Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 836460, 12 pages
doi:10.1155/2011/836460

Research Article

Prime Field ECDSA Signature Processing for Reconfigurable Embedded Systems

Benjamin Glas, Oliver Sander, Vitali Stuckert, Klaus D. Müller-Glaser, and Jürgen Becker

Institute for Information Processing Technology (ITIV), 76131 Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Correspondence should be addressed to Benjamin Glas, glas@kit.edu

Received 27 August 2010; Accepted 10 February 2011

Academic Editor: Gilles Sassatelli

Growing ubiquity and safety relevance of embedded systems strengthen the need to protect their functionality against malicious attacks. Communication and system authentication by digital signature schemes is a major issue in securing such systems. This contribution presents a complete ECDSA signature processing system over prime fields for bit lengths of up to 256 on reconfigurable hardware. By using dedicated hardware implementation, the performance can be improved by up to two orders of magnitude compared to microcontroller implementations. The flexible system is tailored to serve as an autonomous subsystem providing authentication transparent for any application. Integration into a vehicle-to-vehicle communication system is shown as an application example.

1. Introduction

With emerging ubiquity of embedded electronic systems and a growing part of distributed systems and functions even in safety relevant areas, the security of embedded systems and their communication gains importance quickly. One major concern of security is authenticity of communication peers and information exchange. Especially if many different remote participants have to communicate or not all participants are known in advance, asymmetric signature schemes are beneficial for authentication purposes. In contrast to symmetric schemes like the Keyed-Hash Message Authentication Code HMAC [1], asymmetric signature schemes like RSA [2], DSA [3], and the ECDSA scheme [3] considered in this contribution get along without key exchange or predistributed keys, relying usually on a certification authority as trusted third party instead.

This benefit comes at the cost of a much greater computational complexity of these schemes compared to authentication techniques based on symmetric ciphers or solely on hashing. This imposes major problems especially for embedded systems, where resources are scarce.

This contribution presents a hardware-implemented system for complete prime field ECDSA signature processing on FPGAs. It can be integrated as an autonomous subsystem for signature processing in embedded devices. As an application example the integration in a vehicle-to-vehicle communication unit is presented.

The remainder of this paper is organized as follows. In Section 2 some related work is given, Section 3 presents basics of the implemented signature scheme ECDSA, and Section 4 outlines the assumed situation and requirements for the system. The structure and implementation of the signature system itself is presented in Section 5, and Section 6 shows an application example and integration in a wireless communication system. Section 7 details performance and resource usage that are further discussed in Section 8. The paper is concluded in Section 9.

2. Related Work

Since elliptic curves were proposed as basis for public key cryptography in 1985 by Koblitz [4] and Miller [5] independently, many implementations of the prime field Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Cryptography (ECC) in general have been published. Software implementations on general purpose processors

need a lot of computation power. The eBACS ECRYPT benchmark [6] gives values for 256-bit ECDSA of, for example, 1.88 ms for generation and 2.2 ms for verification on an Intel Core 2 Duo at 1.4 GHz and 2.9 ms respectively, 3.4 ms on an Intel Atom 330 at 1.6 GHz. Values for a crypto system based on an ARM7 32-bit microcontroller are given in [7] for a key bit length of 233 bit. Using a comb table precomputation ($w = 4$) 742 ms are needed for a generation and 1240 ms for a verification of an ECDSA signature. An implementation for a RIM Blackberry [8] using an ARM 9EJ-S core realizes 150 ms for a signature generation and 168 ms for a signature verification [9].

To achieve usable throughputs and latencies on embedded systems, various specialized hardware solutions have been proposed, for example, many approaches for implementation of $\mathbb{F}_p$ arithmetic and the ECC primitives point add and point double on reconfigurable hardware. A survey of hardware implementations can be found in [10]. McIvor et al. [11] propose a special ECC processor for $\mathbb{F}_p$ on a Virtex II Pro FPGA, calculating a 256-bit scalar multiplication in 3.86 ms using a clock frequency of 39.5 MHz. Orlando and Paar [12] achieve for a bit length of 192 a scalar multiplication in 3 ms on a Virtex-E FPGA. Güneysu and Paar present in [13] a very fast approach based on special DSP FPGA slices, achieving processing times of 620 μs for a 256-bit scalar multiplication on a Virtex-4 FPGA. The implementation presented here is based on an $\mathbb{F}_p$ ALU presented by Ghosh et al. in [14]. Implementation approaches on CMOS standard cells can be found, for example, in [15, 16], achieving scalar multiplications in 256-bit length in 2.68 ms and 4.3 ms, respectively.

Nevertheless, open implementations of full signature processing units performing complete ECDSA are scarce. Järvinen and Skyttä [17] present a Nios II-based ECDSA system on an Altera Cyclone II FPGA for a key length of 163-bit performing signature generation in 0.94 ms and verification in 1.61 ms.

This contribution presents an FPGA-based autonomous ECDSA system for longer key lengths of 256 bit containing all necessary subsystems for application in embedded systems on reconfigurable hardware.

3. ECDSA Fundamentals

The Elliptic Curve Digital Signature Algorithm (ECDSA) is based on a group structure defined on an elliptic curve E over a finite field $\mathbb{F}_q$. Mostly two types of underlying finite fields are technically used: binary fields $\mathbb{F}_{2^n}$ of characteristic two and prime fields $\mathbb{F}_p$ with large primes p and corresponding characteristic. This paper focuses on prime fields $\mathbb{F}_p$ with characteristic char $(\mathbb{F}_p) \gg 3$. In this case the group E and the respective operation is defined as follows.

Definition 1 (group operation on E). Let E be an elliptic curve over a finite field $\mathbb{F}_p$ of characteristic char $(\mathbb{F}_p) \gg 3$ given by the Weierstrass equation

$$E : y^2 = x^3 + ax + b, \tag{1}$$

Input: Domain parameter $D = (q, a, b, G, n, h)$, secret key d, message m
Output: Signature (r, s)
(1) Chose random $k \in [1, n-1], k \in \mathbb{N}$
(2) Compute $kG = (x_1, y_1)$
(3) Compute $r = x_1 \bmod n$. If $r = 0$ goto step 1.
(4) Compute $e = H(m)$
(5) Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ goto step 1.
(6) **return** (r, s).

Algorithm 1: ECDSA signature generation.

with $a, b \in \mathbb{F}_p$, $4a^3 + 27b^2 \neq 0$, and $P = (x_1, y_1)$, $Q = (x_2, y_2)$ points on E. A group on $E \cup \{\mathcal{O}\}$, $\mathcal{O}$ being the special point at infinity, and the group law

$$+ : E \times E \longrightarrow E, \qquad (P, Q) \longmapsto P + Q =: R = (x_3, y_3) \tag{2}$$

on E is defined by the following

(i) $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E$.

(ii) For $P = (x_1, y_1) \in E$, the point $P = (x_1, -y_1)$ is also $\in E$ and $P + (-P) = \mathcal{O}$.

(iii) For $P \neq \pm Q$ and $P \neq -P$, the operation for $R = P + Q = (x_3, y_3)$ is given by

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2,$$
$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1. \tag{3}$$

(iv) For $P = Q$ and $P \neq -P$, there is $R = 2P = (x_3, y_3)$ defined by

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1,$$
$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1. \tag{4}$$

The set $E \cup \{\mathcal{O}\}$ with the defined group law $+$ is an abelian group with neutral element $\mathcal{O}$. The inverse to a point $P = (x, y)$ is given by $-P = (x, -y)$.

For the use of ECDSA a set of common domain parameters is needed to be known to all participants. These are the modulus p identifying the underlying field, parameters a, b defining the elliptic curve E used, a base point $G \in E$, the order n of G, and the cofactor $h = \text{order}\ (E)/n$. In addition a cryptographic hash function H is needed. The signature generation and verification for a key pair (Q, d), $Q \in E$ being a point on the curve and d a scalar factor with $Q = dG$, can then be performed using the secret key d or the public key Q, respectively. The procedures needed are shown in Algorithms 1 and 2.

Input: Domain parameter $D = (q, a, b, G, n, h)$,
public key Q, message m, signature (r, s).
Output: Acceptance or Rejection of the signature
(1) **if** $\neg(r, s \in [1, n-1] \cap \mathbb{N})$ **then**
(2) **return** "reject"
(3) **end if**
(4) Compute $e = H(m)$
(5) Compute $w = s^{-1} \bmod n$.
(6) Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
(7) Compute $X = (x_X, y_X) = u_1 G + u_2 Q$.
(8) **if** $X = \infty$ **then**
(9) **return** "reject"
(10) **end if**
(11) Compute $v = x_X \bmod n$.
(12) **if** $v = r$ **then**
(13) **return** "accept"
(14) **else**
(15) **return** "reject"
(16) **end if**

Algorithm 2: ECDSA signature verification.

For identification of the most demanding operations, a tracing of the algorithms based on the hardware implementation presented in Section 5 and the possible parallelization was done. Of the total of 395,521 clock cycles needed for signature generation with the modulus $p256$ used (see Section 4), a percentage of 99.8% or 394,752 cycles were spent computing the scalar multiplication kG. For signature verification the amount of cycles spent for the double scalar multiplication $X = u_1 G + u_2 Q$ is even 99.9%. So in the further consideration we focus on these central operations.

4. Setup and Situation

Objective of a digital signature is to guarantee authenticity and integrity of a signed message to the receiver and prove the identity of the sender, including nonrepudiation. The usual method based on an asymmetric primitive like ECDSA contains three protocol steps. First the sender generates a key pair consisting of a secret signature key SK (d in the ECDSA case) and a public verification key VK (Q for ECDSA) and publishes VK to all possible verifiers. To sign a message m of arbitrary length, the sender generates a digest $H(m)$ of the message using a publicly known cryptographic hash function H. This digest is of a fixed length and can be seen as a fingerprint of the message in the sense that finding a different message $m' \neq m$ with $H(m) = H(m')$ is infeasible. This digest is then signed, meaning *encrypted* using the signing key SK of the sender, and sent along with the original plain text message m. The receiver or verifier is then able to verify the signature by *decrypting* the received hash value using the sender's public verification key PK and comparing the decrypted value to the output of H applied to the received plain message. If the two values match, the signature is positively verified (see Algorithm 3).

The security and correctness of the signature method is based on the assumption that a signed value (encrypted with the secret key) can only be verified (decrypted) with knowledge of the corresponding public key and vice versa and that the secret key cannot be computed from the public key. Secondly the mapping of public keys to identities has to be guaranteed in some way. This is usually done using certification authorities as trusted third parties that verify the identity and issue a certificate for the public key.

We assume an embedded system communicating with several peers which are not entirely known in advance. Therefore, the exchanged signed messages are sent with a certificate attached, that is, issued, to a commonly trusted certification authority. As an example scenario the vehicle-to-vehicle (V2V) communication is considered in Section 6.

This contribution focuses on prime field ECDSA as it is proposed for vehicle-to-vehicle communications which is our general focus application (see also Application Example). Implemented are especially two elliptic curves recommended by the U.S. National Institute of Standards and Technology (NIST) in [18] and Certicom Research in [19], namely, the curves $p224$ (secp224r1) and $p256$ (secp256r1) with bit lengths 224 and 256, respectively, and the corresponding domain parameters also given in the standard.

The proposed system works as a security subsystem exclusively performing signature processing and passing and receiving messages m to and from the external system.

5. Signature Processing System

Processing of ECDSA consists of several layers of computation. On the top level the signature generation and verification algorithms as well as the certificate validation are performed. This signature scheme-dependent layer is based on the group operations point add (PA) and point double (PD) in the underlying elliptic curve. These are in turn based on the underlying finite prime field ($\mathbb{F}_p$) arithmetic, that is, modular arithmetic modulo a prime p. For the main operation of signature verification, the double scalar multiplication $kG + rQ$, the respective number of underlying operations needed on each layer to perform a single operation on the respective upper layer is given in Figure 1. In an even higher layer, there is also the communication protocol to consider at least partially as needed for the signature system.

The architecture and presentation of the system reflects this layering. The two upper layers are implemented as finite state machines (FSM) and make use of a basic $\mathbb{F}_p$ arithmetic logical unit (ALU) and some additional auxiliary modules. Figure 2 outlines the structure of the system. The different building blocks are detailed in the following paragraphs.

5.1. $\mathbb{F}_p$ Modular ALU. The central processing is done by a specialized $\mathbb{F}_p$-ALU for primes of maximum 256-bit length. It is based on the ALU proposed by Ghosh et al. in [14]. Figure 3 depicts the implemented structure. The ALU contains one $\mathbb{F}_p$ adder, subtractor, multiplier, and divider/inverter each. All registers and datapaths between

Input: Sender: Hash function H, secret key SK of sender, message m.
Output: Signed message $(m, \mathrm{Sig}(m))$

(1) Compute hash value $H(m)$ of m.
(2) Compute $\mathrm{Sig}(m) = Enc_{SK}(H(m))$ by encrypting the hash digest $H(m)$ using the sender's secret key.
(3) Send $(m, \mathrm{Sig}(m))$ to receiver.

Input: Receiver: Hash function H, public key PK of sender, received packet (m', x).
Output: Proof that Message m' originates from sender.

(1) Compute $y = Dec_{PK}(x)$ by decrypting the received signature x using the public key of sender.
(2) Compute hash value $H(m')$ of received message m'.
(3) **if** $y == H(m')$ **then**
accept signature
(4) **else**
reject signature
(5) **end if**

Algorithm 3: General digital signature procedure.

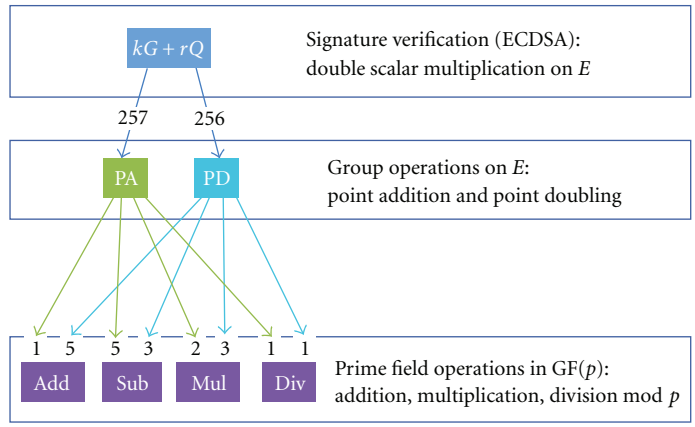

Figure 1: Execution layers of double scalar multiplication on E. On each layer the numbers of operations are given that are needed for a single operation on the respective upper layer.

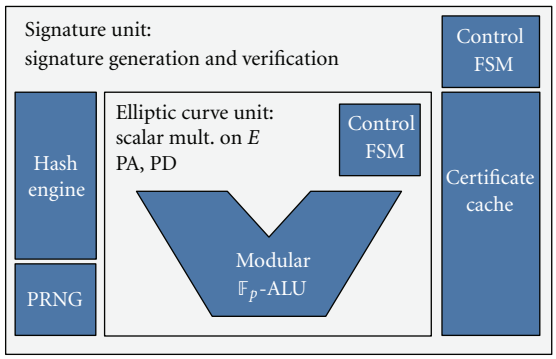

Figure 2: Overview of the signature system.

the modules are 256 bit wide so that complete operands up to 256-bit width (as in the *p256* case) can be stored and transmitted within a single clock cycle. Four inputs, two outputs, and four combined operand/result register as well as a flexible interconnect allow a start of two operations each at the same time as long as they do not use the same basic arithmetic units. The units perform operations independently, so that using different starting points parallel execution in all four subunits is possible. This allows parallelisation especially in the scalar multiplication (see Section 5.2.1).

The $\mathbb{F}_p$-adder and -subtractor perform each operation in a single clock cycle as a general addition/subtraction with subsequent reduction. The $\mathbb{F}_p$ multiplying module computes the modular multiplication iteratively as shift-and-add with reduction mod p in every step. It therefore needs $|p|$ clock cycles for one modular multiplication, $|p|$ being the bit length of the modulus and thereby also the maximum bit length of the operands.

Modular inversion and division is the most complex task of the ALU. It is based on a binary division algorithm on $\mathbb{F}_p$; see [14] for details. The runtime depends on the input values,

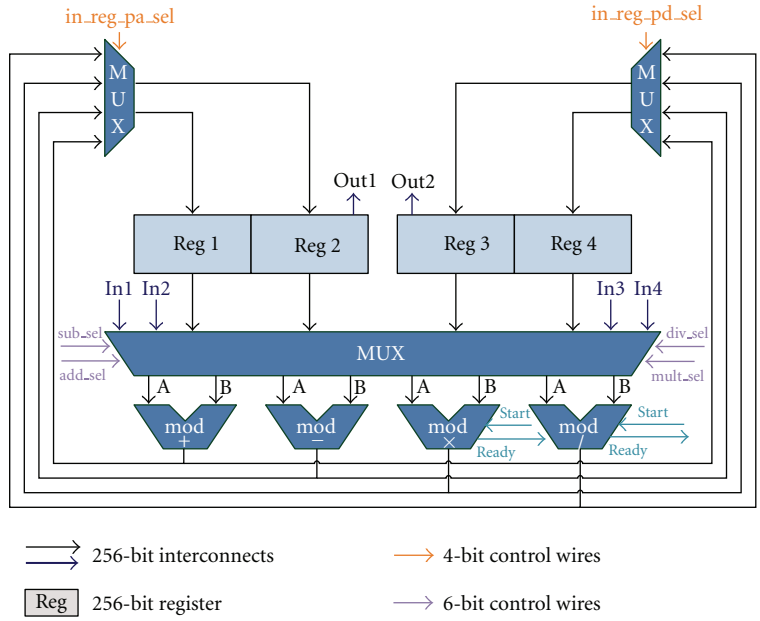

Figure 3: Schematic overview of the $\mathbb{F}_p$ ALU.

Table 1: Hardware execution of point addition.

Step	$\mathbb{F}_p$ unit	No. of cycles
(1) $t_1 = y_2 - y_1$	Sub	1
(2) $t_2 = x_2 - x_1$	Sub	1
(3) $t_2 = t_1/t_2 (= \lambda)$; $t_3 = x_1 + x_2$	div; add	max. $2\lvert p\rvert$
(4) $t_1 = t_2 \cdot t_2$	Mult	$\lvert p\rvert$
(5) $t_1 = t_1 - t_3 (= x_3)$	Sub	1
(6) $t_1 = x_1 - t_1$	Sub	1
(7) $t_1 = t_2 \cdot t_1$	Mult	$\lvert p\rvert$
(8) $t_1 = t_1 - y_1 (= y_3)$	Sub	1
		max. $4\lvert p\rvert + 5$

Table 2: Hardware execution of point doubling.

Step	$\mathbb{F}_p$ unit	No. of cycles
(1) $t_1 = x_1 \cdot x_1$	Mult	$\lvert p\rvert$
(2) $t_2 = t_1 + t_1$	Add	1
(3) $t_1 = t_1 + t_2$	Add	1
(4) $t_1 = t_1 + a$	Add	1
(5) $t_2 = y_1 + y_1$	Add	1
(6) $t_2 = t_1/t_2 (= \lambda)$; $t_3 = x_1 + x_1$	div; add	max. $2\lvert p\rvert$
(7) $t_1 = t_2 \cdot t_2$	Mult	$\lvert p\rvert$
(8) $t_1 = t_1 - t_3 (= x_3)$	Sub	1
(9) $t_1 = x_1 - t_1$	Sub	1
(10) $t_1 = t_2 \cdot t_1$	Mult	$\lvert p\rvert$
(11) $t_1 = t_1 - y_1 (= y_3)$	Sub	1
		max. $5\lvert p\rvert + 7$

maximum runtime being $2\lvert p\rvert$ clock cycles, in the *p256* case therefore up to 512 cycles. Statistical analysis showed an average runtime of $1.5 \cdot \lvert p\rvert$ clock cycles.

ALU control is performed over multiplexer and module control wires and is implemented as a finite state machine presented in the following paragraph. The complete ALU allocates 14256 LUT/FF pairs in a Xilinx Virtex-5 FPGA and allows a maximum clock frequency of 41.2 MHz (after synthesis).

In addition to the 256-bit arithmetic based on the modulus *p256* the ECDSA unit also implements the arithmetic for modulus *p224*. This is done using the same hardware and is also implemented in the overlaying FSM. Theoretically all moduli up to 256-bit width are supported by the ALU. Nevertheless, in the following, all given data refers to the 256-bit key case. Details on resource consumption and performance values are given in Section 7.

5.2. Elliptic Curve Processing. On the elliptic curve E addition of points is defined as group operation. Doubling of a point is specially implemented as it requires a different computation because general point addition is not defined with operands being equal (see Section 3). A comprehensive introduction to elliptic curve arithmetic including algorithms can be found in [20]. To map the algorithms to the implemented specific ALU, the single operation steps have to be scheduled to the respective units. The operation schedules for point addition and point doubling for execution on the ALU are given in Tables 1 and 2.

In the tables, $\lvert p\rvert$ stands for the bit length of the modulus p. In the case of *p256*, this means $\lvert p\rvert = 256$. The execution schedules map the operations to the executing units using

Input: Point $P \in E$; Integer $k = \sum_{i=0}^{l-1} k_i 2^i$ with $k_i \in \{0, 1\}$ and $k_{l-1} = 1$.
Output: Point $Q = kP \in E$.
(1) $P_1 = P$
(2) $P_2 = 2P$
(3) **for** $i = l - 2$ downto 0 **do**
(4) **if** $k = 0$ **then**
(5) $P_1^{\text{new}} = 2P_1^{\text{old}}$
(6) $P_2^{\text{new}} = P_1^{\text{old}} + P_2^{\text{old}}$
(7) **else**
(8) $P_1^{\text{new}} = P_1^{\text{old}} + P_2^{\text{old}}$
(9) $P_2^{\text{new}} = 2P_2^{\text{old}}$
(10) **end if**
(11) **end for**
(12) **return** $Q = P_1$

Algorithm 4: Scalar multiplication in E.

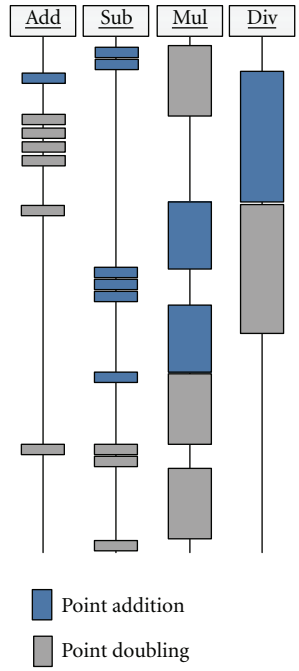

Figure 4: Parallel Scheduling of PA and PD.

three auxiliary register t_1, t_2, t_3 for storing intermediate results. As can be seen in the tables, the third register t_3 is used only once in each point operation and reduces the cycle count in each case by one. If this additional clock cycle is accepted, one 256-bit register can be saved.

5.2.1. Scalar Multiplication on E. Scalar multiplication in step 2 is the central operation of the signature generation of Algorithm 1. Computation is done iteratively using the so-called Montgomery ladder [21, 22] showed in Algorithm 4.

The operations in the branches inside the for-loop, meaning steps 5 and 6 in the if-branch, respectively, steps 8 and 9 in the else-branch, can be executed in parallel. Since it is a point addition and a point doubling each, a real parallel execution on the ALU is possible using a tailored scheduling. Figure 4 depicts the implemented schedule. Although the computation of PA and PD is now done in parallel, a total of five registers for intermediate results is sufficient because the respective t_3 register of PA and PD is not needed at the same time and can therefore be shared.

The execution time using this schedule is $6|p| + 7$ clock cycles for a single pair of point addition and point doubling. Compared to the time of $(4|p| + 5) + (5|p| + 7) = 9|p| + 12$ clock cycles needed for a sequential processing of PA and PD, a performance gain of 33% can be achieved. Execution time for the complete scalar multiplication is therefore at maximum $((|p| - 1) \cdot (6|p| + 7) + (5|p| + 7)) = 6|p|^2 + 6|p|$ clock cycles for the combination of point add and point double.

Input: Point $P, Q \in E$; Integers $k = \sum_{i=0}^{l-1} k_i 2^i$ and $r = \sum_{i=0}^{l-1} r_i 2^i$ with $k_i, r_i \in \{0, 1\}$ and $k_{l-1} \vee r_{l-1} = 1$.
Output: Point $X = kP + rQ \in E$.
(1) Precomputation: $P + Q$
(2) $X = O$ (point at infinity)
(3) **for** $i = l - 2$ downto 0 **do**
(4) $X = 2X$
(5) $X = X + (k_i P + r_i G)$
(6) **end for**
(7) **return** X

Algorithm 5: Simultaneous multiple point multiplication.

5.2.2. Double Scalar Multiplication. For verification of ECDSA signatures two independent scalar multiplications have to be executed (see Algorithm 2, step 7). Instead of computing independently in sequence, it is faster to compute them together using an approach proposed originally by Shamir (see [23]) also known as "Shamir's trick" shown in Algorithm 5.

In contrast to Algorithm 4, the central operations in steps 4 and 5 of Algorithm 5 cannot be parallelized as they depend directly on each other. The maximum time consumption of the algorithm is therefore

$$(4|p| + 5) + |p| \cdot ((5|p| + 7) + (4|p| + 5)) = 9|p|^2 + 16|p| + 5 \quad (5)$$

clock cycles for a double scalar multiplication. This is nevertheless less than the

$$2 \cdot \left(\left(6|p|^2 + 6|p|\right) + (4|p| + 5)\right) = 12|p|^2 + 16|p| + 5 \quad (6)$$

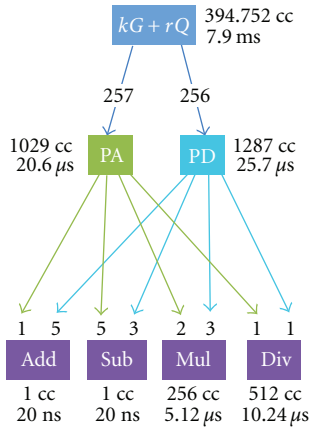

Figure 5: Complexity of execution layers of double scalar multiplication on E. On each level, the number of clock cycles needed for the respective operation is given. The times given refer to a clock frequency of 50 MHz.

cycles that two independent scalar multiplications would consume. Assuming uniform distribution step 5 is omitted in 25% of the cases leaving an estimated runtime of

$$\begin{aligned}&((4|p|+5)+|p|\cdot((5|p|+7)+0.75\cdot(4|p|+5)))\\&\quad=8|p|^2+14.75|p|+5\end{aligned}\tag{7}$$

clock cycles. The composition of the double scalar multiplication on the different levels of computation is shown in Figure 5.

5.3. Signature and Certificate Control System. On top of the elliptic curve (EC) operations and the control FSM performing them, the actual signature algorithms and the certificate verification are implemented. This is done in a separate FSM, controlling the EC arithmetic FSM, some registers, and the auxiliary hashing and random number generation. Figure 6 shows the sequence of operations of the signature verification. See Algorithms 1 and 2 for the implemented procedures.

This FSM is the upmost layer of the signature module and provides a register interface for operands like messages, signatures, certificates, and keys. For integration in an embedded system, it has to be wrapped to support the message format and create the inputs to select the function needed. An example for an integration is given in Section 6.

5.4. SHA2 Hashing Module. The SHA2 hashing unit provides functions SHA-224 and SHA-256 according to the Secure Hash Algorithm (SHA) standard [24]. It is based on a freely available verilog SHA-256 IP-core (available as *SHA IP Core* at http://opencores.com/) adapted with a wrapper performing precomputation of the input data and providing a simple register interface accepting data in 32-bit chunks. In addition the core has been enhanced to support SHA-224.

The unit processes input data in blocks of 512 bit needing 68 clock cycles each at a maximum clock frequency of 120 MHz (after synthesis) and a resource usage of 2277 LUT/FF pairs. After finishing the operation, the result is available in a 256-bit output register.

5.5. Pseudorandom Number Generation. For ECDSA signature generation, a random value k is needed. To provide this k the system incorporates a Pseudorandom Number Generator (PRNG) consisting of two linear feedback shift registers (LFSR), one with 256 bit length, feedback polynomial $x^{255}+x^{251}+x^{246}+1$, and a cycle length of $2^{256}-1$ and a second LFSR with 224 bit length, feedback polynomial $x^{222}+x^{217}+x^{212}+1$, and a cycle length of $2^{224}-1$, both taken from [25].

The LFSR occupies 480 LUT/FF pairs and allows a maximum clocking of 870 MHz although operated in the system in the general system clock of 50 MHz. It is operated continuously to reduce predictability of the produced numbers. The current register content is read out on demand.

For further improvement of the security level, a True Random Number Generator (TRNG) could be integrated. An example implementation of an FPGA-based TRNG can be found in [26].

5.6. Certificate Cache. Usually digital signatures or their respective public keys needed for verification are endorsed by a certificate issued by a trusted third party, a so-called certification authority (CA), to prove its authenticity. Verification of the certificate requires a signature verification itself and is therefore equally complex as the main signature verification of the message. If communicating several messages with the same communication peer using the same signature key, the certificate can be stored hence saving the effort for repetitive verification.

The system incorporates a certificate cache for up to 81 certificates stored in two BRAM blocks. It can be searched in parallel with the signature verification (see Figure 6). Replacement of certificates is performed using a least recently used (LRU) policy.

6. Application Example

The system offers complete ECDSA signature and certificate handling and can be used in a variety of embedded systems seeking authentication and security of communication. As an application example, we show the integration into a vehicle-to-X (V2X) communication system. V2X communication is an emerging topic aiming at information exchange between vehicles on the road and between vehicles and infrastructure like roadside units [27]. This can be used to enhance safety on roads, optimize traffic flow, and help to avoid traffic congestions [28]. Usually two types of broadcasted messages are used, a network beacon sent regularly with a frequency of 2–10 Hz containing status information of the sender and additional event-triggered messages notifying about special events and situations. Latter messages can also be forwarded over several hops to reach receivers outside the direct wireless communication range.

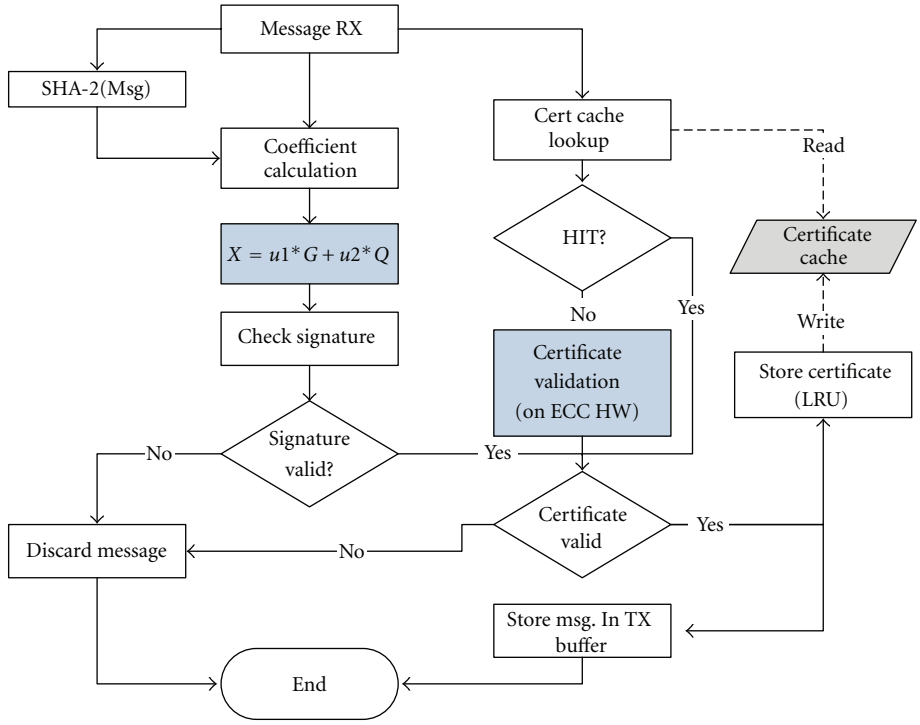

Figure 6: Procedure for signature and certificate verification on the implemented ALU. The blue states mark the main steps.

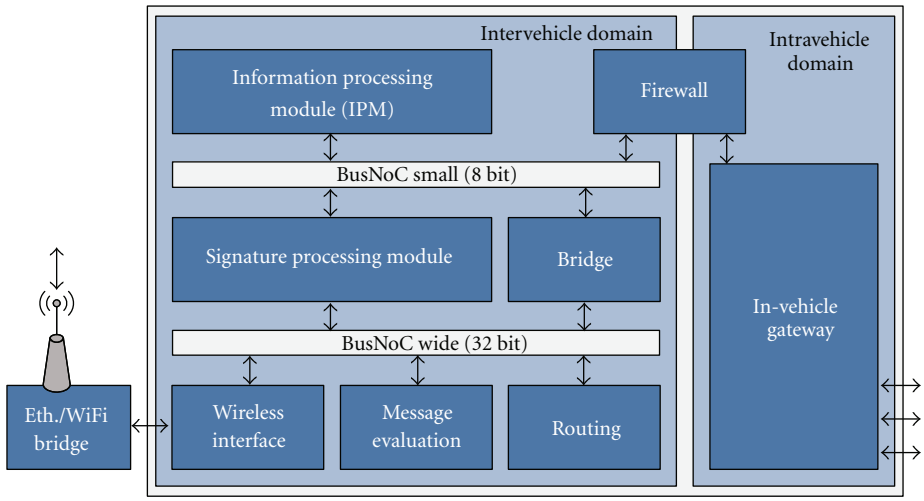

Figure 7: Schematic overview of the V2X-OBU.

To be able to base decisions and applications on information received from other vehicles, trustworthiness of this information is mandatory. To ensure the validity and authenticity of information, signature schemes are used to protect the messages broadcasted by the participating vehicles against malicious attacks [29, 30]. As V2X communication is at present in the process of standardization, no fixed settings are available yet, but the use of ECDSA is proposed in the IEEE 1609 Wireless Access in Vehicular Environments (WAVE) standard draft [31] as well as the proposals of European consortia [32], put together by the COMeSafety project [33].

In the chosen realization V2X communication is performed by a modular FPGA-based On Board-Unit (OBU) presented in [34]; see Figure 7.

It consists of different functional modules connected by a packet-based on-chip communication system [35]. The signature verification system is integrated as a submodule and performs signature handling for incoming and outgoing messages automatically, being therefore transparent to the other modules except for the unavoidable processing latency. It is connected to two different on-chip communication systems, one transmitting unsecured messages over an 8-bit wide communication structure (BusNoC small), and the

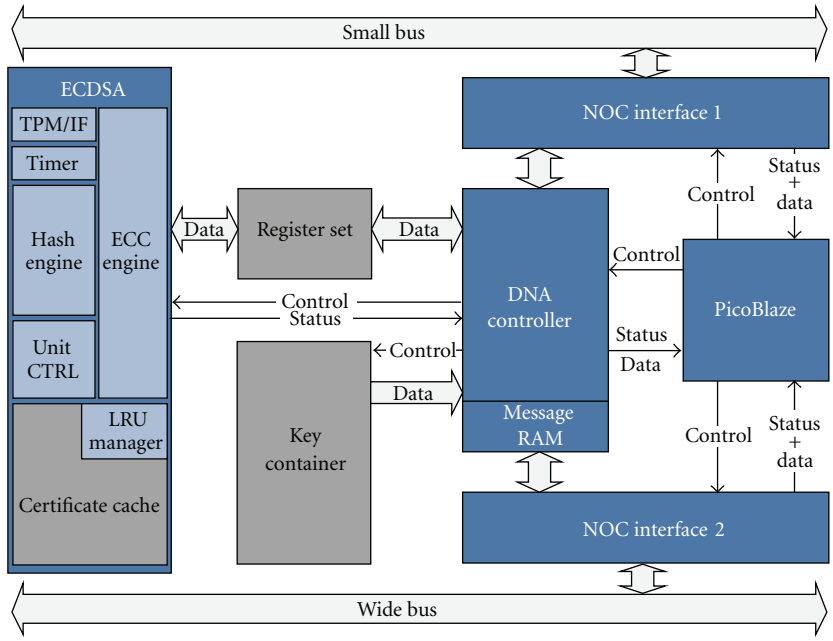

Figure 8: Wrapping of the signature system for V2X integration.

Table 3: Resource usage on an XC5VLX110T with 69.120 LUTs.

	LUT-FF Pairs (synthesis)	Rel. res. usage on FPGA	Max. frequency (MHz)
Signature unit	32.299	46.7%	50
ECDSA unit	24.637	36%	50.1
Hashing unit	2.277	3%	120.8
PRNG	482	0.7%	872.6
$\mathbb{F}_p$-ALU	14.256	20%	41.2
$\mathbb{F}_p$-ADD	858	1.2%	83
$\mathbb{F}_p$-SUB	857	1.2%	92.8
$\mathbb{F}_p$-MUL	2.320	3.4%	42.3
$\mathbb{F}_p$-DIV	5.670	8.2%	73.4

Table 4: Performance of signature verification at 50 MHz.

Verification		secp224r1	secp256r1
Compute time	Worst case	7.23	9.42
(ms/Sig)	Simulated	7.17	9.09
Throughput	Worst case	138	106
(Sig/s)	Simulated	140	110
Latency	Worst case	361.151	471.111
(Cycles/Sig)	Simulated	358.478	454.208

Table 5: Performance of signature generation at 50 MHz.

Generation		secp224r1	secp256r1
Compute time	Worst case	5.56	7.26
(ms/Sig)	Simulated	5.45	7.15
Throughput	Worst case	180	138
(Sig/s)	Simulated	184	140
Latency	Worst case	278.097	362.881
(Cycles/Sig)	Simulated	272.345	357.315

other (BusNoC wide) transmitting only secured messages containing signatures and certificates. These messages are larger because of the additional data, and the latter communication structure is therefore 32 bit wide. A short description of the security system and its system integration is given in [36]. Figure 8 depicts the wrapped signature system with the interfacing to both communication structures.

This interfacing consists of a Direct Network Access (DNA) controller and two interfaces to the Network-on-Chip (NoC) communication structures. An 8-bit PicoBlaze processor controls and configures the components. The DNA controller manages the intramodular procedure and generates the input and control data to the encapsulated ECDSA module. The register set serves as data interface and buffer for intermediate results.

The signature system accepts incoming messages, verifies signatures and certificates, and passes only verified messages on to the Information Processing Module (IPM) for further processing. In case of an invalid signature the outer system (IPM and Routing) is informed. For outgoing messages, signatures are generated, and the corresponding certificate is attached to the message which is then passed on to the wireless interface.

6.1. Key Container. In the V2X environment privacy of participants is of major importance. As messages containing

Table 6: Performance comparison for signature verification and generation for ECDSA on GF(p). Values marked with an asterisk (*) are only for the core operations scalar multiplication and multiple scalar multiplication, respectively, without all pre- and postprocessing and hashing.

	Bit length $\lvert p \rvert$	Hardware resources	Clk (MHz)	Generation (kG)		Verification ($Kg + rQ$)	
				Time	#/s	Time	#/s
Microcontroller implementations							
Drutarovsky and Varchola [7]	233	ARM7	25	742 ms	1.35	1240 ms	0.8
RIM [9]	256	ARM9EJ-S	N.a.	168 ms	5.95	150 ms	6.7
PC processor implementations							
eBACS [6]	256	Motorola PowerPC G4 7410	533	11.7 ms	85.2	14.1 ms	70.7
Petit [37]	256	Intel Pentium D	3400	3.33 ms	300	6.63 ms	151
eBACS [6]	256	Intel Atom 330	1600	2.9 ms	345	3.4 ms	294
eBACS [6]	256	Intel Core 2 Duo U9400	1400	1.88 ms	532	2.2 ms	455
Brown et al. [38]	256	Intel Pentium II	400	*1.67 ms	*599	*6.4 ms	*156
FPGA implementations							
McIvor et al. [11]	256	Xilinx Virtex II Pro, 15755 CLB, 256 MUL	39.5	*3.86 ms	*259	N.a.	N.a.
Orlando and Paar [12]	192	Xilinx Virtex-E, 11416 LUT, 35 BRAM	40	*3 ms	*333	N.a.	N.a.
This paper	256	Xilinx Virtex 5, 14256 LUT/FF pairs	20	7.15 ms	140	9.09 ms	110
ASIC implementations							
Sakiyama et al. [16]	256	243K gates (0.25 μm)	159	*4.3 ms	*233	N.a.	N.a.
Satoh and Takano [15]	256	120K gates (0.13 μm)	138	*2.68 ms	*373	N.a.	N.a.

vehicle type and further information like current position, speed, and heading are continuously broadcasted from twice to up to ten times a second, these messages could easily be used by an eavesdropper to trace participants. To counter such attempts anonymity in the form of pseudonyms is used that are changed on a regular basis. A number of pseudonyms for change are stored directly in the signature module's key container (see Figure 8). It also contains the public keys of trusted certification authorities needed for verification of certificates. The change itself is triggered by a dedicated message sent to the signature processing system by the central information processing module of the C2X system. For all other modules this privacy function is fully transparent as well.

6.2. Caching of Certificates. As V2X communication is not deployed in the fleet so far and also realistic field tests with larger numbers of vehicles are only just beginning (e.g., simTD [39] in Germany), large-scale predictions of message numbers and network behaviour have to be based on simulations and estimations. For an estimation of the expected cache hit rate results from the literature are used. Seada [40] show based on real-world measurements on American freeways that the average communication time between two vehicles is approximately 65 seconds. Based on that and assuming a beaconing frequency of 10 Hz and a sufficient cache size in only one out of 650 messages, the certificate has to be validated. In addition pseudonym change has to be regarded. Papadimitratos et al. [41] propose exchange of pseudonyms every 60 seconds. Assuming stochastical independence of both values, a cache hit rate of 99.68% is possible. Since the communication is regular while the peer vehicle is in range, an LRU strategy is suitable. The required cache size depends strongly on the number of vehicles in range and should therefore be adapted to the expected situations.

7. Resources and Performance

The presented system has been realized using a Xilinx XC5VLX110T Virtex-5 FPGA [42] on a Digilent XUP ML509 evaluation board [43]. The following values refer to an implementation of the complete signature generation and verification unit with interfacing for the application example given previously. Table 3 shows an overview of the resource usage.

After integration of all submodules, the ECDSA unit allows a maximum clock frequency of 50 MHz that has been successfully tested. Table 4 shows signature verification performance values of the ECDSA unit at 50 MHz. Values for signature generation are given in Table 5.

In both tables the *worst case* values given are calculations based on the statistically estimated runtime of the algorithms for scalar multiplication. As these runtimes depend on the operand values, the measured average computation times are different.

Direct comparison of the system's performance is difficult, because implementations of complete ECDSA signature and verification units with certificate handling are scarce. So we can only compare the performance of the GF(p) processing unit, where values are available. Table 6 gives an overview in comparison to some implementations presented already in Section 2.

8. Discussion

The presented system implements the complete ECDSA signature processing in a modular way. As shown in the application example, it can be integrated as an autonomous subsystem to authenticate message traffic and provide verified information to the overlaying system. In comparison to known full implementations (see Section 2), the system's performance of up to 110 verifications per second is by one to two orders of magnitude better than software implementations on microcontrollers, providing sufficient performance for most applications. For high-performance applications like the V2X application example given in detail in Section 6, a still higher throughput of up to 1600 [41, 44], respectively, over 2500 [45] signatures per second is needed though. This can be achieved by a number of optimization steps; see Section 9.

The complete signature module from Section 6 is nevertheless prepared for further improvements. As can be seen in Figure 8, the ECDSA system is encapsulated as a submodule wrapped by the control and communication system that fits to the external system structure. The ECDSA system can therefore easily be replaced by a more performant system without having to adapt the overall system structure.

9. Conclusion and Further Work

We presented a hardware-implemented subsystem for ECDSA signature processing for integration into embedded systems based on reconfigurable hardware. It can be integrated as a stand-alone subsystem performing transparent authentication functionality for communication systems. Applicability of the system has been shown using vehicle-to-X communication as a practical example.

The performance values presented in Section 7 are sufficient for applications like entry control systems or electronic payment, where the number of communication peers is small. For V2X communication even larger throughput is necessary. Further work therefore includes speeding up the computation. Promising approaches that are subject to ongoing work here are the use of windowing techniques on algorithmic level, the tailored use of optimized representations like projective coordinates on mathematical level, and the speedup of the field operations on implementation level, for example, by the use of hardware multipliers. Also the use of low-cost FPGAs and reduction of the footprint is required for the use in embedded systems.

References

[1] FIPS, "Pub 197: Advanced Encryption Standard (AES)," Federal information processing standards publication, U.S. Department of Commerce, Information Technology Laboratory (ITL), National Institute of Standards and Technology (NIST), Gaithersburg, Md, USA, 2001.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] FIPS, *Pub 186-3: Digital signature standard (dss)*, Federal information processing standards publication, U.S. Department of Commerce, Information Technology Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, Md, USA, 2009.

[4] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[5] V. S. Miller, "Use of elliptic curves in cryptography," in *Proceedings of the Advances in Cryptology (CRYPTO '85)*, pp. 417–426, 1986.

[6] eBACS, "ECRYPT Benchmarking of Cryptographic Systems," 2010, http://bench.cr.yp.to/ebats.html.

[7] M. Drutarovsky and M. Varchola, "Cryptographic system on a chip based on actel ARM7 soft-core with embedded true random number generator," in *Proceedings of the 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS '08)*, pp. 164–169, IEEE Computer Society, Washington, DC, USA, 2008.

[8] RIM Blackberry 7230, "Datasheet," 2010, http://pdadb.net/index.php?m=specs&id=1467&view=1&c=rim.

[9] D. Hankerson, "Implementing elliptic curve cryptography (a narrow survey)," in *Proceedings of the Workshop in Implementation of Cryptographic Methods (WIMC '05)*, 2005.

[10] G. Meurice de Dormale and J. J. Quisquater, "High-speed hardware implementations of Elliptic Curve Cryptography: a survey," *Journal of Systems Architecture*, vol. 53, no. 2-3, pp. 72–84, 2007.

[11] C. J. McIvor, M. McLoone, and J. V. McCanny, "Hardware elliptic curve cryptographic processor over GF(p)," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 9, pp. 1946–1957, 2006.

[12] G. Orlando and C. Paar, "A scalable gf(p) elliptic curve processor architecture for programmable hardware," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES '01)*, vol. 2162 of *Lecture Notes in Computer Science*, pp. 348–363, 2001.

[13] T. Güneysu and C. Paar, "Ultra high performance ecc over nist primes on commercial fpgas," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES '08)*, Lecture Notes in Computer Science, pp. 62–78, Washington, DC, USA, 2008.

[14] S. Ghosh, M. Alam, I. S. Gupta, and D. R. Chowdhury, "A robust GF(p) parallel arithmetic unit for public key cryptography," in *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD '07)*, pp. 109–115, Washington, DC, USA, 2007.

[15] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 449–460, 2003.

[16] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF (2^n)," *IEEE Transactions on Computers*, vol. 56, no. 9, pp. 1269–1282, 2007.

[17] K. Järvinen and J. Skyttä, "Cryptoprocessor for Elliptic Curve Digital Signature Algorithm (ECDSA)," Tech. Rep., Helsinki University of Technology, Signal Processing Laboratory, 2007.

[18] NIST, "Recommended elliptic curves for federal government use," Tech. Rep., National Institute of Standards and Technology, U.S. Department of Commerce, 1999.

[19] Certicom Research, "Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters," 2000.

[20] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, New York, NY, USA, 2004.

[21] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[22] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 177, pp. 243–264, 1987.

[23] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, 1985.

[24] FIPS, *Pub 180-2: Secure hash standard (shs)*, Federal information processing standards publication, U.S. Department of Commerce, National Institute of Standards and Technology (NIST), Gaithersburg, USA, 2002.

[25] R. Ward and T. Molteno, "Table of Linear Feedback Shift Registers," 2007, http://www.otagophysics.ac.nz/px/research/electronics/papers/technical-reports/lfsr_table.pdf.

[26] D. Schellekens, B. Preneel, and I. Verbauwhede, "FPGA vendor agnostic true random number generator," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 139–144, August 2006.

[27] "CAR 2 CAR Communication Consortium, *Manifesto—Overview of the C2C-CC System v1.1*," 2007.

[28] Commission of the European Communities, "European transport policy for 2010: time to decide," white paper com 370 final, 2001, http://ec.europa.eu/transport/strategies/doc/2001_white_paper/lb_com_2001_0370_en.pdf.

[29] P. Papadimitratos, L. Buttyan, T. Holczer et al., "Secure vehicular communication systems: design and architecture," *IEEE Communications Magazine*, vol. 46, no. 11, pp. 100–109, 2008.

[30] F. Kargl, P. Papadimitratos, L. Buttyan et al., "Secure vehicular communication systems: design and architecture," *IEEE Communications Magazine*, vol. 46, no. 11, pp. 110–118, 2008.

[31] IEEE Vehicular Technology Society, ITS Committee, *IEEE Trial-Use Standard for Wireless Access in Vehicular Environments (WAVE)—Security Services for Applications and Management Messages*, 2006.

[32] COMeSafety Project, "European ITS Communication Architecture—Overall Framework," 2008, http://www.comesafety.org/.

[33] COMeSafety Project—Communication for eSafety, "Project website," 2010, http://www.comesafety.org/.

[34] O. Sander, B. Glas, C. Roth, J. Becker, and K. D. Müller-Glaser, "Design of a vehicle-to-vehicle communication system on reconfigurable hardware," in *Proceedings of the International Conference on Field-Programmable Technology (FPT '09)*, pp. 14–21, IEEE, 2009.

[35] O. Sander, B. Glas, C. Roth, J. Becker, and K. D. Müller-Glaser, "Priority-based packet communication on a bus-shaped structure for FPGA-systems," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '09)*, pp. 178–183, April 2009.

[36] B. Glas, O. Sander, V. Stuckert, K. D. Müller-Glaser, and J. Becker, "Car-to-car communication security on reconfigurable hardware," in *Proceedings of the IEEE 69th Vehicular Technology Conference (VTC '09)*, Barcelona, Spain, 2009.

[37] J. Petit, "Analysis of ecdsa authentication processing in vanets," in *Proceedings of the 3rd International Conference on New Technologies, Mobility and Security (NTMS '09)*, pp. 388–392, IEEE Press, Piscataway, NJ, USA, 2009.

[38] M. Brown, D. Hankerson, and A. Menezes, "Software implementation of the nist elliptic curves over prime fields," in *Proceedings of the Topics in Cryptology (CT-RSA '01)*, vol. 2020 of *Lecture Notes in Computer Science*, pp. 250–265, Springer, Berlin, Germany, 2001.

[39] simTD, "Sichere Intelligente Mobilität: Testfeld Deutschland. Project webpage," 2008.

[40] K. Seada, "Insights from a freeway car-to-car real-world experiment," in *Proceedings of the 3rd ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH '08)*, pp. 49–55, ACM, New York, NY, USA, 2008.

[41] P. Papadimitratos, G. Calandriello, J. P. Hubaux, and A. Lioy, "Impact of vehicular communications security on transportation safety," in *Proceedings of the IEEE INFOCOM Workshops*, April 2008.

[42] Xilinx Inc., *UG190: Virtex-5 FPGA User Guide. v5.2*, 2009.

[43] Xilinx Inc., *UG347: ML505/ML506/ML507 Evaluation Platform—User Guide, 2009. v3.1.1*, 2009.

[44] Q. Xu, T. Mak, J. Ko, and R. Sengupta, "Vehicle-to-vehicle safety messaging in DSRC," in *Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks (VANET'04)*, pp. 19–28, October 2004.

[45] M. Torrent Moreno, *Inter-vehicle communications: achieving safety in a distributed wireless environment*, Dissertation, Shaker, 2007.