Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 65751, 9 pages
doi:10.1155/2007/65751

Research Article

Supporting Symmetric 128-bit AES in Networked Embedded Systems: An Elliptic Curve Key Establishment Protocol-on-Chip

Roshan Duraisamy,[1] Zoran Salcic,[1] Maurizio Adriano Strangio,[2] and Miguel Morales-Sandoval[3]

[1] *Department of Electrical and Computer Engineering, The University of Auckland, Auckland 1142, New Zealand*
[2] *Department of Information, Systems and Production, University of Rome "Tor Vergata", 00173 Rome, Italy*
[3] *Computer Science Department, National Institute for Astrophysics, Optics and Electronics, 72840 Puebla, Mexico*

Received 14 July 2006; Revised 2 November 2006; Accepted 12 December 2006

Recommended by Sandro Bartolini

The secure establishment of cryptographic keys for symmetric encryption via key agreement protocols enables nodes in a network of embedded systems and remote agents to communicate securely in an insecure environment. In this paper, we propose a pure hardware implementation of a key agreement protocol, which uses the elliptic curve Diffie-Hellmann and digital signature algorithms and enables two parties, a remote agent and a networked embedded system, to establish a 128-bit symmetric key for encryption of all transmitted data via the advanced encryption scheme (AES). The resulting implementation is a protocol-on-chip that supports full 128-bit equivalent security (PoC-128). The PoC-128 has been implemented in an FPGA, but it can also be used as an IP within different embedded applications. As 128-bit security is conjectured valid for the foreseeable future, the PoC-128 goes well beyond the state of art in securing networked embedded devices.

1. INTRODUCTION

Securing communications between low-power, low-resource embedded systems is a relatively new challenge that has arisen with the rapid proliferation of Internet-enabled and other networked devices. Encryption of all transmitted data between networked embedded systems and remote agents, which connect to them for monitoring or remote control purposes, provides a strong means of establishing communication security. However, data encryption presupposes the establishment of secure cryptographic keys, which must be substantially large to reduce opportunities for an attacker with significant computing power to break via brute force or differential attacks. At the same time, embedded devices possess relatively fewer resources to manage large cryptographic keys.

To address this tradeoff between resource usage and cryptographic security, elliptic curve cryptography (ECC) has been proposed as a public key (PK) scheme to enable communicating systems establish keys of relatively small size for an equivalent level of security when compared with PK schemes like RSA [1]. According to [2], 163-bit ECC is known to provide 80-bit equivalent security, similar to 1024-bit RSA, which corresponds to 2^{80} rounds of brute force computation required to break the scheme. Recent research records that even 139-bit ECC can provide 80-bit security [3]. However, the current state-of-art 80-bit requirements are considered secure only until 2010 [1], and only 112-bit and above security will be viable until 2035. For 2036 and beyond, 128-bit security is therefore recommended and will likely become the state of art, according to the current NIST forecasts of microprocessor ability to break cryptographic schemes cited in [1].

Key-agreement protocols are vital to securely establishing encryption keys. To set up the secret key, public key cryptography (Diffie-Hellman key exchange [4]) is used and entity authentication is achieved via digital certificates (X.509). The use of a certificate authority- (CA-) based structure overcomes the vulnerability of basic (unauthenticated) Diffie-Hellman key exchange to man-in-the-middle attacks as all communicating nodes in the network are issued digital certificates. In general, the (complexity-theoretic) security of

Diffie-Hellman key exchange schemes derives from the intractability of the computational Diffie-Hellman (CDH) and the decisional Diffie-Hellman (DDH) problems in the underlying mathematical groups. The elliptic curve analogue of the Diffie-Hellman key agreement algorithms (ECDH) is comparatively more convenient than other groups since it allows efficient storage and implementations. ECDH protocols have been standardized in ANSI X9.63 [5], IEEE-1363-2000 [6], and ISO 15946-3 [7]. For ECC-based systems, the elliptic curve digital signature algorithm (ECDSA) [8] offers the ability to securely sign and verify data that can be used in such certificate-based authentication schemes.

However, key-agreement protocols still need to address a number of security attributes, which would otherwise enable attackers to break the protocol and compromise the established session key. These security attributes include known key security, forward secrecy, key-compromise impersonation resilience, unknown key-share resilience, and key control resilience [9].

Encryption and decryption of data is achieved by symmetric cryptographic schemes. ECC-based methods use a key derivation function (KDF) such as the KDF-1 specified in the IEEE P1363 [6] to derive the session key mask from the elliptic curve session key and perform encryption or decryption via an XOR operation between the mask and the plaintext or ciphertext, respectively. The potential for key or data compromise through known plaintext attacks can be alleviated by using enhanced modes of operation (e.g., cipher block chaining—CBC). The advanced encryption standard (AES) [10] also provides strong symmetric encryption, and is fast becoming a standard encryption scheme of choice.

Merging these different concepts into a comprehensive and secure protocol that can be used in networked embedded devices is a challenge that needs to be addressed. Recent research in this field has for the most part been built upon software-based microprocessor schemes [3, 11–13], and do not always provide integrated support for symmetric encryption such as AES. More recently, a full hardware protocol-on-chip (PoC), which performs secure ECDH operations using ECDSA-based certificates, was developed to meet the current state-of-art 80-bit equivalent security requirements [14]. In this implementation, a 163-bit binary field was used for ECC, and SHA-1 was used as the hashing algorithm both for the KDF and for the generation and verification of messages that are signed and verified via an ECDSA scheme.

However, a system is only as secure as its weakest link, and as the PoC only implements components that have a maximum security of 80 bits, a more comprehensive key agreement solution, which addresses the future 128-bit minimum security requirements, is necessary. In addition, this PoC does not fully address all the security attributes conjectured valid for a protocol attack. Full forward secrecy, for instance, is not guaranteed when embedded devices are being accessed by remote agents.

This paper presents a new implementation of a PoC that supports all security attributes using the elliptic curve key exchange ECKE-2 protocol, a modified version of the ECKE-1 protocol originally proposed in [9], which has been shown to fully address all security attributes. In addition, the elliptic curve components of the PoC have been upgraded to work on a 277-bit binary finite field, which provides equivalent 128-bit security [2]. The hashing algorithm used is an SHA-256 module, which is 128-bit collision resistant and therefore stronger than the 80-bit SHA-1. Symmetric encryption employs 128-bit AES for encryption and decryption of data instead of a pure XOR operation with the result of the ECC KDF-1 specified in [6]. While this system does use more hardware area than the original PoC, the resource usage has been optimized through sharing finite field units in the elliptic curve components. The level of resource requirements will certainly be affordable for future embedded systems that need to be 128-bit secure.

The rest of this paper is organized as follows: Section 2 provides a short overview of elliptic curve cryptography (ECC). Section 3 presents a brief review of the original PoC developed in [14]. Section 4 reviews the ECKE-1 protocol and the modified version ECKE-2 used in the new 128-bit PoC (PoC-128) as well as the security conjectures that this protocol addresses. Section 5 details the various functional modules of the PoC-128. Section 6 compares the timing and synthesis results of the original PoC with the PoC-128, and includes a comparison of both systems with recent related protocol implementations that secure networked embedded devices. Finally, Section 7 concludes this paper with a summary of contributions.

2. REVIEW OF ELLIPTIC CURVE CRYPTOGRAPHY

Elliptic curves over binary fields F_{2^m} or prime fields F_q can be represented by one of the following equations:

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

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

Elliptic curve arithmetic can be performed using either polynomial basis arithmetic or normal basis arithmetic [15]. A hardware polynomial basis implementation over the curve in (1) was used in this research. Points on an elliptic curve are expressed in terms of their coordinates $P(x, y)$. Elliptic curve arithmetic involves addition of two points on a curve to yield another point on the curve:

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

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

and doubling of a point to yield another point:

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

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

Scalar-point multiplication refers to the multiplication of a point P on the curve by a scalar value k to yield another point $R = kP$ on the curve; it is achieved by a combination of

the point addition and point doubling operations over the finite field until the multiplication is complete. The inverse of scalar-point multiplication is said to be intractable for a relatively small finite field size, thus making elliptic curve cryptography very suitable for asymmetric public key systems.

The elliptic curve variant of the Diffie-Hellmann protocol (ECDH) makes use of the intractability of this operation in the underlying group (which by analogy to multiplicative groups is also called the discrete log (DL) problem), and is used to establish a shared secret between two communicating parties. It is assumed that an attacker knows the domain parameters $(a_2, a_6, P(x, y), n)$. Two honest parties A and B with their respective secrets w_A and w_B compute public keys W_A and W_B which they exchange in order to establish the shared secret,

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

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

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

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

A passive attacker only sees W_A and W_B but is unable to determine either w_A or w_B due to the intractability of the DL problem nor can she compute the shared secret K_{AB} because of the intractability of the ECDH problem.

The elliptic curve digital signature algorithm (ECDSA) can be used by one party (the recipient) to verify the authenticity of a message sent by another party (the signer) using the latter's public key. To sign a message, party A with public-private key pair (W_A, w_A) performs the following steps over the elliptic curve $(a_2, a_6, P(x, y), n)$:

(1) generate random value r;
(2) compute the random point $R(x_R, y_R) = rP$;
(3) compute the hash of the message $h = \text{H}(\text{message})$;
(4) signature $s_1 = x_R \pmod n$;
(5) signature $s_2 = ((h + s_1 w_A)/r) \pmod n$.

The signature pair (s_1, s_2) is transferred across to party B, who then uses A's public key W_A to verify that the message was signed by A as follows:

(1) compute the hash of the message $h' = \text{H}(\text{message})$;
(2) compute $u = (h'/s_2) \pmod n$ and $v = (s_1/s_2) \pmod n$;
(3) compute the point on the elliptic curve: $K(x_k, y_k) = uP + vW_A$;
(4) if $x_k \pmod n = s_1$, then the signature has been verified.

3. REVIEW OF A PREVIOUS ELLIPTIC CURVE PROTOCOL-ON-CHIP

The original PoC developed in [14] resides at the network interfaces of embedded devices. Remote agents can connect to these devices provided they have issued certificates by a CA server that the devices can use to authenticate incoming agents. The CA server also functions as a security manager that designates specific nodes that can communicate with one another. Each node in a network possesses a long-term public-private ECC key pair. In ECC, a private key is typically a scalar value in the elliptic curve finite field, and the corresponding public key is a point on the chosen elliptic curve, which is generated by multiplying the private key by a chosen base point on the curve.

When a remote node initiates communication with an embedded device, the remote node first generates a random ephemeral secret, from which an ephemeral public key is computed. The ephemeral public key is signed together with the remote node identity using the remote node's long-term private key via ECDSA, using SHA-1 as the hash function. The signature, remote node identity, and the ephemeral public key are transferred across the communications channel to the embedded device. The signature of the ephemeral public key and the remote node's identity are verified by the PoC using the remote node's certificate, and the PoC then generates its own ephemeral keys and signatures. A common session key is established via the traditional ECDH process. From this session key, a shared secret key mask is derived using the KDF-1 [6], which uses SHA-1 for key derivation. Symmetric encryption and decryption involves an XOR operation with the key mask. It is also possible for a PoC to establish a secure connection with a remote node as described, except that the PoC now functions as the initiator and the remote node as the responder.

The PoC can also establish special communication sessions with the bound CA server, which can be configured to periodically request regeneration of new certificates. Each certificate comprises of the node identity and the node's long-term public key. The CA server maintains a database of all node certificates and a special "CA counter" for each node which assists in synchronizing all communications between the CA server and each node when certificate regeneration is required. Each node also maintains a corresponding "CA counter." The counter is used as part of the ECDSA signature generation and verification routines when nodes are being periodically reconfigured with new certificates by the CA server. This periodic reconfiguration is required to track certificate expiry.

4. THE ECKE-2 PROTOCOL

The ECKE-1 protocol [9] was designed to address all the security attributes of known key security, forward secrecy, key-compromise impersonation resilience, unknown key-share resilience, and key control. The protocol enables two parties to exchange ephemeral public keys as in the normal ECDH protocol; however, the generation of the ephemeral secrets and the session keys involve arithmetic operations that ensure an attacker cannot circumvent the conjectured security attributes merely from transcripts of the data exchanged. Figure 1 depicts the ECKE-2 protocol, which improves the original implementation in [9] for the new PoC-128, together with ECDSA signature generation (SGEN) and verification (SVER) functions for authenticating the ephemeral data transferred.

Strictly speaking, the ECKE-2 key agreement protocol is designed to provide implicit key authentication (IKA), meaning that in a run of the protocol only the two uncorrupted parties involved in the communication should be able

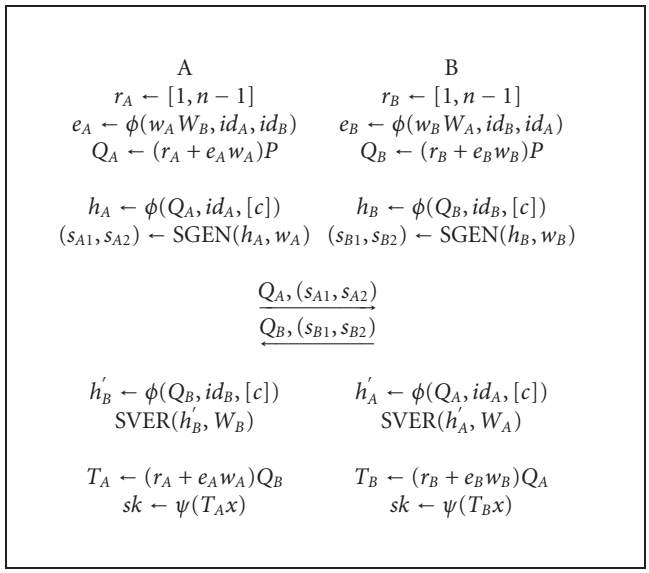

Figure 1: The ECKE-2 protocol.

to establish the session key (since computation of this key by each party requires knowledge of their long-term private keys). The whole point about this key-based authentication mechanism is that it allows the design of efficient protocols. However, in some situations a stronger requirement may be mandated to prevent arbitrary modifications of the protocol flow by an active adversary (e.g., man-in-the-middle attacks cited earlier). For this reason, protocol ECKE-2 makes use of digital signatures (ECDSA) to (explicitly) authenticate the message flows although this results in additional computation (three scalar multiplications are required on each side).

Consider two parties A and B with public-private key pairs W_A, w_A and W_B, w_B, respectively. Such key pairs are associated with a set of domain parameters $(a_2, a_6, P(x, y), n, h, FR, q)$ which describe an elliptic curve $E(F_q)$ (with coefficients a, b) over a finite field F_q, a base point P of order n, the cofactor $h = \#E(F_q)/n$ and an indication FR of the representation used for field elements. The parameters should be appropriately chosen so that no efficient algorithm exists that solves the DL problem in the subgroup $\langle P \rangle$. The domain parameters must undergo a validation process proving the elliptic curve has the claimed security attributes [15]. In the protocol, each side also uses a hash function $\phi(\cdot)$ to produce the long-term shared secret values e_A and e_B and generates a random number to compute the ephemeral keys Q_A and Q_B, which are signed and exchanged with the signatures, as shown in Figure 1. In protocol ECKE-1, the values $e_A = \phi(r_A, w_A, id_A)$ and $e_B = \phi(r_B, w_B, id_B)$ are ephemeral session-specific data while in protocol ECKE-2 they are long-term static keys and therefore may be used across subsequent independent runs (with one less scalar multiplication).

The message digest to be signed at each node is composed of its ephemeral public key, its identity, and an optional CA server counter c, if the PoC is communicating with a CA server. After signature verification, the shared session keys T_A and T_B are generated as per the ECDH process. The shared secret key for symmetric encryption is derived via a KDF that uses the SHA-256 for hashing.

The signature-based ECKE-2 protocol addresses all security attributes as follows.

Known-key security

An attacker with access to previously established session keys (by honest parties) cannot obtain the session keys of future protocol runs. Indeed, keys established in a run of the protocol are unique unless the same players generate identical random nonces in two different sessions. However, the probability of such an event is negligible (in the order of s^2/n, where s is an upper bound on the number of sessions observed by the adversary).

Forward secrecy

Assuming an adversary possesses either one or both the private keys w_A and w_B, deriving the session keys from previous runs of the protocol requires knowledge of the random ephemeral keys r_A and r_B. Given the intractability of the DL problem on the underlying EC group, it is computationally infeasible to obtain these values. Furthermore, even if the adversary is able to obtain this session-specific data, compromise of the long-term private keys w_A, w_B may be hard in practice (e.g., if they are stored in a tamper-proof security module). Thus, the protocol maintains full forward secrecy. Observe that protocol ECKE-2 becomes resistant to the stronger version of forward secrecy against active adversaries (as opposed to passive adversaries that are allowed to corrupt the parties only after the protocol has completed its run).

Unknown key-share resilience

An adversary posing as E cannot deceive A into believing that messages received from E were actually issued by B. Again, this is because although E may have been able to obtain a valid certificate, A can easily verify the identity of E. Without a valid certificate in the first place, which is established when a CA server designates communicating nodes, A will not participate in the protocol.

Key-compromise impersonation resilience

If A's private key w_A is compromised, an adversary E can easily impersonate A to any other party. In passing we note that, contrary to the claims of the author, protocol ECKE-1 is vulnerable to KCI attacks. Indeed, an adversary E knowing w_A may replace the response message of B (Q_B) with $Q_E = r_E P$ (for some random nonce r_E) and have A accept a known session key derived from $r_E Q_A + d_A W_B$. By making use of signatures protocol ECKE-2 is not affected by such a vulnerability since the adversary must obtain w_B (to sign in place of B) or must be able to forge a signature from B.

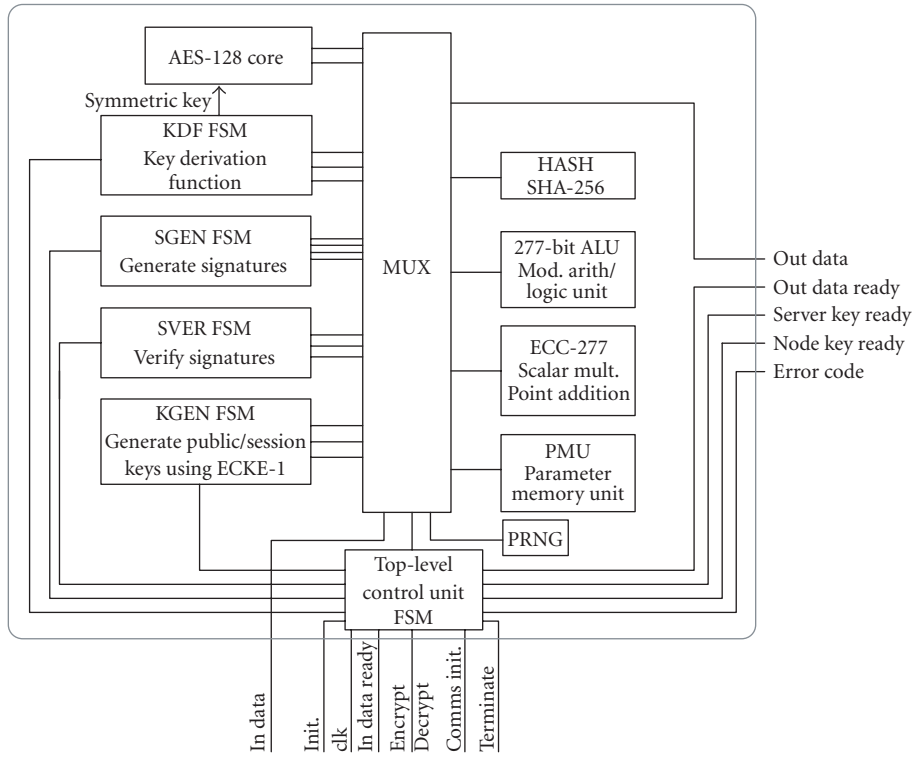

Figure 2: Functional modules of the PoC-128.

Key-control resilience

Key agreement protocols rely on the assumption (which is often implicit) that robust primitives are available for generating random numbers. In some protocols, one of the principals may have a slight advantage in predetermining the value of a random nonce. However, in a run of the ECKE-2 protocol the initiator may be able to select a limited number of bits in its nonce since, in practice, the precomputation must be done before the responder times out.

Identity assurance

The signature-based authentication scheme (with ECDSA) ensures that each node can corroborate the purported identity of any other node it is in communication with, by verifying the authenticity of the associated digital certificates.

5. PoC-128

The functional layout of the PoC-128 is depicted in Figure 2. The entire structure uses a hierarchical finite-state machine (FSM) as in the previous implementation of the PoC, whereby a top-level FSM initiates individual FSMs of the functional modules of the protocol. The ECKE-2 protocol is coordinated by the key generation (KGEN) module, which generates both the ephemeral public-private key pairs and the elliptic curve session key. This session key is then used by the KDF FSM in conjunction with the SHA-256 core to produce a 128-bit symmetric key which is made available to the AES-128 module. The top-level FSM coordinates the signing of the PoC-generated ephemeral keys and the verification of incoming ephemeral keys. Then, the ECDSA signature generation (SGEN) and signature verification (SVER) FSM modules are initiated as appropriate. These, in turn, make use of the SHA-256, ECC-277, and ALU-277 computational modules accordingly.

As with the previous PoC implementation, a parameter memory unit (PMU) is used to store all node configuration data as well as temporary protocol data. The entire datapath is managed by the top-level FSM and a multiplexer (MUX) that enables resource-sharing of the functional units.

AES core

The AES-128 algorithm consists of 10 rounds of computation. Each round transforms a 128-bit input into a 128-bit output, and uses a round key that is derived from the original key. There are four basic stages for the first nine rounds—ByteSub (BS), ShiftRow (SR), MixColumn (MC), and AddRoundKey (ARK). The tenth round does not use the MC stage. Each of these four stages is invertible for decryption.

A pipelined implementation of the AES-128 core is shown in Figure 3. The inputs to the core are 32-bit words, which are sequentially serialized on the clock into an input register in groups of 4 words and a null 32-bit word.

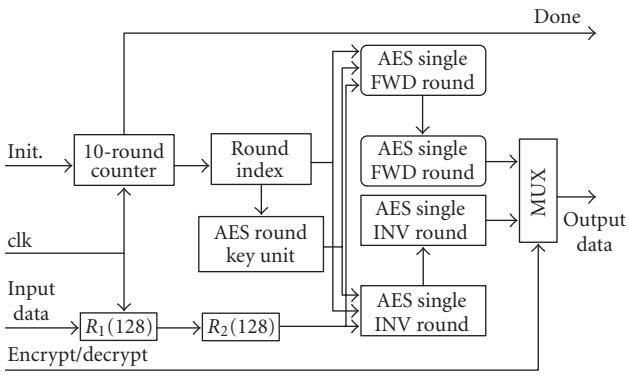

Figure 3: AES-128-pipelined core.

Table 1: Comparison of resource usage between original PoC and the new PoC-128.

	PoC	PoC-128
Device family	Stratix II	Stratix II
Device name	EP2S90F1020C3	EP2S130F1020C4
Combinational functions	34 914	53 246
Registers (flipflops)	19 130	19 456
ALUTs	40 234/72 768 (55%)	58 782/106 032 (55%)
Maximum frequency	78.84 MHz	47.44 MHz
Memory bits	49 152/4 520 448 (1%)	73 728/6 747 840 (1%)

The null word is not encrypted, but serves merely to delimit the 4-word input. The null word is ignored by the encryption/decryption core but allows for adding parity information or other error-checking codes, if necessary. The AES core with two rounds per clock cycle ensures that a single encryption or decryption takes 5 clock cycles to complete. While encryption is performed on the register R2, the input words are serialized into the input register R1. In this manner, encryption/decryption takes place in a reasonably pipelined fashion, and supports a 32-bit interface. This ensures the structure is adaptable to the top-level PoC, which itself assumes an external 32-bit interface for working with a 32-bit microprocessor in the main embedded device application being secured by the PoC.

SHA-256 core

A nonpipelined SHA-256 functional unit, with a block diagram shown in Figure 4, was developed for providing 128-bit secure hashing functionality. Eight registers A–H are used for the temporary SHA-256 variables. The control unit maintains a counter for the round index value which ranges from 0 to 63. The round constant table is implemented as a ROM-unit that selects one of the 64 SHA-256 round-table constants depending on the round index input.

The word generator module provides one of the 16 segments of the input word while the round index is between 0 and 15 inclusive. However, for higher values of the round index, a different combination of the original word segments is used and provided at the output. The X_0 and X_1 blocks are combinational functional blocks that are described in Figure 4.

The X_0 block uses the registers A, B, and C as inputs. The X_1 block combines registers E, F, and G together with the round constant RC and the message word segment MW for that particular round. The register A then obtains the value $X_0 + X_1$, and the register E obtains the value $D + X_1$. The other registers are updated as shown. The final result from the A–E registers is then accumulated in registers h_0–h_7 after 64 rounds of computation.

ECC core

The original 163-bit elliptic curve multiplier/adder module shown in Figure 5(a) is highly parameterized, and therefore changing the finite field size in the module to 277 bits is straightforward. However, when synthesized, the 277-bit unit uses a significant number of resources. Sharing of the 277-bit finite field arithmetic units (F_2^m multiplier, divider and squarer) across the adding and doubling units within the module reduces the total number of resources by over 5 000 adaptive look-up tables (ALUTs), a reduction of almost 27%. A 277-bit multiplier/adder without resource sharing requires 19 049 ALUTs on an EP2S60F1020C3 Stratix II FPGA. This platform provides a total of 48 352 ALUTs, which is equivalent to 60 440 logic elements [16].

This resource sharing is depicted in Figure 5(b). The FSM at the top level of the ECC multiplier/adder module determines whether the ECC adder or the ECC doubler exclusively uses the F_2^m arithmetic units. At an operating frequency of 90 MHz, this increases overall execution time of the unit by only 0.73 milliseconds (the total time for a scalar multiplication is 2.4 milliseconds).

6. RESULTS AND DISCUSSION

The PoC-128 synthesizes successfully on an FPGA Stratix II chip, occupying 58 782 ALUTs as opposed to the 40 234 ALUTs of the original PoC (Table 1). The PoC-128 operates at a lower maximum frequency of 47.44 MHz, compared with the 78.84 MHz of the original PoC. The highest latency is exhibited by the SHA-256 unit at 50 MHz, followed by the AES-128 core at 65 MHz. With a clock-generator specifically for the SHA-256, therefore, the maximum operating frequency of the PoC-128 can be taken up to almost 65 MHz.

Table 2 compares the operation times for both PoC implementations at clock speeds close to their maximum operating frequencies. The PoC-128 takes 61 milliseconds for a complete ECKE-2 protocol run, while the original PoC takes 10 milliseconds with its simple, authenticated ECDH protocol. These results illustrate that with a 46% increase

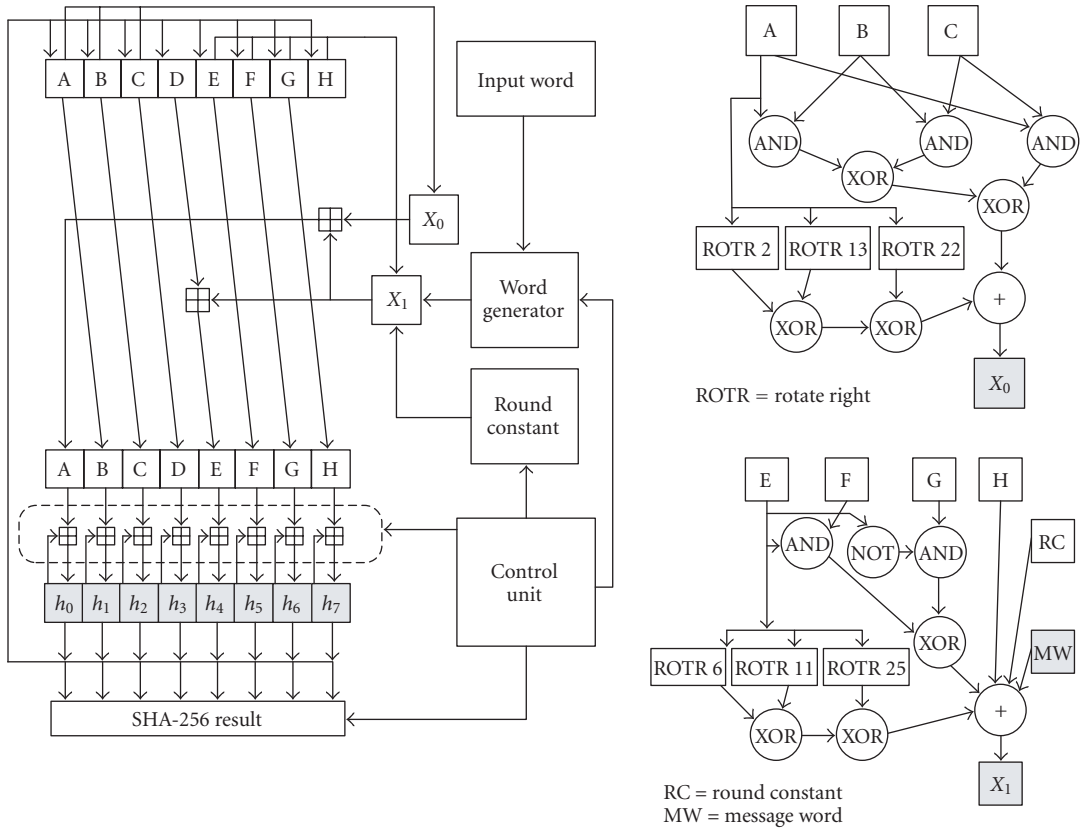

Figure 4: SHA-256 computational module.

Table 2: Comparison of the protocol times in milliseconds for the original PoC and the new PoC-128.

Operation	PoC at 75 MHz	PoC-128 at 45 MHz
Protocol	10	61
Signature generation	1.3	7.3
Signature verification	1.8	11.5
ECC scalar multiplication	0.7	4.8
ECC point addition	0.004	0.012
ALU modulo multiplication	0.37	1.57
ALU modulo division	0.21	0.87

in hardware area, and 51 milliseconds increase in protocol execution run time (using maximum frequencies of both platforms), 128-bit security can be fully ascertained in a secure protocol on a single chip that supports full symmetric encryption with AES. Despite the performance reduction compared to the original PoC, the tradeoff is clearly in increased security.

A comparison of the original PoC and the new PoC-128 against other recent protocol implementations for supporting symmetric encryption is presented in Table 3. As can be observed, of all six implementations, the PoC-128 is the only one that uses the more advanced AES-128 as its mode of symmetric encryption and SHA-256 for all hashing purposes. Thus, the PoC-128 is the only one that provides the required 128 bits security, which is estimated sufficient for the foreseeable future. All other protocol implementations use SHA-1 for hashing and 80-bit equivalent public key schemes, and they may or may not themselves support symmetric encryption. Although [12] does use AES, its implementation involves SHA-1 and uses ECC-132 optimal extension fields which have a maximum equivalent security of only 80 bits, and hence its total security is only 80-bit equivalent. Moreover, all the elliptic curve implementations apart from the PoC-128 use only 80-bit equivalent elliptic curve operations, and thus cannot meet the 128-bit symmetric security requirements of [1]. Consequently, the PoC-128 is a significant step forward in the development of protocol implementations for securing networked embedded systems as it goes well beyond the current state of art requirements and caters to embedded systems security of the future.

7. CONCLUDING REMARKS

Symmetric encryption of data exchanged between two embedded nodes in a network is an important part of securing such nodes. In a network of embedded systems accessed by remote agents, the challenge of establishing a cryptographic key for symmetric encryption can be addressed by strong public key mechanisms. In this paper, we have proposed an elliptic curve-based protocol-on-chip using elliptic curve

Table 3: Comparison of protocol performance and features.

Protocol	[3]	[11]	[12]	[13]	PoC	PoC-128
Cryptosystem	ECC 160 prime	ECC 132 optimal extension	ECC 160 prime	RSA 1024	ECC 163 binary	ECC 277 binary
Total time (ms)	140	3000	760	> 14 500	11	61
Implementation	Software	Software	Software	Software	Hardware	Hardware
Certificate usage	Yes	No	Yes	No	Yes	Yes
Number of transactions	4	4	5	2	2	2
Symmetric encryption	None	3DES	AES or MSR (modular square root)	None	ECES	AES-128
Hash function	SHA-1	None	SHA-1	None	SHA-1	SHA-256

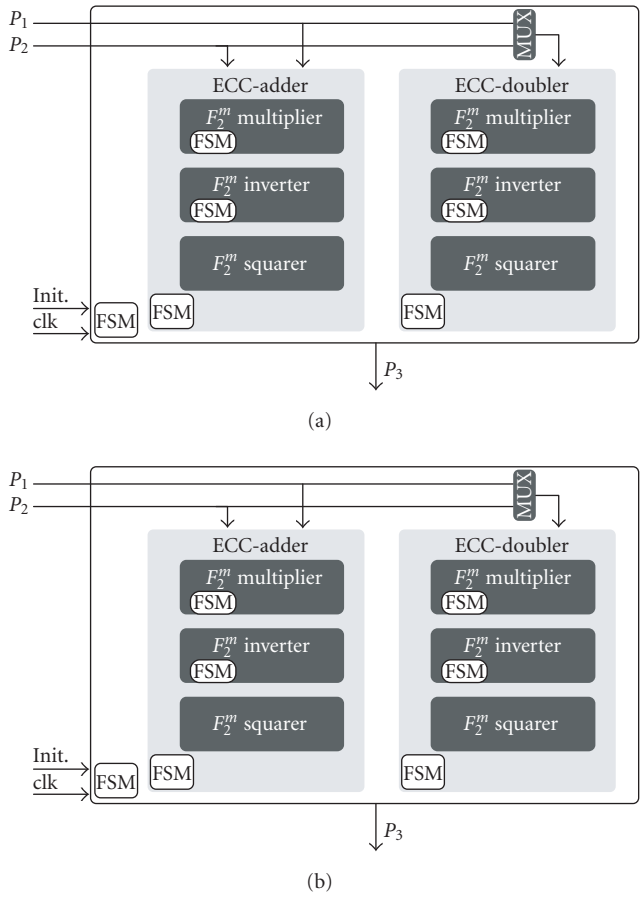

Figure 5: (a) Original 163-bit core, (b) 277-bit core with resource sharing.

components that have been extended to operate on a 277-bit polynomial basis field to provide 128-bit equivalent security. Certificates issued by a special CA server enable nodes to authenticate one another. Signing and verifying of exchanged data is accomplished at each node using ECDSA over a 277-bit field, which employs SHA-256 as the hashing algorithm. The ECKE-2 protocol, a signature-based version of the original ECKE-1, enables secure establishment of a 277-bit ECC session key between two nodes. This is used to derive a 128-bit symmetric key (using the SHA-256 component), which can be used by the nodes to encrypt and decrypt all exchanged data via AES-128. All components thereby provide a total of 128 bits of security, thus ensuring that the entire system is secure, according to current forecasts, well beyond 2036. While we envisage software versions of the protocol being used in remote agents and on the network CA server, shortest key agreement execution time of 61 milliseconds can be achieved via a full hardware implementation. With this in consideration, the entire system has been described and implemented as a complete hardware module, a protocol-on-chip (PoC) for use at the network interfaces of the embedded devices.

REFERENCES

[1] 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, 2004.

[2] P. Panjwani and Y. Poeluev, "Additional ECC Groups For IKE," IPSec Working Group, INTERNET-DRAFT, 1999.

[3] M. Aydos, T. Yanik, and Ç. K. Koç, "High-speed implementation of an ECC-based wireless authentication protocol on an ARM microprocessor," *IEE Proceedings: Communications*, vol. 148, no. 5, pp. 273–279, 2001.

[4] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[5] ANSI X9.63, "Public Key Cryptography for the Financial Services: Key Agreement and Key Transport using Elliptic Curve Cryptogrphy," American National Standards Institute, 2001.

[6] IEEE-P1363-2000, "Standard Specifications for Public Key Cryptography," Institute of Electrical and Electronics Engineers, 2000.

[7] ISO/IEC-15946-3, "Information Technology-Security Techniques—Cryptographic Techniques based on Elliptic Curves-Part 3: Key Establishment," International Standards Organization, 2002.

[8] ANSI-X9.62-1998, "Public Key Cryptography for the Financial Services: The Elliptic Curve Digital Signature Algorithm," American National Standards Institute, 1999.

[9] M. A. Strangio, "Efficient Diffie-Hellmann two-party key agreement protocols based on elliptic curves," in *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC '05)*, vol. 1, pp. 324–331, Santa Fe, NM, USA, March 2005.

[10] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," National Institute of Standards and Technology, 1999.
[11] S. Kumar, M. Girimondo, A. Weimerskirch, C. Paar, A. Patel, and A. S. Wander, "Embedded end-to-end wireless security with ECDH key exchange," in *Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS '03)*, vol. 2, pp. 786–789, Cairo, Egypt, December 2003.
[12] Q. Huang, J. Cukier, H. Kobayashi, B. Liu, and J. Zhang, "Fast authenticated key establishment protocols for self-organizing sensor networks," in *Proceedings of the 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '03)*, pp. 141–150, San Diego, Calif, USA, September 2003.
[13] R. Watro, D. Kong, S.-F. Cuti, C. Gardiner, C. Lynn, and P. Kruus, "TinyPK: securing sensor networks with public key technology," in *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '04)*, pp. 59–64, Washington, DC, USA, October 2004.
[14] R. Duraisamy, Z. Salcic, M. Morales-Sandoval, and C. Feregrino-Uribe, "A fast elliptic curve based key agreement protocol-on-chip (PoC) for securing networked embedded systems," in *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '06)*, pp. 154–161, Sydney, Australia, August 2006.
[15] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Professional Computing, Springer, New York, NY, USA, 2004.
[16] "Stratix II Device Handbook, Volume 1," Altera, 2006.