The Energy Cost of SSL in Deeply Embedded Systems

Vipul Gupta
Michael Wurm

The Energy Cost of SSL in Deeply Embedded Systems

Vipul Gupta and Michael Wurm

SMLI TR-2008-173 June 2008

Abstract:

As the number of potential applications for tiny, battery-powered, “mote”-like, deeply embedded devices grows, so does the need to simplify and secure interactions with such devices. Embedding a secure web server (capable of HTTP over SSL, *aka* HTTPS), enables these devices to be monitored and controlled securely via a user-friendly, browser-based interface.

This paper presents the first empirical energy analysis of the Internet’s dominant security protocol, SSL, on highly constrained devices. We have enhanced Sizzle, our tiny-footprint HTTPS stack, with energy conserving features and measured its performance on a Telos mote. We show that the key exchange phase, which consumes much more energy than bulk encryption and authentication, amortizes well over the transmission of a few kilobytes of application data. Such amortization is easily attained with features like session reuse and persistent HTTP(S), both of which are supported by Sizzle. The extra energy cost of encrypting and authenticating application data with SSL is around 15%. With the addition of an application-level, duty-cycle based approach to low-power listening for incoming services requests, a pair of alkaline batteries can power Sizzle for over a year under a variety of application scenarios.

Sun Labs
16 Network Circle
Menlo Park, CA 94025

email addresses:
vipul.gupta@sun.com
mwurm@sime.com

For information regarding the SML Technical Report Series, contact Jeanie Treichel, Editor-in-Chief <jeanie.treichel@sun.com>.All technical reports are available online on our website, http://research.sun.com/techrep/.

The Energy Cost of SSL in Deeply Embedded Systems*

Vipul Gupta, Michael Wurm
Sun Microsystems Laboratories
16 Network Circle, UMPK16-160
Menlo Park, CA 94025
http://research.sun.com/projects/crypto
vipul.gupta@sun.com, mwurm@sime.com

June 12, 2008

Abstract

As the number of potential applications for tiny, battery-powered, "mote"-like, deeply embedded devices grows, so does the need to simplify and secure interactions with such devices. Embedding a secure web server (capable of HTTP over SSL, *aka* HTTPS), enables these devices to be monitored and controlled securely via a user-friendly, browser-based interface.

This paper presents the first empirical energy analysis of the Internet's dominant security protocol, SSL, on highly constrained devices. We have enhanced Sizzle, our tiny-footprint HTTPS stack, with energy conserving features and measured its performance on a Telos mote. We show that the key exchange phase, which consumes much more energy than bulk encryption and authentication, amortizes well over the transmission of a few kilobytes of application data. Such amortization is easily attained with features like session reuse and persistent HTTP(S), both of which are supported by Sizzle. The extra energy cost of encrypting and authenticating application data with SSL is around 15%. With the addition of an application-level, duty-cycle based approach to low-power listening for incoming service requests, a pair of alkaline batteries can power Sizzle for over a year under a variety of application scenarios.

1 Introduction

A single development – the introduction of network-enabled cell phones was responsible for doubling, between 2001 and 2003, the number of devices connected to the World Wide Web. As impressive as this trend has been, industry watchers predict an even more dramatic increase in the next few years. Today more than 3 billion devices have Web access, and that number is expected to

*The experiments reported in this paper were performed in 2005. We believe this research has become even more relevant in light of two subsequent developments: (i) a recent industry trend towards adapting standard Internet protocols for wireless sensors and similar resource-constrained devices [1], and (ii) the incorporation of elliptic curve cryptography-based cipher suites in SSL/TLS implementations from major vendors and open source projects.

grow to 14 billion within five years, driven primarily by the proliferation of tiny, battery-powered, wireless sensors and similar deeply embedded devices.

Industrial, agricultural, health care, environmental, security, and military users, among others, are beginning to recognize how wireless sensors can revolutionize their operations. As these devices become available at commodity prices, staggering numbers of them will start turning up everywhere imaginable.

Interaction with these devices is made challenging by the fact that they typically lack a display or input mechanism and their users are often unskilled in managing and administrating computers. Consider the task of remotely configuring or monitoring a radiation sensor or an industrial process control sensor. By integrating a secure web server into such a device, one can perform these tasks simply and easily using a web browser. Besides the advantage of a familiar user interface and platform independence, this approach also offers communication security[1] in the form of SSL, a protocol supported by virtually every browser.

Sizzle [2] is a tiny web server that supports both HTTP and HTTPS. The latter is widely used on the Internet for security sensitive applications like online banking and e-commerce. Sizzle runs efficiently within the tight computing, networking and memory constraints of mote-like devices [3].

These devices are typically powered by batteries – and maximizing battery lifetime is crucial to keeping maintenance costs low. This paper presents the first empirical energy analysis of a secure web server implementation for deeply embedded systems exemplified by the motes. Its main contributions are:

- We describe architectural enhancements to Sizzle aimed at reducing the energy spent in idle listening for incoming service requests.
- We analyze the power consumption of an SSL handshake and bulk data transfer, as well as the underlying cryptographic algorithms, on highly constrained devices
- We investigate the energy overhead of security by comparing HTTPS data transfers (using both RSA and Elliptic Curve Cryptography (ECC) [4] public-key mechanisms) against HTTP transfers and give battery lifetime estimates for different application scenarios.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 presents relevant aspects of Sizzle including a brief overview of the SSL protocol. Section 4 describes the main characteristics of the Telos device[2] and special energy conserving enhancements we made to Sizzle in response to those traits. The results of our energy measurements and computations are presented in Section 5. Finally, Section 6 summarizes our conclusions.

2 Related Work

Gura *et al*. [6] compared ECC and RSA operations on 8-bit processors and showed that public-key cryptography is feasible on mote-like devices. ECC is a resource efficient public-key cryptosystem

[1] An important requirement for many applications.

[2] The newest member in the family of sensor platforms, called "motes", developed at UC Berkeley and running TinyOS [5].

that offers a comparable level of security as RSA while using smaller keys and faster computations. This computational advantage over RSA increases with the decrease in processor word size, making ECC the technology of choice for resource constrained devices.

In [2] we showed that it is possible to implement a complete secure web server stack that runs efficiently within the tight memory and CPU constraints of the Mica2 and Mica2dot motes. We also presented ideas for reducing data transmission across the wireless link using a template-based scheme for compressing SSL handshake messages and a customizable version of the Mozilla browser capable of sending short HTTP headers. Our performance analysis in that paper focused on memory and CPU usage but did not consider energy consumption. In [7], we presented an energy analysis of public-key cryptography for the older generation Mica2 motes but the analysis was based on an abstract protocol and did not involve empirical measurements.

Potlapally *et al*. [8] analyzed the energy cost of SSL for PDAs and Hadjat *et al*. analyzed the energy cost of Elliptic Curve Diffie-Hellman (ECDH) key exchange and AES encryption on WINS nodes[3] [9]. However, those results do not directly translate to the mote family of devices, which have much tighter resource limitations.

Radio communication is a significant contributor to the total energy consumption in a wireless sensor system. The energy efficiency of the IEEE 802.15.4 low-power, wireless networking standard has been examined by Bougard *et al*. [10] and a number of alternative low-power MAC protocols have been proposed [11, 12, 13]. A common goal is to minimize active listening – the time a radio spends in receive mode to detect network traffic – by employing duty-cycle based schemes. This is done by synchronizing the communicating parties to use only certain time slots, or with the help of special hardware functionality which allows a radio to detect traffic while staying in a low power mode most of the time.

Unfortunately, the most popular implementation of IEEE 802.15.4, the Chipcon CC2420, does not offer hardware support for low power listening. Our alternative approach, described in Section 4.2, conserves energy at the application level. This results in a scheme that is simple, portable across different radio platforms, and customizable for specific applications.

3 SSL and Sizzle Overview

The SSL protocol [14] offers encryption, authentication and integrity protection on top of a streaming data-transport mechanism like TCP. It allows communicating parties to choose a set of cryptographic algorithms, called a *cipher suite*, which determines the method of key exchange, signature verification, encryption and keyed hashing (to detect data tampering). Cipher suites that use ECC-based key exchange have been standardized for SSL [15, 16]. ECC support is now available in SSL/TLS implementations from major vendors (notably Firefox, Internet Explorer) and can be enabled in the Apache web server [17, 18].

The two main components of the SSL protocol are the Handshake protocol and the Record Layer protocol. In the handshake, both parties agree on a cipher suite, authenticate each other and establish a *shared secret* using public-key cryptography (see Figure 1). The record layer uses symmetric keys, derived from the shared secret, to perform bulk encryption and authentication of application data. Since public-key cryptography is expensive, SSL supports abbreviated hand-

[3] StrongARM based wireless sensors developed at Rockwell Scientific.

shakes, where the parties reuse a shared secret established in a previous full handshake.

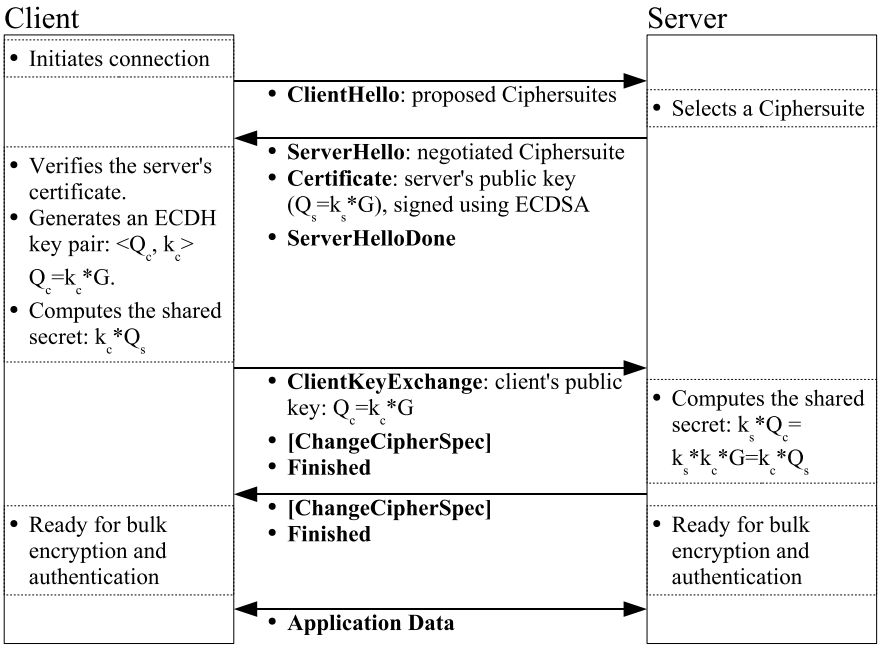

Figure 1: Minimal ECDH-based SSL Handshake. The most computationally expensive part is the EC point multiplication, which occurs when the server processes the ClientKeyExchange message.

Sizzle brings the well established security properties of SSL to highly constrained devices. It implements a number of optimizations for reducing resource usage while maintaining compatibility with standard SSL. It supports 1024-bit RSA and 160-bit ECC (which provides equivalent security) as well as SHA1, MD5 and RC4. These cryptographic algorithms are written in C except for the most time critical portions (*e.g.*, big integer operations) which are written in assembly language.

Figure 2 shows the architecture we presented in [2]. It uses a star topology[4] in which the central device is connected to a PC and acts as a gateway, forming a bridge between the TCP/IP network on one side and a network of 802.15.4 devices on the other. Each wireless device is mapped to a different TCP-port on the gateway and users can communicate with the Sizzle stack running on the device via a standard web browser.

A special reliable data transfer protocol is used between the gateway and wireless devices. This protocol runs directly on top of the TinyOS radio stack and ensures lossless, in-order delivery of messages whose size is limited only by available buffer space.[5] A long message is split into multiple, fixed-size TinyOS packets and the receiver explicitly acknowledges the first and the last packets. Remaining packets are transmitted using a NACK scheme to avoid the overhead of acknowledging every single packet.

In order to minimize the overhead of sending multiple short messages, the gateway buffers TCP segments until a complete SSL record (or plain HTTP request) has been assembled before forwarding it. The wireless device processes the data received and either sends additional data

[4]Support for a multi-hop mesh topology can be added without significant modifications.

[5]On the Telos, more than 5KB of RAM is available for buffers.

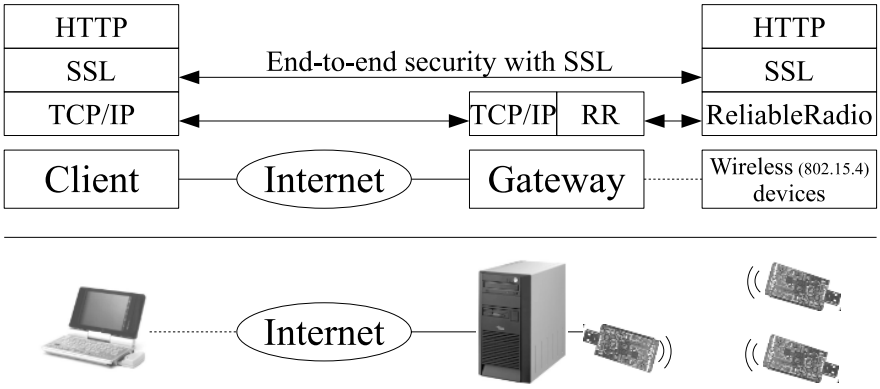

Figure 2: Gateway-based architecture to make wireless devices accessible from the Internet. Even though the TCP/IP connection ends at the gateway, the security provided by SSL extends end-to-end.

in response (*e.g.*, when responding to the ClientHello with a combined ServerHello, Certificate and ServerHelloDone message or when responding to an HTTP request with an HTTP response) or transmits a "ready" control message to indicate its readiness to process the next chunk of data (*e.g.*, when waiting for the ChangeCipherSpec message after processing the ClientKeyExchange). In the former situation, data transmission from the wireless device is an implicit signal to the gateway that it is ready to receive the next chunk of information.

4 Experimental Platform

4.1 Telos Motes

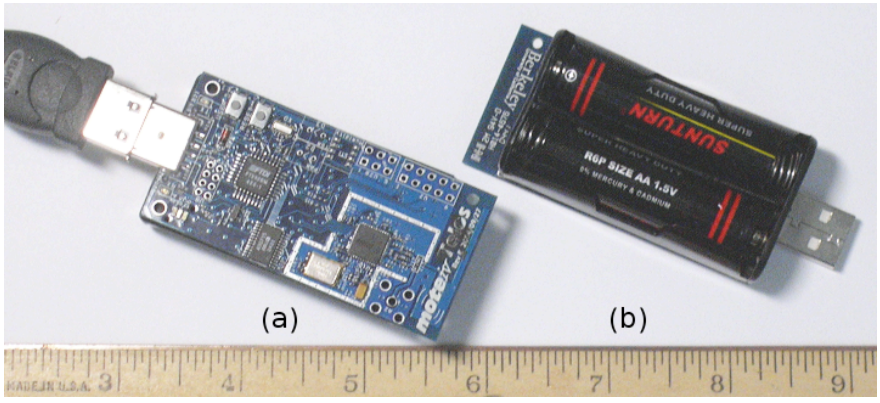

Figure 3: The Telos (Rev B) "mote" has a 16-bit CPU and an IEEE 802.15.4 radio. (a) A Telos device connected to a PC via USB can be used as a base station for relaying data packets between the USB and radio links. (b) It is powered by two AA batteries and application-specific sensors can be attached to digital and analog input pins or to an I^2C bus interface.

The Telos motes [19, 20], depicted in Figure 3, are the successors to the Mica family of motes

(Mica2dot, Mica2 and MicaZ) and offer greater computing power and higher radio bandwidth (compared to the Mica2dot and Mica2). They are equipped with the MSP430 16-bit CPU from Texas Instruments, rated for 8 MHz, a Chipcon CC2420 IEEE 802.15.4 compatible radio, and an onboard USB connector to download applications.

The MSP430 microcontroller integrates 48KB of FLASH, 10KB of RAM, and several peripherals including a multiply-accumulate unit. The multiply-accumulate unit performs computations of the form ACC = ACC + (a · b), which can be utilized to speed up the multiplication of long integers – a core operation in both RSA and ECC computations. The MSP430 also offers a rotate instruction which speeds up SHA1 and MD5 by almost 40%. The microcontroller uses very little power in active and standby modes and supports fast transition between the two modes. Table 1 shows the measured current draw of the overall system in different operating modes.

Vcc	Idle	CPU	Radio-RX	Radio-TX	CPU+Radio-RX
2.2	0.004	2.1	20.5	17.2	22.6
2.6	0.006	3.0	20.6	17.2	23.6
3.0	0.008	3.9	20.6	17.3	24.5

Table 1: Measured current draw (in mA) for common operating modes of the Telos mote at different supply voltages with the CPU operating at 8MHz. In the overall system, the radio consumes considerably more energy than the CPU, and the receive mode draws more current that the transmit mode. Since the radio must be in receive mode to listen for packets, special care must be taken to minimize idle listening in energy constrained applications.

The TinyOS operating system has been ported to the Telos and offers an abstraction layer for accessing attached sensors and the radio. The TinyOS radio stack implements the *B-MAC* protocol [11] which relies on special hardware support, as found on the older CC1000 radio in the Mica2dot and Mica2 motes, to facilitate low-power listening. The lack of this feature in the CC2420 results in a constant current draw of 20mA which limits the battery lifetime to about four days. However, the radio interface exposes methods to enter a low-power mode for application-driven power management.

4.2 Energy-conservation in Sizzle

Given that the radio consumes energy at a much higher rate than the microcontroller, we've enhanced Sizzle to turn off the radio whenever possible. Normal web servers are always on and ready to receive incoming TCP connection requests. If such a web server were to use a duty cycle based approach – sleeping most of the time and waking up only periodically – TCP packets delivered to the web server when it's asleep would not be acknowledged. The TCP stack on the client would incorrectly interpret packet loss as a sign of network congestion and unnecessarily throttle back its transmission rate.

Our Sizzle architecture (see Figure 2) terminates the TCP connection at a mains powered (hence, always on) gateway. As described in the last part of Section 3, the gateway only sends data to a wireless device if it has recently received some data or the "ready" control message.

This makes it easy to have the device go to sleep without worrying about data loss. For example, the device can turn off its radio while processing the ClientKeyExchange SSL record (this can take a few seconds if an RSA decryption is involved) knowing that the gateway won't transmit the ChangeCipherSpec message until the device turns on its radio again and transmits a "ready" message.

We have introduced a new control message called the "ready-sleep" in the special wireless protocol used across the radio link in Figure 2. This message is sent by the wireless device to inform the gateway that the device has turned on its radio for a short period, but if the gateway does not initiate a new service request (for an SSL handshake or HTTP(S) data transfer) during this period, the device will turn off the radio and go to sleep. The device sleeps most of the time but wakes up periodically and sends the "ready-sleep" message to poll the gateway for pending service requests. If the gateway does not respond within a short time period, the device powers down again as shown in Figure 4.

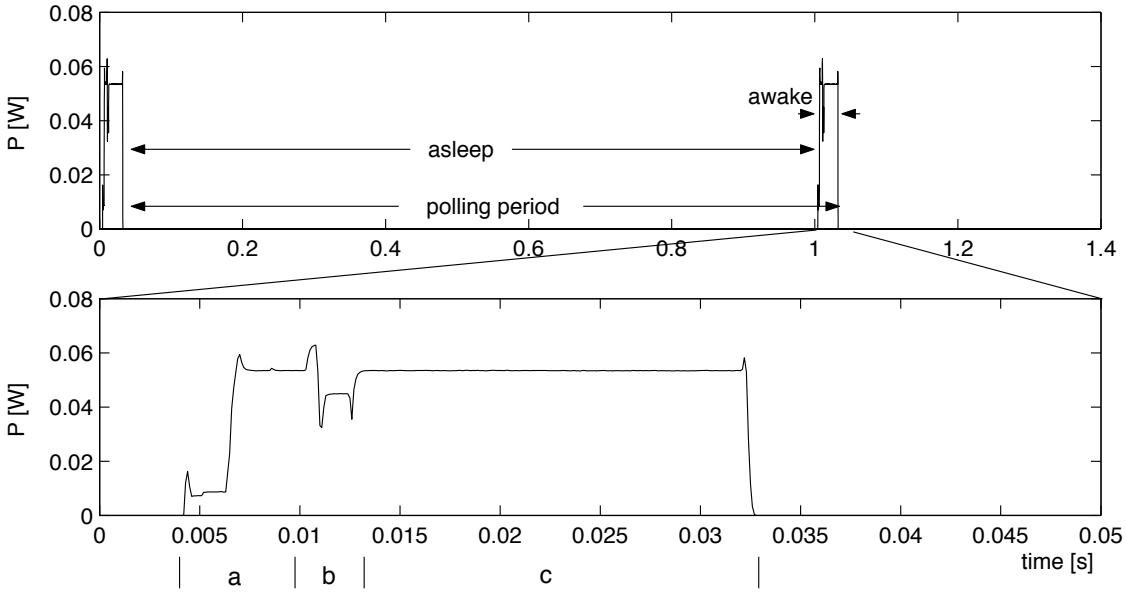

Figure 4: Measured power consumption while the mote polls the gateway for pending service requests. (a) The CPU wakes up and activates the radio. (b) The mote transmits the "ready-sleep" message and leaves its radio turned on for a short period. (c) If the gateway does not respond during the short period, the mote powers down again. Each polling attempt consumes roughly 1.5mJ of energy.

Our experiments indicate that staying awake and listening for 20ms gives the gateway enough time to respond. By adjusting how long the mote sleeps in between consecutive polling attempts, one can trade-off overall power consumption for response latency.

The fact that each device periodically transmits "ready-sleep" messages can be used by the gateway to discover the presence of active devices in its neighborhood. We have added a web server at the gateway and it creates a dynamic web page listing currently active devices. Device entries in the list include discovered information such as the device name, manufacturer, serial number, *etc*. Most importantly, each entry also includes a URL that acts as an entry point for interacting with the Sizzle server embedded inside that device. This new functionality greatly

simplifies the overall user experience of integrating a newly purchased device. After turning on a device, the user directs his web browser to the fixed URL of the "device listing" page. Shortly after the device is discovered, it automatically appears on this page with a link to its embedded web server. Clicking the link brings up the starting web page for the device with overview information and additional links for monitoring and/or configuring it.[6] The "device listing" web page uses the refresh meta tag to force periodic reloading by the browser. This eliminates the need for any explicit user action in order to discover a newly introduced device.

4.3 Battery Characteristics

We use a pair of alkaline batteries [21] as the basis of our energy analysis due to their low price and easy availability. The total energy capacity and the shape of the discharge curve are dependent on the current draw. In low power applications, the discharge curve tends to be more linear and the batteries deliver their full rated capacity.

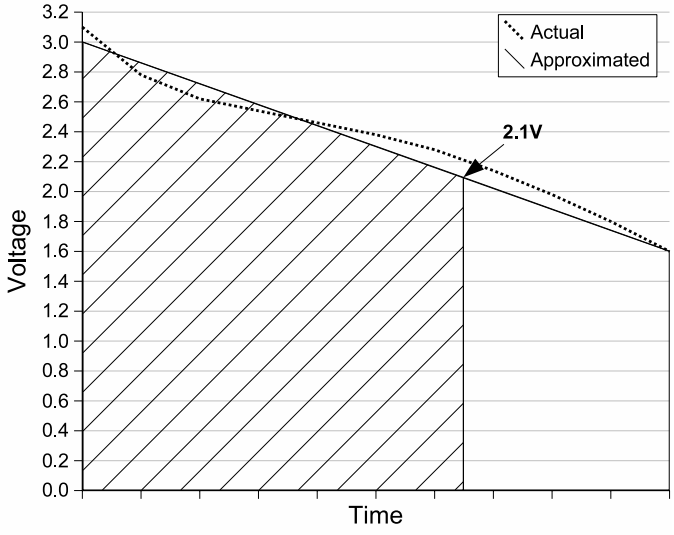

Figure 5: Battery voltage over time, assuming constant current draw. Because the cut-off voltage of the Telos radio is 2.1V, only the energy represented by the shaded area can be used.

The typical capacity of a pair of alkaline batteries is 2.6Ah, or 9,360As. Assuming a linear voltage drop from 3V to 1.6V (a mean voltage of 2.3V), the total energy is E_{total} = U · I · T = 2.3V · (9,360As / T) · T = 21,500J.

The cut-off voltage of the Telos mote is determined by the CC2420 radio chip which requires 2.1V to operate. As a result, only 71% of the total energy can be utilized (see Figure 5), *i.e.*, E_{avail} = 71% · E_{total} = 15,300J

According to the manufacturer's claim, 85% of the initial capacity remains after 4 years. We assume a constant self discharge of 4% per year – $P_{self-discharge}$ = 4% · 21,500J / ($t_{sec-per-year}$) = 27.8uW.

[6]Sizzle supports user authentication using passwords over SSL and this can be used to implement device-specific access control policies.

All energy figures and computations reported in Section 5 are based on a supply voltage of 2.6V which is close to the mean voltage in the batteries' life cycle, as they discharge from 3V to 2.1V.

5 Empirical Results

In this section, we first investigate the energy costs of the cryptographic algorithms used in our SSL implementation. We then analyze the cost of handshakes and bulk data transfers before presenting battery lifetime estimates for different use cases.

We utilized one of the hardware timers built into the microcontroller to measure the execution time of various cryptographic operations and the CPU's static power consumption to arrive at the energy costs shown in Table 2. As expected, ECC uses far less energy than RSA. However, both kinds of public-key operations consume orders of magnitude more energy than RC4, MD5 or SHA1. Because of its simplicity, the per-byte energy cost of RC4 is extremely small. The RC4 key setup cost is only incurred once per SSL handshake and persistent HTTPS allows multiple data transfers to reuse the results of one handshake.

Asymmetric	Key Exchange	Ratio
RSA	46,690.00	12
ECC	3,780.00	1
Symmetric	**Key Setup**	**Per Byte**
RC4	10.40	0.03
Hash	**One Time**	**Per Byte**
MD5	1.56	0.24
SHA1	1.05	0.26

Table 2: Energy costs (in uJ) of different cryptographic algorithms. The cost of an ECC key exchange is equal to the cost of encrypting 110KB of data using RC4 and to the cost of hashing roughly 15KB of data using MD5 or SHA1.

Figure 6 combines three different plots showing (from top to bottom) the radio activity in an SSL handshake with ECDH key exchange, the corresponding power consumption at each instant, and accumulated energy consumption since the start of the handshake. These plots were obtained by using a stabilized voltage source and measuring the voltage drop on a sense resistor with a digital oscilloscope. We corrected the data to eliminate errors introduced by the resistor and verified the accuracy of our results by comparing them against static measurements presented above. Similar measurements were also made for an RSA handshake and for an abbreviated handshake. The RSA plot is similar to the ECC plot except the public-key operation (indicated by region b in the bottom plot in Figure 6) takes 12 times longer and the longer Certificate and ClientKeyExchange messages result in slightly more data transfer. Based on these measurements, if just 10% of the available battery energy were devoted to SSL handshakes, one could perform nearly 20,000 RSA handshakes or 50,000 ECC handshakes or 90,000 abbreviated handshakes. Assuming a desired

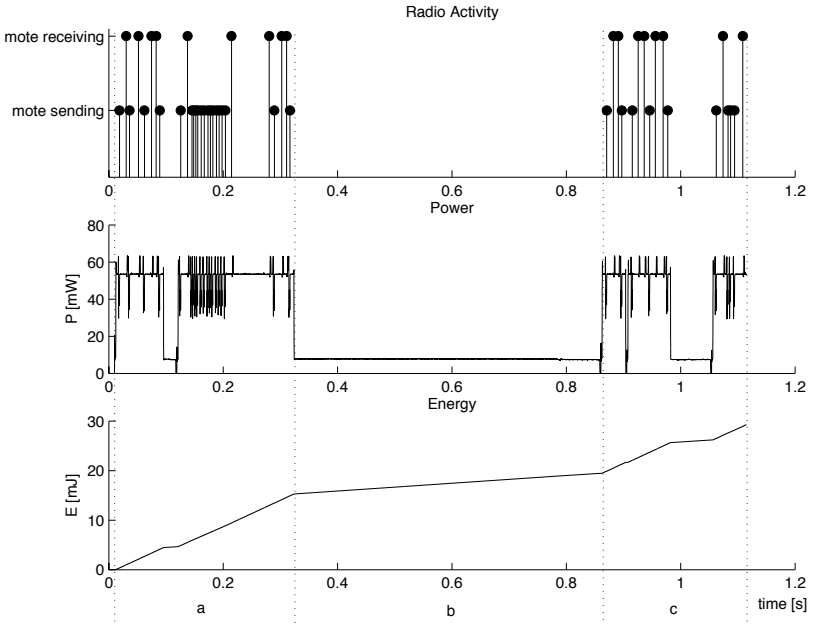

Figure 6: Power consumption during an ECC handshake. (a) Transmission of Hello messages, Certificate and ClientKeyExchange. (b) Public-key operation. (c) ChangeCipherSpec and Finished messages. The public-key operation uses a relatively small fraction of the energy spent in the handshake, because of the low power consumption of the CPU compared to the radio.

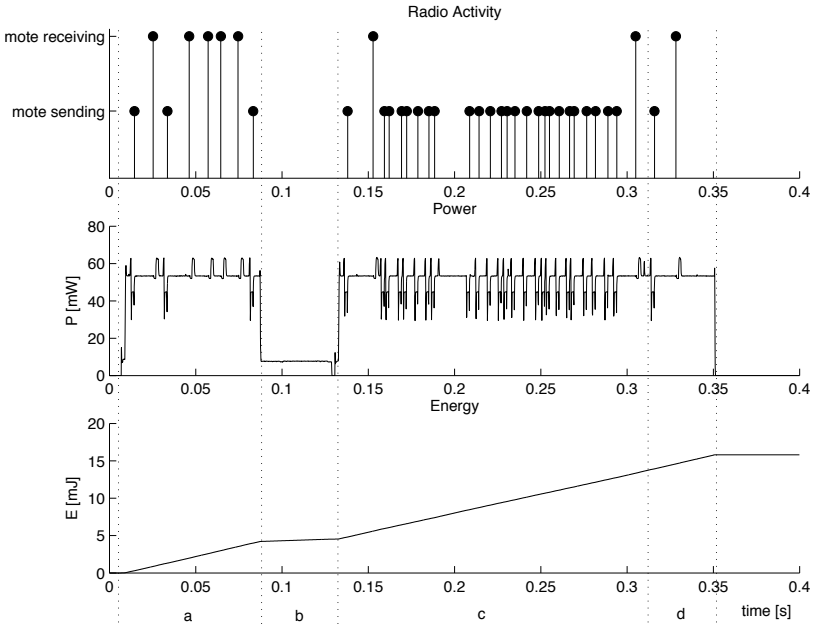

Figure 7: Application data transfer with persistent HTTPS. (a) Reception of the HTTP request (100 bytes). (b) Processing of the request and generation of the response (including symmetric cryptography and message authentication). (c) Transmission of the response (500 bytes). (d) Transmission of "ready-sleep" before the mote powers down.

battery lifetime of 500 days, this translates to 40 RSA handshakes or 100 ECC handshakes or 180 abbreviated handshakes per day.

We used thresholds in the power plots to determine the times where the CPU, the radio or both were active to estimate the percentage of energy consumed by the radio. Those results are shown in Table 3. While the energy cost of communication is much higher than the energy cost of computation for an ECC handshake, the situation is reversed for RSA. The abbreviated handshake does not involve expensive public-key operations, and almost all the energy is spent in communication.

	Absolute Cost		Relative to RSA		
	Time [ms]	Energy [mJ]	Time	Energy	Energy in Communication
RSA Handshake	6,410	76.9	100.0%	100.0%	41%
ECDH Handshake	1,110	29.7	17.3%	38.6%	82%
Abbreviated Handshake	422	16.5	6.6%	21.5%	93%

Table 3: Costs of three different types of SSL handshakes in terms of time and energy. Compared to an RSA handshake, an ECC handshake results in energy savings of more than 60% and time savings of more than 80%. The abbreviated handshake reduces costs even further.

Persistent HTTPS reuses an already open TCP connection (for which an SSL handshake has already occurred) for multiple subsequent HTTP data transfers. The communication overhead of application data transfer can be further reduced using a modified version of the Mozilla web browser, which can be configured to issue short HTTP requests [22], containing only essential header elements. This reduces the length of a typical HTTP request from about 400 bytes to less than 100 bytes. Figure 7 shows radio activity, instantaneous power consumption, and accumulated energy costs in such a transaction. It is also worth noting that AJAX-based web applications naturally exchange much smaller amounts of data compared to the size of a typical web page.

Transaction Size [bytes]		Time [ms]	Energy [mJ]	Data Rate [bits/s]	Energy/Byte [mJ]	Energy in Communication
Insecure	600	222	11.2	21,622	.019	98%
Insecure	1000	313	15.9	25,559	.016	98%
Secure	600	293	12.9	16,382	.022	96%
Secure	1000	400	18.0	20,000	.018	96%

Table 4: Comparison of HTTP and HTTPS data transfers in terms of time and energy. While the additional overhead of security is 30% in terms of time, the energy overhead is only 15%. These results do not consider the cost of the initial handshake.

We also compared the transmission times and energy costs of secure (HTTPS) and insecure (HTTP) communication for different page sizes. Those results are shown in Table 4. Even in the case of secure communication, the computational cost is only a small fraction of the total cost. The

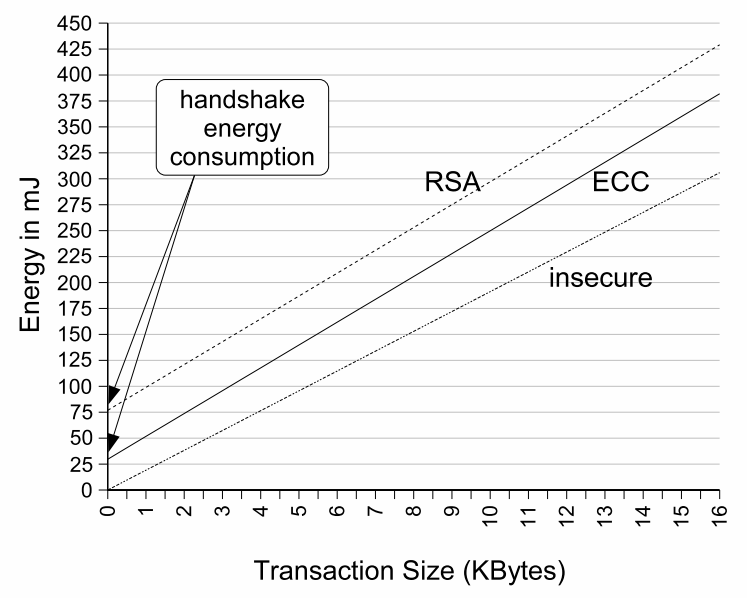

Figure 8: Energy consumed as a function of application data size. The points where the plots intersect the Y axis denote the energy spent in an SSL handshake (76.9mJ for RSA, 29.7mJ for ECC), while the slopes indicate the energy needed to transfer unit data. The slopes for ECC and RSA plots are identical and only 15% steeper than the slope for insecure communication. These plots assume SSL record size does not exceed 1KB; the slope will be lower if larger SSL records are utilized.

greater energy consumption is mainly due to the overhead of transmitting additional information: SSL record headers and message authentication codes. As indicated in Table 4, this overhead decreases when larger SSL records are used to transmit data. Note that at roughly 20uJ per byte, one can transmit over half a gigabyte of data over SSL while using approximately 65% of the available battery capacity. This translates to over 1MB of data per day for 500 days.

Figure 8 plots energy consumed for transferring different amounts of application data (potentially using multiple HTTPS requests and responses) following a single ECC or RSA handshake. A plot for plain HTTP is also included to indicate the additional energy overhead of security. While the energy cost of an SSL handshake is relatively high, it soon becomes marginal when persistent HTTPS is used to amortize that cost across multiple data transfers (see Figure 9).

As mentioned in Section 4.2, one can adjust the rate at which Sizzle polls the gateway for pending service requests to lower average energy consumption at the expense of increasing response latency and vice versa. Since each poll consumes roughly 1.5mJ of energy, 10% of the available battery capacity can power nearly one million polls. For a desired battery lifetime of 500 days, this translates to 2,000 polls per day or one poll every 45 seconds. The impact of different polling rates and application communication characteristics on battery lifetime is captured in Figure 10.

To isolate the impact of security from the impact of idle listening, we consider an additional scenario in which the device initiates data transfers on its own instead of polling for incoming requests. The device accumulates sensor readings in a buffer and transmits them whenever that buffer is full. A realistic size for this buffer is 4KB, given that the Telos has a total of 10KB RAM of which the Sizzle stack already uses about 4KB at run time. Figure 11 illustrates the impact of

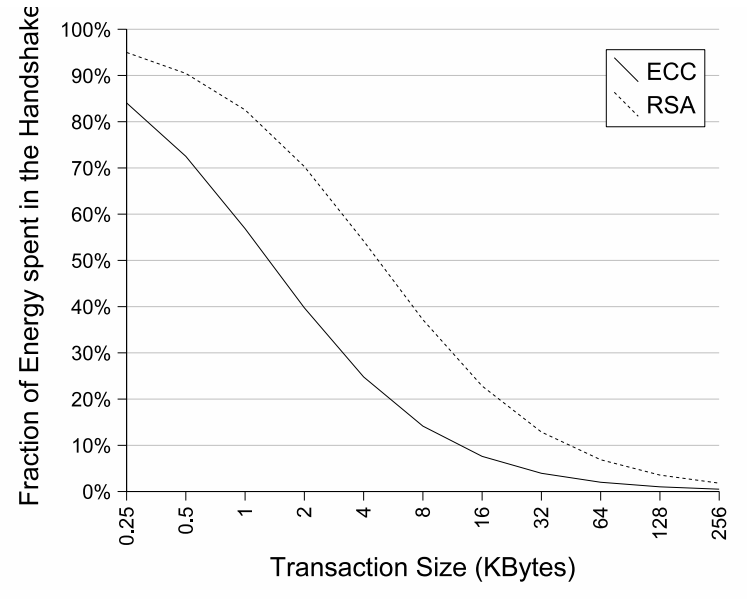

Figure 9: Fraction of the energy spent in an SSL handshake with increasing transaction size. The point where the cost of data transfer exceeds the handshake cost is at about 1.3KB for ECC and at 4.3KB for RSA.

full handshake frequency on battery lifetime. It highlights the effectiveness of Sizzle features like session reuse and persistent HTTPS (which minimize the need for full handshakes) in increasing battery lifetime.

6 Conclusion

We presented the design of a secure web server architecture for deeply-embedded devices with several energy conserving features. We implemented these features in the Sizzle HTTPS stack and performed energy measurements on the Telos mote. This platform uses an ultra-low-power microcontroller, the MSP430, so the energy cost of communication plays a very significant role.

Our measurements show that using ECC, rather than RSA, for key establishment in SSL results in energy savings of nearly 60% and time savings of 80%. The energy cost of bulk data transfer over SSL is quite small (around 20uJ per byte) and represents about a 15% overhead compared to insecure data transfer. Overall, support for features like abbreviated handshakes and persistent HTTPS, ensures that the introduction of SSL does not increase energy costs significantly.

In realistic scenarios, a duty-cycle based approach to polling for incoming connections has the greatest impact on energy conservation. Battery lifetimes of several months can be expected while keeping service latency within user-acceptable limits. For example, consider allocating just 5% of the available energy capacity from a pair of alkaline batteries to SSL handshakes, 10% to polling, 25% to SSL bulk data transfer and the remaining 60% to powering the device and attached sensors. This allocation can power 1 million polling attempts, 9,000 ECC handshakes, 28,000 abbreviated handshakes and 270MB of data transfer. For an estimated battery lifetime of 1 year, this translates to 1 poll every 30 seconds, 1 full handshake and 3 abbreviated handshakes every hour and half a

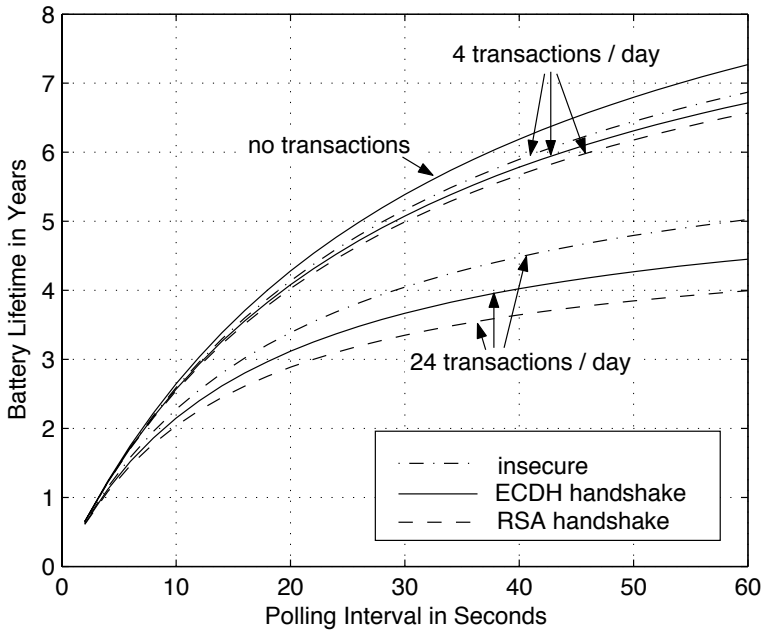

Figure 10: Battery lifetime estimation for the web server, as a function of the polling interval. Curves are shown for different numbers of average transactions per day using insecure and secure communication, with RSA and ECDH handshake. We assume that one transaction consists of a handshake and the transfer of 10 HTML pages, with 500 bytes each. When keeping the polling interval in a user-acceptable range, the energy consumed by communication and security has almost no impact.

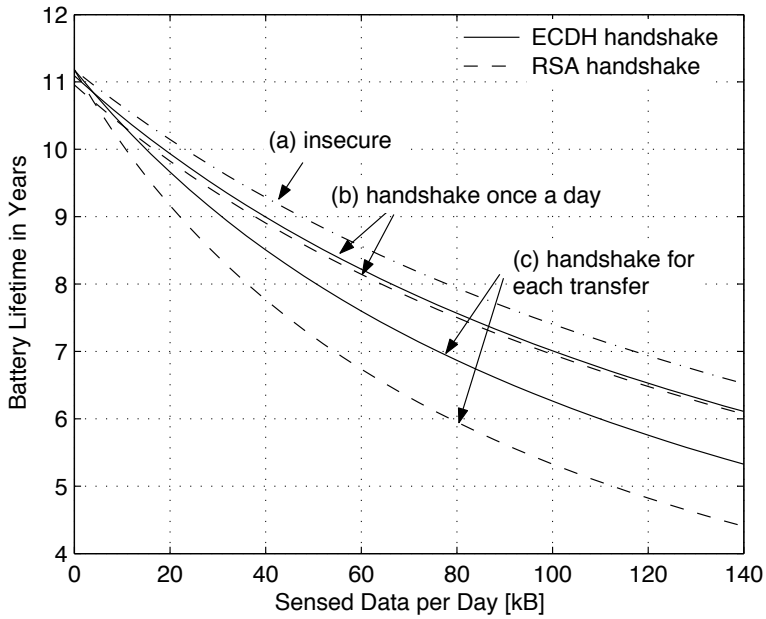

Figure 11: Battery lifetime estimation when the device initiates a transfer whenever its buffer is filled with sensor readings. In case (a), the transactions are not secured, in case (b), a full SSL handshake is performed once a day, and in case (c), a full handshake is performed for each transfer.

kilobyte of application data every minute.

7 Acknowledgments

The authors wish to thank Yu Zhu, Matthew Millard, Stephen Fung, Nils Gura, Hans Eberle, Sheueling Chang-Shantz, Arun Patel and Arvinderpal Wander for their help in several aspects of this project.

References

[1] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," IETF RFC 4919, August 2007.

[2] Vipul Gupta, Matthew Millard, Stephen Fung, Yu Zhu, Nils Gura, Hans Eberle, and Sheueling Chang-Shantz, "Sizzle: A Standards-based End-to-End Security Architecture for the Embedded Internet," in *Third IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2005, Kauai, Hawaii.

[3] Joe Polastre, Robert Szewczyk, Cory Sharp, and David Culler, "The Mote Revolution: Low Power Wireless Sensor Network Devices," 2004, Hot Chips.

[4] Scott A. Vanstone, "Next Generation Security for Wireless: Elliptic Curve Cryptography," *Computers and Security*, vol. 22, no. 5, August 2003.

[5] TinyOS Community Forum, "An open-source OS for the networked sensor regime," http://www.tinyos.net/.

[6] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang-Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," in *CHES '2004 Workshop on Cryptographic Hardware and Embedded Systems*. August 2004, Lecture Notes in Computer Science, Springer-Verlag, Cambridge, MA.

[7] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. Chang Shantz, "Energy Analysis of Public-Key Cryptography for Wireless Sensor Network," in *Third IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2005, Kauai, Hawaii.

[8] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha, "Analyzing the Energy Consumption of Security Protocols," August 2003, IEEE/ACM Intl Symposium on Low Power Electronics and Design (ISLPED).

[9] A. Hodjat and I. Verbauwhede, "The energy cost of secrets in ad-hoc networks," September 2002, IEEE CAS Wkshp. Wireless Communication and Networking.

[10] B. Bougard, F. Catthoor, D. C. Daly, A. Chandrakasan, and W. Dehaene, "Energy Efficiency of the IEEE 802.15.4 Standard in Dense Wireless Microsensor Networks: Modeling and Improvement Perspectives," 2005, Design, Automation and Test in Europe, pp. 196–201.

[11] Joe Polastre, Jason Hill, and David Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," November 2004, SenSys'04.

[12] Amre El-Hoiydi and J.-D. Decotignie, "WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks," in *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004), Lecture Notes in Computer Science, LNCS 3121*. July 2004, pp. 18–31, Springer-Verlag.

[13] Tijs van Dam and Koen Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proc. 1st Intl. Conf. on Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, November 2003, ACM, pp. 171–180.

[14] Alan O. Freier, Philip Karlton, and Paul C. Kocher, "The SSL Protocol — Version 3.0," IETF Internet Draft, November 1996.

[15] Tim Dierks and C. Allen, "The TLS Protocol — Version 1.0," January 1999, IETF RFC 2246.

[16] Simon Blake-Wilson, Nelson Bolyard, Vipul Gupta, Chris Hawk, and Bodo Möller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)," RFC 4492, May 2006.

[17] Vipul Gupta, Douglas Stebila, and Sheueling Chang-Shantz, "Integrating Elliptic Curve Cryptography into the Web's Security Infrastructure," in *The 13th International World Wide Web Conference – Alternate Track Papers and Posters*, May 2004, New York City.

[18] V. Gupta, "Expose ECC cipher suites (IETF RFC 4492) in OpenSSL to Apache," https://issues.apache.org/bugzilla/show_bug.cgi?id=40132.

[19] Moteiv Corporation, "Telos product information," http://www.moteiv.com/products/.

[20] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," April 2005, IPSN/SPOTS.

[21] The Gillette Company, "Alkaline Technical Bulletin," http://www.duracell.com/oem/Pdf/others/ATB-5.pdf.

[22] V. Gupta and M. Wurm, "Configurable option to reduce the size of HTTP requests," https://bugzilla.mozilla.org/show_bug.cgi?id=300811.

About the Authors

Vipul Gupta is a Distinguished Engineer at Sun Microsystems Laboratories where his research interests include wireless sensor networks, resource efficient cryptography, and mobile computing. He has authored over thirty conference/journal publications in these and related areas and is an active contributor to various standards organizations and open source projects. At Sun, he has been a co-recipient of two Sun Labs Technology Transfer Awards, the 2004 Chairman's Award for Innovation and an honoree of the Computerworld Horizon Award 2006. His development of the world's smallest secure web server, Sizzle (about the size of a quarter-dollar coin) received the Mark Weiser Best Paper Award at the IEEE Pervasive Computing and Communications Conference in 2005. He received his Ph.D. in Computer Science from Rutgers University and a BTech in Computer Science and Engineering from the Indian Institute of Technology, New Delhi.

Michael Wurm graduated from the Graz University of Technology in Austria with a degree in Computer Science and Telecommunications. After working on security implementations for the embedded Internet at Sun Microsystems Laboratories (during an internship from January through August 2005) and traveling all over Europe for a year, he is now back in the Bay Area working at a security startup.