USENIX Association

Proceedings of the 9th USENIX Security Symposium

Denver, Colorado, USA
August 14–17, 2000

PGP in Constrained Wireless Devices

Michael Brown* Donny Cheung* Darrel Hankerson† Julio Lopez Hernandez‡
Michael Kirkup* Alfred Menezes*

Abstract

The market for Personal Digital Assistants (PDAs) is growing at a rapid pace. An increasing number of products, such as the PalmPilot, are adding wireless communications capabilities. PDA users are now able to send and receive email just as they would from their networked desktop machines. Because of the inherent insecurity of wireless environments, a system is needed for secure email communications. The requirements for the security system will likely be influenced by the constraints of the PDA, including limited memory, limited processing power, limited bandwidth, and a limited user interface.

This paper describes our experience with porting PGP to the Research in Motion (RIM) two-way pager, and incorporating elliptic curve cryptography into PGP's suite of public-key ciphers. Our main conclusion is that PGP is a viable solution for providing secure and interoperable email communications between constrained wireless devices and desktop machines.

1 Introduction

It is expected that there will be more than 530 million wireless subscribers by the year 2001, and over a billion by 2004 (see [46]). Efforts are underway, most notable among them the Wireless Application Protocol (WAP) [50], to define and standardize the emerging wireless Internet. Users will access wireless services including telephony, email and web browsing, using a variety of wireless devices such as mobile phones, PDAs (such as the PalmPilot), pagers, and laptop computers equipped with wireless modems. Many wireless devices are constrained by limited CPU, memory, battery life, and user interface (e.g., small screen size, or a lack of graphics capabilities). Wireless networks are constrained by low bandwidth, high latency, and unpredictable availability and stability. The purpose of this paper is to examine the viability of using PGP for providing secure and interoperable email communications between constrained wireless devices and desktop machines.

There are two popular standards for email security: S/MIME and PGP. S/MIME [40] provides confidentiality and authentication services to the MIME (Multipurpose Internet Mail Extensions) Internet email format standard. PGP (Pretty Good Privacy) [8, 16] is an email security standard that has been widely used since it was first introduced by Zimmermann in 1991 [52]. While it appears that S/MIME will emerge as the industry standard for commercial and organizational use, it also appears that PGP will remain the choice for personal email security for many users in the years to come.

The specific goals of this project were three-fold:

1. Port the basic PGP functionality to the RIM pager, and implement a workable key management system and a usable user interface that is appropriate for the RIM pager environment.
2. Achieve interoperability with existing PGP implementations for workstation and PalmPilot platforms.
3. Incorporate standards-based and commercial-strength elliptic curve cryptography into PGP's suite of public-key algorithms.

The remainder of this paper is organized as follows. §2 provides a brief history of PGP, and summarizes the security services offered by PGP. A description of the RIM two-way pager including hardware, software, user interface, development tools, and the paging environment, is provided in §3. A brief overview of the PalmPilot is presented in §4. Elliptic curve cryptography is introduced in §5, along with a description of our implementation. We provide timing comparisons of our ECC implementation with RSA and DL implementations on a variety of platforms. Our experience with porting PGP to the RIM pager is described in §6. Our implementation, including a description of the user interface and key management facilities, is presented in §7. In §8, we describe some possible directions for future work. Finally, §9 makes concluding remarks.

*Dept. of Combinatorics and Optimization, University of Waterloo, Canada. Emails: {mk3brown, dccheung, mkirkup, ajmeneze}@uwaterloo.ca

†Dept. of Discrete and Statistical Sciences, Auburn University, USA. Email: hankedr@mail.auburn.edu. Supported by a grant from Auburn University COSAM.

‡Institute of Computing, State University of Campinas, Brazil, and Dept. of Computer Science, University of Valle, Colombia. Email: julioher@dcc.unicamp.br

2 Pretty Good Privacy

2.1 History of PGP

The history of the Pretty Good Privacy (PGP) application is both interesting and convoluted, and encompasses issues in national security, personal privacy, patents, personalities, and politics; see, for example, [16]. A myriad of PGP releases emerged, in part due to US Government restrictions on exports.

The initial PGP application was released in 1991. According to [16] this was an "emergency release" prompted in part by a proposed anti-crime bill which would require eavesdropping ability for the US Government on all communications systems. An RSA-based public-key scheme was used, along with a symmetric-key algorithm developed by Zimmermann known as Bass-O-Matic.

Security concerns over Bass-O-Matic resulted in its replacement with IDEA in PGP 2. A commercial version of PGP was developed in 1993 with ViaCrypt (which had a license from Public Key Partners for RSA). Although RSA Data Security had released a reference implementation (RSAREF) of RSA that could be used for non-commercial purposes, there were interface and other difficulties preventing its use in PGP. In 1994, RSAREF 2.0 was released and included changes which MIT recognized would solve the interface problems. This eventually led to PGP 2.6, a version which could be used freely for non-commercial purposes, and which quickly leaked out of the US and developed into several international variants.

MIT PGP 2.6.2 increased the ceiling on the maximum size of an RSA modulus (from 1024 to 2048 bits, although ViaCrypt reports a patch correcting certain bugs with the longer moduli). The symmetric-key cipher is IDEA, a 64-bit block cipher with 128-bit keys; MD5 is used as the hash function, having digest length of 128 bits. A dependency tree for various US and international versions and variants may be found via [38].

Work on PGP 3 began in 1994, and was released by PGP Inc (formed by Zimmermann) as PGP 5 in May 1997.[1] New algorithms were present, including DSA [34] for signatures, an ElGamal public-key encryption scheme [12], the Secure Hash Algorithm (SHA-1) [35] with 160-bit message digests, and the symmetric-key ciphers CAST and Triple-DES (64-bit block ciphers with key sizes of 128 and 168 bits, respectively).

In August of 1997, the IETF was approached concerning a proposal to bring PGP to a standards body as a protocol. An OpenPGP working group was formed. Using PGP 5 as the base, a format specification was promoted to a Proposed Standard by the IESG in October 1998. The resulting IETF specification for OpenPGP [9] describes an unencumbered architecture, although compatibility with PGP 2.6 was encouraged. A reference implementation was written by Tom Zerucha and provided in a form suitable for scanning to circumvent US export restrictions [8].

In December 1999, Network Associates (which had acquired PGP Inc in December 1997) was granted a license by the US Government to export PGP. An international PGP project [25], which had been making PGP available world-wide by scanning paper copies that were (legally) exported from the US, announced that the lifting of the ban on strong encryption "marks the end of the PGPi scanning and OCR project, which started with PGP 5.0i in 1997."

Several OpenPGP-compliant applications have been developed. The reference implementation by Zerucha [8] relies on the OpenSSL library [37], and has been used by Zerucha as the basis for a PalmPilot implementation. The standard does not require the use of patented algorithms, and applications such as GNU Privacy Guard [18], released in 1999 as a replacement for PGP, can be both compliant and distributable without patent restrictions (since it does not include IDEA or RSA).

2.2 PGP security services

KEY GENERATION AND STORAGE. PGP allows a user to generate multiple key pairs (public-key/private-key pairs) for each public scheme supported. Different key pairs are generated for public-key encryption and for digital signatures. The key pairs, together with public keys of other users, are stored in a file called the key ring.

Information stored with a public key includes the user's name, email address, trust and validity indicators, key type, key size, expiry date, fingerprint (e.g., the 160-bit SHA-1 hash of the formatted public key), and a key ID (e.g., the low order 64 bits of the fingerprint).

Private keys are not stored directly in the key ring. Instead, the user selects a passphrase which is salted and hashed to derive a key k for a symmetric encryption scheme. The private key is encrypted using k, the passphrase is discarded, and the encrypted private key is stored. Subsequently, when the user wishes to access a private key (in order to decrypt a message or sign a message), the passphrase must be supplied so that the system can regenerate k and recover the private key.

CRYPTOGRAPHIC SERVICES. PGP uses a combination of symmetric-key and public-key methods to provide authentication and confidentiality.

A message can be signed using the private key from a suitable public-key signature scheme. The recipient can

[1] Callas [8] notes that ViaCrypt had released several products with a version number of 4 although they were derivatives of PGP 2, and "it was easier to explain why three became five than to explain why three was the new program and four the old one."

verify the signature once an authentic copy of the signer's corresponding public key is obtained. The OpenPGP standard requires support for SHA-1 as a hash algorithm and the DSA, and encourages support for the MD5 hash function and RSA as a signature algorithm.

The use of symmetric-key algorithms (such as DES) alone for encryption is supported, although PGP is known more for the confidentiality provided by a combination of public-key and symmetric-key schemes. Since public-key encryption schemes tend to be computationally expensive, a session key is used with a symmetric-key scheme to encrypt a message; the session key is then encrypted using one or more public keys (typically, one for each recipient), and then the encrypted message along with each encrypted session key is delivered. The standard requires support for an ElGamal public-key encryption scheme and Triple-DES; support for RSA, IDEA, and CAST is encouraged.

Signatures and encryption are often used together, to provide authentication and confidentiality. The message is first signed and then encrypted as described above.

KEY MANAGEMENT. The OpenPGP standard does not have a trust model. An OpenPGP-compliant PGP implementation could support a hierarchical X.509-based public key infrastructure (PKI). The trust model employed by existing PGP implementations is a combination of direct trust and the web of trust. In the former, user *A* obtains *B*'s public key directly from *B*; fingerprints facilitate this process as only the fingerprints have to be authenticated. In the web of trust model, one or more users can attest to the validity of *B*'s public key by signing it with their own signing key. If *A* possesses an authentic copy of the public key of one of these users, then *A* can verify that user's signature thereby obtaining a measure of assurance of the authenticity of *B*'s public key. This chaining of trust can be carried out to any depth.

3 RIM's Pager

3.1 Overview

The RIM wireless handheld device is built around a custom Intel 386 processor running at 10 MHz. Current models carry 2 Mbytes of flash memory and 304 Kbytes of SRAM. There is a fairly conventional (if rather small) keyboard with a 6- or 8-line by 28 character (depending on font) graphical display. A thumb-operated trackwheel takes the place of a conventional mouse (see Figure 1).

A set of applications including a calendar and address book are commonly installed; even the occasional game of Tetris (falling blocks) is possible with efficient use of the graphical display. The main attraction is the wireless communication features, in particular, email solutions. The integrated wireless modem is essentially invisible, with no protruding antennae. The device is roughly 3.5in x 2.5in x 1in (89mm x 64mm x 25mm) and weighs 5 ounces (142 g) with the single AA battery (there is also an internal lithium cell). RIM claims that the battery will last roughly three weeks with typical usage patterns.

A docking cradle can be used to directly connect the device to a serial port. Software for Microsoft Windows is provided to download programs and other information, and to synchronize application data. An RS-232 compatible serial port on the pager runs at 19200 bps.

To be slightly more precise, RIM has two hardware devices, the 850 and the 950, which are combined with software to provide communications solutions. We used RIM's BlackBerry solution [6] which uses the same hardware as the RIM Inter@ctive Pager 950. The 950 is more of a 2-way pager, sold in Canada by Cantel and in the US by BellSouth Wireless Data. The BlackBerry is sold directly by RIM and includes features such as single mailbox integration and PIM synchronization to the device.

The RIM 850 looks very similar to the 950 device, but runs on a different wireless network (ARDIS for the 850 as opposed to Mobitex for the 950). The RIM 850 is resold through American Mobile Satellite Corporation (AMSC) in the US, and is part of the AMSC and SkyTel eLink solution.

3.2 Software development

The BlackBerry Software Developer's Kit (SDK) is designed to make use of the features in Microsoft's C++ compiler packages. The SDK is freely available from [41]. A handheld application is built as a Windows DLL, a process which allows use of development and debugging facilities available for Windows. However, only a small subset of the usual library calls may be used, along with calls to SDK-supplied routines. The resulting DLL is then stripped of extraneous information and ported into the handheld operating system.

For simplicity, the multitasking is cooperative. An application is expected to periodically yield control; in fact, failure to yield within 10 seconds can trigger a pager reset. As an example, public-key operations tend to be computationally expensive, and it was necessary to insert explicit task yields in the code developed for this paper.

The SDK includes a simulator which can be used to test applications on the handheld operating system without having to download to the device (the images in this paper are snapshots of the simulator). A radio device (RAP modem) can be connected via serial port to the host machine so that applications running in the simulator can communicate with the Mobitex network. Alternately, a pager in the cradle can be used to exchange email with the simulator, provided that the pager is in coverage.

Figure 1: The RIM pager.

The simulator is essential for serious development, although testing on the pager can reveal bugs not found in the simulator. For example, we managed to link applications in such a way that they would work in the simulator but fail on the pager. At one point, we carelessly used some instructions introduced on the Intel 486, which would work in the simulator when running on a 486-or-better, but would fail on a 386.

3.3 File system

The pager relies on flash memory to store non-volatile data. Writing to flash is significantly more expensive than reading, primarily because flash is a write-once, bulk-erase device. Rewriting a single word of flash involves saving the contents of the 64K sector, erasing, and rewriting the entire sector. The longest step in this operation is erasing the sector, and takes approximately 5 seconds. A log-structured file system is employed in order to maintain acceptable performance. Periodically, the expensive process of committing the log updates is performed in order to free file system space.

The programming interface to the file system is generally through a relatively small number of high-level database-style calls. Handles are used to read and update databases and variable-length records, a simple but effective method to cooperate with the updating process of the log-structured file system. It is possible to use stream-style I/O operations of the type familiar to C programmers, which we occasionally found useful for testing code fragments developed on more traditional systems.

4 The PalmPilot

For comparison, our crypto routines were also run on the PalmPilot, a very popular PDA based on a 16 MHz Motorola 68000-type "Dragonball" processor.[2] Recent models carry 2–4 MB of memory in addition to ROM, although considerable expansion is possible. In 1999, wireless capabilities were introduced on the Palm VII. The communications model differs from the RIM device; in particular, the Palm does not qualify as a pager in the usual sense. There is an antenna which must be physically activated and then the device can request information. A NiCad battery charged from two AAA batteries common in the Palm series is used to power the radio.

Ian Goldberg had adapted portions of Eric Young's well-known SSLeay library (now OpenSSL [37]) for use on the PalmPilot [19]. The resulting library was used by Zerucha in building a Palm version of his reference OpenPGP, and by Daswani and Boneh [11] in their paper on electronic commerce.

We used Palm development tools based on the GNU C compiler (gcc-2.7.2.2). Timings were done on a Palm V running PalmOS 3.0. There are code segment and stack restrictions which must be considered in the design of a larger application, and our code had to be divided into several libraries in order to accommodate the Palm.

[2] According to [39], "Even after two rounds of Microsoft's best Windows CE efforts, PalmPilot OS devices still represent 80% of all palmtop sales."

5 Elliptic Curve Cryptography

5.1 Introduction

Elliptic curve cryptography (ECC) was proposed independently in 1985 by Neal Koblitz [27] and Victor Miller [33]. For an introduction to ECC, the reader is referred to Chapter 6 of Koblitz's book [29], or the recent book by Blake, Seroussi and Smart [7].

The primary reason for the attractiveness of ECC over RSA and discrete log (DL[3]) public-key systems is that the best algorithm known for solving the underlying hard mathematical problem in ECC (the elliptic curve discrete logarithm problem, ECDLP) takes fully exponential time. On the other hand, the best algorithms known for solving the underlying hard mathematical problems in RSA and DL systems (the integer factorization problem, and the discrete logarithm problem) take subexponential time. This means that the algorithms for solving the ECDLP become infeasible much more rapidly as the problem size increases than those algorithms for the integer factorization and discrete logarithm problems. For this reason, ECC offers security equivalent to that of RSA and DL systems, while using significantly smaller key sizes.

Table 1 lists ECC key lengths and very rough estimates of DL and RSA key lengths that provide the same security (against known attacks) as some common symmetric encryption schemes. The ECC key lengths are twice the key lengths of their symmetric cipher counterparts since the best general algorithm known for the ECDLP takes $(\sqrt{\pi 2^k})/2$ steps for k-bit ECC keys, while exhaustive key search on a symmetric cipher with l-bit keys takes 2^l steps. The estimates for DL security were obtained from [2]. The estimates for RSA security are the same as those for DL security because the best algorithms known for the integer factorization and discrete logarithm problems have the same expected running times. These estimates are roughly the same as the estimates provided by Lenstra and Verheul in their very thorough paper [31].

The advantages that may be gained from smaller ECC parameters include speed (faster computation) and smaller keys and certificates. These advantages are especially important in environments where processing power, storage space, bandwidth, or power consumption are at a premium such as smart cards, pagers, cellular phones, and PDAs.

[3]Examples of DL systems are the ElGamal public-key encryption scheme and the DSA signature scheme which is specified in the Digital Signature Standard. PGP documentation refer to these two schemes as Diffie-Hellman/DSS or DH/DSS.

5.2 Selecting ECC parameters

NOTATION. In the following, $\mathbb{F}_q$ denotes a finite field of order q, and E denotes an elliptic curve defined over $\mathbb{F}_q$. $\#E(\mathbb{F}_q)$ denotes the number of points on the elliptic curve E. The point at infinity is denoted by $\mathcal{O}$. There is a group law for adding any two elliptic curve points. If k is an integer and $P \in E(\mathbb{F}_q)$ is a point, then kP is the point obtained by adding together k copies of P; this process is called scalar multiplication.

DOMAIN PARAMETERS. ECC domain parameters consist of the following:

q	—	the field size.
FR	—	method used for representing field elements.
a, b	—	elements of $\mathbb{F}_q$ which determine the equation of an elliptic curve E.
G	—	the base point of prime order.
n	—	the order of G.
h	—	the cofactor: $h = \#E(\mathbb{F}_q)/n$.

The primary security parameter (see §5.4) is n. The ECC key length is thus defined to be the bitlength of n. Typical choices for q are an odd prime (in which case $\mathbb{F}_q$ is called a *prime field*) or a power of 2 (in which case $\mathbb{F}_q$ is called a *binary field*).

CURVES SELECTED. For this project, we chose binary fields $\mathbb{F}_{2^m}$, for $m = 163$, 233 and 283. Suitably chosen elliptic curves over these fields provide at least as much security as symmetric-key ciphers with key lengths 80, 112 and 128 bits respectively (see Table 1). A polynomial basis representation was used to represent field elements. Such a representation is defined by a reduction polynomial $f(x)$, which is an irreducible binary polynomial of degree m. For each field $\mathbb{F}_{2^m}$, we chose a random curve over $\mathbb{F}_{2^m}$ and a Koblitz curve [28] over $\mathbb{F}_{2^m}$ from the list of elliptic curves recommended by NIST for US federal government use [34]. The salient features of the Koblitz curves are provided in Table 2. Koblitz curves have special structure that enable faster elliptic curve arithmetic in some environments (see [44, 45]). The number of points on each of the chosen curves is almost prime; that is, $\#E(\mathbb{F}_{2^m}) = nh$, where n is prime and $h = 2$ or $h = 4$. Since $\#E(\mathbb{F}_{2^m}) \approx 2^m$, it follows that the ECC key length is approximately equal to m. Security implications of these choices are discussed in §5.4.

5.3 ECC protocols

KEY GENERATION. An entity A's public and private key pair is associated with a particular set of EC domain parameters $(q, \text{FR}, a, b, G, n, h)$. This association can be assured cryptographically (e.g., with certificates) or by context (e.g., all entities use the same domain parameters).

Symmetric cipher key lengths	Example algorithm	ECC key lengths for equivalent security	DL/RSA key lengths for equivalent security
80	SKIPJACK	160	1024
168	Triple-DES	224	2048
128	128-bit AES	256	3072
192	192-bit AES	384	7680
256	256-bit AES	512	15360

Table 1: ECC, DL, and RSA key length comparisons.

m	163
$f(x)$	$x^{163}+x^7+x^6+x^3+1$
E	$Y^2+XY=X^3+X^2+1$
n	4000000000000000000020108A2E0CC0D99F8A5EF
h	2
m	233
$f(x)$	$x^{233}+x^{74}+1$
E	$Y^2+XY=X^3+1$
n	8000000000000000000000000000069D5BB915BCD46EFB1AD5F173ABDF
h	4
m	283
$f(x)$	$x^{283}+x^{12}+x^7+x^5+1$
E	$Y^2+XY=X^3+1$
n	1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE9AE2ED07577265DFF7F94451E061E163C61
h	4

Table 2: Koblitz curves selected.

To generate a key pair, entity A does the following:

1. Select a random integer d from $[1, n-1]$.
2. Compute $Q = dG$.
3. A's public key is Q; A's private key is d.

PUBLIC KEY VALIDATION. This process ensures that a public key has the requisite arithmetic properties. A public key $Q = (x_Q, y_Q)$ associated with domain parameters $(q, \text{FR}, a, b, G, n, h)$ is validated using the following procedure:

1. Check that $Q \neq \mathcal{O}$.
2. Check that x_Q and y_Q are properly represented elements of $\mathbb{F}_q$.
3. Check that Q lies on the elliptic curve defined by a and b.
4. Check that $nQ = \mathcal{O}$.

The computationally expensive operation in public key validation is the scalar multiplication in step 4. This step can sometimes be incorporated into the protocol that uses Q – this is done in the ECAES below. Public key validation with step 4 omitted is called *partial* public key validation.

ELLIPTIC CURVE AUTHENTICATED ENCRYPTION SCHEME (ECAES). The ECAES, proposed by Abdalla, Bellare and Rogaway [1], is a variant of the ElGamal public-key encryption scheme [12]. It is efficient and provides security against adaptive chosen-ciphertext attacks.

We suppose that receiver B has domain parameters $D = (q, \text{FR}, a, b, G, n, h)$ and public key Q. We also suppose that A has authentic copies of D and Q. In the following, MAC is a message authentication code (MAC) algorithm such as HMAC [30], ENC is a symmetric encryption scheme such as Triple-DES. KDF denotes a key derivation function which derives cryptographic keys from a shared secret point.

To encrypt a message m for B, A does:

1. Select a random integer r from $[1, n-1]$.
2. Compute $R = rG$.
3. Compute $K = hrQ$. Check that $K \neq \mathcal{O}$.
4. Compute $k_1 \| k_2 = \text{KDF}(K)$.
5. Compute $c = \text{ENC}_{k_1}(m)$.
6. Compute $t = \text{MAC}_{k_2}(c)$.
7. Send (R, c, t) to B.

To decrypt ciphertext (R, c, t), B does:

1. Perform a partial key validation on R.
2. Compute $K = hdR$. Check that $K \neq \mathcal{O}$.
3. Compute $k_1 \| k_2 = \text{KDF}(K)$.
4. Verify that $t = \text{MAC}_{k_2}(c)$.
5. Compute $m = \text{ENC}_{k_1}^{-1}(c)$.

The computationally expensive operations in encryption and decryption are the scalar multiplications in steps 2-3 and step 2, respectively.

ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM (ECDSA). The ECDSA is the elliptic curve analogue of the DSA [34]. SHA-1 is the 160-bit hash function [35].

We suppose that signer A has domain parameters $D = (q, \text{FR}, a, b, G, n, h)$ and public key Q. We also suppose that B has authentic copies of D and Q.

To sign a message m, A does the following:

1. Select a random integer k from $[1, n-1]$.
2. Compute $kG = (x_1, y_1)$ and $r = x_1 \bmod n$. If $r = 0$ then go to step 1.
3. Compute $k^{-1} \bmod n$.
4. Compute $e = \text{SHA-1}(m)$.
5. Compute $s = k^{-1}\{e + dr\} \bmod n$. If $s = 0$ then go to step 1.
6. A's signature for the message m is (r, s).

To verify A's signature (r, s) on m, B should do the following:

1. Verify that r and s are integers in $[1, n-1]$.
2. Compute $e = \text{SHA-1}(m)$.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $u_1 G + u_2 Q = (x_1, y_1)$.
6. Compute $v = x_1 \bmod n$.
7. Accept the signature if and only if $v = r$.

The computationally expensive operations in signature generation and signature verification are the scalar multiplications in step 2 and step 5, respectively.

5.4 Security issues

HARDNESS OF THE ECDLP. It can easily be verified that the elliptic curves $E(\mathbb{F}_q)$ chosen resist all known attacks on the ECDLP. Specifically:

1. The number of points, $\#E(\mathbb{F}_q)$, is divisible by a prime n that is sufficiently large to resist the parallelized Pollard rho attack [36] against general curves, and its improvements [15, 48] which apply to Koblitz curves.
2. n does not divide $q^k - 1$ for all $1 \leq k \leq 30$, confirming resistance to the Weil pairing attack [32] and the Tate pairing attack [13].
3. $\#E(\mathbb{F}_q) \neq q$, confirming resistance to the Semaev attack [43].
4. All binary fields $\mathbb{F}_{2^m}$ chosen have the property that m is prime, thereby circumventing recent attacks [14, 17] on the ECDLP for elliptic curves over binary fields $\mathbb{F}_{2^m}$ where m is composite.

SECURITY OF ECAES. The ECAES modifies the ElGamal encryption scheme by using the one-time Diffie-Hellman shared secret, $hrdG$, to derive secret keys k_1 and k_2 The first key k_1 is used to encrypt the message using a symmetric cipher, while the second key k_2 is used to authenticate the resulting ciphertext. The latter provides resistance to chosen-ciphertext attacks. Some formal justification of ECAES security is provided in [1], where it is proven to be semantically secure against adaptive chosen-ciphertext attack on the assumption that the underlying symmetric encryption and MAC schemes are secure, and assuming the hardness of certain variants of the elliptic curve Diffie-Hellman problem.

In order to correctly balance the security of the ECAES cryptographic components, one should ideally employ a $\frac{k}{2}$-bit block cipher and a k-bit hash function for HMAC when using a k-bit elliptic curve (see Table 1). Our implementation used the 112-bit block cipher Triple-DES in CBC-mode and the 160-bit hash function SHA-1 for all 3 choices of ECC key lengths (163, 233 and 283). A future version of our implementation should allow for a variable output-length hash function (e.g., the forthcoming SHA-2) and a variable-length block cipher (e.g., the AES).

SECURITY OF ECDSA. ECDSA is the straightforward elliptic curve analogue of the DSA, which has been extensively scrutinized since it was proposed in 1991. For a summary of the security properties of the ECDSA, see [26].

Our implementation used the 160-bit hash function SHA-1 for all 3 choices of ECC key lengths (163, 233 and 283). As with the ECAES, a future version of our ECDSA implementation should allow for a variable output-length hash function.

5.5 Timings

This section presents timings for the ECC operations on a Pentium II 400 MHz machine, a PalmPilot and the RIM pager, and compares them with timings for RSA and DL operations.

ECC TIMINGS. Our ECC code was written entirely in C on a Sun Sparcstation and, in order to ensure porta-

bility, no assembler was used. We encountered no problems in porting the code to the Pentium II, RIM pager, and PalmPilot platforms, although some changes were required in order to cooperate with the 16-bit options used in the Palm version of the "big number" library of OpenSSL. No effort was made to optimize the ECC code for these particular platforms; it is very likely that significant performance improvements could be obtained by optimizing the ECC (and DL and RSA) code for these platforms. Further details of our ECC implementations are reported in [23].

For other ECC implementation reports, see [42] for a C implementation of elliptic curve arithmetic over $\mathbb{F}_{2^{155}}$, [49] for a C/C++ of elliptic curve arithmetic over $\mathbb{F}_{2^{191}}$ and over a 191-bit prime field, and [22] for an assembly language implementation of elliptic curve arithmetic over a 160-bit prime field on a 10 MHz 16-bit microcomputer.

Tables 3, 4 and 5 present timings of our implementation for ECC operations using the Koblitz curves and random curves over $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$.

RSA TIMINGS. The RSA code, written entirely in C, was taken from the OpenSSL library [37]. Tables 6 and 7 present timings for 512, 768, 1024, and 2048-bit RSA operations.

DL TIMINGS. The DSA and ElGamal code, also written entirely in C, was obtained from the OpenSSL and OpenPGP libraries. For ElGamal, the prime p was chosen to be a safe prime; that is $p = 2q + 1$ where q is also prime. Table 8 presents timings for 512, 768 and 1024-bit DSA and ElGamal operations. For encryption, the per-message secret key is not of full length (i.e., the bitlength of p), but of bitlength 200 + (bitlength of p)/32; this explains why ElGamal encryption is faster than ElGamal decryption. The ElGamal operations could be sped up significantly if DSA-like parameters were used (i.e., $p = kq + 1$, where q is a 160-bit prime).

COMPARISON. The performance of all three families of public-key systems (ECC, RSA and DL) are sufficiently fast for PGP implementations on a Pentium machine—it hardly matters whether a user has to wait 10 ms or 100 ms to sign and encrypt a message.

On the pager, RSA public-key operations (encryption and signature verification) are faster than ECC public-key operations, especially when the public exponent is $e = 3$. For example, verifying a 1024-bit RSA signature takes about 300 ms, while verifying a 163-bit ECC signature (using a Koblitz curve) takes about 1,800 ms. On the other hand, RSA private-key operations (decryption and signature generation) are slower than ECC private-key operations. For example, signing with a 1024-bit RSA key takes about 16,000 ms, while signing with a 163-bit ECC key takes about 1,000 ms. ECC has a clear advantage over RSA for PGP operations that require both private key and public key computations. Signing-and-encrypting together takes 16,400 ms with 1024-bit RSA (using $e = 3$), and 2800 ms with 163-bit ECC (using a Koblitz curve). Verifying-and-decrypting together takes 16,200 ms with 1024-bit RSA, and 2,900 ms with 163-bit ECC.

Similar conclusions are drawn when comparing RSA and ECC performance on the PalmPilot.

Private key operations with 2048-bit RSA are too slow for the pager and the PalmPilot, while 233-bit ECC and 283-bit ECC operations are tolerable for PGP applications on the pager.

Since domain parameters are used in our ECC implementation, ECC key generation only involves a single scalar multiplication and thus is very fast on the pager. RSA, ElGamal and DSA key generation on the pager is prohibitively slow. However, ElGamal and DSA key generation would be feasible on the pager if precomputed domain parameters (primes p and q, and generator g) were used.

5.6 Interoperability

The elliptic curves and protocols were selected to conform with the prevailing ECC standards and draft standards.

The Koblitz and random curves over $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$ are from the list of NIST recommended curves [34]. The representations, for both field elements and for elliptic curve points, are compliant with the ANSI X9.62 [4], ANSI X9.63 [5], IEEE P1363 [24] and FIPS 186-2 [34] standards. In addition, the Koblitz curve over $\mathbb{F}_{2^{163}}$ is explicitly listed in the WAP wTLS specification [51].

Our ECDSA implementation conforms to the security and interoperability requirements of ANSI X9.62, IEEE P1363, and FIPS 186-2. Our ECAES implementation conforms to the security and interoperability requirements of ANSI X9.63. The cryptographic components HMAC and Triple-DES (in CBC mode) of ECAES are compliant, respectively, with RFC 2104 [30] and ANSI X9.52 [3].

6 Porting PGP to the Pager

There are now a number of cryptographic libraries and PGP applications which have received extensive development and for which source code is available; see, for example, cryptlib by Peter Gutmann [20] and Crypto++ by Wei Dai [10]. Our plan was to adapt existing code, adding public-key schemes based on elliptic curves. For comparisons and development, it was essential that the

	Koblitz curve over $\mathbb{F}_{2^{163}}$			Random curve over $\mathbb{F}_{2^{163}}$		
	RIM pager	PalmPilot	Pentium II	RIM pager	PalmPilot	Pentium II
Key generation	751	1,334	1.47	1,085	1,891	2.12
ECAES encrypt	1,759	2,928	4.37	3,132	5,458	6.67
ECAES decrypt	1,065	1,610	2.85	2,114	3,564	4.69
ECDSA signing	1,011	1,793	2.11	1,335	2,230	2.64
ECDSA verifying	1,826	3,263	4.09	3,243	5,370	6.46

Table 3: Timings (in milliseconds) for ECC operations over $\mathbb{F}_{2^{163}}$ on various platforms.

	Koblitz curve over $\mathbb{F}_{2^{233}}$			Random curve over $\mathbb{F}_{2^{233}}$		
	RIM pager	PalmPilot	Pentium II	RIM pager	PalmPilot	Pentium II
Key generation	1,552	2,573	3.11	2,478	3,948	4.58
ECAES encrypt	3,475	5,563	7.83	6,914	11,373	13.99
ECAES decrypt	2,000	2,969	4.85	4,593	7,551	9.55
ECDSA signing	1,910	3,080	4.03	3,066	4,407	5.52
ECDSA verifying	3,701	5,878	7.87	7,321	11,964	14.08

Table 4: Timings (in milliseconds) for ECC operations over $\mathbb{F}_{2^{233}}$ on various platforms.

code run on several platforms in addition to the RIM device.

Our initial work was with GNU Privacy Guard (GnuPG) [18], an OpenPGP-compliant freely distributable replacement for PGP, which was nearing a post-beta release in 1999. Initial tests on the pager with several fragments adapted from GnuPG sources were promising, and the code appeared to be ideal for adding the elliptic curve routines and testing on Unix-based and other systems. However, it appeared that untangling code dependencies for our use on the pager would be unpleasant. (Perhaps a better understanding of GnuPG internals and design decisions would have changed our opinion.)

Jonathan Callas suggested that we look again at the OpenPGP reference implementation [8], which we had put aside after initial testing revealed a few portability and alignment problems in the code. The reference implementation relied on the OpenSSL library [37].

The OpenPGP reference implementation is surprisingly complete for the amount of code, although it is admittedly a little rough on the edges.[4] The code was developed on a Linux/x86 system, and modifications were required for alignment errors which prevented the program from running on systems such as Solaris/SPARC. In addition, some portability changes were required, including code involving the "long long" data type. For the RIM pager, the separation of the PGP code from the well-tested OpenSSL library, along with the small size of the OpenPGP sources, were definite advantages. Finally, it should be noted that the OpenSSL libraries build easily on Unix and Microsoft Windows systems, and are designed so that adding routines such as the elliptic curve code is straightforward.

Although applications for the pager are built as Windows DLLs, the pager is not a Windows-based system. There are significant restrictions on the calls that can be used, extending to those involving memory allocation, time and character handling, and the file system. There is no floating-point processor on the pager. In order to adapt code developed on more traditional systems, we wrote a library of compatibility functions to use with the pager. Some functions were trivial (such as those involving memory allocation, since the SDK included equivalent calls); others, such as the stream I/O calls, were written to speed testing and porting and cannot be recommended as particularly robust or elegant.

We used portions of OpenSSL 0.9.4, along with the library in the OpenPGP reference implementation. Relatively few changes to OpenSSL were required, and could be restricted to header files in many cases. The elliptic curve routines were integrated, including additions to the scripts used to build OpenSSL. For some platforms, OpenSSL can be built using assembly-language versions of certain key routines to improve execution speed. Some of these files for the Intel x86 include instructions (such as bswap) which were introduced for the 486, and cannot be used on the pager.

The OpenPGP sources were modified to correct the alignment bugs and portability problems mentioned above, and necessary changes were made for the elliptic curve schemes (public-key algorithms 18 and 19 in the

[4] Zerucha writes that he wasn't "careful about wiping memory and preventing memory leaks and other things to make the code robust" [8].

	Koblitz curve over $\mathbb{F}_{2^{283}}$			Random curve over $\mathbb{F}_{2^{283}}$		
	RIM pager	PalmPilot	Pentium II	RIM pager	PalmPilot	Pentium II
Key generation	2,369	4,062	4.50	3,857	6,245	6.88
ECAES encrypt	5,227	8,579	11.02	11,264	18,273	20.86
ECAES decrypt	2,932	4,495	6.78	7,498	12,046	13.88
ECDSA signing	2,760	4,716	5.64	4,264	6,816	8.08
ECDSA verifying	5,485	9,059	11.46	11,587	18,753	21.15

Table 5: Timings (in milliseconds) for ECC operations over $\mathbb{F}_{2^{283}}$ on various platforms.

	512-bit modulus			768-bit modulus		
	Pager	Pilot	Pentium II	Pager	Pilot	Pentium II
RSA key generation	73,673	189,461	346.77	287,830	496,356	953.01
RSA encrypt ($e = 3$)	213	317	1.13	388	587	1.87
RSA encrypt ($e = 17$)	262	410	1.28	451	753	2.17
RSA encrypt ($e = 2^{16} + 1$)	428	743	1.90	793	1,347	3.32
RSA decrypt	2,475	5,858	11.05	7,905	16,262	28.05
RSA signing	2,466	5,751	10.78	7,889	16,047	27.72
RSA verifying ($e = 3$)	99	200	0.40	214	413	0.78
RSA verifying ($e = 17$)	147	293	0.56	273	577	1.07
RSA verifying ($e = 2^{16} + 1$)	314	623	1.17	616	1,221	2.24

Table 6: Timings (in milliseconds) for 512-bit and 768-bit RSA operations on various platforms.

OpenPGP specification [9]). The compatibility library, along with a few stream-to-memory conversion functions allowed fairly direct use of the OpenPGP sources on the pager.

The only code tested exclusively in the pager environment involved the user interface (see §7.1). The SDK provides a fairly powerful and high-level API for working with the display and user input. The difficulties we encountered were mostly due to the lack of support in the API for direct manipulation of messages desired in a PGP framework. In part, this reflects a deliberate design decision by BlackBerry to develop a robust and intuitive communication solution which provides some protection against misbehaving applications.[5]

The pager DLLs for the interface and PGP library were over 400 KB in combined size. This includes all of the OpenPGP required algorithms and recommended algorithms such as IDEA and RSA, along with the new schemes based on elliptic curves. For a rough comparison, the code size for the main executable from the OpenPGP reference implementation (with the addition of the elliptic curve routines) is 300–400 KB, depending on platform.

[5] During our work on this project, BlackBerry modified the API to provide some of the access needed to smoothly integrate PGP into their mail application.

7 Implementation

7.1 User interface

PGP in any form has not been an easy application for novices to manage properly, in part due to the sophistication required, but also because of poor interface design [47]. The goals for our user interface design were rather modest: that a user who is familiar with using PGP on a workstation, and is comfortable operating the RIM device, should, without having to refer to a manual or help pages, be easily able to figure out how to use PGP on the pager and avoid dangerous errors (such as those described in [47]). As mentioned in §3.1, the graphics capabilities and screen size of the RIM device are very limited. This forced us to keep our PGP implementation simple and only offer the user the essential features.

A glimpse of our user interface is provided in Figures 1–5. Clicking on the PGP icon (see Figure 1) displays the list of users whose keys are in the public key ring (see Figure 2). Selecting a user name displays the menu shown in Figure 3, which allows the user to view the key's attributes, compose a new key, delete a key, or send a key.

	1024-bit modulus			2048-bit modulus		
	Pager	Pilot	Pentium II	Pager	Pilot	Pentium II
RSA key generation	580,405	1,705,442	2,740.87	—	—	26,442.04
RSA encrypt ($e = 3$)	533	1,023	2.70	1,586	3,431	7.26
RSA encrypt ($e = 17$)	683	1,349	3.23	2,075	4,551	9.09
RSA encrypt ($e = 2^{16} + 1$)	1,241	2,670	5.34	4,142	8,996	16.57
RSA decrypt	15,901	36,284	67.32	112,091	292,041	440.78
RSA signing	15,889	36,130	66.56	111,956	288,236	440.69
RSA verifying ($e = 3$)	301	729	1.23	1,087	2,392	4.20
RSA verifying ($e = 17$)	445	1,058	1.76	1,585	3,510	6.10
RSA verifying ($e = 2^{16} + 1$)	1,008	2,374	3.86	3,608	7,973	13.45

Table 7: Timings (in milliseconds) for 1024-bit and 2048-bit RSA operations on various platforms.

	512-bit modulus			768-bit modulus			1024-bit modulus		
	Pager	Pilot	PII	Pager	Pilot	PII	Pager	Pilot	PII
ElGamal key gen	—	—	51,704	—	—	219,820	—	—	1,200,157
ElGamal encrypt	7,341	17,338	19.13	16,078	34,904	35.91	26,588	73,978	67.78
ElGamal decrypt	8,704	19,060	22.55	26,958	56,708	59.53	57,248	148,059	144.73
DSA key gen	—	—	3,431	—	—	14,735	—	—	54,674
DSA signing	2,955	6,329	7.53	6,031	11,875	15.55	9,529	25,525	24.28
DSA verifying	5,531	12,389	14.31	11,594	24,277	26.13	18,566	52,286	47.23

Table 8: Timings (in milliseconds) for DL operations on various platforms.

Figure 2: Listing of PGP keys.

Figure 3: The main menu.

7.2 Key generation and storage

The main PGP menu (Figure 3) has an option "New Key" for creating a key pair. Users can enter their name, email address, pager PIN, and select a key type and key length (see Figure 4). The key types and key sizes presently available are ECC (random curve or Koblitz curve; over $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ or $\mathbb{F}_{2^{283}}$), DH/DSS (512/512, 768/768, 1024/1024, 1536/1024 or 2048/1024 bits), and RSA (512, 768, 1024, 1536 or 2048 bits). The DH/DSS and RSA key sizes are the ones available in many existing PGP implementations. For the DSA, the maximum bitsize of the prime p is 1024 bits in conformance with the DSS [34]. For ECC, separate key pairs are generated for public-key encryption and digital signatures.

Public keys and private keys are stored in separate key rings. Public key attributes (see Figure 5) can be viewed using the "View Key" function available on the main menu. As required by OpenPGP, private keys are encrypted under a user-selected passphrase, and the encrypted private key is stored. The passphrase has to be entered whenever a private key is required to sign or decrypt a message.

7.3 Cryptographic services

The three basic PGP services are available: sign only, encrypt only, or sign-and-encrypt. Users can decide to sign an email, or to encrypt an email, after composing the message. The user is prompted for the passphrase to unlock the private signing key, and to select the public encryption key of the intended recipient. In addition to the times given in Tables 3–8 for the main operations, there is additional overhead which can be apparent to the

Figure 4: Screen for creating a new key pair.

Figure 5: Screen for viewing a (portion of the) public key's attributes.

user. Verifying the passphrase, for example, may require 20 seconds if the default iteration count is used when hashing the salted passphrase; our implementation used a smaller default iteration count. A small amount of time is added for interaction with the database filesystem for large memory transfers.

7.4 Key management

The key management system we implemented was the simplest one possible—the direct trust model (see §2.2). A menu item is available (see Figure 3) for emailing one's public key to another user. A function is also available for extracting and storing a public key received in an email message. If desired, a public key can be authenticated by verifying its fingerprint by some direct means (e.g., communicating it over the telephone—authenticity is provided by voice recognition).

8 Future Work

The following are some directions for future work.

RANDOM NUMBER GENERATION. Many systems implement a "random gathering device" which attempts to use environmental noise (keyboard data, system timers, disk characteristics, etc.) to build a cryptographically secure source of random bits [21]. Our pager application used only a rather simple (and most likely not sufficiently secure) seeding process involving the clock and a few other sources. A more sophisticated solution is essential, perhaps tapping into the radio apparatus as a source.

CODE SIZE. No serious effort was made to minimize the size of the programs loaded to the pager. There is some code linked from the OpenSSL cryptographic library which could easily be removed (in fact, we were somewhat surprised that the library with the added elliptic curve routines could be used with relatively few modifications for the pager). The library routines adapted from OpenSSL and OpenPGP along with various glue needed to adapt to the pager accounts for approximately 3/4 of the 370 KB loaded on the device (with the remainder attributed to code involving the screen and user-interface). If some interoperability can be sacrificed, then the code size can also be reduced by removing routines such as CAST or some of the hash algorithms.

MAKING THE OPENPGP CODE MORE ROBUST. The OpenPGP reference implementation provides minimal diagnostics and can easily break on bad data. The occasional segmentation fault triggered by bad user data may be merely unpleasant when an application is used on a workstation; such errors on the pager are completely unacceptable. Our application corrects some of the most troublesome shortcomings, but better error-handling is needed.

KEY MANAGEMENT. We would like to implement an X.509-based PKI or the web of trust model. In either case, we would implement a key server for retrieving and storing keys in a key repository. This would involve setting up a proxy wireless server with which the pager would communicate directly. The proxy server in turn would communicate with existing key servers on the Internet.

9 Conclusions

IMPLEMENTING PGP ON THE RIM PAGER. The 32-bit architecture, relatively sophisticated operating system and development environment, and relatively large memory size means that development for the pager is closer to that done for more traditional systems than the small size might suggest. The user interface must be customized for the device, but "generic code" which does not involve file I/O moves fairly easily to the pager.

On the other hand, it appears likely that such devices will continue to have processors which run much more slowly than their desktop counterparts. Long delays in handling encrypted messages or signatures will be a considerable annoyance for users of this type of device. While we used a significant amount of the available memory on the pager, it would be desirable to reduce the resource consumption in a production version of PGP. Battery life will continue to be a major concern, and the overhead of authentication and confidentiality competes with the need to minimize transmissions from the device.

INTEROPERABILITY. The goal of interoperability was met. All of the required algorithms from RFC 2440 are included, along with several listed as recommended and the elliptic curve routines. Our PGP implementation interoperated with existing implementations for the PalmPilot and workstations.

ELLIPTIC CURVE CRYPTOGRAPHY. Elliptic curve solutions fit particularly well into the constrained environment. 1024-bit and 2048-bit RSA private-key operations are too slow for PGP applications, while the performance of 163-bit, 233-bit and 283-bit ECC operations is tolerable for PGP applications. If PGP (or other email security solutions) is to be used for securing email communications between constrained wireless devices and desktop machines, then our timings show that ECC is preferable to RSA since the performance of the latter on some wireless devices is too slow, while both systems perform sufficiently well on workstations.

GENERAL. This paper concentrated on PGP, although the results are more widely applicable. Many of the services targeted at the growing wireless market will require security solutions involving the cryptographic mechanisms used by PGP. The constraints on small wireless devices are likely to be with us for some time, and will require a balance of usability, computational requirements, security, and battery life.

Acknowledgements

The authors would like to thank Jonathan Callas for some enlightening discussions about PGP, and Herb Little for answering our numerous questions about the RIM pager.

References

[1] M. Abdalla, M. Bellare and P. Rogaway, "DHAES: An encryption scheme based on the Diffie-Hellman problem", preprint, 1999. Available from http://www.cs.ucdavis.edu/~rogaway/papers

[2] ANSI X9.30-1, "The digital signature algorithm (DSA) (revised)", American Bankers Association, working draft, July 1999.

[3] ANSI X9.52, "Triple data encryption algorithm modes of operation", American Bankers Association, 1998.

[4] ANSI X9.62, "The elliptic curve digital signature algorithm (ECDSA)", American Bankers Association, 1999.

[5] ANSI X9.63, "Elliptic curve key agreement and key transport protocols", American Bankers Association, working draft, August 1999.

[6] Blackberry, http://www.blackberry.net

[7] I. Blake, G. Seroussi and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[8] J. Callas, *OpenPGP Specification and Sample Code*, Printers Inc. Bookstore, Palo Alto, March 1999.

[9] J. Callas, L. Donnerhacke, H. Finney and R. Thayer, "OpenPGP message format", Internet RFC 2440, November 1998.

[10] W. Dai, Crypto++. http://www.eskimo.com/~weidai/cryptlib.html

[11] N. Daswani and D. Boneh, "Experimenting with electronic commerce on the PalmPilot", *Financial Cryptography '99*, Lecture Notes in Computer Science, **1648** (1999), Springer-Verlag, 1-16.

[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, **31** (1985), 469-472.

[13] G. Frey and H. Rück, "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves", *Mathematics of Computation*, **62** (1994), 865-874.

[14] S. Galbraith and N. Smart, "A cryptographic application of Weil descent", *Codes and Cryptography*, Lecture Notes in Computer Science, **1746** (1999), Springer-Verlag, 191-200.

[15] R. Gallant, R. Lambert and S. Vanstone, "Improving the parallelized Pollard lambda search on binary anomalous curves", to appear in *Mathematics of Computation.*

[16] S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995.

[17] P. Gaudry, F. Hess and N. Smart, "Constructive and destructive facets of Weil descent on elliptic curves", preprint, January 2000. Available from http://www.hpl.hp.com/techreports/2000/HPL-2000-10.html

[18] GNU Privacy Guard, http://www.gnupg.org

[19] I. Goldberg, "Pilot stuff from the ISAAC Group", http://www.isaac.cs.berkeley.edu/pilot/

[20] P. Gutmann, "Cryptolib". http://www.cs.auckland.ac.nz/~pgut001/cryptlib

[21] P. Gutmann, "Software generation of practically strong random numbers", *Proceedings of the Seventh USENIX Security Symposium*, 1998, 243-257.

[22] T. Hasegawa, J. Nakajima and M. Matsui, "A practical implementation of elliptic curve cryptosystems over $GF(p)$ on a 16-bit microcomputer", *Proceedings of PKC '98*, Lecture Notes in Computer Science, **1431** (1998), 182-194.

[23] D. Hankerson, J. Lopez Hernandez and A. Menezes, "Software implementations of elliptic curve cryptography over fields of characteristic two", draft, 2000.

[24] IEEE P1363, "Standard specifications for public-key cryptography", ballot draft, 1999. Drafts available at http://grouper.ieee.org/groups/1363

[25] The International PGP Home Page, http://www.pgpi.org

[26] D. Johnson and A. Menezes, "The elliptic curve digital signature algorithm (ECDSA)", Technical report CORR-34, Dept. of C&O, University of Waterloo, 1999. Available from http://www.cacr.math.uwaterloo.ca

[27] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, **48** (1987), 203-209.

[28] N. Koblitz, "CM-curves with good cryptographic properties", *Advances in Cryptology – Crypto '91*, Lecture Notes in Computer Science, **576** (1992), Springer-Verlag, 279-287.

[29] N. Koblitz, *A Course in Number Theory and Cryptography*, 2nd edition, Springer-Verlag, 1994.

[30] H. Krawczyk, M. Bellare and R. Canetti, "HMAC: Keyed-hashing for message authentication", Internet RFC 2104, February 1997.

[31] A. Lenstra and E. Verheul, "Selecting cryptographic key sizes", *Proceedings of PKC 2000*, Lecture Notes in Computer Science, **1751** (2000), Springer-Verlag, 446-465.

[32] A. Menezes, T. Okamoto and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *IEEE Transactions on Information Theory*, **39** (1993), 1639-1646.

[33] V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology – CRYPTO '85*, Lecture Notes in Computer Science, **218** (1986), Springer-Verlag, 417-426.

[34] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS Publication 186-2, February 2000. Available at http://csrc.nist.gov/fips

[35] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS Publication 180-1, April 1995. Available at http://csrc.nist.gov/fips

[36] P. van Oorschot and M. Wiener, "Parallel collision search with cryptanalytic applications", *Journal of Cryptology*, **12** (1999), 1-28.

[37] OpenSSL, http://www.openssl.org

[38] PGP versions, http://www.paranoia.com/~vax/pgp_versions.html

[39] David Poguet, *PalmPilot: The Ultimate Guide*, 2nd edition, O'Reilly & Associates, 1999.

[40] B. Ramsdell, "S/MIME version 3 message specification", Internet RFC 2633, June 1999.

[41] RIM Software Developer's Kit (SDK), http://developers.rim.net/handhelds/sdk

[42] R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast key exchange with elliptic curve systems", *Advances in Cryptology – Crypto '95*, Lecture Notes in Computer Science, **963** (1995), Springer-Verlag, 43-56.

[43] I. Semaev, "Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p", *Mathematics of Computation*, **67** (1998), 353-356.

[44] J. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves", *Advances in Cryptology – Crypto '97*, Lecture Notes in Computer Science, **1294** (1997), Springer-Verlag, 357-371.

[45] J. Solinas, "Improved algorithms for arithmetic on anomalous binary curves", Technical report CORR-25, Dept. of C&O, University of Waterloo, 1999. Available from http://www.cacr.math.uwaterloo.ca

[46] W@P white paper, 1999, http://www.wapforum.org/what/whitepapers.htm

[47] A. Whitten and J. Tygar, “Why Johnny can’t encrypt: A usability evaluation of PGP 5.0”, *Proceedings of the Eighth USENIX Security Symposium*, 1999.

[48] M. Wiener and R. Zuccherato, “Faster attacks on elliptic curve cryptosystems”, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, **1556** (1999), Springer-Verlag, 190-200.

[49] E. De Win, S. Mister, B. Preneel and M. Wiener, “On the performance of signature schemes based on elliptic curves”, *Algorithmic Number Theory, Proceedings Third Intern. Symp., ANTS-III*, Lecture Notes in Computer Science, **1423** (1998), Springer-Verlag, 252-266.

[50] Wireless Application Protocol Forum, “Wireless Application Protocol”, John Wiley & Sons, Inc., 1999. See also http://www.wapforum.org/

[51] Wireless Application Protocol Forum, “Wireless Transport Layer Security Specification”, chapter 16 of [50], 1999. Drafts available at http://www.wapforum.org

[52] P. Zimmermann, *The Official PGP User’s Guide*, MIT Press, 1995.