformly employed throughout the software package. Thus, we can consider that the relative computational differences between various algorithms belonging to the same class (asymmetric, hash, and symmetric) are largely due to disparities in the complexity of their constituent operation steps and are not strongly tied to the data structures and software programming techniques used. In addition, most of the hardware platforms used in embedded devices are either SA-1110-based or very similar to it. Therefore, the conclusions drawn here are broadly applicable to other protocols such as WTLS, IPSec, etc., since they use the same cryptographic algorithms. However, there are some limitations to this approach arising out of the usage of particular number-theoretic algorithms. For each number-theoretic operation, there exist many different algorithms to perform it, and they have varying performance. Though the library chooses the optimal mathematical algorithms for the various operations, the state-of-the-art keeps changing. Thus, the observations from our empirical analysis would be relevant in the context of the algorithms employed for different mathematical operations.

5.1.1 Asymmetric Algorithms

Computationally hard mathematical problems form the basis of public-key cryptosystems. RSA is based on the hardness of integer factorization, while the digital signature algorithm (DSA) and Diffie-Hellman (DH) are based on that of the discrete logarithm problem in integer fields. The

TABLE 1
Energy Cost of Digital Signature Algorithms

Algorithm	Key size ($bits$)	Key generation (mJ)	Sign (mJ)	Verify (mJ)
RSA	1,024	270.13	546.50	15.97
DSA	1,024	293.20	313.60	338.02
ECDSA	163	226.65	134.20	196.23
ECDSA	193	281.65	166.75	243.84
ECDSA	233	323.30	191.37	279.82
ECDSA	283	504.96	298.86	437.00
ECDSA	409	1034.92	611.40	895.98

elliptic curve digital signature algorithm (ECDSA) and elliptic curve Diffie-Hellman algorithm (ECDH) provide security based on the discrete logarithm problem defined on elliptic curves. The basic mathematical operation in RSA, DSA, and DH is modular exponentiation and point multiplication on elliptic curves forms the core arithmetic operation in ECDSA and ECDH [32]. If $Security_{integer}(k1)$ and $Security_{elliptic}(k2)$ denote the security provided by RSA/DSA/DH algorithms employing a key of $k1$ bits and ECDSA/ECDH algorithms using a key of size $k2$ bits, respectively, then research has shown that the following equivalence exists between them [33]:

$$Security_{integer}(163) \equiv Security_{elliptic}(1024),$$
$$Security_{integer}(283) \equiv Security_{elliptic}(3072), \text{and}$$
$$Security_{integer}(409) \equiv Security_{elliptic}(7680).$$

For example, a 1,024-bit modulus RSA offers the same level of protection from cryptanalytic attacks as a 163-bit ECDSA.

Table 1 compares the energy consumed by the three federal information processing standard (FIPS)-approved asymmetric algorithms for generating and verifying signatures in security protocols: RSA, DSA, and ECDSA. We show the energy consumed by ECDSA for different elliptic key sizes in addition to 1,024-bit RSA and DSA. The energy values are reported for the three main steps associated with digital signature algorithms: key generation, signature creation (Sign), and signature verification (Verify). We assume a priori generation of the parameters used in the key generation process, as is the case in resource-constrained devices. We can see that 163-bit ECDSA is energy-efficient compared to 1,024-bit DSA. However, 163-bit ECDSA and 1,024-bit RSA digital signature algorithms have complementary energy costs. RSA performs signature verification efficiently, while ECDSA imposes a smaller cost for signature generation. We can see that the energy costs of sign and verify are much more symmetric in ECDSA than in RSA. ECDSA uses point multiplication where a scalar of the order of the degree of the curve is multiplied with a fixed point (called the base point) on the elliptic curve to get another point on the curve. Point multiplication is employed by both the sign and verify operations. In ECDSA, the verify operation requires some extra steps involving modular multiplication for validating the signature and, therefore, consumes more energy than the sign operation. The huge discrepancy in the energy costs of sign and verify operations in RSA results from the significant difference in the sizes of the keys employed (which are used as exponents in the modular exponentiation operation). In the sign operation, the private key is used which has the same size as the modulus, and the much smaller public key (it is usually 3 or 17) is used in the verify operation.

TABLE 2
Energy Cost of Key Exchange Algorithms

Algorithm	Key size ($bits$)	Key generation (mJ)	Key exchange (mJ)
DH	1,024	875.96	1,046.5
ECDH	163	276.70	163.5
DH	512	202.56	159.6

Asymmetric algorithms are also widely used for performing key exchange. Table 2 compares the standard algorithms used for key exchange, Diffie-Hellman (DH) and its elliptic curve analogue (ECDH). We observe that a 163-bit ECDH consumes much lesser energy than a 1,024-bit DH key exchange. The energy cost of the DH algorithm can be drastically reduced by decreasing the size of keys from 1,024 bits to 512 bits. However, this benefit does come at the cost of reduced security.

Modular exponentiation and point multiplication can be done in different ways [32]. In the implementations used in our experiments, modular exponentiation was performed by combining Montgomery reduction with a sliding window exponentiation. The window size was set at 6. There were two alternatives available for realizing point multiplication: the Montgomery without precomutation (MWP) method and the width-w nonadjacent (wNAF) method. We studied the energy consumption of the two algorithms and found out that the wNAF method is efficient only when multiple point multiplications are performed with respect to the same point.

5.1.2 Hash Algorithms

Table 3 summarizes the energy cost of commonly-used hashing algorithms. In general, hash algorithms are the least complex of the cryptographic algorithms and should intuitively incur the least energy cost. From Table 3, MD2 and HMAC are observed to be more compute-intensive than the rest of the hash algorithms. HMAC is a keyed hash, and as the bit-width of the key is increased from 0 (no key) to 128 bits, the energy cost varies by a very small amount. SHA and SHA1 are newer hash algorithms and have a larger number of steps than MD4 and MD5. Also, SHA and SHA1 are supposed to have better collision resistance, i.e., probability of two inputs mapping to the same hash value, than MD4 and MD5. These benefits of SHA (and SHA1) come at the cost of a slightly higher energy cost than MD4 and MD5.

5.1.3 Symmetric Ciphers

Symmetric ciphers can be chosen from two classes, *block* and *stream*, for use in a security protocol. Block ciphers operate on similar-sized blocks of plaintext and ciphertext. Examples of block ciphers include DES, 3DES, AES, etc. Stream ciphers, such as RC4, convert a plaintext to ciphertext one bit (or byte) at a time. Before a block or stream cipher starts the encryption/decryption operation, the input key (usually 64 bits) is expanded in order to derive a distinct and cryptographically strong key for each round (*key setup*). Encryption or decryption in symmetric algorithms then proceeds through a repeated sequence

TABLE 3
Energy Consumption Characteristics of Hash Functions

Algorithm	MD2	MD4	MD5	SHA	SHA1	HMAC
Energy (μJ/B)	4.12	0.52	0.59	0.75	0.76	1.16

(rounds) of mathematical computations. We begin by comparing the energy consumption of various symmetric algorithms and go on to examine the energy costs of features that are particular to symmetric algorithms and their consequences.

Fig. 5 shows variations in energy consumption due to the use of different symmetric ciphers. Energy numbers for the key setup phase and energy-per-byte numbers for encryption/decryption phases are shown for each cipher. The results are reported for one specific mode of each block cipher—ECB or electronic code book, where a given plaintext block always encrypts to the same cipher-text block for the same key (the impact of different modes on energy is explored later in Section 5.1.4). The only exception is RC4, which is a stream cipher. From the results displayed in Fig. 5, we make the following observations:

- RC4 is supposed to be a fast and efficient stream cipher, which is suitable for encrypting data in high-speed networking applications. However, we see that it has a significant encryption cost compared to other symmetric ciphers. Further analysis of the operation of the algorithm shows that the majority of the energy is consumed in memory accesses resulting from cache misses. In the 3.93 μJ used in encrypting a byte of data, 3.44 μJ (87.53 percent) is spent in memory-related operations and the remaining 0.49 μJ (12.47 percent) on computations.
- Blowfish exhibits the greatest contrast between the energy costs of key setup and encryption/decryption: The energy cost of key setup is the highest, while that of encryption/decryption ranks as one of the lowest. Blowfish is a 64-bit cipher which performs encryption using simple operations and is designed to be efficient on 32-bit processors with a reasonably-sized data cache. On the other hand, key setup is a complex operation involving 521 iterations in which subkey arrays totaling 4,168 bytes are generated. This algorithm is suitable for applications where the key is not changed frequently (thereby allowing the significant overhead of key setup to be amortized by the low encryption cost).
- In terms of energy of key setup and encryption, IDEA is on par with AES. IDEA is a 64-bit cipher where the constituent operations in encryption are performed on 16-bit blocks. IDEA is supposed to have very good cryptanalytic properties, thereby combining efficiency with acceptable security.
- AES has competitive energy costs, and its cryptanalytic properties have been well-studied. The round operations in AES operate on 8-bit data blocks and are amenable to implementation efficiency on 8-bit processors. However, optimizations exist to make AES run extremely fast on 32-bit processors at the cost of some space overhead (upto 4KB) [34]. In this case, the round operations are transformed into table lookups. The AES implementation under study has this optimization (discussed in detail in Section 5.1.4). Moreover, the table lookups can be done in parallel and this feature can be exploited by multithreaded processors to get further gains in performance.

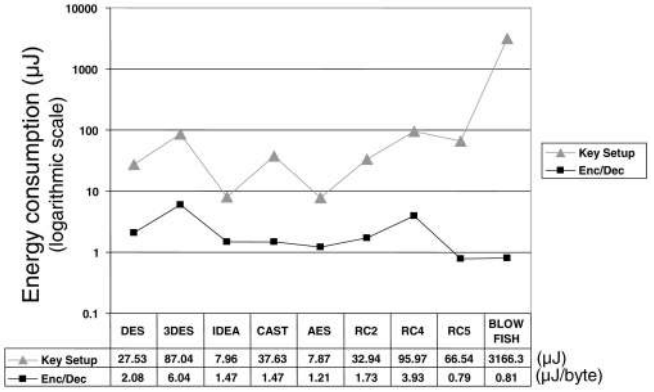


Fig. 5. Energy consumption data for various symmetric ciphers.

The encryption energy overhead of symmetric algorithms is also considerably influenced by their operational characteristics, like key size, cipher mode, etc. These effects are investigated in a later section.

5.1.4 Energy Costs of Implementation Choices

Most symmetric ciphers (block) perform encryption/decryption by passing the input data through multiple iterations of a fixed sequence of operations. This fixed sequence of operations is collectively referred to as a *round*. In software implementations of symmetric ciphers, some characteristics of the operations which make up the *round* are exploited to enhance the performance of the implementation. Two popular techniques employed for improving performance are table look-ups and loop unrolling. In this section, we evaluate the effect of these popular optimizing techniques on the energy consumption by studying them in the context of AES.

Some of the mathematical transformations used in a round can be implemented as predetermined tables. Table look-ups allow faster execution of the corresponding round operations at the expense of increase in code size. In loop unrolling, code implementing the round operations is expanded across the loop iterations, i.e., the body of the loop is replicated once for every reduction in the number of loop iterations. Loop unrolling increases the number of instructions relative to the branch and overhead instructions present in a loop implementation. This results in a better scheduling of instructions in the processor pipeline and, thus, improved performance. However, too much unrolling can result in a drastic increase in code size. In constrained environments, like embedded systems, which usually have a small-footprint on-chip memory, bloating of code size raises the number of capacity misses in the cache, thereby increasing the number of memory accesses which are expensive with respect to energy and performance.

A round in AES consists of four operations, namely, ByteSub, ShiftRows, MixColumns, and AddRoundKey. Among the four operations, ByteSub is most suitable for being implemented as a table look-up. When performance is an issue, the four operations in a round can be realized as

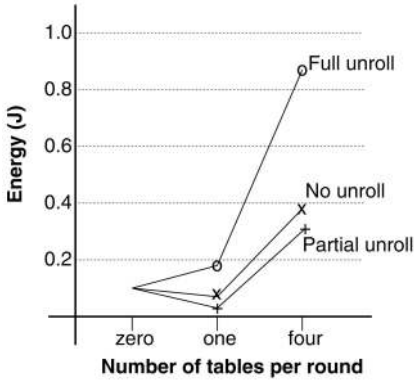


Fig. 6. Energy consumption of AES as a function of table look-ups.

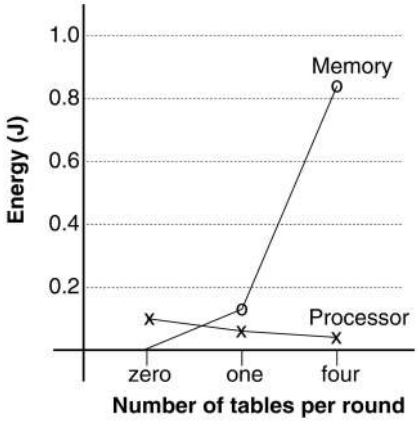


Fig. 7. Energy consumed by AES in processor and memory.

four table look-ups on 32-bit processors [34]. Similarly, the degree of loop unrolling can be varied. In our experiments, we observed the energy consumed in encrypting and decrypting a 60KB data file using a 128-bit key as the number of tables per round are varied (none, one, and four), in addition to altering the degree of loop unrolling (none, partial, and full). In the case of one table per round, only the ByteSub operation is implemented as a table and, in partial unrolling, the loop is unrolled for half the number of times it is done in full unrolling. The results of the experiments are shown in Fig. 6, on which we base the following observations:

- Full unrolling consumes the maximum energy among the three degrees of unrolling. The increase in code size due to full unrolling has a negative impact on the cache behavior, thereby resulting in an increase in expensive memory accesses. The presence of table look-ups further increases the memory traffic and, therefore, we see energy consumption increasing with the number of tables.
- Partial unrolling gives the most energy-efficient behavior among the three types of unrolling. We can see that partial unrolling extracts the benefits of loop unrolling without appreciably affecting the cache behavior. The inclusion of a single table further improves the energy efficiency, however, when the number of tables is increased to four, it worsens (due to an increase in memory traffic).
- Even in the absence of loop unrolling, the presence of four tables leads to an increase in memory accesses (and, thus, energy). The presence of a single table improves energy efficiency.

From the observations above, we can see that partial unrolling gives better energy efficiency than no and full loop unrolling. Similarly, a single table look-up was observed to be the most energy-efficient option with respect to the number of table look-ups. The reason for this is substantiated in Fig. 7, which shows the energy dissipated in the processor and memory in the presence of partial loop unrolling, as the number of tables per round is varied. We see that, as the number of tables is increased, the energy consumed in the memory increases, while the processor energy consumption decreases. This is explained by the fact that tables trade computation in the arithmetic and logic units of the processor for look-ups in memory. However, the rate of energy consumption increase in the memory is much higher than that of reduction in the processor. The crossover point of these two divergent curves with respect to energy consumption occurs in the vicinity of one table per round. Thus, we conclude that optimum energy efficiency is achieved in the presence of a single table per loop and partial loop unrolling (in embedded processor environments).

5.1.5 Energy Consumption versus Security Trade-Offs

If different security levels can be provided by a cryptographic algorithm, each with its associated energy consumption characteristic, a security protocol has the option to adapt the level of security commensurate with the current state of the battery of the system with a view of extending its life. This is best exemplified in symmetric algorithms where the security level can be altered by adjusting functional parameters, like cipher modes, key size, and number of rounds. We show that each of these parameters has a considerable effect on the energy cost of the algorithms, thereby resulting in energy-security trade-offs of practical interest.

The different cipher modes of operation of a block cipher result in algorithmic variants with different energy consumption characteristics and also security levels. We illustrate this fact in the context of AES. The simplest mode is the ECB, but it is susceptible to cryptanalytic attacks. The remaining modes (cipher block chaining (CBC), cipher-feedback mode (CFB), and output-feedback mode (OFB)) employ a feedback mechanism so that the encryption of a plaintext block is made dependent on the results of encryption of previous plaintext blocks. These modes differ in the manner in which the feedback loop is realized. Due to the feedback mechanism, even for the same key, a given plaintext will not always map to the same ciphertext. Thus, CBC, OFB, and CFB offer greater resistance to cryptanalytic attacks than ECB [3]. Also, the size of the key has an effect on the security offered by an algorithm: The larger the key size, the greater the security offered [3]. Table 4 presents the energy consumption of the AES algorithm for various operating modes and key sizes. From the table, we can observe that:

- The energy consumption for the key set-up phase and encryption increases with the key size.
- ECB mode is the most energy-efficient mode for encryption, and the energy cost of encryption increase across CBC, OFB, and peaks for CFB mode.

TABLE 4
Energy Costs of AES Variants

Key size (bits)	Key setup (μJ)	ECB (μJ/B)	CBC (μJ/B)	CFB (μJ/B)	OFB (μJ/B)
128	7.83	1.21	1.62	1.91	1.62
192	7.87	1.42	2.08	2.30	1.83
256	9.92	1.64	2.29	2.31	2.05

Studies have shown that the above conclusions are generic and hold for other block ciphers too.

The number of rounds of execution has a proportional effect on the security of the algorithm. Table 5 identifies different security levels for the RC5 cipher, obtained by changing the number of rounds used in the cipher for a given key and block size (128 bits). Each entry indicates the data (number of attempts) needed for a successful attack against RC5 using differential and linear cryptanalysis techniques. The symbol $>$ denotes the case when the attacks are deemed impossible even theoretically. We measured the energy consumption of RC5 for various security levels, and the detailed energy versus security trade-off curve is shown in Fig. 8. This shows a scheme for lowering the energy consumption by adjusting the security level from high to mid to low, achieved by changing the number of RC5 rounds from 20 to 16 to 8, respectively.

We also analyzed the combined effect of key size and number of rounds on the energy cost of key setup for RC5. From Table 6, we can see the cost of key setup steadily increasing with key size and number of rounds.

5.2 Energy Analysis of the SSL Protocol

Fig. 9 shows the typical (client-side) sequence of operations for a secure session that uses the SSL protocol. The first stage involves loading the client certificate from local storage, optionally decrypting it using a symmetric cipher and performing an integrity check. Note that the above cryptographic operations are optional and are not part of the SSL protocol. This is only a means of ensuring that user's certificates stored on his local machine were not tampered with by some unauthorized agents. Once the SSL handshake initiates a session, the client and server begin a sequence of exchanges which result in the client-side operations shown in the figure. The operations include 1) *server authentication*, where the client verifies the digital signature of the trusted certificate authority (CA) on the server certificate through decryption using the public key of the CA followed by an integrity check, 2) *client authentication*, where the client generates a digital signature by hashing some data using the MD5 and SHA-1 algorithms, concatenating the digests, and encrypting the result with its private key, and 3) *key exchange*, where the client generates a 48-byte premaster secret (used to generate the secret key for the record stage) and encrypts it with the public key of the server. Once the connection is established, secure transmission of data proceeds through the SSL record stage.

TABLE 5
Multiple Levels of Cryptanalytic Difficulty in RC5

Rounds	4	6	8	10	12	14	16
DC-C	2^{19}	2^{42}	2^{58}	2^{83}	2^{106}	2^{123}	$>$
DC-K	2^{74}	2^{86}	2^{94}	2^{106}	2^{118}	$>$	$>$
LC	2^{47}	2^{95}	2^{119}	$>$	$>$	$>$	$>$

**DC-C: Differential cryptanalysis (chosen plain-text). DC-K: Differential cryptanalysis (known plain-text). LC: Linear cryptanalysis (known plain-text).*

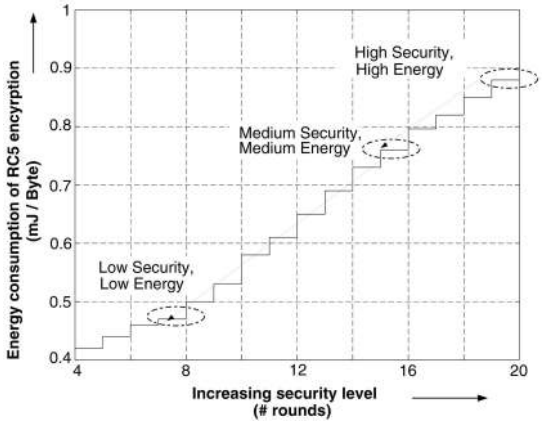


Fig. 8. Energy consumption versus security trade-off for RC5 encryption.

In the next two sections, we discuss the energy consumption of the SSL protocol with respect to that consumed by computation and by communication, respectively. In the concluding section, we discuss the techniques for optimizing the energy consumption of different stages of the SSL protocol, i.e., handshake and record stages.

5.2.1 Energy Cost of Computation in SSL Protocol Processing

The computation in SSL protocol processing is divided among the operations in the handshake and record stages. Fig. 10 examines the energy consumption contributions from the handshake and record stages of the SSL protocol for various transaction sizes. We can see that, for small transaction sizes (up to 256KB), the SSL handshake protocol dominates the overall energy consumption, e.g., 98.9 percent for 1KB transactions, while, for large transactions, the energy consumption of the SSL record protocol is significant, e.g., 80.4 percent for 1MB transactions. Therefore, we have to optimize the energy dissipated by the handshake protocol to improve the energy efficiency of small transactions and similarly target the record protocol for significantly improving the energy consumption behavior of large transactions. Usually, the operating environment of a device determines whether the small transactions dominate or the large ones do.

TABLE 6
Energy Consumption of RC5 Key Setup

Key size (bits)	Rounds	Energy (μJ)
64	8	36.60
	12	51.00
	16	68.53
128	8	65.80
	12	66.96
	16	71.48
256	8	122.34
	12	126.10
	16	131.12

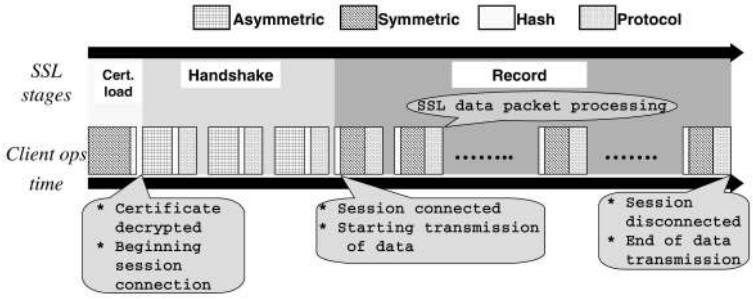


Fig. 9. Sequence of client-side operations for an SSL session.

The SSL handshake and record stages include cryptographic and noncryptographic operations. It is of interest to find out how the total energy consumed in SSL data transactions is divided between cryptographic and noncryptographic processing. This information will enable us to calculate the upper bound on the energy savings we can achieve by improving the energy efficiency of cryptographic algorithms. Fig. 11 summarizes our findings on the energy consumption of the cryptographic and noncryptographic components of the SSL computation for three different transaction sizes (1KB, 100KB, 1MB). Cryptographic processing includes processor cycles spent in execution of symmetric, asymmetric, and hash algorithms as part of SSL protocol execution. Noncryptographic processing encompasses all the system functions that are necessary for sending and receiving data over a network. Examples of this would be networking functions operating at different layers of the protocol stack (socket, TCP, IP, and network interface), memory management for buffering packets after reception and before transmission, etc. Based on the data in Fig. 11, we make the following observations:

- Energy used by cryptographic processing contributes a significant percentage to the total energy dissipated. For example, in a 1KB SSL-enabled data transaction, 58 percent of the total energy dissipated is due to cryptographic algorithms. We also note that the energy contribution from cryptographic processing steadily decreases with the size of the data transactions.
- For small-sized data transactions, the energy dissipated by cryptographic processing is made up almost entirely of contributions from asymmetric algorithms. However, as the size of the transaction increases, symmetric algorithms replace asymmetric algorithms as a dominant contributor to the total energy dissipated by cryptographic processing. The energy contribution of hash algorithms also increases with data size, but remains a minor fraction of the total energy consumption. For example, in a 1KB data transaction, the energy dissipated by asymmetric algorithms forms more than 90 percent of the energy consumption of cryptographic processing (and roughly 56 percent of the overall energy consumption). When the data transaction size is increased to 1 MB, the contribution of asymmetric algorithms to the energy consumed by cryptographic processing reduces to 23 percent (12 percent of overall consumption) and that of symmetric algorithms increases to 67 percent (37 percent of overall consumption).

Thus, smaller transactions benefit greatly from optimization of asymmetric algorithms for energy, while an improvement in energy efficiency of symmetric algorithms significantly improves the energy cost of large transactions.

5.2.2 Communication Energy Cost in SSL Protocol Processing

We now analyze the communication energy overhead resulting from the transmission and reception of extra bytes resulting from SSL-related security processing. The

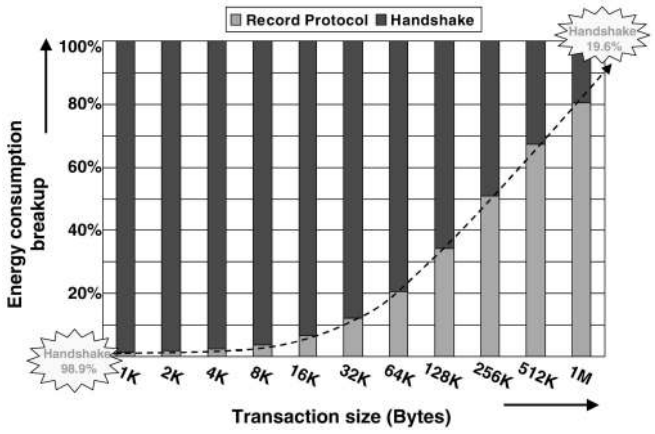


Fig. 10. Variation of energy consumption contributions from the SSL handshake and record stages with increasing transaction sizes.

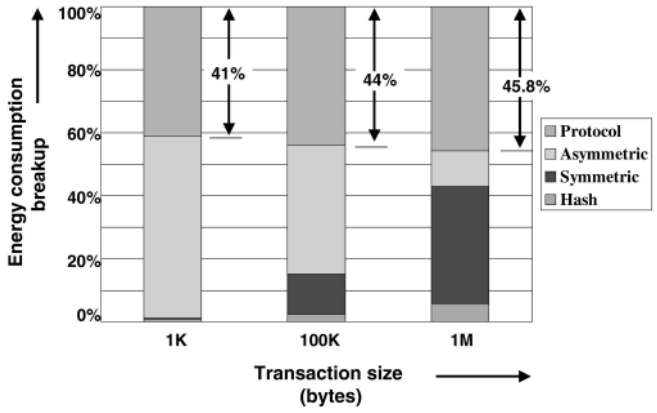


Fig. 11. Break-up of SSL energy consumption into cryptographic and noncryptographic components.

TABLE 7
Profile of the Data Exchanged between the Client and Server in an SSL Connection

Transaction size (*bytes*)	Direction	Handshake (*bytes*)	Record (*bytes*)	Change cipher spec (*bytes*)	Alert (*bytes*)	Total (*bytes*)
1K	C → S	183	20	32	1	236
	S → C	1,627	1,088	32	1	2,748
10K	C → S	183	20	32	1	236
	S → C	1,627	10,432	32	1	12,092
100K	C → S	183	20	32	1	236
	S → C	1627	104,000	32	1	105,660
1M	C → S	183	20	32	1	236
	S → C	1,627	1,064,960	32	1	1,066,620

extra bytes that are transmitted or received by the client are part of the packets which belong to one of the following four protocol layers of SSL, namely: *handshake*, *record*, *alert*, and *change cipher spec*. The packets belonging to the SSL handshake, SSL change cipher spec, and SSL alert layers are exchanged for the purpose of initiating, maintaining, and closing the SSL connection, whereas the SSL record layer is responsible for securely transmitting the data. Therefore, the extra bytes result from two sources: first, from the protocol management packets exchanged by the nonrecord SSL layers, and, second, from the extra information (like, the hash value, SSL header, etc.) appended to the data in the packets by the SSL record layer. In Table 7, we show the actual number of bytes, categorized according to the SSL protocol layer to which they belong, that are transacted between a client and server for four different files (having sizes 1KB, 10KB, 100KB, and 1MB) in an SSL connection. These numbers were obtained by using *ssldump* [36], a software package which can be used to read SSL traffic on networks.

In Table 7, the first column gives the size of the transaction. The second column shows the direction of the flow of bytes exchanged, i.e., "$C \rightarrow S$" stands for "client to server" and, similarly, "$S \rightarrow C$" means "server to client." Therefore, in the "$C \rightarrow S$" flow, the bytes are transmitted by the client, whereas, in the "$S \rightarrow C$" flow, it receives them. The next four columns show the amount of data generated by the four layers of the SSL protocol. The last column gives the total number of bytes transmitted and received by the client. The iPAQ, acting as the client, consumes different amounts of energy for transmission and reception, which are $3.36uJ/Byte$ and $2.64uJ/Byte$, respectively. Table 8 shows the energy overhead incurred by the client due to the transaction of the extra bytes resulting from SSL processing. The first column gives the size of the file, and the second column gives the energy expended in receiving this file from the server without using SSL. The fourth column shows the actual number of bytes handled by the client, in order to receive the file, of size as given by the entry in the first column, from the server using SSL. The fifth column gives the energy consumed by the client in a file transaction using SSL. Finally, the last column shows the communication energy overhead of SSL security processing which is the difference between the energy values given in the fifth and second columns.

From Table 8, we can see that the ratio of the communication energy overhead to the theoretically needed energy is very high for small-sized data transactions. Thus, communication energy cost of security imposes a significant burden for small data transactions. This is primarily due to the large number of bytes exchanged in the nonrecord layers of the SSL protocol. However, as the size of the data transaction increases, the communication overhead of security forms a smaller and smaller fraction of the theoretically required energy, i.e., the impact of the communication energy overhead is reduced. This trend is illustrated in Fig. 12. In this figure, the two tuples indicate the transaction size and the percentage communication energy overhead. We can see that, for a transaction of size 1KB, the percentage communication energy overhead is a very significant 197.69 percent, and it falls to 1.75 percent for a 1M-sized transaction. Also, the SSL client needs to update its SSL certificate database from time to time by including new ones and energy is expended in downloading them over the wireless interface. A typical SSL certificate provided by a certificate authority is around 1,200 bytes, and the energy cost of updating a certificate comes to $3.168mJ$.

5.2.3 Scope for Optimizing SSL Protocol Processing

Having examined the energy consumption characteristics of the SSL protocol, both with respect to computation and communication, we now analyze how the energy consumption of the handshake and record stages is affected by various protocol-level services as well as cryptographic algorithm parameters. Specifically, we describe how the use of client authentication impacts the energy consumption due to SSL handshake and how the choice of cipher-suite affects the energy consumption of SSL handshake and record stages, respectively. We conclude this section by enumerating the various ways in which the energy consumption of SSL protocol processing can be optimized.

Impact of Client Authentication and Asymmetric Cipher Choice on SSL Handshake. We investigated the energy cost of the SSL handshake protocol using the RSA algorithm and the ECC algorithms (ECDSA/ECDH) to implement various public-key operations. The results of our analysis are presented in Fig. 13. The SSL handshake can be performed between a server and a client with or without client authentication. In the case of handshake without client authentication, the following operations are performed by the client and the server:

- **RSA-based handshake.** The client performs two RSA public key operations (verify and encrypt), and the server performs an RSA private key operation (decrypt).

TABLE 8
Communication Energy Overhead in the Client Due to SSL Security Processing

Transaction size ($bytes$)	Energy needed without SSL overhead (uJ)	Action	Data exchanged when using SSL ($bytes$)	Energy spent with SSL overhead (uJ)	Energy overhead of SSL usage (uJ)
1K	2,703.36	Tx	236	8,047.68	5,344.32
		Rx	2,748		
10K	27,033.6	Tx	236	32,715.84	5,682.24
		Rx	12,092		
100K	270,336.00	Tx	236	279,735.36	9,399.36
		Rx	105,660		
1M	2,768,240.64	Tx	236	2,816,669.76	48,429.12
		Rx	1,066,620		

- **ECC-based handshake.** The client performs verification using ECDSA, and an ECDH operation is performed to compute the shared secret. The server performs an ECDH operation to calculate the shared secret.

If client authentication is required, some extra operations need to be performed by the client and the server. These are:

- **RSA-based handshake.** The client performs an RSA private key operation (sign). The server performs two extra RSA public key operations (verify).
- **ECC-based handshake.** The client performs an additional signing operation using ECDSA, and the server performs two extra verification operations using ECDSA.

Fig. 13 shows the energy consumed by the SSL handshake process using RSA or ECC algorithms for the handheld functioning as a client or a server. Though the handheld typically behaves as the client in a majority of transactions, it may sometimes be required to play the part of the server. In order to investigate this scenario, we allowed the handheld to perform the server operations for collecting the corresponding energy data. Energy data were also collected for studying the impact of client authentication in all the cases. With respect to the client energy cost, we can see from the figure that RSA-based handshake is much more efficient than the ECC-based handshake when there is no client authentication in the SSL handshake stage. However, in the presence of client authentication in SSL handshake, the ECC-based handshake consumes less energy than the RSA-based handshake. In general, we believe that various protocol-level parameters have interdependent effects on energy, leading to many interesting trade-offs.

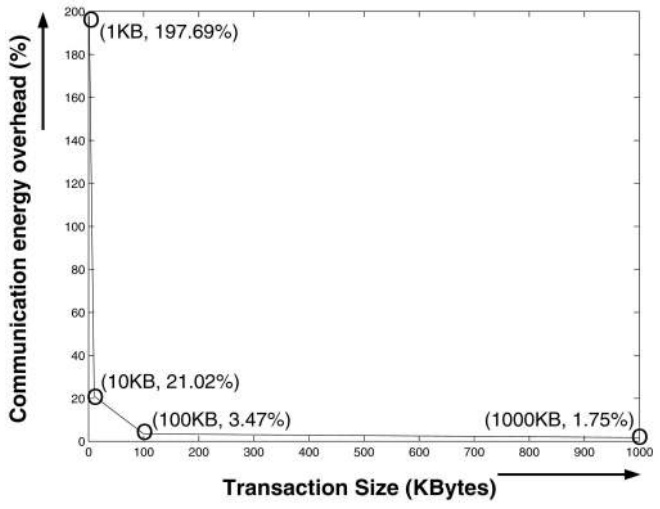


Fig. 12. Percentage communication energy overhead of SSL usage for varying transaction sizes.

Impact of Cipher-Suite Choice on SSL Record Energy Consumption. The energy cost of the SSL record stage is mainly determined by the amount of bulk data that is transmitted. Analysis of the cipher suites shows that careful choices of cryptographic algorithms need to be made in order to optimize energy during the record stage. Consider the following two cipher suites, ECC-BLOWFISH-SHA1 and ECC-AES-SHA. A cursory examination would conclude that the second cipher suite is more energy-efficient, given the very high cost of key setup in BLOWFISH. However, Fig. 14 shows that, if the amount of data transacted is greater than 7.9KB, then, in fact, the first cipher suite is more efficient. This is because the cost of key setup in BLOWFISH is gradually amortized, and the advantages of BLOWFISH come into play.

Fig. 14 illustrates the energy consumption of two cipher suites, RSA-RC5-SHA1 and ECC-3DES-SHA. The public-key algorithm (RSA or ECC) is used in the SSL handshake stage and the symmetric-key algorithm (RC5 or 3DES) is used for bulk encryption in the SSL record stage. The figure shows that, for data sizes smaller than 21KB, ECC-3DES-SHA is more energy-efficient because ECC is simpler than RSA (and asymmetric energy consumption dominates that of small data transactions). However, for transactions where there are significant bulk data (greater than 21KB) to encrypt, RSA-RC5-SHA1 consumes less energy, because, for large data transfers, energy consumption of symmetric ciphers dominates the total energy spent and RC5 is much simpler than 3DES. This shows that a judicious choice of cryptographic algorithms can greatly reduce the amount of energy consumed.

Scope for Optimizing SSL. The energy analyses of the SSL protocol and the constituent cryptographic algorithms provide us with insights into the opportunities available for saving energy during protocol execution. These opportunities emerge from 1) the presence of security levels (ranging from high to low) for each cryptographic algorithm and security protocol with each level correlated with distinct energy consumption characteristics, 2) the availability of parameters in a cryptographic algorithm such as

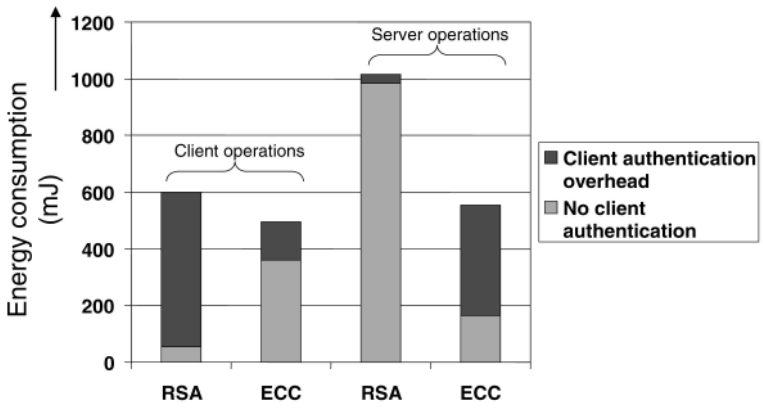


Fig. 13. Energy consumption for client and server operations in SSL handshake under the presence or absence of client authentication.

key sizes, number of rounds, etc., that can be tuned for energy efficiency and security level, and 3) the availability of parameters in a security protocol such as services provided, cipher/cipher-suite used for a given security service, etc. It should be remembered that, for the above optimizations to be practical, the capabilities of both the interacting parties (server and client) should be enhanced.

We now examine the security versus energy tradeoffs that result from variation of key size, cryptographic algorithm, cipher parameter implementation choices, and protocol steps.

- **Key size.** The size of the keys used in cryptographic primitives is flexible. The key size determines the strength of the cryptographic primitive, and also the amount of computation done. This is *especially* true for asymmetric algorithms. For example, in modular exponentiation (ME)-based asymmetric algorithms, like RSA, etc., if k is the number of bits in the key, the number of modular multiplications necessary to compute one ME operation is $1.5k$ (when the LR binary algorithm is used). Since small-sized keys can be easily broken by an adversary, the keys cannot be made arbitrarily smaller. Cryptographic studies [37] have shown that, to maintain an acceptable level of security, key sizes should be greater than or equal to 64 bits, 512 bits, and 163 bits for *symmetric algorithms*, *RSA and discrete logarithm (DL)-based*, and *elliptic curve (EC)-based asymmetric algorithms*, respectively. Table 9 shows the key sizes advisable for the two levels of security for various types of algorithms. Key size has a significant effect on the energy consumed by asymmetric algorithms as observed in the case of DH key exchange, where the energy consumption for key exchange increases from $159.6mJ$ to $1046.5mJ$, in going from a 512-bit key to a 1,024-bit key (Table 2).
- **Choice of algorithm.** Different cryptographic primitives exist to realize the same security functionality. It is known that the different choices give the same level of protection with *equivalent sized keys* [37]. For example, in asymmetric algorithms, a 1,024-bit RSA key is calculated to give the same level of protection as a 163-bit ECC key. However, in spite of this observation, the choice of algorithm to be used in a session is still open. This is due to a variety of reasons, such as the algorithms supported by the end parties, the perceived confidence in the algorithms by the users,[2] and the nature of the applications. The following observations can be made regarding algorithm selection for achieving energy efficiency:

  1. AES offers a good mixture of security and energy efficiency. Its security properties have been well-studied, and it was found to offer high resistance to linear and differential cryptanalysis. In addition, it also has a low energy cost for both key setup and encryption. However, recent studies have pointed out that it might be susceptible to algebraic attacks [38]. In terms of high resistance to cryptanalytic attacks, other algorithms which fare well are 3DES and IDEA [4]. When lower energy consumption is a higher priority, RC5 and Blowfish serve as possible candidates (Fig. 5). Blowfish is the ideal choice

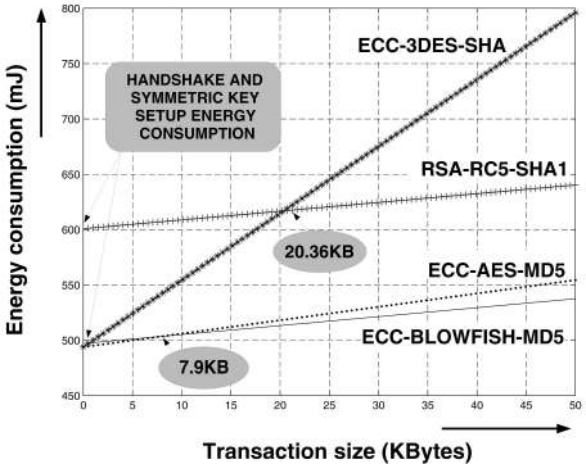


Fig. 14. The impact of cipher suite selection on energy consumption during the SSL handshake and record stages.

2. The cryptographic algorithms are as secure as the limitations of the present cryptanalytic techniques. A new discovery, either in mathematics or cryptanalysis, can undermine the security of a cryptographic algorithm which was hitherto considered to be unbreakable. Usually, the longer an algorithm stays unbroken, the greater is the trust placed in it. For example, though ECC is eminently suited for small form factor devices and is gaining wide acceptance, some parties prefer to use RSA just because it has been around longer, and no successful attacks have been carried out against it.

TABLE 9
Key Sizes for the Two Security Levels

Type of algorithm	Keysize in bits	
	Low security	High security
Symmetric	64	≥ 128
RSA and DL	512	$\geq 1{,}024$
EC	163	≥ 190

when large amounts of data are to be transmitted with a low frequency of key refreshes.

2. When there is a choice in the symmetric algorithm to be used (as in the latter case mentioned above), the size of the data to be encrypted should be an important factor in making a decision. Within the available set of algorithms, the one which can transmit the data with the least energy consumption should be selected. The energy cost of symmetric encryption, Si, can be estimated by a simple equation, given by:

$$Energy\_cost_{Si} = Key\_setup(Si) + Energy\_per\_byte(Si) \times Data\_size,$$

where $Key\_setup(Si)$ is the energy cost of expanding the symmetric key for algorithm Si, energy expended per byte for encryption/decryption using algorithm Si is given by $Energy\_per\_byte(Si)$, and $Data\_size$ is the total size of the data to be encrypted by algorithm Si. Fig. 5 gives the Key_setup and $Energy\_per\_byte$ values for different symmetric algorithms.

3. For digital signatures, RSA consumes much less energy for verification compared to signature generation, while ECDSA displays a complementary behavior. In a battery-constrained handheld, if the frequency of signature generation is much *smaller* than verification, then it is advisable to use RSA, while ECDSA is suggested if the frequency of signature generation is much higher. However, if both the operations are performed with similar frequencies on the handheld, then it might be a good idea to use ECDSA because the total energy dissipated for both signature generation and verification is less than in the case of RSA and DSA (Table 1).

- **Cipher Parameters of an Algorithm.** The cipher parameters of an algorithm influence the manner in which the ciphertext is produced. The cipher parameters have a significant influence on symmetric algorithm execution (and, thereby, the energy dissipation), and include the number of rounds and the mode of operation (ECB, CBC, CFB, and OFB). The recommendations, in terms of selecting the cipher parameters for obtaining energy efficiency by an appropriate selection of cipher parameters, are listed next:

  1. For a high security level, the number of rounds in a symmetric algorithm execution should be more than for a low security level. In RC5, where the number of rounds is variable, for low security, the number of rounds is set to eight, and, for high security, the number of rounds can be set at 12 or higher (Fig. 8).
  2. The choice of mode is dictated by the nature of the application and its desired security level. The ECB mode consumes the least energy, but is susceptible to statistical attacks using the plaintext. Therefore, it is suited for encrypting short random data with a low security level. For a high security level and for sending normal data, it is advisable to use one of the CBC, CFB, or OFB modes. They provide more security compared to the ECB mode at the expense of a higher energy consumption. In the CBC mode, the present plaintext block is XORed with the previously generated ciphertext block to provide better protection against statistical attacks. In CFB and OFB modes, the amount of ciphertext given as feedback to be XORed with the present plaintext block can be varied in sizes having increments of 8 bits. These two modes are suited for networking applications where there is streaming data and, therefore, one does not need to wait for the entire block to arrive to do encryption (as is necessary in CBC). Table 4 gives the energy consumption values for these modes in the case of the AES algorithm.

- **Implementation Choices.** In Section 5.1.4, we showed that memory accesses have a considerably higher energy cost than computations in the datapath of the processor. However, techniques like table lookups and loop unrolling are frequently employed for performance reasons. Usually, aggressive use of these measures results in an increase in the number of memory accesses, thereby decreasing energy efficiency. Thus, for the sake of energy, we should be judicious in the extent to which we employ these techniques that increase memory traffic. Depending on the energy and performance constraints, the right balance between the fraction of computation, which can be done on the processor, and that which is implemented though table lookups should be achieved (Fig. 7).
- **Protocol Steps.** The operations in security protocols can be divided into two mutually exclusive subsets: compulsory and optional. The optional steps are usually included for extra security. Depending on the security requirements of a task, the optional steps are either included in the protocol execution or not. In SSL handshake, the *client authentication* step is optional. The decisions with regard to execution of protocol steps, necessary to achieve energy efficiency, can be stated as follows:

  1. The following energy saving measures are valid when the handheld is acting as a client (Fig. 13):

     - If a high level of security is desired, client authentication should be included in the handshake. Using ECC is much more energy-efficient than RSA.

- When the security requirements are low, it is advisable to skip client authentication in SSL handshake. In this case, using RSA results in significantly higher energy efficiency than ECC.

2. When the handheld is acting as a server, using ECC results in better energy efficiency both in the presence and absence of client authentication.

Implementing the suggestions proposed for optimizing the energy efficiency of the SSL protocol do not entail making major changes to the existing structure and implementation of the protocol. They determine the parameters for energy-efficient protocol execution based on some characteristics of the secure session, like the size of the data transaction, amount of security desired, whether client authentication is required, amount of charge left in the battery, etc. The information governing energy-efficient parameter selection can be easily stored as a set of parameter values, and a set of rules which specify the conditions under which the parameters are applicable. This decision-making system can be naturally incorporated into the SSL Handshake protocol which determines the parameter values applicable to the session to be established and fixes them for the whole length of the session. This approach has an obvious limitation that both the client and server wishing to establish a secure connection should be able to support the protocol parameters aimed at achieving energy efficiency. Except for this issue, we cannot foresee any practical matters related to existing protocol implementations which will limit the usage of the proposed recommendations.

6 Conclusions

In this work, we presented a framework for analyzing the energy consumption of cryptographic algorithms and security protocols. We examined several cryptographic algorithms from the three main classes—asymmetric, symmetric, and hash, and observed that:

1. asymmetric and hash algorithms have the highest and least energy costs, respectively,
2. the energy cost of asymmetric algorithms is dependent on the key size, while that of symmetric algorithms is not significantly affected by the key size,
3. the energy consumption of a symmetric algorithm depends not only on the bulk data encryption/decryption cost but also on the key set-up cost,
4. wide variations in energy consumption exist within the same family of cryptographic algorithms, and
5. the level of security provided by a cryptographic algorithm can be traded off for energy savings by tuning parameters such as key size, number of rounds, etc.

We also studied the energy consumption profile of the SSL protocol and saw that the energy costs of the handshake and record stages of the SSL protocol vary depending on parameters like the functionality desired in the handshake, size of bulk data transacted, etc. Furthermore, we used our analysis to detail the opportunities available for making SSL (and other security protocols) energy-efficient.

References

[1] US Department of Commerce, *The Emerging Digital Economy II*, http://www.esa.doc.gov/508/esa/TheEmergingDigitalEconomyII.htm, 1999.

[2] W.W.W. Consortium, *The World Wide Web Security FAQ*, http://www.w3.org/Security/faq/www-security-faq.html, 1998.

[3] W. Stallings, *Cryptography and Network Security: Principles and Practice.* Prentice Hall, 1998.

[4] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C.* John Wiley and Sons, 1996.

[5] LAN MAN Standards Committee of the IEEE CS, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification: IEEE standard 802.11*, 1990.

[6] IPSec Working Group, http://www.ietf.org/html.charters/ipsec-charter.html, 2000.

[7] SSL 3.0 Specification, http://wp.netscape.com/eng/ssl3/, 1996.

[8] Wireless Application Protocol 2.0—Technical White Paper, http://www.wapforum.org/, Jan. 2002.

[9] Compaq iPAQ Pocket PC, http://h20022.www2.hp.com, 2002.

[10] D.W. Carman, P.S. Kruus, and B.J. Matt, *Constraints and Approaches for Distributed Sensor Security*, Technical Report 00-010, Network Assoc. Labs, 2000.

[11] J. Goodman, A. Chandrakasan, and A. Dancy, "Design and Implementation of a Scalable Encryption Processor with Embedded Variable DC/DC Converter," *Proc. Design Automation Conf.*, pp. 855-860, June 1999.

[12] Z. Shi and R. Lee, "Bit Permutation Instructions for Accelerating Software Cryptography," *Proc. IEEE Int'l Conf. Application-Specific Systems, Architectures, and Processors*, pp. 138-148, July 2000.

[13] J. Burke, J. McDonald, and T. Austin, "Architectural Support for Fast Symmetric-Key Cryptography," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 178-189, Nov. 2000.

[14] N. Potlapally, S. Ravi, A. Raghunathan, and G. Lakshminarayana, "Optimizing Public-Key Encryption for Wireless Clients," *Proc. IEEE Int'l Conf. Comm.*, pp. 1050-1056, May 2002.

[15] S. Ravi, A. Raghunathan, N. Potlapally, and M. Shankaradass, "System Design Methodologies for Wireless Security Processing Platform," *Proc. Design Automation Conf.*, pp. 777-782, June 2002.

[16] D. Boneh and N. Daswani, "Experimenting with Electronic Commerce on the PalmPilot," *Proc. Financial Cryptography*, pp. 1-16, Feb. 1999.

[17] W. Freeman and E. Miller, "An Experimental Analysis of Cryptographic Overhead in Performance-Critical Systems," *Proc. Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems*, pp. 348-357, Oct. 1999.

[18] S.K. Miller, "Facing the Challenges of Wireless Security," *Computer*, pp. 46-48, July 2001.

[19] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha, "Securing Electronic Commerce: Reducing the SSL Overhead," *IEEE Network*, pp. 8-16, July 2000.

[20] D.S. Wong, H.H. Fuentes, and A.H. Chan, "The Performance Measurement of Cryptographic Primitives on Palm Devices," *Proc. Ann. Computer Security Applications Conf.*, pp. 92-101, Dec. 2001.

[21] S. Ravi, A. Raghunathan, and N. Potlapally, "Securing Wireless Data: System Architecture Challenges," *Proc. Int'l Symp. System Synthesis*, pp. 195-200, Oct. 2002.

[22] A. Hodjat and I. Verbauwhede, "The Energy Cost of Secrets in Ad-Hoc Networks," *Proc. IEEE CAS Workshop Wireless Comm. and Networking*, Sept. 2002.

[23] M. Jakobsson and D. Pointcheval, "Mutual Authentication for Low-Power Mobile Devices," *Proc. Financial Cryptography*, pp. 178-195, Feb. 2001.

[24] D.S. Wong and A.H. Chan, "Mutual Authentication and Key Exchange for Low Power Wireless Communications," *Proc. IEEE Military Comm. Conf.*, pp. 39-43, Oct. 2001.

[25] Y.W. Law, S. Dulman, S. Etalle, and P.J.M. Havinga, *Assessing Security-Critical Energy-Efficient Sensor Networks*, Technical Report TR-CTIT-02-18, Univ. of Twente, The Netherlands, July 2002.

[26] R. Karri and P. Mishra, "Minimizing Energy Consumption of Secure Wireless Session with QoS Constraints," *Proc. Int'l Conf. Comm.*, pp. 2053-2057, May 2002.

[27] H. Feistel, "Cryptography and Computer Privacy," *Scientific Am.*, pp. 15-23, May 1973.

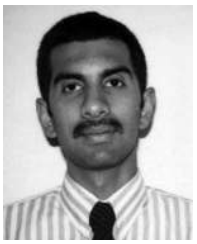
[28] K.H. Rosen, *Elementary Number Theory and its Applications.* Addison-Wesley Publishing Co., 1985.

[29] OpenSSL Project, http://www.openssl.org, 2001.

[30] Familiar Project, http://familiar.handhelds.org, 2002.
[31] National Instruments Corp., http://www.ni.com, 2001.
[32] A. Menezes, P.V. Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.
[33] V. Gupta, S. Gupta, S. Chang, and D. Stebila, "Performance Analysis of Elliptic Curve Cryptography for SSL," *Proc. ACM Workshop Wireless Security*, pp. 87-94, Sept. 2002.
[34] J. Daemen and V. Rijmen, "Rijndael, the Advanced Encryption Standard," *Dr. Dobb's J.*, pp. 137-139, Mar. 2001.
[35] Y.L. Yin, "The RC5 Encryption Algorithm: Two Years On," *RSA Laboratories' Cryptobytes*, vol. 2, pp. 14-15, 1997.
[36] SSLdump Project, http://www.rtfm.com/ssldump/, 2002.
[37] A.K. Lenstra and E.R. Verheul, "Selecting Cryptographic Key Sizes," *J. Cryptology: J. Int'l Assoc. for Cryptologic Research*, vol. 14, no. 4, pp. 255-293, 2001.
[38] Counterpane Internet Security: Crypto-Gram Newsletter, http://www.counterpane.com/crypto-gram.html, 2002.

Nachiketh R. Potlapally is currently pursuing the PhD degree in electrical engineering at Princeton University, Princeton, New Jersey. He is working toward proposing novel architectures to enable efficient security processing in resource-constrained embedded devices. He is a student member of the IEEE.

Srivaths Ravi received the BTech degree in electrical and electronics engineering from the Indian Institute of Technology, Madras, India, in 1996, and the MA and PhD degrees in electrical engineering from Princeton University, Princeton, New Jersey, in 1998 and 2001, respectively. He is, at present, a research staff member with NEC Labs America, Inc., Princeton. His research interests include various aspects of embedded system security. He is responsible for designing the MOSES security architecture for NEC's application chips in 3G cell phones. He has several publications in leading ACM/IEEE conferences and journals on VLSI, including invited contributions and talks at the International Symposium on System Synthesis (2002), the International Conference on VLSI Design (2003, 2004), and the Design Automation and Test in Europe Conference (2003). His papers have received awards at the International Conference on VLSI design in 1998, 2000, and 2003. He received the Siemens Medal from the Indian Institute of Technology, Madras, India in 1996. He is a member of the program committees of the VLSI Test Symposium (VTS) and the Design Automation and Test in Europe (DATE), and has served as the Compendium Chair in the 20th Anniversary Committee of the VTS. He is a member of the IEEE.



Anand Raghunathan (S'93, M'97, SM'00) received the BTech degree in electrical and electronics engineering from the Indian Institute of Technology, Madras, India, in 1992, and the MA and PhD degrees in electrical engineering from Princeton University, Princeton, New Jersey, in 1994 and 1997, respectively. He is currently a senior research staff member at NEC Laboratories America in Princeton, New Jersey, where he leads several research projects related to System-on-Chip architectures, design methodologies, and design tools, with emphasis on high-performance, low power, and testable designs. He has coauthored a book (*High-Level Power Analysis and Optimization*) and six book chapters, and has presented several full-day and embedded conference tutorials in the above areas. He holds or has filed for 18 US patents in the areas of advanced system-on-chip architectures, design methodologies, and VLSI CAD. He has received best paper awards at the IEEE International Conference on VLSI Design (one in 1998 and two in 2003) and at the ACM/IEEE Design Automation Conference (1999 and 2000), and three best paper award nominations at the ACM/IEEE Design Automation Conference (1996, 1997, and 2003). He received the patent of the year award (an award recognizing the invention that has achieved the highest impact) from NEC in 2001. He has served as a member of the technical program and organizing committees of several leading conferences and workshops. He was program chair of the IEEE VLSI Test Symposium in 2003. He has served as associate editor of the *IEEE Transactions on CAD*, the *IEEE Transactions on VLSI Systems*, and *IEEE Design & Test of Computers*. He is currently a senior member of the IEEE and vice chair of the Tutorials & Education Group at the IEEE Computer Society's Test Technology Technical Council. He was a receipient of the IEEE Meritorious Service Award and was elected a Golden Core Member of the IEEE Computer Society in 2001, in recognition of his contributions.

Niraj K. Jha (S'85-M'85-SM'93-F'98) received the BTech degree in electronics and electrical communication engineering from Indian Institute of Technology, Kharagpur, India in 1981, the MS degree in electrical engineering from the State University of New York at Stony Brook, New York, in 1982, and the PhD degree in electrical engineering from the University of Illinois, Urbana, Illinois in 1985. He is a professor of electrical engineering at Princeton University. He has served as an associate editor of the *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*. He is currently serving as an editor of the *IEEE Transactions on Computer-Aided Design*, the *IEEE Transactions on VLSI Systems*, the *Journal of Electronic Testing: Theory and Applications* (JETTA), and the *Journal of Embedded Computing*. He has served as the guest editor for the JETTA special issue on high-level test synthesis. He has also served as the program chairman of the 1992 Workshop on Fault-Tolerant Parallel and Distributed Systems and the 2004 International Conference on Embedded and Ubiquitous Computing. He is the director of the Center for Embedded System-on-a-Chip Design funded by the New Jersey Commission on Science and Technology. He is the recipient of the AT&T Foundation Award and NEC Preceptorship Award for research excellence, the NCR Award for teaching excellence, and a Princeton University Graduate Mentoring Award. He has coauthored three books titled *Testing and Reliable Design of CMOS Circuits* (Kluwer, 1990), *High-Level Power Analysis and Optimization* (Kluwer, 1998), and *Testing of Digital Systems* (Cambridge University Press, 2003). He has also authored four book chapters. He has authored or coauthored more than 270 technical papers. He has coauthored six papers which have won the Best Paper Award at ICCD'93, FTCS'97, ICVLSID'98, DAC'99, PDCS'02, and ICVLSID'03. Another paper of his was selected for "The Best of ICCAD: A collection of the best IEEE International Conference on Computer-Aided Design papers of the past 20 years." He has received 11 US patents. His research interests include low power hardware and software design, computer-aided design of integrated circuits and systems, digital system testing, and distributed computing. He is a fellow of the ACM and the IEEE.