Implementing Identity-based Key Agreement in Embedded Devices

Giovanni Schmid[1] and Francesco Rossi[2]
[1]*High Performance Computing and Networking Institute (ICAR), Naples, Italy*
[2]*University of Naples Parthenope, Naples, Italy*

Keywords: ID-based Cryptography, Key Agreement, Embedded Device.

Abstract: In recent years, a substantial work has been devoted to the design of cryptographic protocols with reduced computational load and power consumption. However, implementations are scarce, especially in case of embedded devices. In this paper, we discuss the implementation over elliptic curves of a two-party key agreement protocol for the Raspberry PI platform. The protocol requires just one round to derive an (implicit) authenticated session key, and it makes use of identity-based cryptography, which fits very well some application scenarios, allowing for more efficiency than the certificate-based approach. Our implementation takes advantage of the portability and security features of the Java programming language and, due to a modular design, it can be easily extended to encompass other identity-based schemes and protocols. We run a set of tests in order to verify the correctness of our implementation and to measure its performances in term of computing time. Our results demonstrate that getting secure communications using low cost, resource constrained devices is viable to the point that it can be used for real world applications.

1 INTRODUCTION

In recent times, we are witnessing an increasing use of embedded devices, also in relation to services and applications that require secure communications. Embedded devices have limited computational capabilities and power resources, and these limitations will be presumably hold also in the future, since the driving forces in this area of hardware design point towards an increasing miniaturization and more functionalities. On the other hand, security in communications is obtained due to cryptographic protocols, which however can result in a not negligible overhead, enough to jeopardize the operation of embedded devices. A class of especially expensive cryptographic protocols, both in term of memory usage and computing time, is that for *key agreement* (Menezes et al., 2010). Key agreement is at the core of secure peer-to-peer communications, and its high costs are because it requires asymmetric cryptography. Once agreed upon a key, a set of peers can used such key to secure their communications due to symmetric cryptographic mechanisms. Symmetric mechanisms are much less expensive than asymmetric ones and they are feasible even on embedded devices. From the above, it then follows that efficient, reliable implementations of key agreement protocols play a crucial role in the realization of secure distributed applications for embedded devices. The implementation of a key agreement protocol is a very complex task. Besides the subtleties concerning the software coding of reliable cryptographic mechanisms, a cryptographic protocol requires the management of communications through synchronization among parties, packet assembling and disassembling, data exchange, and error handling. The present work concerns the implementation on the Raspberry PI platform (Monk, 2014) of a protocol derived from the two-party key agreement protocol of (Fiore and Gennaro, 2010) (FG protocol, for short). The FG protocol uses *identity-based* (ID-based) cryptography (Shamir, 1985), thus avoiding the set-up and management of public key certificates. ID-based cryptography can be valuable whenever trust can be enforced centrally through a *key distribution center* (KDC), allowing for better memory efficiency and less communication overhead than certificate-based approaches. Moreover, the FG protocol is optimal as number of communication rounds, requiring just one round to get (implicit) key authentication. In order to match the constrained resources of embedded devices such as the Raspberry PI, we have implemented the FG protocol over elliptic curves. Elliptic curves are very promising to enforce strong cryptography on embedded devices, since they allow to scale much better with respect to key sizes (i.e. security thresholds). Our implementation is in the Java programming language and, due to a modular design, it can be easily extended to encompass other identity-based schemes and protocols. Although Java cannot achieve the performance of full-compiled languages (e.g., C, C++), it is well suited for coding distributed applications and services, because of its portability and built-in security features (De Caro and Iovino, 2011). The paper is organized as follows. Section 2 discusses related work. Section 3 outlines some notions and notations that will be used through the rest of the paper. In Section 4 we describe our elliptic version of the FG protocol. Section 5 describes our Java implementation for the Raspberry PI device, discussing design choices and issues. Section 6 summarizes the results of various tests we have done in order to evaluate performance. Finally, in Section 7 we draw conclusions and sketch out future work.

2 RELATED WORK

A two-party key agreement protocol allows two parties, who each have a long-term key, to agree as peers upon a common secret key by exchanging messages with each other (Menezes et al., 2010). Later, the shared secret key may be used to secure communication sessions among these parties, due to one or more of the fast mechanisms offered by symmetric cryptography.

Starting from the pioneering work of (Diffie and Hellman, 1976), many key agreement protocols have been proposed over the years in the literature. However, many of these protocols lack a sound security analysis, and some of them have been actually proven insecure. On the other hand, many provably secure key agreement protocols are far from being optimal, and require a significant overhead in local computations and/or communication bandwidth. Comprehensive surveys on key agreement can be found in (Boyd and Mathuria, 2003) and (Menezes et al., 2010). (Fiore and Gennaro, 2010) introduced a provably secure, very efficient protocol, which requires only twice the amount of bandwidth and computation of the unauthenticated basic Diffie-Hellman protocol. This protocol is identity-based and outperforms MQV (Law et al., 2003), the most efficient authenticated two-party Diffie-Hellman based protocol in the public-key model. The concept of *identity-based* (ID-based) cryptography was introduced by (Shamir, 1985). The innovation was the use of identity attributes, instead of digital certificates, for data encryption and signature. This avoids the generation and management of users' certificates, reducing significantly the complexity and computational cost of protocols. Pioneered by the work of (Joux, 2000), various ID-based key agreement protocols (e.g., (McCullagh and Barreto, 2005), (Okamoto et al., 2005), (Chen et al., 2007)) have been proposed so far. In order to work, however, these protocols require groups over elliptic curves where special mappings, called *bilinear pairings* (Miller, 2004), are available. Such groups must be carefully selected for getting adequate performance without a downgrade in security. Conversely, the FG protocol works over any cyclic group. This results in the possibility of choosing much smaller groups than in the case of pairing-based protocols to get a given threshold of security, since one can use "regular" elliptic curves, rather than the ones that admit efficient pairings computations for high security levels. This paper concerns an implementation of the FG protocol on the Raspberry PI platform. Although libraries exist that support identity-based cryptography and bilinear pairing operations (e.g., the reference benchmark C library PBC of (Lynn, 2007) and its Java porting jPBC of (De Caro and Iovino, 2011)), at the time being, to the best of our knowledge, there are no known implementations of ID-based key-agreement protocols. Further, there are few implementations also for other classes of key-agreement protocols. Actually, only the Charm library (Akinyele et al., 2011) implements software components for message serialization, data transmission, and error handling.

3 PRELIMINARIES

In this section we first describe how ID-based cryptosystems work, and then we give some basic notions about elliptic curves, showing how to choose them for our scopes.

3.1 ID-based Cryptosystems

In ID-based cryptography (Shamir, 1985), a *trusted key generation center* (KGC) first generates a master public/secret key pair, publishing the public key alongside with any other public parameter which uniquely identifies the Diffie-Hellmann system to be used and the set of strings chosen as user identifiers. In a second phase, the KGC gives to each user having identity U a secret key S_U, which is a function of the string U and the KGC's secret key. Using their own S_U, users can then perform cryptographic tasks. These two phases performed by the KGC are named *Setup* and *Extract*, respectively.

A first advantage of ID-based systems is the versatility with which identifiers may be chosen. Since identities can be arbitrary string, they can be selected according to the function and attributes of the parties. For example, a network device might be identified by its MAC address, its "role" in the network (e.g., "router A"), or its physical position (e.g., latitude and longitude).

A second advantage of identity-based schemes is that they can significantly simplify authentication management, bypassing the certification issues. All a party needs to know in order to communicate with a given peer is its own secret key, the public information of the KGC, and the identifier of the peer.

3.2 Elliptic Curves

An *Elliptic curve* $E(\mathbb{F}_q)$ over a field of q elements $\mathbb{F}_q$ is the set of solutions $(x,y) \in \mathbb{F}_q$ of an equation of the form:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad (1)$$

together with an additional point at infinity, denoted as O, which represents the direction of the y-axis. Equation (1) reduces to a simpler form according to the field *characteristic* p, a prime number such that $q = p^d$ with $d \geq 1$. In the following we will consider only elliptic curves for which equation (1) reduces to:

$$y^2 = x^3 + ax + b \ . \qquad (2)$$

Such curves are related to fields for which $d = 1$. In this case $q = p$, and the number of points (i.e. the *order*) of $E(\mathbb{F}_p)$ is given by $p+1-t$, where $|t| \leq 2\sqrt{p}$. Moreover, for any such t there is an elliptic curve of equation (2) having exactly order $p+1-t$. Curve of this type must be defined over fields having a large prime p characteristic, so that the Diffie-Hellmann problem over a cyclic group in $\mathbb{F}_p$ can be sufficiently hard. In our tests (Section 6) we used the three ordinary curves P-160, P-192 and P-224; these are curves defined in the NIST standard (Brown et al., 2001) having orders in bit size of 160, 192 and 224, respectively.

4 THE eFG PROTOCOL

Accordingly to any ID-based cryptosystem, the FG protocol (Fiore and Gennaro, 2010) requires the identifiers A, B of the two communicating peers, and all other information made public by the KGC which uniquely identifies the system. In case of the elliptic version eFG of the protocol, this public information consists of:

- the elliptic curve E to be used;
- the point $P \in E$ of prime order l, with $\gcd(l, p-1) = l$, which generates the cyclic group $\mathbb{G} \subset E$ where the Diffie-Hellmann system will be considered;
- the two hash functions
$$h : \{0,1\}^* \to \mathbb{Z}_l^*, \quad H : \mathbb{G} \times \mathbb{G} \to \{0,1\}^n \ ,$$
where n is a suitable positive integer;
- the public key $P_{pub} = sP \in \mathbb{G}$ of the KGC, where $s \in \mathbb{Z}_l^*$ is the corresponding secret key, chosen randomly by the KGC.

Moreover, as result of an interaction with the KGC through a properly protected, out-of-band channel each user receives its own private key related to its identity U. In case of the eFG protocol this key is the output of the Extract algorithm 1, and consists in a couple $(R_U, s_U) \in \mathbb{G} \times \mathbb{Z}_l^*$, which represents the elliptic version of the Schnorr's signature (Schnorr, 1991) of the string U under the public key P_{pub}.

Algorithm 1: eFG Extract algorithm.

```
KGC: randomly select $r_U \in \mathbb{Z}_l^*$;
KGC: compute $R_U = r_U P$;
KGC: compute $s_U = r_U + sh(U \| R_U)$;
KGC: send message $\{R_U, s_U\}$ to U
```

Protocol's computations can be subdivided in three steps. In the first step, each user U asynchronously calculates its ephemeral private-public key couple $(t_U, T_U) \in \mathbb{Z}_l^* \times \mathbb{G}$. The second step is a communication step in which each user U sends to the other party a packet with its identifier U alongside with the two points $R_U, T_U \in \mathbb{G}$. Finally, in the third step each user U computes a couple of points $(Z_U, Z'_U) \in \mathbb{G} \times \mathbb{G}$ and derives the key as $K = H(Z_U, Z'_U)$. The computation of (Z_U, Z'_U) requires the private information of U, together with P_{pub} and the information received by U from its peer due to the previous communication step. Since each user computes the same couple (Z_U, Z'_U), the computed key K is actually the same for both users. Note that the protocol actually does not rely on message authentication, since each of the two messages exchanged at the step two is actually not signed by the sender. However, the way to derive (Z_U, Z'_U) (Alg. 2) assures that only the two parties with identities A, B can compute the key K.

It easy to show that protocol eFG is *correct*, meaning that it actually results in a key agreement between

Algorithm 2: eFG protocol.

```
A: randomly select $t_A \in \mathbb{Z}_l^*$
    compute $T_A = t_A P$
B: randomly select $t_B \in \mathbb{Z}_l^*$
    compute $T_B = t_B P$
A: send message $\{A, R_A, T_A\}$ to $B$
B: send message $\{B, R_B, T_B\}$ to $A$
A: compute $Z_A = (t_A + s_A)(T_B + R_B + h(B||R_B)P_{pub})$
            $Z'_A = t_A T_B$
            $K = H(Z_A, Z'_A)$
B: compute $Z_B = (t_B + s_B)(T_A + R_A + h(A||R_A)P_{pub})$
            $Z'_B = t_B T_A$
            $K = H(Z_B, Z'_B)$
```

users A and B. Indeed:

$$\begin{aligned} Z_A &= (t_A + s_A)(T_B + R_B + h(B||R_B)P_{pub}) \\ &= (t_A + r_A + sh(A||R_A))(t_B P + r_B P + h(B||R_B)sP) \\ &= (t_A + r_A + sh(A||R_A))((t_B + r_B + sh(B||R_B))P) \\ &= (t_A P + r_A P + sh(A||R_A)P)(t_B + s_B) \\ &= (T_A + R_A + h(A||R_A)P_{pub})(t_B + s_B) \\ &= Z_B \end{aligned} \tag{3}$$

and

$$Z'_A = t_A T_B = t_A t_B P = T_A t_B P = Z'_B$$

It can be proven that protocol eFG is secure under the *strong Diffie-Hellmann assumption*, if the hash functions h and H behaves as *random oracles* (for these notions and the security proof (Fiore and Gennaro, 2010)). For each node, the eFG protocol requires 4 point-scalar multiplications, 2 point additions, 2 hash computations. Moreover, each node has to send and receive one message, which composes of the sender identifier and two points in $\mathbb{G}$.

5 eFG IMPLEMENTATION

In this section, we discuss about software design and development on the Raspberry PI platform, in order to implement the eFG Protocol discussed in the previous section. Our design complies with modular programming, emphasizing the separation of the functionality offered by a program into independent and interchangeable modules. This way, software gains in reliability, security and reusability. Modules can be separately tested and verified, and they can be used in other software as well. In our case, some of the modules developed to implement the eFG protocol can be reused for other protocols, such as ID-based group key agreement protocols.

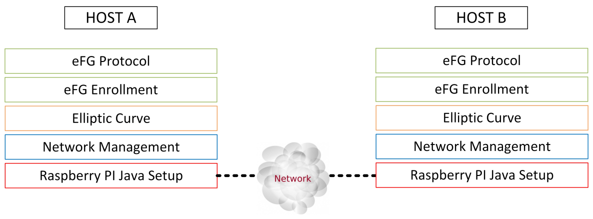

Figure 1: Software modules involved in the key agreement process performed through eFG protocol.

Our implementation is in the Java programming language. Java allows for portability and is of widespread use in mobile platforms. The Raspberry PI device (Monk, 2014) includes two Java ARM technologies for speeding up performance, *ARM Floating Point and Jazelle* (”ARM”, 2014).

ARM Floating Point technology provides hardware support for floating point operations in half and single double-precision floating point arithmetic. Instead, ARM Jazelle technology is a combined hardware-software solution to increase the performance of the Java Virtual Machine (JVM).

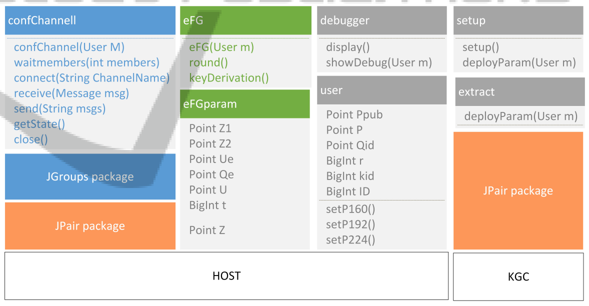

Figure 2: The Java classes provided in the eFG protocol implementation. The three columns on the left are related to the host device, whilst the lost column on the right concerns the key generation center.

Figure 2 shows the Java class related to the implementation of the eFG protocol. The implementation includes packages JPair (Dong, 2010) and JGroups (JGroups, 2014), that will be described in the next sections. Class *user* stores all public parameters, plus user’s private and public keys that are generated by the KGC through Algorithm 1. The *confChannel* class is devoted to the configuration of the communication channel: it includes methods to open/close the channel and to send/receive data. Class *eFG* implements the eFG protocol, using class *eFGparam* for storing local parameters needed for computations. Finally, the key generation center (KGC) implementation includes classes Setup and Extract that are used for the generation of the public system parameters and the user public/private key couple, respectively. In the sequel, we give a brief description for each of the implemented module and its functions.

5.1 Raspberry PI Java Setup

The *Raspberry PI Java Setup* module relates to Raspberry PI operating system configuration, which in our case is the *Raspian* OS (”Raspian”, 2014). In order to run our application, written in the Java programming language, we have installed *Java SE Embedded*, release 8. This release introduces a new concept called Compact Profiles, which enables reduced memory footprint for applications that do not require the entire Java platform. At the time being, the three profiles *compact1*, *compact2* and *compact3* are supported, each being a superset of the APIs in the previous profile. The Compact Profiles feature is very useful for small devices, because it allows for a smaller size o the Java code. A JRE can be configured with a compact profile, reducing its footprint for deployment along with a compact profile application. We developed our application with NetBeans 8, which supports compact profiles and allows the deployment of a Java application directly on an embedded platform such as the Raspberry PI. Moreover, we installed in Raspian the P4J library (”Pi4j”, 2014), that allows to control the GPIO of a Raspberry PI device through Java APIs.

5.2 Network Management

The *Network Manager module* is devoted to user synchronization, data transmission, and error handling. We implemented it using the JGroups toolkit (JGroups, 2014). For any given TCP or UDP communication channel, JGroups offer services for joining and leaving, membership detection and notification, detection and removal of crashed users, and the sending and receiving of unicast and multicast messages. The most powerful feature of JGroups is its flexible protocol stack, which allows developers to adapt it for the best matching of application requirements with network characteristics. For protocol eFG we have chosen an UDP asynchronous communication channel for better performance.

The *Message Serialization Manager* module is dedicated to data serialization. Hosts A and B need a precise order to build packets for delivery. For example, referring to first data transaction of protocol eFG, host A has to make a UDP packet which composes of the three field A, R_A, T_A (Algorithm 2). For allowing recipient B to process the packet correctly, such fields must be properly identified and must be put in the right order. This serialization is implemented by a Java class that splits and merges data using special characters as separators.

5.3 Elliptic Curve (EC) Module

The *EC module* implements all mathematical operations required by elliptic curve cryptography. These include finite fields and elliptic curve generation, integer arithmetic over elliptic curves and hash function computation. The API offered by this module is used by the higher level modules Enrollment and Protocol, discussed in the next section. The EC module is implemented using an extended version of the JPair library (Dong, 2010) which was completed by adding the standard NIST curves P-160, P-192, P-224 described in Section 3.2.

5.4 Enrollment Module

Enrollment is the step first required in ID-based cryptosystems. It corresponds to the registration of users to a specific crypto-service offered by a key generation center (KGC), and is realized through the Setup and Extract classes. The Enrollment is executed one-time and simulates the workaround of a KGC deploying off-line public parameters on devices through a secure, out-of-band communication channel.

The scope of the Setup class is the setting of the KGC crypto-parameters for the intended service. The operation performed by the Setup class are the following:

- Selection of an elliptic curve $E(\mathbb{F}_p)$ and of a point $P \in E$ having as prime order the order of E;
- Choice of a pseudo-random integer $s \in \mathbb{Z}_l^*$, and computation of $P_{pub} = sP$. The private/public key couple of the KGC is (s, P_{pub});
- Instantiation of the two functions:

$$H : \mathbb{G} \times \mathbb{G} \to Z_l^*, \quad h : \{0,1\}^* \to Z_l^*$$

The scope of the Extract class is the generation of the private keys for users A and B. As we told in Section 4 these private keys are the elliptic version of Schnorr's signatures of strings A and B under public P_{pub}. This class corresponds to Algorithm 1 and, given the output returned by class Setup, it does the following:

- Computation of digests $H(A)$, $H(B)$, $h(A)$ and $h(B)$;
- Choice of the random values $r_A, r_B \to Z_l^*$;
- Computation of the points
  $R_A = r_A P$ and $R_B = r_B P$;
- Computation of the two modulo l integers
  $s_A = r_A + sh(A||R_A)$;
  $s_B = r_B + sh(B||R_B)$;

The private keys of users A and B are given by (R_A, s_A) and (R_B, s_B), respectively.

Module *Protocol* implements a Java class for the eFG protocol. This Java class includes all the methods required by Algorithm 2 and the eFGparam subclass, which is used to store local variables.

6 PERFORMANCE

The Raspberry PI (Monk, 2014) is an embedded device developed in UK by the Raspberry PI foundation. Its design is based around a Broadcom BCM2835 SoC, which includes an ARM1176JZF-S 700 MHz processor, 512 Megabytes of RAM and an SD card for booting and long-term storage. A set of tests were performed to measure eFG protocol performances. Each computed value is the average of 1000 executions, in order to take account of time outliers due to JVM memory management and HotSpot optimizations ("Oracle", 2014). Table 1 shows the elliptic curves used in tests, along with the order (in bit size) of the group $\mathbb{G}$ considered on the curve, and the corresponding order of a standard group in which the discrete logarithm problem (DLP) has a difficulty comparable to that of the DLP in $\mathbb{G}$.

Table 1: The elliptic curves used in tests are the NIST standards P-160, P-192 and P-224. These are ordinary curves whose equation differs only for the known term b_i (Brown et al., 2001).

Name	Equation	Order of $\mathbb{G}$	DLP Difficulty
P-160	$y^2 = x^3 - 3x + b_1$	160-bit	1024-bit
P-192	$y^2 = x^3 - 3x + b_2$	192-bit	2048-bit
P-224	$y^2 = x^3 - 3x + b_3$	224-bit	4096-bit

Overall, our Java code footprint with all the cryptographic material needed to perform the tests required just 106 KB of memory versus a the maximum of 100 MB supported by the Jazelle architecture for each device. Table 2 reports computing times (ms) of the Setup and Extract algorithms for each of the elliptic curves of Table 1. In our case, the computing time of the Extract algorithm encompasses the creation of the private key for both users A and B.

Table 2: Computation times (ms) for the Setup and Extract algorithms.

Algorithm	P-160	P-192	P-224
Setup	101	102	104
Extract	2407	2497	2499

Table 3 reports the computing times (ms) required by each user for the execution of the three different steps in protocol eFG (Section 4). Each value is the average of the times spent by A and B for each step. The computing time for the communication step (Step 2) was measured using two different network interface cards, wired LAN (IEEE 802.3u Ethernet) and WiFi (IEEE 802.11g).

Table 3: Computing time (ms) for each step of the eFG protocol.

Curve	Step 1	Step 2 (wired LAN)	Step 2 (WiFi)	Step 3
Curve P-160	482	34	90	1956
Curve P-192	584	42	111	2174
Curve P-224	656	48	132	2296

Finally, Table 4 reports the total computing time (ms) required on average by each of the two communicating parties to compute the shared key. As for the previous case, each value is the average of the times spent by users A and B. Notice that these values are greater than the totals obtained by summing the values related to each step, since they include time spent by parties in synchronization. As before, results are related to each of the curves given in Table 1 and consider both the cases of communication over a wired LAN and through WiFi.

Table 4: Total computing time (ms) of the eFG protocol in case of wired LAN and WiFi communications.

Curve	Total ct wired LAN	Total ct WiFi
Curve P-160	2002	2060
Curve P-192	2250	2302
Curve P-224	2384	2471

All the above results clearly show that using the eFG protocol for key agreement using low cost, resource constrained embedded devices, such as the Raspberry PI is viable to the point that it can be used for real world applications.

7 CONCLUSIONS AND FUTURE WORK

Implementations of advanced key-agreement protocols are scarce, especially in case of embedded devices. However, the security of networking through embedded devices can greatly benefit from the implementation of advanced key-agreement protocols. This paper describes an implementation for embedded devices of a two-party ID-based key agreement protocol. We considered a variant of a fast, one-round protocol introduced in (Fiore and Gennaro, 2010), named eFG protocol. The eFG protocol makes use of identity-based cryptography and is implemented on elliptic curves, in order to achieve good performances both in computing time and memory allocation size. These goals were confirmed by experimental tests, which show that eFG is viable for enabling secure communications among resource-constrained, embedded devices such as the Raspberry PIs.

As a next step, we are going to test a porting of our implementation on wireless sensor network equipped with low-pan devices (IEEE 802.15.4). More future work includes the implementation of a related set of protocols to manage the case of authenticated key agreement for dynamic groups, plus some new elliptic curves and related bilinear parings. Moreover, we are working on a real-word scenario testbed in which such protocols operate to secure a network of *Raspberry PI* devices deployed for environmental monitoring in hostile environments.

REFERENCES

Akinyele, J. A., Green, M. D., and Rubin, A. D. (2011). Charm: A framework for rapidly prototyping cryptosystems. Cryptology ePrint Archive, Report 2011/617. http://eprint.iacr.org/.

”ARM” (2014). Arm jazelle. http://www.arm.com/products/processors/technologies/jazelle.php.

Boyd, C. and Mathuria, A. (2003). *Protocols for authentication and key establishment*. Springer.

Brown, M., Hankerson, D., López, J., and Menezes, A. (2001). *Software implementation of the NIST elliptic curves over prime fields*. Springer.

Chen, L., Cheng, Z., and Smart, N. P. (2007). Identity-based key agreement protocols from pairings. *International Journal of Information Security*, 6(4):213–241.

De Caro, A. and Iovino, V. (2011). jpbc: Java pairing based cryptography. In *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pages 850–855, Kerkyra, Corfu, Greece, June 28 - July 1.

De Caro, A. and Iovino, V. (2011). jpbc: Java pairing based cryptography. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 850–855. IEEE.

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654.

Dong, C. (2010). Jpair: A quick introduction. https://personal.cis.strath.ac.uk/changyu.dong/jpair/intro.html.

Fiore, D. and Gennaro, R. (2010). Identity-based key exchange protocols without pairings. In *Transactions on Computational Science X*, pages 42–77. Springer.

JGroups (2014). Jgroups toolkit. http://www.jgroups.org/.

Joux, A. (2000). A one round protocol for tripartite diffie–hellman. In *Algorithmic number theory*, pages 385–393. Springer.

Law, L., Menezes, A., Qu, M., Solinas, J., and Vanstone, S. (2003). An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134.

Lynn, B. (2007). *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford University.

McCullagh, N. and Barreto, P. S. (2005). A new two-party identity-based authenticated key agreement. In *Topics in Cryptology–CT-RSA 2005*, pages 262–274. Springer.

Menezes, A. J., Van Oorschot, P. C., and Vanstone, S. A. (2010). *Handbook of applied cryptography*. CRC press.

Miller, V. S. (2004). The weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261.

Monk, S. (2014). *Raspberry Pi Cookbook*. O'Reilly Media, Inc., 1st edition.

Okamoto, T., Tso, R., and Okamoto, E. (2005). One-way and two-party authenticated id-based key agreement protocols using pairing. In *Modeling Decisions for Artificial Intelligence*, pages 122–133. Springer.

”Oracle” (2014). Java se hotspot at a glance. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html.

”Pi4j” (2014). Pi4j project. http://pi4j.com/.

”Raspian” (2014). Raspbian operating system. http://www.raspbian.org/.

Schnorr, C.-P. (1991). Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174.

Shamir, A. (1985). Identity-based cryptosystems and signature schemes. In *Advances in cryptology*, pages 47–53. Springer.