Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Communications 27 (2004) 1716–1729
computer communications
www.elsevier.com/locate/comcom

On a framework for energy-efficient security protocols in wireless networks

Phongsak Prasithsangaree*, Prashant Krishnamurthy

Telecommunications Program, School of Information Science, University of Pittsburgh, 135 N. Bellefield Ave, Pittsburgh, PA 15260, USA

Received 25 May 2004; accepted 25 May 2004
Available online 1 July 2004

Abstract

Network security is an important issue especially in wireless networks where the network is open and the network perimeter is not exactly known. This makes wireless networks more vulnerable to attacks such as eavesdropping, message interception and modifications. Several security protocols designed for wired-line networks have been adopted for use in wireless networks. However, they may not be suitable for wireless networks and devices since scenarios and capabilities applicable to wired-line networks may not be valid in wireless networks. For example, wireless devices often have limited battery power, and performing several message exchanges used in typical wired-line security protocols may rapidly deplete the devices' battery. Cryptographic primitives consume energy and could degrade the battery performance of wireless networks. In this paper, we classify energy saving mechanisms for security protocols in wireless networks. We apply these energy saving mechanisms to existing security protocols and demonstrate the reduction in energy consumption that is possible with the suggested approaches.

Keywords: Energy efficiency; Security protocol; Wireless networks; Energy consumption; Adaptive security

1. Introduction

In the past few years, wireless data networks have been exponentially growing. Many business and information technology applications have relied on wireless data networks such as IEEE 802.11 Wireless Local Area Networks (WLANs). The threats to those networks are also growing. The first version of the security protocol, Wired Equivalent Privacy (WEP), of the WLAN standard is widely known to be vulnerable to several attacks [1–3]. Due to the discovery of vulnerabilities of WLANs, many business and government sectors have temporarily ceased to adopt WLANs in their firms because of the increased threats to their businesses [4]. Thus, providing security services is no longer optional for wireless network operators.

A significant problem in today's wireless networks is that the available battery power is limited. The battery power will be even more limited in the future due to the increasing imbalance between computing power and battery capabilities. According to Moore's law, the number of transistors in a chip will be doubled roughly every 18 months [5]. However, the capacity of batteries is growing linearly, and this introduces a *battery gap* which is the difference between the power required for computing and the battery capacity [6]. While it is believed that fuel cells, which offer more capacity, will replace traditional batteries, this is unlikely to happen in the near future due to the size, cost and safety of fuel cells [7]. Thus, the battery power will still be a very limited resource for small wireless devices for many years to come.

A security protocol typically provides security services such as data confidentiality, message authentication and integrity. The security services or features may form part of a higher-level security policy. The security protocol utilizes security mechanisms comprising one or more primitives such as ciphers for encryption and hash functions or message digests for message authentication and integrity. These primitives form the core of security protocols and they may consume a fair amount of energy for computation. As reported in Ref. [8], encrypting only 13.6 kilobytes of data using the Blowfish cipher with a 32-bit key on

* Corresponding author.
E-mail addresses: phongsak@sis.pitt.edu (P. Prasithsangaree), prashant@sis.pitt.edu (P. Krishnamurthy).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.05.022

a handheld device drained about 75% of the battery power. For sufficient security strength today, it is recommended that key sizes of at least 80 bits be employed. Usually, a longer key implies more operations and the battery can be drained even more quickly. Additionally, the wireless transmission process also consumes a fair amount of energy. While energy efficient transmission protocols have been widely studied for wireless networks [9], very few results consider energy efficient security protocols over a wireless link.

Because of the limited battery power of devices in wireless networks, providing security services without draining the battery quickly is a challenging task. We need security protocols that are energy efficient but at the same time provide security levels comparable to those existing in wired-line networks. Simply borrowing security protocols from wired-line networks for use in wireless networks is likely to be inadequate because they typically do not consider energy efficiency as a factor in the protocol design. They are also not adaptive to the environment—which in the case of wireless networks is known to be dynamic and heterogeneous. For example, it may be advantageous to use one encryption scheme for short packets and another for larger packets. The choice of the packet size may itself depend on the channel conditions. Therefore, we believe there is need for a security framework that can be used to design energy-efficient security protocols that are adaptive to the environment.

In this work, we present approaches towards an adaptive framework for security protocols used in wireless networks such as WLANs. In Section 2, we describe some related work. In Section 3, we discuss methods of computing the energy consumption of cryptographic primitives and how they vary with algorithm, packet size, key size and number of operational rounds. Section 4 describes three methods for energy conservation for security protocols. Section 5 applies two of these methods and demonstrates the possibility of conserving energy in security protocols. Limitations of the work and conclusions are presented in Sections 6 and 7.

2. Related work

Several papers have proposed energy efficient communications protocols for wireless devices. Most of them have focused on protocols between a mobile station (MS) and an access point (AP) or base station. To our knowledge, very little work has been done on developing a framework for energy efficient security protocols in wireless networks. In this section, we provide a brief overview of energy efficient communication protocols and some work related to energy efficiency of security protocols.

2.1. Energy efficient communications protocols

Many techniques for improving the energy efficiency of wireless communication protocols have been proposed by the research community at the link, network, operating system, and device levels in wireless systems [10]. Such techniques are applied at different layers in the protocol stack and they aim to reduce the energy consumption of a wireless device. A typical procedure to improve energy efficiency works as follows. The factors that consume the most energy in each layer are examined and methods that can retain essential functionality while saving some energy are applied. Some of these techniques are straightforward. At the circuit level, it is possible to use better fabrication methods, reduced clock speeds, and so on to reduce power consumption. At the operating system level, the way in which the CPU schedules processes can be optimized towards reducing energy consumption. At the protocol level, in order to maintain proper functionality, certain changes may be required in the protocol design itself. For example, if a MS goes into a sleep mode, some mechanisms will have to be created to wake it up periodically to see whether there are messages intended for it. Also special management messages (like the traffic indication map-TIM in 802.11) are required to indicate the presence or absence of messages to a MS that may be waking up from sleep mode.

Most energy-saving techniques at the communication protocol level are focused on the link layer because the wireless link is the place where energy consumption becomes important. The link layer is composed of two sublayers, the Logical Link Control (LLC) and the Medium Access Control layers. The LLC sublayer often implements error control using Automatic Repeat Request (ARQ) and/or Forward Error Correction (FEC). These mechanisms consume extra bandwidth and energy due to additional transmission overhead or packet retransmission. ARQ is used to retransmit packets that are lost or corrupted during transmission. The energy consumed by retransmission can be high in wireless environments when a channel is in a bad state and error rates are high. To reduce the energy consumption due to retransmission, one method used is avoiding persistence in retransmitting data. Such methods trade the number of retransmission attempts for a reduced probability of successful transmission. Alternatively, another method defers retransmissions when the channel error rate is high. This method employs an adaptive mechanism that adjusts data transmission according to channel conditions. In Ref. [11], an example of an adaptive ARQ protocol is presented. During the normal operation of the ARQ, the MS sends data packets and receives packet acknowledgments (ACK). When a packet is lost or corrupted and the corresponding ACK is not received, regular transmission is stopped. Instead of retransmitting the lost packet, the MS enters a probing mode by periodically sending a short packet (probe). Regular transmissions resume only when an ACK for a probe is received, which very likely reflects an improved channel condition. Such a protocol is energy efficient in that sending probe packets consumes less energy than persistently retransmitting the lost packet which may be much larger in size. It is possible to combine ARQ and FEC adaptively depending on the channel conditions to further improve the energy efficiency as in Ref. [12].

2.2. Other work related to energy efficient security protocols

Recently, some work has focused on studying the energy consumption of cryptographic primitives and investigating methods of reducing it using a variety of techniques. However, most of these studies are not at the protocol level. In Ref. [13], the computational complexity of public key cryptography (PKC) on an embedded processor has been studied. This work concentrates on employing mathematical techniques and parameters to improve the performance of public key cryptography used in the Handshake protocol of the Secure Socket Layer (SSL) protocol. Gupta et al. [14] studied the performance of Elliptic Curve Cryptography (ECC) based algorithms for the Handshake protocol. Several experiments were run on an MS client and a server, and the results were compared to those of RSA in term of latency at the MS client and the server. ECC algorithms are believed to be computationally less intensive because of smaller key sizes and hence suitable for wireless devices [15]. In Ref. [16], Law et al. studied the energy consumption of encryption for sensor networks. In their work, the efficiency of code sizes and two algorithms (RC5 and Tiny Encryption Algorithm-TEA) are studied. Yuan and Qu proposed an energy saving technique that uses dynamic voltage scaling to reduce the energy consumption of public key encryption such RSA, DSA, and ElGamal [17]. In Ref. [18], an optimization of the energy consumption of the SSL protocol is studied. The technique in this case is based on using a compression algorithm to reduce the size of the messages exchanged by the protocol to reduce the power consumed by encryption and transmission.

In our previous work, we studied the energy consumption of CAST, IDEA, and Triple-DES symmetric key encryption schemes on wireless handheld devices [8]. An approach similar to the use of short probe packets in Ref. [11] towards energy efficient security protocols was also first suggested in Ref. [8], but there was no extensive study of the design of such protocols. The energy consumption of RC4 and AES was considered by us in Ref. [19] with WLANs in view. Based on our results, we suggested an energy efficient security protocol that adaptively changes the encryption algorithms according to the packet size in Ref. [19]. Once again, no actual design of the protocol was considered.

All of the above techniques except the suggestion in Ref. [19] lack adaptability to wireless environments, which are known to be continually changing and heterogeneous. For example, different types of packets exchanged in a protocol may need different security services. In WLANs, only data packets may need confidentiality while control and management packets (which do not contain any secret data) may require only authentication. The size of the packets may affect the energy consumption of the security primitives. Some ciphers are ‘energy efficient’ only when they are used to encrypt large packets. Within wireless environments where packet sizes can be diverse, we can increase the energy efficiency by using different ciphers for different packet sizes. In this paper, we propose a framework for energy efficient security protocol design based on our extensive studies on the performance of security primitives. Then, we use our framework to show two preliminary approaches for reducing the energy consumption of security protocols in WLANs.

3. Computation of the energy consumption of security primitives

In order to design a framework for energy efficient security protocol design, the primary step is to determine how much energy is consumed by security primitives under various circumstances. In what follows, we assume that the primitives are built in software or firmware and make use of the CPU of a MS. We do not consider the situations where a separate cryptographic processor or hardware implementations are used for performing computationally intensive operations.

Energy consumption of security primitives can be measured in many ways. The first method is to directly measure the energy consumption of each component of a device such as a laptop or a PDA. This method gives the actual value of energy consumption due to each hardware component of the device. One of the components will be the CPU operations for the encryption algorithm. However, this method is probably too complex to measure only the energy consumption of a security primitive. This method is used in research work by Viredaz and Wallach to measure the general energy consumption in a pocket personal computer device [20].

The second method used to measure energy consumption is to assume that an average amount of energy is consumed by normal operations and to determine the extra energy consumed by an encryption scheme [21]. This method simply monitors the percentage of remaining battery level by using the standard application programming interface (API) of Advanced Power Management (APM) or Advanced Configuration and Power Interface (ACPI) in laptops or other MSs. One limitation of this method is that the extra energy consumed by one encryption can be very small (with secret key algorithms and hash functions but not public key algorithms) compared to that by normal operations. Thus, the granularity of power measurement is coarse and the amount of energy consumption measured by this method is unlikely to be precise. The accuracy of this method could be improved by using an external multi-meter that can quickly read the level of current drawn by one encryption and it also directly shows the impact on the battery.

Third, the energy consumption of the cryptographic primitives can be measured *indirectly* by counting the number of computing cycles, which are used by the CPU in computations related to cryptographic operations. The number of cycles is usually used by cryptographers to evaluate new encryption algorithms such as those which were proposed for the Advanced Encryption Standard (AES) [22]. Each cycle of the CPU spent for an instructional set consumes some amount of energy. The number of CPU cycles and the energy consumption has a linear relationship on average in which an increase in the number of cycles increases the amount of energy consumption proportionally. In Ref. [23], details of the numbers of cycles taken up by each instruction of Intel 486DX2 processor and also the amounts of current drawn from a battery are provided. The number of cycles can be affected by preemptive multi-tasking in an operating system. However, by performing the same task many times and by averaging the amount of energy consumed in each cycle, we can convert the number of cycles used by a security primitive into a fairly good estimate of the amount of energy consumed. This technique is also used in the research work by Carmen et al. [24]. A more accurate measurement would actually use the time variation of current in computing the energy rather than the average value.

While the first method shows the real energy consumption and gives the overall energy performance and the second method actually reads out the battery level in a mobile device, in this work, we use the indirect method to determine the energy consumption of a security algorithm. To convert cycles to the energy consumed, we need to know approximately how much current (I) is drawn for one computation cycle (C), the operating voltage (V), and the CPU clock frequency (F) of the processor. For example, an Intel Mobile Pentium III processor has two working modes, full power mode and low power mode. In the full power mode, the processor is operating at 1.60 V with a maximum current of 16.6 amp at the speed of 800 MHz. In the low power mode, it is running at 650 MHz, and operating at 1.35 volts with a maximum current of 12.0 amp [25]. From the above variables, the energy consumption (E) of each security algorithm can be computed as $E = (C \times V \times I/F)$. We do not have a benchmark tool that can measure how much current is drawn for each instruction or each cycle for the Mobile Pentium III processor that we use in our experiments. Therefore, we use an approximate value in this work. On average, each cycle consumes approximately 270 $>$ mA on an Intel 486DX2 processor [23] or 180 mA on Intel Strong ARM [26]. Based on these, we make the approximation that the average current drawn is close to 200 mA. This number is fixed for all energy consumption calculations, and it could be changed easily for energy calculations if the true average current for the processor is known.

3.1. Secret and public key algorithms for wireless systems

In this section, we provide a quick overview of encryption algorithms considered in the rest of the paper. We consider both secret and public key algorithms.

The encryption algorithm in WEP is based on RC4, a stream cipher designed by Ron Rivest in 1987. RC4 is also widely used in many applications today and in other wireless networks such as CDPD [10]. Using a shared secret key, RC4 generates a key stream of pseudo-random numbers. Before generating the key stream, a key expansion process and an initial permutation process are required, which introduce computational overhead for each packet. It is possible to cache the outputs of these processes into memory for better efficiency. As keys in WEP and RC4 make use of initial vectors that change (sometimes for each packet), this may not be entirely possible. One feature of RC4 is that the computational load does not depend on the key size or the number of operational rounds (a repeated operation to increase security that is common in block ciphers). The encryption of data using RC4 is based on simply XORing the pseudo-random numbers from the stream with the plaintext data. Consequently, RC4 is fast and efficient once the initial processes are completed. It can be written using only a few lines of codes and requires only 256 bytes of RAM. It typically uses only 7 CPU clock cycles per byte of output on a Pentium® CPU architecture [27]. Hence, it has been one of the best encryption schemes during the past decade. However, Fluhrer and other researchers have discovered vulnerabilities in the RC4 algorithm [1] for short keys.

The weaknesses in RC4 and loopholes in the WEP protocol have resulted in a standards effort for security in WLANs (IEEE 802.11i) that is proposing a new security protocol based on the Advanced Encryption Standard (AES) [28]. AES (previously called Rijndael) is a block cipher that has a variable key length of 128, 192, or 256 bits to encrypt data blocks of 128 bits (192 and 256 bit blocks are also possible with Rijndael). Both block and key lengths are extensible to multiples of 32 bits. To produce a block of ciphertext, a plaintext is first manipulated with the key (called Add Round Key process), and it then goes through several operational rounds (known as Rijndael rounds). The more the operational rounds, the more is the strength of the AES algorithm (but more will also be the computation). The Add Round Key requires creation of round keys that is a computational overhead for AES encryption. AES encryption is fast and flexible. It can be implemented on various platforms especially in small devices and smart cards. AES has been rigorously tested for security loopholes for a few years before it was standardized by the National Institute of Standards and Technology (NIST). It is considered to be very secure for more than seven rounds. In terms of the choice of algorithms in WLANs, both RC4 and AES have different tradeoffs. AES for instance, needs to be used in the counter mode to generate a pseudorandom stream in

WLANs. Additionally, a study of their energy consumption is needed to decide on their use in security protocols. A preliminary study of energy consumption of RC4 and AES algorithms for WLAN is presented in Ref. [19]. Other secret key algorithms considered in this paper are RC5, Blowfish and f8. RC5 has been considered in the literature for sensor networks [16]. Blowfish is supposed to be a light-weight encryption scheme suitable for handheld computers (although it has a key expansion process that requires additional computational time). UMTS systems employ a variation of AES in their f8 algorithm that operates in the cipher-block chaining (CBC) mode.

Hash functions are also as important as the above mentioned ciphers. Hash functions are used to provide data integrity in digital signatures and public key certificates. They are also used in Message Integrity Codes to provide message authentication and integrity that can rapidly detect malicious message injection into or modification of messages in networks. The secure hash algorithm (SHA-1) and MD5 are two commonly used hash functions we consider in this work.

We briefly describe two public key schemes, RSA and ECC, here that are primarily used for digital signature creation and verification or for key exchange. RSA was invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. The algorithm is based on the integer factorization problem. Basically, the security depends on the fact that given the product of two very large prime numbers, it is computationally hard to find the prime numbers. RSA has been popular for over 25 years. The security of ECC is based on a mathematically hard problem called the Elliptic Curve Discrete Log Problem (ECDLP). Given P and $Q = kP$ where P and Q are discrete points on an elliptic curve and k is a random integer, it is computationally hard to find k given P and Q. The operation kP is called scalar multiplication. To prevent a successful attack, k must be a large integer, and the number of discrete points on the elliptic curve must also be large. Algorithms (such as Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signatures and Elliptic Curve Diffie-Hellman (ECDH) for key exchange) have been developed based on ECDLP [29]. Several elliptic curves (ECs) are recommended by NIST in the FIPS 186-2 standard [30]. The standard recommends three types of ECs, a prime curve, a random binary curve and a Koblitz binary curve, each of which has five different key sizes. Prime curves or elliptic curves over prime fields are popular because many existing efficient techniques in software used with prime number based cryptographic systems such as RSA can also be used here. The sizes of keys in the case of prime curves are 160, 224, 256, 384, or 521 bits. However, in terms of hardware implementation, binary curves or elliptic curves over binary fields are preferred [31]. There are two types of binary curves: Random and Koblitz curves. The random curve is simply an EC over a random field F_2^m where m is a prime number. The Koblitz curves are anomalous binary curves with special efficient techniques that reduce scalar multiplication time [32]. The sizes of keys in binary curves are 163, 233, 283, 409, or 571 bits.

3.2. Modeling energy consumption of security primitives

Security can be provided at different levels using different settings of a secret key encryption algorithm. It is believed that with a strong secret key cipher, an 80-bit key is sufficient for security [33]. This belief is based on the argument that a strong cipher has no known short-cut attacks. Using a brute force search with an 80-bit key requires 2^{79} searches on average. This is considered to be infeasible with today's technology. Thus, a large key size would provide more strength against a brute force attack. For example, with a 128-bit encryption key, a brute force attack would need to try (on average) 2^{127} different keys to find the right key. On the contrary, a long key may increase computation and hence energy consumption. Key sizes of 1024 bits for RSA and of 160 bits for ECC are recommended [33] because of mathematical attacks that can be used against these schemes.

The strength of a cipher depends not only on the key size but also the number of operational rounds which normally perform a routine of data diffusion and manipulation. The number of rounds is an important factor that provides security strength against attacks such as linear cryptanalysis [34] and differential cryptanalysis [35]. Unlike the brute-force attack where 2^{k-1} trials (on average for a k bit key) are required to break the key, cryptanalytic attacks can be more efficient. Depending on the algorithm, a higher number of rounds may be required for robustness against such attacks. A good security algorithm should not perform too few or too many operational rounds. Too few rounds would make the encryption vulnerable to cryptanalysis attacks, and too many rounds would be unnecessary, and importantly it consumes more energy. Thus, to provide the same security strength, each algorithm may require a different number of operational rounds.

For example, AES needs at least 6 rounds for a 128-bit key, 7 rounds for a 192-bit key, and 9 rounds for a 256-bit key to provide enough security strength against known attacks [22]. However, the proposed standard adds more rounds for providing a security margin. Thus, AES performs 10 rounds for a 128-bit key, 12 rounds for a 192-bit key and 14 rounds for a 256-bit key [36]. RC5 is a block cipher with a variable key size, a variable input size, and variable numbers of rounds of operations. There is a differential attack that requires 2^{53} chosen plaintexts for 12 rounds and 2^{68} for 15 rounds. However, the input of RC5 is only 64 bits long; therefore, such an attack is impossible with at least 16 rounds. The linear attack, which is less powerful than the differential attack, is impossible after 6 rounds. The recommendation from Rivest, the creator of RC5, is that RC5 should have at least 12 rounds to provide enough security strength, or 16 rounds to provide complete security strength [37]. Blowfish is a flexible block cipher with a variable key length and a customizable S-box (a non-linear diffusion box). The key for Blowfish can be up to 448 bits. There is a differential attack which requires 2^{4r+1} chosen plaintexts where r is the number of rounds. Therefore, with 16-round Blowfish, such an attack is completely ineffective because the input size in the case of Blowfish is 64 bits long.

Clearly, different combinations of key sizes and rounds will also have different energy consumption. In wireless communications, as we will see later, the data packet size is also another factor that impacts the energy consumption. For instance, some security algorithms are more efficient only if they are used to encrypt longer packets. In the next sections, we explore the performance of different security settings in terms of energy consumption. We use the cryptographic software library from OpenSSL version 0.9.7a [38] in our work. While other libraries do exist [39], we use OpenSSL because it has been widely used in the research community and in many open-source research projects. It has been rigorously reviewed by many cryptographers and programming experts for its correctness and performance. It is possible to optimize cryptographic algorithms for a given platform. However, writing ones own cryptographic code is not considered a good idea [39] and it is better to use well-tested crypto-libraries.

3.3. Energy consumption of secret key cryptographic primitives

In this section, we discuss the performance of different cryptographic primitives in terms of the energy consumption as a function of three parameters: data packet sizes, key sizes and operational rounds. In our experiments, we use the indirect method with the laptop (described earlier in Section 3) to estimate the energy consumption. In each experiment, we repeatedly execute the cryptographic function on a 5.5 MB file with different parameters 500 times and obtain the average value. The variance of the energy consumption was less than 1% in our experiments.

3.3.1. Encryption with different packet sizes

In a data network, the packet size often has an effect on performance. A long data packet can increase throughput in a wired network. Each packet encryption also has an overhead due to the key expansion process required before encryption as discussed earlier. The key expansion of some encryption algorithms needs subtle data diffusion and manipulation. For example, in the case of Blowfish, a 128-bit key expansion process takes around 98 and 94% of the total energy to encrypt a 16 and 256 byte long packet, respectively. A long packet would thus reduce the security overhead and improve the security performance. However, longer packets are easily subject to corruption and loss in wireless networks because the channel can alternate between a good state and a bad state. Thus, a smaller packet size is desirable in terms of packet losses.

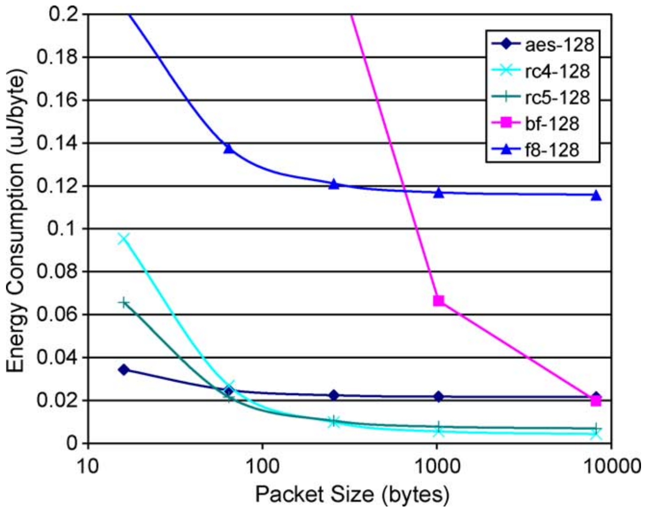

Fig. 1. Energy consumption of encryption algorithms with different packet sizes.

Fig. 1 shows the energy consumption of different encryption algorithms using a 128-bit key with different packet sizes. The key size is kept constant at 128 bits. The graph should be read as follows: If a 5.5 MB file is encrypted with AES with a 128 bit key by breaking it up into 1000 byte packets, encrypting each byte will consume 0.02 μJ on average. From the figure, we can see that AES encryption consumes far less energy than others for small packets because the key expansion overhead is small. Packet sizes also only slightly affect AES. RC4 and RC5 consume more energy than AES for smaller packets, but are better for larger packets. Blowfish has a significant key preparation overhead and it is not suitable for packet transmission [40]. The f8 algorithm is fairly expensive due to its CBC mode of operation.

3.3.2. Encryption with different key sizes

The second important parameter for encryption we consider is the size of the encryption key. In this measurement, we consider the energy consumption with different key sizes. To eliminate the effect of overhead, we measure energy consumption due to encryption only with the key preparation already completed.

Fig. 2 shows the amount of energy consumed with different key sizes. It can be seen that AES consumes different amounts of energy for different key sizes. With a longer key, the encryption needs to do more ‘Rijndael’ rounds which result in more computations. AES with 128, 192, and 256 bit keys requires 10, 12, and 14 rounds, respectively as already discussed. RC4 encryption relies on a random number generator or a key stream generator to produce a key stream. Generating a key from the stream consumes very little energy and it is independent of the key size. The energy consumption of RC5 encryption is also likely to be independent of the key size. Encryption with a longer key in RC5 does require more additions and rotations of 32-bit words. These operations are very simple

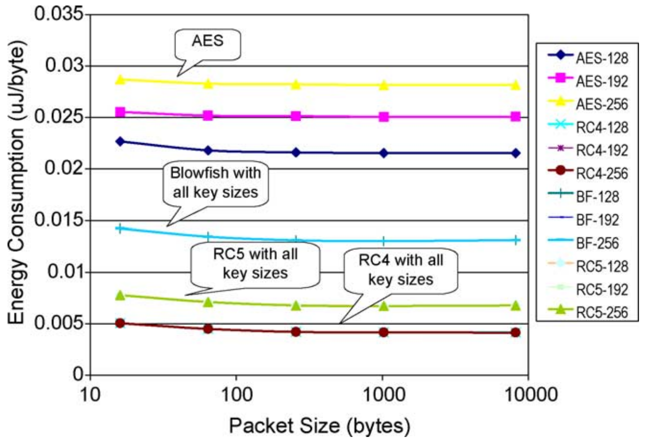

Fig. 2. Energy Consumption of only encryption with different key sizes.

and increasing the key size does not significantly increase the energy consumption of the encryption. In Blowfish, the computational load is also independent of the key size. No matter what the key size is, Blowfish always uses 16 rounds of a Feistel-like network. However, the key size has a large impact on the key expansion before encryption for RC5 and Blowfish.

3.3.3. Encryption with different number of rounds

Fig. 3 shows the energy consumption of encryption algorithms with different numbers of rounds of RC5, AES and Blowfish. The key expansion process is not considered here. RC4 is not shown here since it is a stream cipher not based on operational rounds. The algorithms are used to encrypt packets of 1024 bytes. It is shown that AES consumes energy at a higher rate than others as the number of rounds increases. RC5 is probably a good cipher for constrained devices in terms of energy consumption. However, it has a high overhead due to key expansion.

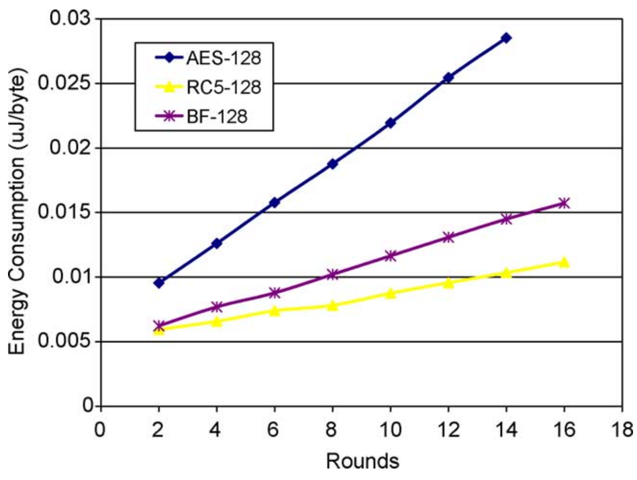

Fig. 3. Energy consumption of only encryption with different numbers of operational rounds.

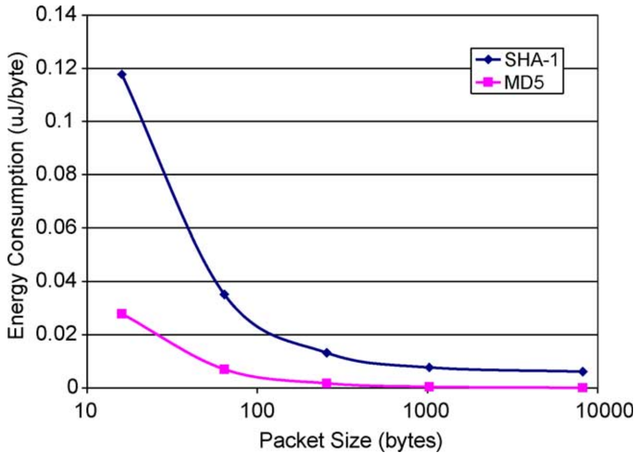

Fig. 4. Energy consumption of hash functions.

3.3.4. Performance of Hash functions

Fig. 4 shows the energy consumption of SHA-1 [41] and MD5 [42] hash functions with different packet sizes. It can be seen that SHA-1 consumes much more energy than MD5 especially where the packet size is small. To create a digest of a packet, SHA-1 processes 512 bits at a time, and continues for as many 512-bit blocks as in the packet. Each 512-bit block is passed through four rounds, each of which are identical and have 20 operations. Each operation performs a non-linear function, adding, and shifting of bits. MD5 also has four rounds of operations, but each round has only 16 operations. Each operation in MD5 is much simpler than that in SHA-1. This results in less computation and energy consumption of MD5 than those of SHA-1. Note that the SHA-1 produces a 160-bit hash which is longer and more secure than a 128-bit hash produced by MD5. MD5 is also known to have security loopholes unlike SHA-1 [43].

3.3.5. Performance of RSA and ECC

Public key algorithms are primarily used for digital signatures and key exchange. To test the performance of RSA and ECC, we use the same experiment set as described in Section 3. In Fig. 5, we show the energy consumption of the RSA digital signature algorithm and ECDSA with different key sizes. It is seen that the RSA signing algorithm consumes more energy than the ECDSA signing algorithm. For example, the RSA signing algorithm with a 4096-bit key needs about 180 mJ while ECDSA with an equivalent 283-bit key requires only about 7% of that energy. In contrast, we can see that the RSA verifying algorithm consumes a very small amount of energy compared to ECDSA. In the case of binary curves, the ECDSA signing process consumes about half the energy consumed by the corresponding ECDSA verifying algorithms. However, ECDSA signing and verifying algorithms with prime curves consume almost the same amount of energy.

Fig. 6 shows the energy consumption of the RSA encryption/decryption and the ECDH algorithm, which are

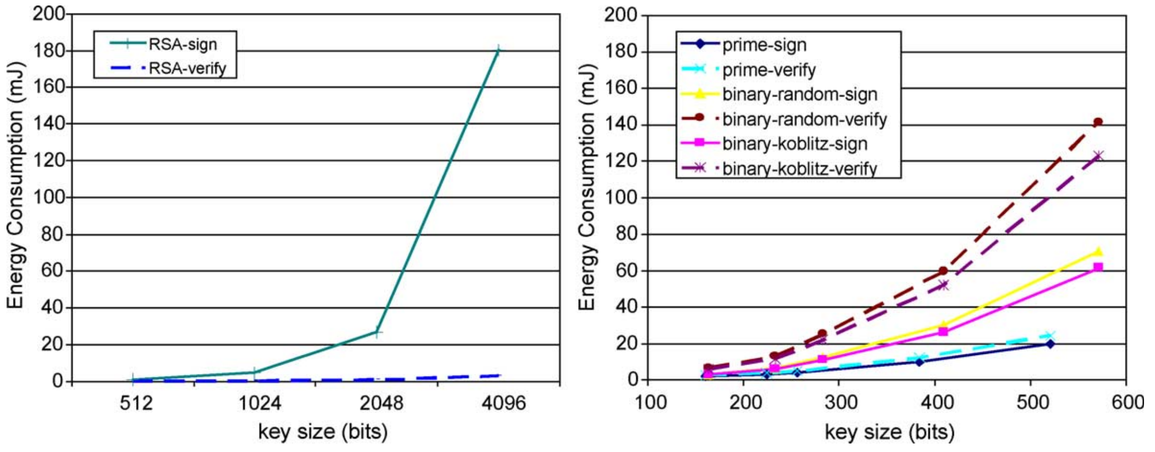

Fig. 5. Energy consumption of different digital signature schemes with different key sizes.

used to exchange a pre-master secret between a MS client and a server. The energy consumed by RSA encryption and decryption does not depend on the size of the pre-master secret. In transport layer security SSL/TLS [44], the pre-master secret size is 48 bytes (384 bits). Therefore, encrypting the pre-master secret using 1024-bit RSA saves about 9 and 25 mJ compared to that using the 384-bit ECDH (prime curve) and the 409-bit ECDH (Koblitz curve), respectively. Also, one observation here is that public key algorithms consume several orders of magnitude more energy than secret key algorithms.

3.4. Energy consumption of transmission

A significant source of energy consumption also comes from message transmission and reception. To calculate the energy consumed by transmitting and receiving messages at a MS client device, we use the transmission energy model from Ref. [45] with a Lucent 802.11b PC card operating in the infrastructure mode. This model uses a linear fit based on experiments to determine the amount of energy consumed by transmission or reception per byte. For transmission, $E = 0.48x + 431$ (μJ) and for reception $E = 0.12x + 272$ (μJ) where E is the energy consumed and x is the number of bytes transmitted or received. All performance measurements are only done at the MS where the device is more resource-constrained than at devices in the fixed network. From the number of transmitted and received bytes, we can estimate the total energy consumed by message transmission from this model from [45].

4. A framework for energy efficient security protocols

Based on the evaluation of energy consumption of security primitives, we see that there is potential for reducing the energy consumption of security protocols for wireless networks. We propose three different methods by which energy can be conserved that can be applied towards the design of energy efficient security protocols.

4.1. Method 1: Eliminating/replacing most energy consuming parts

The first and obvious way to reduce energy consumption is to eliminate or replace the most energy consuming part of existing standard protocols. Besides transmission and reception of packets, the most energy consuming part of

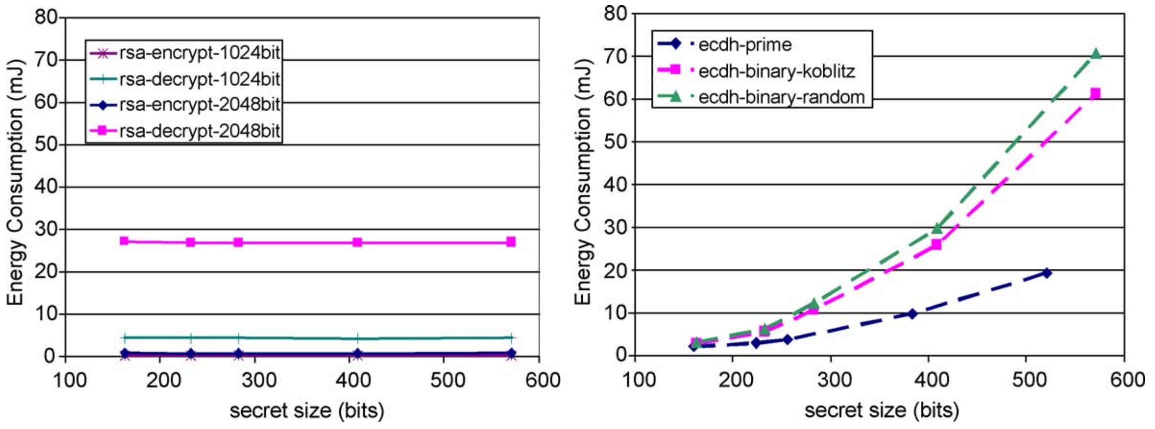

Fig. 6. Energy consumption of RSA encrypt/decrypt and ECDH key exchange algorithms.

the security protocols is due to cryptographic functions such as encryption, key generation, and digital signatures. From our findings as described earlier, different primitives consume different amounts of energy. Looking at several standard protocols, we also find that most of them allow different primitives to be used during a protocol transaction, but the protocols are not adaptive after a primitive has been selected for use for a whole session. However, we believe it is possible to carefully select a combination of primitives to be used in a single session, which is more energy-efficient than a standard one. As an example, assuming that both RC4 and AES with 128 bit keys provide similar security levels, it is better to use AES for shorter packets and RC4 for longer packets to provide confidentiality as it is in current protocols. If message integrity is required, it brings up another dimension (see Method 3).

4.2. Method 2: Modifying protocol primitives and transactions

The second method to reduce energy consumption of security protocols is to modify the protocols themselves. Beyond changing the security primitives, we can modify the protocol primitives (messages) and transactions. Some protocols are not energy-efficient since they have to exchange several primitives and messages; for example, a Kerberos client needs to exchange at least six messages before it can access network services [46] and employing it on a WLAN may result in a larger consumption of energy than required. However, the protocol modification should not change the security level already defined in the existing protocol. To ensure security equivalence, formal methods are often used to verify security strength and to find flaws of a protocol. A well-known formal method is BAN logic [47]; however, it has limitations in that it does not capture the aspects of real systems [48,49]. In a different way, we can also show security equivalence by reasoning or arguments based on lessons learned [50,51]. To implement protocol modifications and achieve energy savings, we can make use of security protocol standards that allow us to 'plug-in' any security transaction. One example of such a protocol for authentication is the Extensible Authentication Protocol (EAP) [52]. In this paper, we do not show an example of this method.

4.3. Method 3: Using an entirely new security framework

The last method we suggest is to take the greenfield approach where we start from scratch and come up with security protocols that consume the least energy. Such a method will offer tunable and adaptive security services. As an example, consider data integrity for packets over a WLAN. MD5 is more efficient than SHA-1 for shorter packets, but it is also less secure. MD5 may still be used for shorter packets that do not need a lot of security (it would be sufficient if the security is not breached for a few days instead of a few years). Another example is a modified communications/security protocol that adapts itself to channel conditions. For instance, knowing that the channel is bad, short probe packets can be transmitted with a low level of security (simply providing confidentiality for the duration of the probes). Both of these examples require the development of a policy for determining how much security is required for a given communication session (what is reasonable) and how this can be mapped into the appropriate primitives. We do not address the development of policies in this paper, but it is part of our on going work. We believe this method that tunes the security protocol based on a policy and known performance measures would significantly reduce energy consumption due to adaptive security provisioning and may also reduce the energy consumed by signal/message transmission.

Table 1
A comparison of different energy saving methods

Methodology	Method 1	Method 2	Method 3
Complexity	low	low	medium-high
Compliance	yes	maybe	no
Adaptability	no	no	yes
Energy Savings	low	medium	high

A comparison of the three energy saving methods for security protocols is shown in Table 1. The first method improves the energy saving by simply analyzing several existing security protocols and changing some protocol components. This method will still be compliant with existing standards; however, the energy saving is probably small. The second method probably offers more energy saving. However, it may or may not be backwards compatible with existing protocols, but it is within the scope of the standard. This method may also introduce a little complexity. The third method starts from scratch and creates a security protocol that utilizes a new energy efficient framework. This method introduces complexity ranging from medium to high and probably will not comply with any existing standards; however, we believe it has the potential to conserve the most energy.

5. Applying the framework

In this section, we show two simple examples of using the proposed energy saving framework described in Section 4 to conserve energy for security provisioning in WLANs. In the first example, we use Method 1, where we replace the most energy consuming part with something that consumes lesser energy. This example looks at a very simple variation of the handshake protocol for SSL where the energy consuming RSA signature algorithm is replaced by ECDSA, but other RSA processes are left as they are. In the second example, we employ a technique that is a part of Method 3. We adaptively change the encryption algorithm in a simple home WLAN. In the current WLAN standard, every packet transmitted over a wireless channel can be encrypted with RC4 algorithm in the secure mode [53]. However, to provide 'reasonable' security, the encryption needs not be equally applied to all packets. As an example, if only confidentiality services are required for some packets, encryption should only be applied to confidential data packets, and not to control and management packets such as the beacon and acknowledgment packets. Also if the packet size is small, we may use AES instead of RC4 since AES has less overhead [28]. Employing both RC4 and AES and switching between them are not possible in the current WLAN standard, but this example shows the potential energy savings that can be achieved under a good framework.

5.1. A variation of the handshake protocol for SSL

The SSL/TLS protocol is widely used to secure various network application protocols. It is the de facto standard for web-based transactions for e-commerce (such as online banking and shopping). It is used to secure application protocols such as email (IMAP, POP3, and ACAP) and network file systems (NFS). Recently, it has been used as part of the Extensible Authentication Protocol (EAP) standard and IEEE 802.1X standard for authentication in WLANs [54,55], and it is part of the Windows® XP release to support wireless 802.11 security [56]. Fig. 7 shows the protocol primitives of the SSL/TLS Handshake protocol [44]. First, the MS client and the server exchange random numbers (or nonces) and negotiate a 'cipher suite' with ClientHello and ServerHello. The cipher suite specifies which cryptographic algorithms can be used to provide confidentiality, authentication and data integrity. The server then sends its certificate signed by a Certificate Authority (CA) which is trusted by the MS and the server. The server also requests client authentication with CertRequest message.

The MS uses *PKC_Verify* process to validate the server's certificate by using the CA's public key (which is distributed beforehand). If it is valid, the MS obtains the server's public key from the server's certificate. The MS then sends its certificate (ClientCert) signed by the CA to

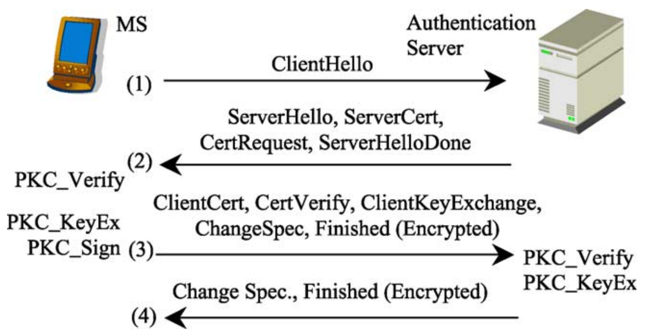
MS
Authentication Server
(1)
ClientHello
ServerHello, ServerCert,
CertRequest, ServerHelloDone
(2)
PKC_Verify
ClientCert, CertVerify, ClientKeyExchange,
ChangeSpec, Finished (Encrypted)
PKC_KeyEx
PKC_Sign
(3)
PKC_Verify
PKC_KeyEx
Change Spec., Finished (Encrypted)
(4)

Fig. 7. The handshake protocol with client authentication.

the server. The server validates the MS's certificate (using *PKC_Verify*). This process is usually optional for web browsers. The client also initiates a key exchange process (*PKC_KeyEx*) to exchange a pre-master secret. During this process, the MS sends an encrypted key in the Client-KeyExchange message to the server so that the server can generate the same pre-master secret by decrypting it using its private key.

Additionally, the MS needs to authenticate itself to the server by showing possession of its private key. The MS produces a message digest of previously exchanged messages and signs the digest with its private key (using *PKC_Sign*), and sends the signed digest within the CertVerify message. Using the MS's public key obtained from the MS's certificate, the server verifies the received digest (using *PKC_Verify*). If it is valid, the MS truly possesses the private key that pairs with the public key sent to the server. At this point, the MS and the server are authenticated to each other, and they already share a master key derived from the key exchange process. To complete the Handshake protocol, the MS and the server send the encrypted message digest of all previously exchanged messages to ensure the integrity of the messages and to show possession of the shared master key. Note that the three cryptographic processes, *PKC_Verify, PKC_KeyEx, PKC_Sign,* are different depending on which PKC algorithm (RSA or ECC) is used.

Fundamentally, at the MS, the Handshake protocol is composed of three processes: server authentication, key exchange, and MS authentication. In the server authentication process, a MS uses a digital signature verification (*PKC_Verify*) process to validate a server's certificate. This process uses either RSA or ECDSA verification algorithm but not both. The key exchange (*PKC_KeyEx*) process establishes a shared key which uses either RSA encryption/decryption or ECDH. The MS authentication (*PKC_Sign*) process enables an MS to prove its identity by digitally signing a message and sending it to a server. This process uses either RSA or ECDSA digital signing algorithm.

From our experiments (described in Section 3), we see that ECDSA has advantages over the RSA signature algorithm in terms of energy consumption. However, signature verification using ECDSA consumes more energy than the RSA signature verification algorithm. RSA verification, like encryption is quite cheap in terms of energy consumption. In the key exchange process, the RSA encryption algorithm consumes much less energy than the ECDH algorithm to exchange a 48-byte pre-master secret. This asymmetrical nature of energy consumption of the RSA and ECC algorithms is exploited in our proposed variation of the Handshake protocol.

The goal of the proposed variation of the Handshake protocol is to reduce energy consumption at only the MS side where the mobile device is energy constrained. The variation introduces additional computational load at

Table 2
The total of average energy consumption of a SSL/TLS AKE session

ECC Key Size (RSA Key Size)	Energy consumption (mJ)		
	RSA	ECC random curve	Proposed
163 (1024)	5.07	13.02	3.82
233 (2048)	28.51	25.86	8.14
283 (4096)	186.05	50.13	18.27

the fixed server-side device, but this would be tolerable at the server. In this variation, the MS needs to hold an ECC-based certificate and the server needs to hold an RSA-based certificate. The variation also requires more memory space to store both RSA and ECC modules than the traditional SSL protocol. However, it would not be a constraint since the cost of memory is smaller than the cost of battery.

The variation is that, in Fig. 7, the MS uses the RSA verification algorithm to validate the server's RSA certificate in the *PKC_Verify*process. Upon the certificate request from the server, the MS sends its ECC certificate and not an RSA certificate. Then, in the *PKC_KeyEx* process, the MS generates and encrypts a pre-master secret using RSA encryption algorithm along with the server's public key (obtained from the server's RSA certificate). To prove its identity in the *PKC_Sign* process, the MS signs all previously exchanged messages using ECDSA. The security strength of the proposed protocol is the same as that of the ECC and RSA algorithms if the proper key sizes are selected (160 bit keys with ECC are equivalent to 1024 bit keys in RSA).

Table 2 shows the total average energy consumption on the client wireless device using different Handshake protocols and different key sizes. These results are from the comparison of RSA, ECC with the random binary curve, and the proposed variation of the Handshake protocol. From Table 3, we see that by using the variation of the protocol, we can save energy up to 90% compared to 4096-bit RSA or up to 70% compared to the 283-bit ECC Handshake protocol. Note that the energy savings is small for smaller key sizes.

5.2. An energy efficient security protocol for home WLANs

We first study the distribution of the sizes and types of packets typically transmitted and received by a wireless

Table 3
The amount of energy saved by using the proposed protocol

ECC key size (RSA key size)	Energy saved using variant (mJ)	
	Compared to RSA	Compared to ECC
163 (1024)	1.25 (24.79%)	9.2 (70.70%)
233 (2048)	20.37 (71.46%)	17.72 (68.54%)
283 (4096)	167.78 (90.18%)	31.86 (63.55%)

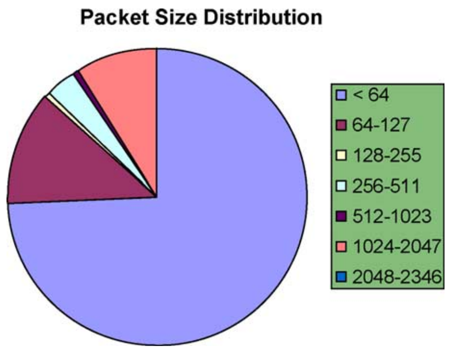

Fig. 8. The packet size distribution over one hour period of a home WLAN.

device over a home WLAN. We used a wireless sniffer to capture 802.11 network packets (about 10 MB over one hour) and obtained their packet size distribution of which about 8 MB are data packets and the rest are control and management packets as shown in Fig. 8. It is seen that around 75% of packets have a very small size (between 14 and 63 bytes) and they are control and management packets. The data packet is typically about 1536 bytes (which also depends on the maximum transfer unit (MTU) size). Data packets are less likely to be transmitted over a WLAN network since most WLAN users are idle or leave their laptops on but not using them [57] for communications. Although this packet distribution is from a small home network where there are only two MSs and one AP, the distribution of packets is likely to be similar for larger networks where users tend to access the network in bursts but management and control packets are transmitted periodically. For instance, the beacon packet in IEEE 802.11 WLANs is transmitted approximately once every 100 ms.

Using Method 3 as described in Section 4, we use different cryptographic primitives to adaptively provide security to packets based on their size and type. We show the results of how much its energy can be saved for security protocols in WLANs. We propose (see Fig. 9) an energy efficient security protocol with a simple adaptive mechanism. We simply look at the packet size and type and apply different cryptographic primitives to them. We use keyed MD5 [42] to provide message authentication to management packets such as an 802.11 ACKs (14 bytes long),

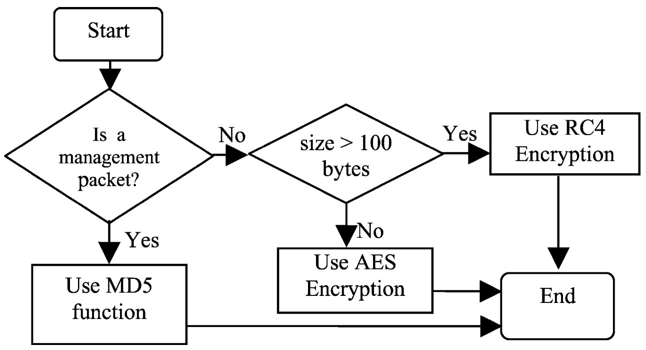

Fig. 9. A simple energy efficient adaptive security protocol.

Table 4
A Comparison of energy consumption using the fixed and adaptive security protocols

Algorithm	fixed RC4	fixed AES	**Adaptive**
Energy (mJ)	138.223	252.453	**91.579**
Percent saving	**33.74%**	**63.72%**	–

beacon packets (72 bytes long), and other short 802.11 management packets. The assumption here is that we need to provide only message authentication to the control and management packets to prevent packet modification. Additionally, the contents of these packets are not private; hence, encryption is unnecessary unless complete privacy is required (and perhaps to thwart traffic analysis). For data packets, we use RC4 or AES to provide confidentiality. To be more adaptive, RC4 is used only with the packets whose sizes are more than 100 bytes; otherwise AES is used. However, there are security implications beyond the simple mechanisms used here. For instance, if both RC4 and AES use the same key, if RC4 was broken and the key compromised, AES would also be broken. The detailed relationship between security levels, protocols and energy consumption is part of our ongoing work. From the results of the security performance in previous sections, we can compute the energy consumption of RC4, AES, and MD5 as a function of packet sizes.

For the scenario where we captured packets (10 MB in total), we calculate the energy consumed for each packet based on its packet size and type. Table 4 shows the results of energy consumption using a fixed algorithm (AES or RC4) and the proposed adaptive algorithm. For the fixed algorithm, only one selected encryption algorithm is used to encrypt all packets. We use a 128-bit keys for both RC4 and AES functions. The key size is selected as the minimum requirement for today's security from the study by Lenstra and Verheul [33]. From the table, we see that by using the adaptive algorithm we can save about 33 and 63% energy compared to using fixed RC4 and AES encryption algorithms respectively. To provide stronger security by using AES and save energy, we could also fragment a long packet into smaller packets and use AES to encrypt them. Additionally, smaller packets are likely to be less susceptible to wireless channel errors, and hence, we could save much more energy. Although the fragmentation would give significant energy efficiency, it will also lower transmission throughput (thereby increasing the energy consumed) and increase latency. The tradeoffs in this approach needs a more detailed investigation.

6. Limitations of this work

This work demonstrates the benefits of applying energy saving techniques in the design of security protocols for wireless networks. A major limitation of this work is that we do not tie it with the security requirements of the wireless network and the implications it may have on the security provided to a wireless network. For a given network or application we need a security policy-a statement of what is allowed and what is not allowed. The security services that implement the security policy can then be defined (e.g. confidentiality for data packets, only authentication for probes, ACKs and management packets). Then the way these security services use security mechanisms and primitives adaptively to conserve energy can be determined. Wherever possible, we have pointed out the pitfalls of blindly adopting the approaches presented in the paper. In the case of the variation of the handshake protocol for SSL, there needs to be a new protocol version that would support mixed RSA/ECC algorithms. In the home WLAN example, we have ignored the way the communicating parties would identify the encryption scheme that is used with a particular packet. We have also ignored how different keys would be used with different encryption schemes, how they are exchanged and so on. If the management and control packets are authenticated, the assumption here is that all MSs share the same secret with an AP and that secret is used in the MD5 hash. Since MD5 has security loopholes, this secret may have to be more securely updated. We have not considered that aspect as well. The way we estimate the energy consumption of different cryptographic primitives can be possibly improved. Other Crypto libraries like Crypto++ and Cryptlib [39] can be tested to see if there are fundamental differences in the amount of energy consumed. Finally, we have not investigated in detail, the impact of the radio channel nor have we designed protocols that can exploit knowledge of good and bad states of the channel to save on the energy consumed.

7. Conclusions and future work

This paper describes a framework for designing energy efficient security protocols. We discuss computation of the energy consumed by cryptographic primitives, the operations that consume more energy, the impact of key sizes, packet sizes and operational rounds on the energy consumed and how they vary with different algorithms. We use these results to suggest three approaches for designing efficient security protocols. We apply two of these approaches in simple examples to demonstrate the potential for saving energy. We also discuss limitations of this work. As part of our ongoing work, we are looking at developing a policy to tie the security requirements of a WLAN with the design of the security protocol. We are also investigating better methods for estimating the energy consumed by cryptographic primitives and understanding how we can exploit knowledge of the radio channel to reduce energy consumption in WLANs.

Acknowledgements

This work was partially supported by the NIST CIP Grant No. 60NANB1D0120 titled "A Survivable and Secure Wireless Information Architecture". We also would like to thank Prof. Daniel Mossè for his advice on the measurement of energy consumption.

References

[1] S. Fluhrer, I. Mantin, A. Shamir, Weaknesses in the Key Scheduling Algorithm of RC4, in Proceeding of the 8th Annual Workshop on Selected Areas in Cryptography (SAC2001), Toronto, Ontario, Canada, August (2001).

[2] N. Borisov, I. Goldberg, D. Wagner, Intercepting Mobile Communications: The Insecurity of 802.11, Rome, Italy, 11, 2001.

[3] W.A. Arbaugh, N. Shankar, J. Wang, Your 802.11 Network has no Clothes, in Proceedings of the First IEEE International Conference on Wireless LANs and Home Networks, Singapore, December, 2001, pp. 131–144.

[4] J. Cox, Serious security weakness in 802.11b wireless LANs exposed, News from Network World Fusion, http://www.nwfusion.com/news/2001/0806ieee.html, August 6, 2001, [last access Feb. 29, 2004].

[5] G.E. Moore, Cramming more components onto integrated circuits, Electronics 8 (38) (1965).

[6] K. Lahiri, A. Raghunathan, S. Dey, D. Panigrahi, Battery-Driven System Design: A New Frontier in Low Power Design, Asia South Pacific Design Automation Conference (ASP-DAC)/International Conference on VLSI Design, January (2002) 261–267.

[7] L.D. Paulson, Will Fuel Cells Replace Batteries in Mobile Devices?, Computer (2003).

[8] N. Ruangchaijatupon, P. Krishnamurthy, Encryption and Power Consumption in Wireless LANs, The Third IEEE Workshop on Wireless LANs, September, Newton, Massachusetts, 2001.

[9] C.E. Jones, K.M. Sivalingam, P. Agrawal, J.C. Chen, A Survey of Energy Efficient Network Protocols for Wireless Networks, Wireless Networks 7 (2001) 343–358.

[10] K. Pahlavan, P. Krishnamurthy, Principles of Wireless Networks-A Unified Approach, Prentice Hall, Englewood Cliffs, NJ, 2002.

[11] M. Zorzi, R. Rao, Energy constrained error control for wireless channels, IEEE Personal Communications 4 (6) (1997).

[12] P. Lettieri, C. Schurgers, M. Srivastava, Adaptive link layer strategies for energy efficient wireless networking, Wireless Networks 5 (5) (1999) 339–355.

[13] N.R. Potlapally, S. Ravi, A. Raghunathan, G. Lakshminarayana, Optimizing Public-Key Encryption for Wireless Clients, in Proceedings of the International Conference on Communications (ICC 2002) New York City, New York, May (2002).

[14] V. Gupta, S. Gupta, S. Chang, D. Stebila, Performance Analysis of Elliptic Curve Cryptography for SSL, The First ACM Workshop on Wireless Security, September 2002, Atlanta, GA, USA, 2002.

[15] K. Lauter, The advantages of elliptic curve cryptography for wireless security, IEEE Wireless Communications February (2004) 62–67.

[16] Y.W. Law, et al., Assessing Security-Critical Energy-Efficient Sensor Networks, IFIP WG 11.2 Small Systems Security Conference, Athens, Greece.

[17] L. Yuan, G. Qu, Design Space Exploration for Energy-Efficient Secure Sensor Network, in Proceeding of the 13th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'02), July 17–19, 2002, San Jose, California.

[18] R. Karri, P. Mishra, Optimizing the Energy Consumed by Secure Wireless Sessions-Wireless Transport Layer Security Case Study, Mobile Networks and Applications 8 (2003) 177–185.

[19] P. Prasithsangaree, P. Krishnamurthy, Analysis of Energy Consumption of RC4 and AES Algorithms in Wireless LANs, GLOBECOM 2003, December 1–5, 2003, San Francisco, CA.

[20] M.A. Viredaz, D.A. Wallach, Power Evaluation of a Handheld Computer: A Case Study, Technical Report (2000/2001).

[21] S. Hirani, Energy Efficiency of Encryption Schemes in Wireless Devices, Telecommunications Program, University of Pittsburgh, Pittsburgh, Pennsylvania, 2003.

[22] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, E. Roback, Report on the Development of the Advanced Encryption Standard (AES), Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, October (2000) 2.

[23] K. Naik, D.S.L. Wei, Software implementation strategies for power-conscious systems, Mobile Networks and Applications 6 (3) (2001) 291–305.

[24] D.W. Carmen, P.S. Kruus, B.J. Matt, Constraints and Approaches for Distributed Sensor Network Security, Technical Report 00-010, NAI Labs, September, 2000.

[25] Intel Corp., The Datasheet of Mobile Intel Pentium III Processor in BGA2 and Micro-PGA2 Packages, Doc. No. 283653-002.

[26] A. Sinha, A. Chandrakasan, JouleTrack—A Web Based Tool For Software Energy Profiling, in Proceedings of the 38th Design Automation Conference, Las Vegas, June, 2001.

[27] B. Schneier, D. Whiting, Fast Software Encryption: Designing Encryption Algorithms for Optimal Software Speed on the Intel Pentium Processor, Lecture Notes in Computer Science 1267 (1997) 242–259.

[28] National Institute for Standards and Technology (NIST), Advanced Encryption Standard (AES), FIPS 197, 2001. http://csrc.nist.gov/encryption/aes/ [last access Feb. 29, 2004].

[29] A.J. Menezes, Elliptic Curve Public Key Cryptosystems, 1st ed., Kluwer, Dordrecht, 1993.

[30] US Department of Commerce/National Institute of Standards and Technology, Digital Signature Standard (DSS), FIPS PUB, 186-2, January 2000.

[31] D. Hankerson, J.L. Hernandez, A. Menezes, Software Implementation of Elliptic Curve Cryptography Over Binary Fields, Cryptographic Hardware and Embedded Systems (CHES), Lecture Notes in Computer Science, Springer, Berlin, 2000, pp. 1–24.

[32] N. Koblitz, CM Curves with Good Cryptographic Properties, Proceedings of Crypto'91, Springer, Berlin, 1992, pp. 279–287.

[33] A.K. Lenstra, E.R. Verheul, Selecting Cryptographic Key sizes, Journal of Cryptology 14 (4) (2001) 255–293.

[34] M. Matsui, Linear Cryptanalysis Method for DES Cipher, in Proceedings of Advances in Cryptology—Eurocrypt'93, Springer, Berlin, 1994, pp. 386–397.

[35] E. Biham, A. Shamir, Differential Cryptanalysis of DES-like Cryptosystems, Journal of Cryptology 4 (1) (1991) 3–72.

[36] J. Daemen, V. Rijmen, AES Proposal: Rijndael, The Standard Proposal, September, 1999.

[37] R.L. Rivest, The RC5 encryption algorithm, in Proceedings of Fast Software Encryption Conference, Leuven, Belgium, Lectures Notes in Computer Science LNCS 1008, 1995, pp. 86–96.

[38] OpenSSL Software Distribution, http://www.openssl.org/ [last access Feb. 29, 2004].

[39] J. Viega, G. McGraw, Building Secure Software: How to Avoid Security Problems the Right Way, Addison-Wesley, Reading, MA, 2001.

[40] B. Schneier, Applied Cryptography: Protocols, Algorithms and Source Code in C, 2nd ed., Addison-Wesley, Reading, MA, 1996.

[41] National Institute for Standards and Technology (NIST), Secure hash standard, FIPS 180-1, April 1995.

[42] R. Rivest, The MD5 message-digest algorithm. April 1992, IETF Request for Comments RFC 1321

[43] H. Dobbertin, The status of MD5 after a recent attack, RSA Labs' CryptoBytes 2 (2) (1996).

[44] T. Dierks, C. Allen, The TLS Protocol Version 1.0, IETF Request for Comments RFC 2246, January 1999.

[45] L.M. Feeney, M. Nilsson, Investigating the energy consumption of a wireless network interface in an ad hoc networking environment, in Proceedings of IEEE INFOCOM, Anchorage, AK, 2001.

[46] B. Clifford Neuman, Ts'o Theodore, Kerberos: An Authentication Service for Computer Networks, IEEE Communications 32 (9) (1994) 33–38.

[47] M. Burrows, M. Abadi, R. Needham, A logic of authentication, Transactions of Computer Systems 8 (1) (1990) 18–36.

[48] M. Abadi, M.R. Tuttle, A semantics for a logic of authentication (extended abstract), in Proceedings of the tenth annual ACM symposium on Principles of distributed computing, Montreal, Quebec, Canada (1991) 201–216.

[49] V.D. Gligor, R. Kailar, S. Stubblebine, L. Gong, Logics for cryptographic protocols-virtues and limitation, in Proceedings of the IEEE Computer Security Foundations Workshop IV, Franconia, New Hampshire, June (1991) 219–226.

[50] R. Anderson, R. Needham, Programming Satan's computer, in: J. van Leeuven (Ed.), Computer Science Today: Recent Trends and Developments, Lecture Notes in Computer Science Series, vol. 1000, Springer, Berlin, 1995.

[51] M. Abadi, Security protocols and their properties, in: F.L. Bauer, R. Steinbrueggen (Eds.), Foundations of Secure Computation, 20th International Summer School, Marktoberdorf, Germany, IOS Press, 2000.

[52] L. Blunk, J. Vollbrecht, B. Aboba, J. Carlson, Extensible Authentication Protocol (EAP), IETF draft-ietf-eap-rfc2284bis-03.c.txt, April, 2003. Work in progress.

[53] IEEE P802 working group, IEEE P802.11 Standard, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999 Edition.

[54] B. Aboba, D. Simon, PPP EAP TLS Authentication Protocol, IETF Requests for Comments: 2716, October 1999.

[55] IEEE P802 working group, IEEE Standard, IEEE 802.1X-2001 Standards for Local and Metropolitan Area Networks: Port-Based Network Access Control, 2001.

[56] Microsoft Corp., Wireless 802.11 Security with Windows XP, white paper, October 15, 2002.

[57] D. Kotz, K. Essien, Analysis of a Campus-wide Wireless Network, in Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking, 2002, pp. 107–118.