©2001 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.”

A fast elliptic curve based key agreement protocol-on-chip (PoC) for securing networked embedded systems

Roshan Duraisamy and Zoran Salcic
Department of Electrical and Computer Engineering, University of Auckland, New Zealand
{rdur007, z.salcic}@auckland.ac.nz

Miguel Morales-Sandoval and Claudia Feregrino-Uribe
Computer Science Department, National Institute for Astrophysics, Optics and Electronics
México
{mmorales, cferegrino}@inaoep.mx

Abstract

Securing communication sessions between networked embedded systems is a major challenge that needs to be addressed as an increasing number of such systems become Internet-enabled. Securely and quickly establishing a session key between communicating nodes in a network requires authentication of node identities. In this paper, we propose a simple elliptic-curve based key negotiation protocol suitable for fully hardware implementation as a protocol on chip (PoC). The protocol uses the elliptic curve variants of both the Diffie Hellmann exchange and the Digital Signature Algorithm. Timing results demonstrate that an end-to-end protocol run can be performed in as little as 28ms on a 25MHz clock, which is several times faster than previous microprocessor-based implementations of similar protocols. The results indicate that session keys can thereby have shorter lifetimes, since there is little computational overhead in re-generating them, thus enhancing overall security of the networked embedded systems.

1. Introduction

Securing communication sessions between networked embedded systems is a major challenge that needs to be addressed as an increasing number of such systems become Internet-enabled. Embedded systems, e.g. thermostat controllers, wireless sensors, and video surveillance units may be installed in a single network, and other agents such as PDAs, mobile phones and computers may need to connect to these networks in order to monitor or control them remotely. Any compromise of the security of communications between such systems could have devastating, and sometimes even fatal, consequences. Cryptographic operations provide one method of securing data transmitted between these systems, the main goal being the secure establishment of session keys for encryption and decryption of all exchanged data.

In the recent years, a number of solutions have been proposed to achieve this goal [1]-[4]. Majority of these solutions rely on the availability of general-purpose microprocessors and sufficient memory to perform cryptographic operations. However, some embedded systems may be unable to provide general purpose microprocessors that can establish session keys for communication or those processors may be unable to provide that service fast enough. For that reason, specialised cryptoprocessors are sometimes added, but they typically support multiple cryptographic algorithms, which may be unnecessary in many embedded applications. However, there is a growing need for quick, secure negotiation of session keys for use in communication between networked embedded systems.

Elliptic curve cryptography (ECC) has become the public key scheme of choice, primarily because of the relatively strong security it provides for a much smaller key size as opposed to similarly strong schemes like RSA [5]. For instance, 1024-bit RSA is sometimes considered equivalent to 139-bit ECC [1]. This lower number of bits for the same level of security means that hardware implementations of ECC potentially require fewer resources and are computationally less expensive for the same level of security provided by other cryptographic schemes. Consequently, ECC is expected to provide an optimal solution for generating encryption keys for securing networked embedded systems.

The Diffie Hellmann algorithm [6], originally proposed in the 1970s, offers a simple, flexible method of securely establishing a single shared session key through an exchange of various parameters which a third party cannot use to determine the final key. A variant of this algorithm, the Elliptic Curve Diffie Hellmann (ECDH) has been developed

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

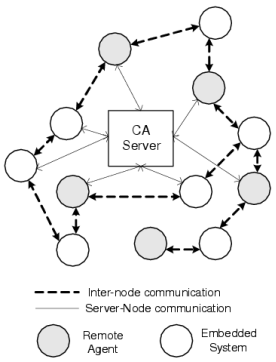

Figure 1. A network of nodes authenticated by a Certificate Authority (CA) server

for elliptic curves. However, the ECDH, as well as the original Diffie Hellmann algorithm, is susceptible to man-in-the-middle attacks, whereby an intervening party can masquerade as an honest party and intercept communications between two nodes. To counter this attack, honest parties need to be able to authenticate each other and proceed with the Diffie Hellmann protocol only after successful authentication.

The Elliptic Curve Digital Signature Algorithm (ECDSA), a variant of the Digital Signature Algorithm (DSA), can be used by communicating parties to sign respective pieces of data using their private key [7]. At authentication stage, a recipient can use a sender's public key to verify that the signature generated by the sender was in fact signed using the sender's private key. When combined with the ECDH, therefore, secure authentication protocols can be developed. Thus, to address the challenge of securely establishing session keys between various nodes in a network of embedded systems and remote connection agents, fast implementations of ECDH and ECDSA algorithms are needed.

This paper presents a simple key agreement protocol consisting of both algorithms implemented on a single chip that can reside at the network interfaces of communicating embedded nodes. It is customized and implemented as a protocol on chip (PoC) and can also be used in remote agent nodes that need to connect to the network and access data on one or more embedded system nodes Figure 1. The implementation of the PoC re-uses a fast scalar elliptic curve multiplier that had earlier been developed as part of an ECC-based encryption and compression module [9]. The implementation assumes a 163-bit binary field ($F_{2^{163}}$) for elliptic curve computations and has been shown to provide fast establishment of session keys between two parties barring only network constraints.

The organization of this paper is as follows. Section 2 briefly reviews elliptic curve cryptography, including the ECDH and ECDSA algorithms, which are essential to the proposed key agreement protocol. Section 3 summarizes related research in the area of secure protocols for low-power devices such as wireless sensors, and hardware implementations of elliptic curve cryptography. In Section 4 we describe the key agreement protocol that has been implemented on chip, including details of authentication and key exchange. The actual functional modules on the chip are described in Section 5. Section 6 documents the results of synthesis and simulation on an FPGA prototype. Finally, Section 7 concludes this paper with recommendations for further research.

2. Review of Elliptic Curve Cryptography (ECC)

Elliptic curves are generally represented by one of the following equations [7]:

$$y^2 + xy = x^3 + a_2x^2 + a_6 \tag{1}$$

$$y^2 + y = x^3 + a_4x + a_6 \tag{2}$$

The constants can either be polynomial or normal basis numbers. In this paper, a polynomial basis implementation over the curve defined in equation 1 was assumed. While normal basis arithmetic implementations often lead to fast and efficient implementations in hardware, field inversion routines are known to utilize far more resources when implemented using normal basis in hardware [8].

Elliptic curve points are identified by their coordinates $P(x, y)$. Addition and doubling of points are essential to elliptic curve arithmetic. Addition of two different points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ on a curve yields a third point $P(x_3, y_3) = P(x_1, y_1) + P(x_2, y_2)$ as defined by the following equations:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right) + x_1 + x_2 + a_2 \tag{3}$$

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 + x_3) - y_1 \tag{4}$$

Doubling of a point on the curve to yield $P(x_3, y_3) = 2P(x_1, y_1)$ is achieved using the following equations:

$$x_3 = \left(x_1 + \frac{y_1}{x_1}\right)^2 + \left(x_1 + \frac{y_1}{x_1}\right) + a_2 \tag{5}$$

$$y_3 = x_3\left(x_1 + \frac{y_1}{x_1} + 1\right) + x_1^2 \tag{6}$$

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

where $P_1(x_1, y_1)$ is not the point at infinity.

Scalar-point multiplication over an elliptic curve refers to the multiplication of a point on a curve by a scalar value to yield another point on the curve. Scalar multiplication involves a combination of doubling and adding points starting from the original point on the curve until the multiplication is complete.

Encryption & Decryption: An elliptic curve is defined over the domain parameters $(a_2, a_6, P(x, y), n)$ where $P(x, y)$ is a base point chosen on the curve and n is the prime number order of the point. In ECC, communicating parties use a public-private key pair, where the private key is a scalar secret k, and the public key is the point $K = kP(x, y)$. When sending a message, an honest party A obtains a random value r and requests B's public point $B = bP$. A then computes the random point $R = rP$ and the secret key $S = rB$. The x-component of S is used to generate a key by applying a key derivation function, and the result is XORed with the message to be encrypted. The message is sent to B along with the random point R. B decrypts the message by computing the same shared secret $S = bR$.

Elliptic Curve Diffie Hellmann (ECDH): The Diffie Hellmann protocol can be used to establish a shared secret between two communicating parties. The elliptic curve version assumes a public elliptic curve, where the domain parameters $(a_2, a_6, P(x, y), n)$ are known to an attacker. Two honest parties A and B generate their respective secrets w_A and w_B and compute public keys accordingly:

$$A : W_A = w_A P(x, y) \tag{7}$$

$$B : W_B = w_B P(x, y) \tag{8}$$

They exchange public keys and can compute the same shared secret, from which a bit mask can be generated using a key derivation function.

$$A : S_{AB} = w_A W_B = w_A w_B P(x, y) = w_B w_A P(x, y) \tag{9}$$

$$B : S_{BA} = w_B W_A = w_B w_A P(x, y) = w_A w_B P(x, y) \tag{10}$$

An attacker only sees the values W_A and W_B but cannot compute the secret key $S_{AB} = S_{BA}$ because of the intractability of the elliptic curve discrete logarithm problem.

Elliptic Curve Digital Signature Algorithm (ECDSA): The ECDSA can be used to verify the authenticity of a message transferred between two recipients. In this algorithm, the signer's public key is used by the recipient to verify that the signer in possession of the corresponding secret key did send the message transferred across [7]. Signature generation for a party A with public-private key pair (W_A, w_A) entails the following steps using the elliptic curve domain parameters $(a_2, a_6, P(x, y), n)$:

1. Generate a random value r modulo n and compute the random point $R(x_R, y_R) = rP(x, y)$
2. The first component of the signature is: $s_1 = x_R(\mathbf{mod} n)$
3. Compute the hash of the message: $h = \mathbf{Hash}(message)$
4. The second component of the signature is: $s_2 = \left(\frac{h + s_1 w_A}{r}\right)(\mathbf{mod} n)$

The signatures are transferred across along with the message to party B, which is already in possession of A's public key . B can then verify that the message was indeed sent by A, using as follows:

1. Compute the hash of the message: $h' = \mathbf{Hash}(message)$.
2. Compute $u = \frac{h'}{s_2}(\mathbf{mod} n)$ and $v = \frac{s_1}{s_2}(\mathbf{mod} n)$.
3. Compute the point on the elliptic curve: $N(x_N, y_N) = uP + vW_A$.
4. If $x_N(\mathbf{mod} n) = s_1$ then the signature has been verified and the message has been authenticated.

3. Related Work

In the last few years, a number of authentication protocols have been devised and implemented for use in embedded sensor networks. [1] is an implementation of a protocol on a 32-bit ARM7TDMI RISC microprocessor that uses the ECDSA as its basis for authentication, targeting mobile phones, handheld devices and smartcards. The authentication protocol involves a server, a terminal user trying to connect to the network, and a certificate authority that enables the user and the server to authenticate one another and establish a session key. The implementation is a complete software library of ECC functions that can operate on a wide range of elliptic curves.

In [2], an end-to-end wireless security protocol using ECDH was implemented in software on a Chipcon CC1010 (based on 8-bit 8051 architecture), targeting wireless sensor networks. The aim of this implementation was to establish secure end-to-end connections between a sensor and a base

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE
IEEE COMPUTER SOCIETY

station without the need for cryptographic co-processors. Unfortunately, because there is no way for communicating parties to authenticate one another in this scenario, the protocol implemented is unable to withstand the man-in-the-middle attack.

Another recent advancement in securing communications between a sensor and a security manager is [3], which is targeted primarily at self-organizing networks. This is a hybrid authentication scheme that moves the more complex cryptographic functions to the server, while at the same time maintaining de-centralized online key management. Communicating nodes are authenticated using public-key certificates that are generated offline before nodes join the network. The ECC operations were implemented on a Mitsubishi M16C microprocessor.

TinyPK [4] is an RSA based public key protocol that allows authentication and key agreement between a sensor network and a third party as well as between two sensor networks. Here again, a certificate authority (CA) is used to sign a third party's public key, but TinyPK does not use certificates as it assumes that nodes have little processing power to deal with certificates. Because of this, there is no way to deal with a third party private key that has been compromised. Authentication is performed at a sensor node by verifying an external party's public key. After successful authentication, a sensor node encrypts its session key as well as a nonce (a timestamp) using the third party's public key and sends it back to the third party. In this manner, a secure session key is established for communication. The Diffie Hellmann protocol was also implemented as part of TinyPK, as the RSA public-private operations were found to be too slow. The TinyPK was implemented on UC Berkeley's MICA 2 platform in the TinyOS development environment.

The research cited focuses on implementing the cryptographic protocols on general-purpose microprocessors. This implies that network nodes each have to provide an additional general purpose microprocessor as well as the additional memory required to perform the cryptographic operations. In addition, the end-to-end session times of these protocols are typically in the range of hundreds of milliseconds to seconds, which may be adequate for target applications involving human user access [1], [2]. However, where speed of key agreement is particularly an issue, e.g. where remote agents need to control embedded nodes in real-time, a hardware implementation clearly offers considerable advantages.

One such hardware implementation of an elliptic curve cryptosystem was developed in [10], which is able to generate 4000 digital signatures per second on a 233-bit binary field Koblitz curve. The complete ECDSA algorithm is implemented, and the entire chip can operate at up to 100MHz. Another very recent implementation is described in [9], which is the first hardware architecture that combines elliptic curve cryptography and lossless data compression. The high speed offered by these hardware implementations makes them very attractive for implementing entire protocols on chip, thereby also removing the need for an additional microprocessor. Hence, our approach was to use an existing hardware based elliptic cryptosystem and to develop a simple secure protocol on a single chip that can be configured for use in networked embedded systems and remote agents.

4. Protocol Design

In the proposed protocol, we address the challenge of authenticating communicating parties using digital certificates. Certificates are generated by a trusted certificate authority (CA) and comprise of node public keys and node identities. We use ECDSA to authenticate public keys generated from private keys.

The certification generation authority is assumed to be a trusted and secured server that maintains a database of the identities of honest nodes and their long-term public keys. (Depending on the network environment, a powerful computing CA server may not always be available.) For test purposes, the server was implemented using the C code for ECDSA and ECDH originally made available in [7]. The CA server code accesses a database of test node identities and their corresponding long-term public keys. The CA server makes these public keys available to all trusted nodes, i.e. nodes that have been previously registered with the CA server. All nodes are bound to this single CA server in the network via the CA server certificate which comprises of a unique identifier, the CA identity S_{srv}, and the CA's long-term public key $W_{srv}(x, y)$. In addition, each node keeps a counter that tracks the number of sessions that the node and the CA server have previously established. A corresponding counter is maintained for each node in the CA server's database. The counter value forms a part of the ECDSA message when server-node communications are established in order to protect against replay attacks. The counter value is not transferred across but is kept in memory at both ends, and is incremented simultaneously upon termination of the session. If the counter values no longer synchronize, a compromise, or at least an error, can certainly be assumed at one end, and a physical reset of the PoC will become necessary. Until then, that node identity will be treated as unauthorized on the server. In case of compromise of the CA server private key, none of the nodes will successfully communicate with the CA server since the counter will not be synchronized, and a new CA binding must be issued.

Inter-node communication is coordinated by the CA server, whereby the server establishes two separate Diffie-Hellmann sessions between honest parties A and B, and separately deploys the public keys and signatures of A and

IEEE COMPUTER SOCIETY

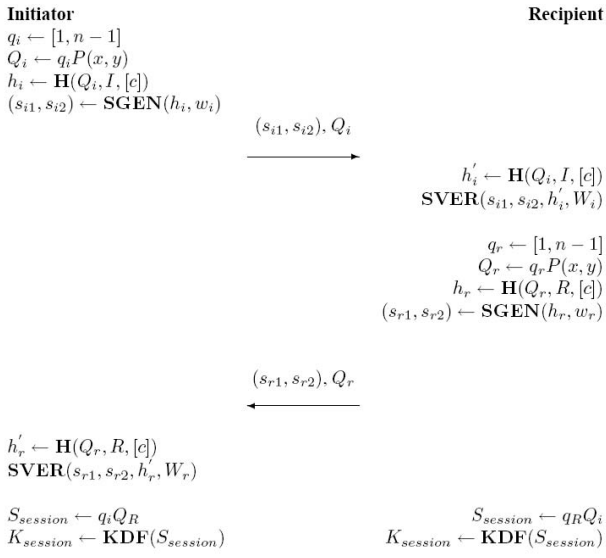

Figure 2. Simple authenticated session key establishment protocol.

B to one another, i.e. A receives the certificate of B and vice-versa. The server can then designate A or B as the initiator of all communications between them. Figure 2 outlines the protocol for both CA server-node and inter-node communication. The initiator first of all generates a random temporary secret q_i from which a temporary Diffie Hellmann public key Q_i is derived. This public key forms part of the message to be signed during the signature generation phase. The identifier I of the initiator is concatenated with the message. If the initiator is either the CA or a node requesting to communicate with the CA, then the node-specific server counter c also forms part of the ECDSA message. The ephemeral public key Q_i together with the rest of the ECDSA message is then signed by the signature generator (SGEN) module using the initiator's long-term private key w_i and sent across along with the signatures (s_{i1}, s_{i2}) and a token message to initiate communication. The recipient of the initiator's signature and temporary public key uses the initiator's long-term public key W_i to verify that the temporary public key Q_i has indeed been generated by the initiator. The message to be hashed as part of the ECDSA verification step (SVER) is re-calculated using the temporary public key Q_i and the initiator's identity I. If the initiator is the CA or a node that requested to communicate with the CA, then the node-specific counter c is again included as part of the re-calculated message, which is then hashed and processed as part of the verification stage.

When the initiator's temporary Diffie-Hellmann key has been authenticated, the recipient generates its own ephemeral secret q_r and corresponding ephemeral Diffie-Hellmann key Q_r and signs it using its long-term private key w_r. This ephemeral key is sent back to the initiator which then verifies it. After successful authentication at both ends, the communicating parties establish the same shared secret $S_{session}$, which is then passed through a key derivation function to obtain an XOR mask $K_{session}$ that can be used for direct encryption and decryption of data as described in [7].

When the session key is established, communicating parties can exchange encrypted data. Either communicating party reserves the right to terminate the connection, in which case an encrypted terminate op-code is sent across and acknowledged, and the session key is automatically made invalid. The protocol interface we have developed allows for the embedded application to terminate the session if required. Session keys typically also have a lifetime that can be specified as part of the protocol configuration.

When a key expires, or when a session is terminated, the node-specific counter is incremented at both ends as long as one of the communicating parties is the CA server. This ensures against replay attacks on the server-node communication. As long as the counter is synchronized at both ends, successful session key establishment is guaranteed. This also ensures that even if either the server private key or the node private key is compromised, previous session keys will not be recovered from transcripts by an adversary. This guarantees full forward secrecy, as opposed to the protocol developed in [3], which only offers half forward secrecy in such a situation. If it happens that the counters are no longer synchronized, a physical reset of the PoC will be required, as mentioned earlier.

Inter-node communication, on the other hand, does not use node-specific counters because of the amount of memory required to store counters for every other party that a node may be in communication with. In this case, the safety against replay attacks and compromise of previous keys can be established if the CA server is configured to command nodes to re-generate their long-term public-private key pair upon expiry of their certificates. The PoC also supports re-generation of the long-term key pair. When such a request is to occur, the server sends a plaintext op-code message, which is signed using the server's private key. The recipient verifies the op-code and the signature, sets up a temporary session key, and invokes a pseudo-random number generator to generate a new value, which is then cryptographically hashed to yield the new private key. The result is multiplied by the elliptic curve base point, and the computed point is saved as the new long-term public key. This public key is encrypted using the temporary session key and sent back to the server, which then updates its database with the new long-term public key of the node.

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE
IEEE COMPUTER SOCIETY

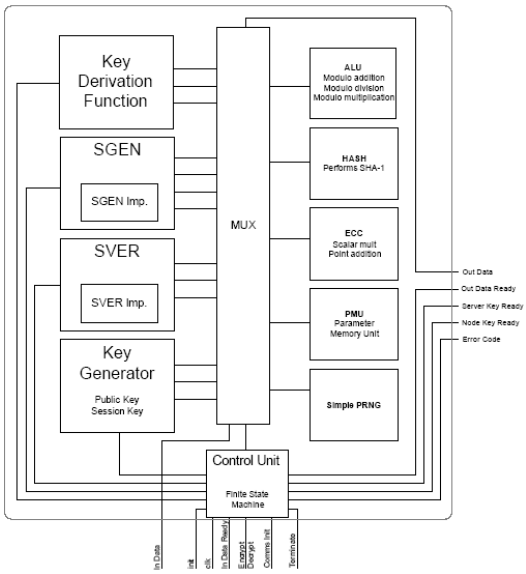

Figure 3. Chip Layout

5. PoC Architecture

The elliptic scalar multiplier and point adder form the core of the elliptic curve cryptoprocessor. As indicated before, we have re-used a very fast cryptoprocessor that had previously been developed in conjunction with a lossless data compression engine [9]. The performance figures make such a cryptoprocessor suitable for a hardware implementation of the key agreement protocol. The block diagram of Figure 3 describes the top-level architecture of the system.

The top-level PoC functional units, which are described in more detail in the remaining part of this section, include the Signature Generation and Verification Units, Public/Session Key Generation Unit, Key Derivation Function Unit, Parameter Memory Unit, and Pseudo-Random Number Generator. These functional units require service from low level cryptographic processing elements and arithmetic logic unit as needed. The entire protocol is driven by a hierarchical finite state machine (HFSM). The top-level FSM in the PoC control unit initiates the lower level FSMs in each unit accordingly. Some of these units interact with the lower-level elliptic curve scalar multiplier unit, which also functions as an elliptic point adder depending on an input operation signal. Another lower-level unit is the big integer arithmetic and logic unit (ALU), which processes 164-bit signed integers and is used by both the ECDSA signature generation and signature verification modules.

Parameter Memory Unit: The parameter memory unit (PMU) is a RAM block that stores all the node configuration data, such as the node's identifier, long- and short-term secrets, long- and short-term public keys, and server bindings (i.e. server identifier and long-term public keys). The only part of the PMU that is public to the world is the incoming address of the signature and public key of an external party that requests communication with the node. This part of the PMU is considered "volatile", i.e. it holds no long-term data. In addition, it can only be updated when there is no communication session in progress, i.e. the previous session has expired or been terminated. Any request to update any other part of the memory unit in the PoC, whether anonymously (while no session is in progress) or even during authenticated inter-node communication is denied. The CA server, however, has the ability to update the server binding information, as well as to instruct the PoC to regenerate its public-private key pair once a session has been set up between the CA server and the node. All such commands can only be processed once a session key has been established with the CA server. This restriction ensures that the PMU is kept relatively secure, and that a session key establishment routine cannot be interfered with by a network attacker.

Signature Generation and Verification Units: Each of these units comprises of two hierarchical FSMs. The lower-level FSM of the signature generator directly computes the signature from a set of input parameters. The higher-level FSM provides the interface to the PMU and sets up the input parameters to the lower-level. It also provides the interface to the ALU, the ECC multiplier-adder module, and the hashing module. The hashing module is an embedded implementation of the SHA-1 algorithm. This hierarchical structure allows for easy re-configuration of the PMU without affecting the lower-level. The same applies for the signature verification unit, SVER. The big integer ALU, which is used by these units, performs 164-bit modulo addition, modulo division and modulo multiplication, which are the main operations required for ECDSA.

Public/Session Key Generation Unit: This unit generates either a temporary Diffie Hellmann public key, which is then signed by the PoC, or the final shared session key depending on the stage of the protocol. The core operation is the elliptic curve point-scalar multiplication.

Key Derivation Function (KDF) Unit: When a shared session key is generated, it needs to be fed into a key derivation function module that can provide a secret key mask for use in encryption. This key derivation function [7] follows the draft specification of IEEE P1363 and computes a recursive SHA-1 hash of the session key generated beforehand concatenated with a number of the node parameters (e.g. node identifier, identifier of the other party, etc.) to yield the secret key mask. In addition, the KDF unit performs encryption and decryption once the session key mask is generated. This key mask is saved in a 160-bit register,

IEEE COMPUTER SOCIETY

and is not exposed to the top-level PoC interface. Therefore the top-level FSM of the PoC has no access to the final key mask generated or to the session key for that matter. These values are not saved in the parameter memory unit either, thus further ensuring their safety. The KDF unit performs encryption and decryption of incoming 32-bit data to the PoC when the encrypt/decrypt signal goes high. Encryption involves a simple XOR operation between the data and the key mask.

Pseudo-Random Number Generator (PRNG): The generation of cryptographically strong random numbers was not the focus of this research, and so a simple linear shift feedback register, using a random seed, is used as the PRNG in this module. This level of simplicity was sufficient to test proof-of-concept of the PoC operation, although in a real-world deployment, a random number generator with a higher level of unpredictability is required, since it is used to generate an ephemeral secret for ECDH and a random number for digital signature generation. All random numbers generated for signature generation are performed modulo n, where n is the order of the base point of the elliptic curve.

6. Implementation Results

Table 1 displays the results of synthesis of the PoC design on an Altera Stratix II FPGA device. The PoC successfully synthesizes at a 163-bit binary field size. The PMU, which stores partly pre-loaded node configuration data, only ever uses 38 192-bit values for a single protocol run. However, to support computing multiple session keys for simultaneous communication with multiple parties, the memory unit provides 256 192-bit locations implemented in FPGA internal memory blocks. An 80-bit counter value is used for synchronizing communication sessions, and the node and server identities each span 192 bits.

Table 1. PoC FPGA synthesis results

Device	Stratix II EP2S90F1020C3
Total pins	81/759 (10%)
Total memory bits	49,152/4,520,448 (1%)
Total ALUTs	40,234/72,768 (55%)
Total registers (FFs)	20,236

The timing results for the various modules, as well as for the entire end-to-end protocol operation that includes initiation, authentication and key generation, are shown in Table 2. For prototype verification a 25MHz clock was chosen to drive the operation. An end-to-end session protocol only requires one exchange (two transactions) of ephemeral ECDH data together with the generated signature, thus keeping the communications overhead to a minimum. The cryptographic operations are evenly balanced at both ends of the protocol, since each party has to perform ECDH and ECDSA signature generation and verification routines. An end-to-end run of the protocol takes only 28 ms to execute at 25MHz.

Table 2. Timing results for sub-modules (clocked at 25MHz)

Unit	Time (ms)
End to End Protocol	28
Signature Generation	3.9
Signature Verification	5.5
Public Key Generation	5.3
Session Key Generation	2.1
ECC Scalar Multiplication	2.1
ECC Point Addition	0.012
ALU Modulo Multiplication	1.1
ALU Modulo Division	0.64

In Table 3 end-to-end protocol session times and other protocol-related features are compared across a range of implementations developed in recent years. A direct comparison between these systems is not possible, since the five cited implementations do not possess identical levels of security. However, the results in this table give an indication of relative performance and illustrate that a full hardware implementation can yield very fast key agreement. Thus, with the PoC, session keys can have much shorter lifetimes, since they can be re-generated quickly and more often if necessary without detriment to application performance. Shorter key lifetimes greatly enhance overall security of the system, since less transaction data is available per session key in transcripts of any session that a malicious third party can use to decrypt the data transmitted.

7. Concluding Remarks

Networked embedded devices often operate in insecure environments, where the data transferred between them is visible to a third party. Consequently, cryptographic key agreement protocols are needed to safeguard the transmitted data. In this paper, we have proposed a simple elliptic curve based protocol that uses ECDSA to authenticate communicating parties and ECDH to establish the session key. Certificates are issued by a special CA server, and nodes (embedded devices and any remote agents) can initiate communication between one another. The protocol has been implemented on a single chip utilizing a high-speed scalar-point multiplier, and can reside at the network interfaces of nodes that can support such a device. The high speed of key

IEEE COMPUTER SOCIETY

Table 3. Comparison of protocol performance and features

Protocol	Cryptosystem	End-to-end time (ms)	Implementation	Certificate Usage	Number of Transactions
[1]	ECC 160 Prime	140	Software	Yes	4
[2]	ECC 132 OEF1	3000	Software	No	4
[3]	ECC 160 Prime	760	Software	Yes	5
[4]	RSA 1024	> 14500	Software	No	2
Our PoC	ECC 163 Binary	28	Hardware	Yes	2

negotiation also permits session keys to have much shorter lifetimes, thereby enhancing the overall security of the communication session.

We envisage a number of enhancements to the existing design. The PoC can be extended to support more advanced protocols, such as the Elliptic Curve Key Exchange v1.0 (ECKE-1) protocol [11], which was designed to address various security attributes, such as known key security, forward secrecy, key-compromise impersonation resilience, unknown key-share resilience, and key control. The various components of the PoC, e.g. big integer arithmetic and hashing, can be used to achieve the computations required by such protocols. Additionally, each PoC is bound to a single CA server. This could pose a scalability problem when nodes using the PoC have to switch between multiple networks. Therefore we would like to develop a special-purpose PoC for nodes that require bindings to multiple CA servers.

Acknowledgements

The authors wish to thank Ravikesh Chandra of the University of Auckland for permitting the use of his SHA-1 implementation in the Key Derivation Function.

References

[1] M. Aydos, T. Yanik, C. K. Koc. High-speed implementation of an ECC-based wireless authentication protocol on an ARM microprocessor. *IEE Proceedings Communications.* Volume 148, Issue 5, 2001.

[2] S. Kumar, M. Girimondo, A. Weimerskirch, C. Paar, A. Patel, A. S. Wander. Embedded end-to-end wireless security with ECDH key exchange. *Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems, MWSCAS '03.* Volume 2, 27-30, 2003.

[3] Q. Huang, J. Cukier, H. Kobayashi, B. Liu and J. Zhang. Fast Authenticated Key Establishment Protocols for Self-Organizing Sensor Networks. *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications.* ACM Press, 2003.

[4] R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, P. Kruus. TinyPK: securing sensor networks with public key technology. *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks.* ACM Press, 2004.

[5] J. Krasner. Using Elliptic Curve Cryptography (ECC) for Enhanced Embedded Security: Financial Advantages of ECC over RSA or Diffie-Hellmann (DH). Embedded Market Forecasters, American Technology, Inc., 2004.

[6] W. Diffie and M.E. Hellman. New directions in cryptography. *In IEEE Transactions on Information Theory.* 1978.

[7] M. Rosing. Implementing Elliptic Curve Cryptography. Manning Publications, 1999.

[8] I. F. Blake, R. M. Roth, G. Seroussi. Efficient Arithmetic in GF(2n) through Palindromic Representation. Visual Computing Department, Hewlett Packard Laboratories, 1998.

[9] M. Morales-Sandoval and C. Feregrino-Uribe. A Hardware Architecture for Elliptic Curve Cryptography and Lossless Data Compression. *Proceedings of the 15th International Conference on Electronics, Communications and Computers.* 2005.

[10] B. Guoqiang, H. Zhun, Y. Hang, C. Hongyi; L. Ming; C. Gang; Z. Tao; Chen Zhihua. A high performance VLSI chip of the elliptic curve cryptosystems. *Proceedings of the 7th International conference on Solid-state and Integrated Circuits techlonogy.* Volume 3, 2004.

[11] M. A. Strangio. Efficient Diffie-Hellmann two-party key agreement protocols based on elliptic curves. *Proceedings of the 2005 ACM symposium on Applied computing SAC '05.* ACM Press, 2005.

IEEE COMPUTER SOCIETY