Low-Latency ECDSA Signature Verification – A Road Towards Safer Traffic –

Miroslav Knežević, Ventzislav Nikov, and Peter Rombouts

NXP Semiconductors

Abstract. Car-to-car and Car-to-Infrastructure messages exchanged in Intelligent Transportation Systems can reach reception rates up to and over 1000 messages per second. As these messages contain ECDSA signatures this puts a very heavy load onto the verification hardware. In fact the load is so high that currently it can only be achieved by implementations running on high end CPUs and FPGAs. These implementations are far from cost-effective nor energy efficient. In this paper we present an ASIC implementation of a dedicated ECDSA verification engine that can reach verification rates of up to 27.000 verifications per second using only 1.034 kGE.

Keywords: ECDSA, signature verification, NIST P-256, Car2Car, Car2X, ITS.

1 Introduction

Everyone nowadays is familiar with the problems caused by the rising traffic densities on our roads and in our cities. Traffic congestion is increasingly crippling efficient movement inside and towards cities resulting in mounting travel times, fuel consumption and consequent air pollution. In the past these problems have been addressed by expanding road infrastructure, however most cities nowadays have reached the limit at which further investments in infrastructure become impractical or prohibitively expensive. We have reached a point at which not the infrastructure but the traffic flow itself should become more efficient.

At the same time cars have become more "intelligent". Sensors in the car constantly monitor environmental and road conditions as well as the car's internal systems; braking power is distributed based on road conditions, lights and wipers are switched on automatically and some cars even brake automatically when an obstacle is approached too closely. GPS allows cars and their drivers to know at all times where they are and finding an alternative route is done at the touch of a button. Even when things go wrong the car is aware; when accelerometers detect a crash the air bags are inflated and some cars even call for help.

Intelligent Transportation Systems (ITS) is an umbrella term for all systems used to improve the safety and efficiency of transportation systems, including air, rail, roads and waterways. ITS make use of a variety of communication, computing, control and sensing technologies to make traffic information available in real time, enabling transport network and vehicle operators alike to respond more quickly to changing circumstances and even to anticipate them.

Although the precursors for ITS, the traffic control systems for air and rail traffic were already deployed in the late 1920's [21], the adoption of ITS for road traffic has been much slower. Nevertheless the more recent history has seen a massive adoption of roadside ITS in the form of automatic road enforcement, variable speed limits and automatic road tolling, amongst others. The further integration of cars and their advanced sensing capabilities into ITS is then only a logical next step, giving rise to Cooperative-ITS (C-ITS) in which information about the vehicles, their location and the road environment can be shared amongst vehicles and with the infrastructure.

The cornerstone for C-ITS is wireless Car-to-Car (C2C) and Car-to-Infrastructure (C2I) communication to transmit all this data coming from cars and infrastructure in a reliable way. An important aspect in this is the authenticity of the messages exchanged in C-ITS, as faked messages from e.g. priority vehicles could otherwise be (ab)used to influence traffic lights and cause disturbances to the traffic flow, for instance. To this end both the European (ETSI 103 097 [7]) and US (IEEE 1609.2 [11]) ITS security standards mandate

the use of digital signatures in the safety messages. In order to limit the impact on the message size the standards have opted to use Elliptic Curve based signatures over RSA based signatures. The mandated signature algorithm is ECDSA with curve P-256 [17] and the remainder of this work will focus on it.

Performance Requirements. Figure 1 represents a typical dense traffic situation where a car is surrounded by many other cars that are constantly moving in and out its transmission range. Now, consider this scenario occurring on a five- or six-level urban stack interchange with traffic moving on at least three lanes in each direction. It is clear to see that this easily brings us to a situation where the car is surrounded by several hundred vehicles, resulting in a large flood of Cooperative Awareness Messages (CAM) [6] to be verified every second. Assuming the channel bandwidth limitation (6 Mb/s, 256 byte messages and channel utilization of 33 %), the number of messages is limited to around $1,000$ per second. This throughput can only be offered in a cost effective way by a highly efficient ECDSA signature verification engine such as the one developed in this work.

However the latency is more important than throughput in the considered use case where the time to react on a message is very important. Examples are high-priority messages police, ambulance, etc. In fact, it is really crucial that these messages get processed very fast. Therefore we would like to stress that in this paper we focus on optimizing latency and not throughput, and they differ for example multiple-core solutions are not suitable. Moreover our goal is to achieve this very low latency in reasonable area.

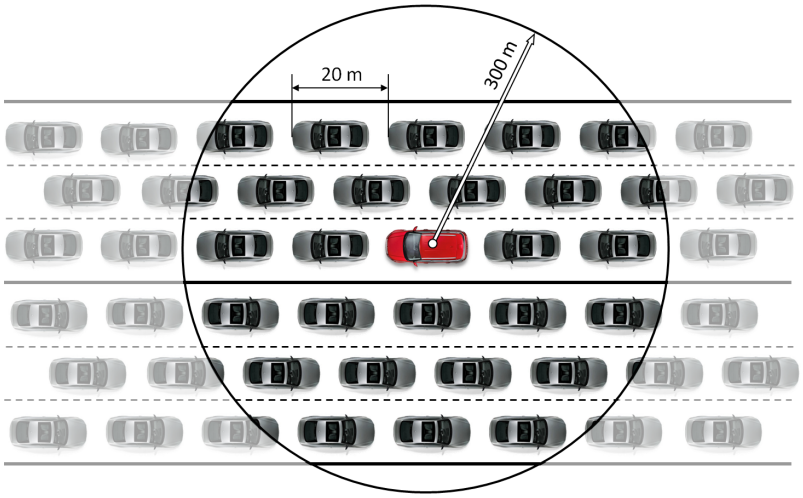

Fig. 1: Communication radius in a dense traffic (dimensions are not scaled proportionally).

Our Contribution. We introduce a high-speed ECDSA signature verification core over the NIST P-256 curve. The systematic approach and significant optimization efforts on every single level result in an architecture that substantially outperforms all the reported results available in literature. We prototype a core that satisfies the throughput requirement of $1,000$ verifications per second with minimal area and additionally perform a thorough exploration of the complete design space, which yields several more designs achieving outstanding performance.

Our goal was to design an ECDSA Signature Verification engine with very low latency, in that respect we have achieved the best results ever published, using an industry compliant technology (worst case CMOS65). We did consider some of the newer implementation techniques but also missed many. Our investigations however show that the basic "well-known" approaches are sufficient and maybe the best for ASIC (HW likes simplicity). Our approach is similar to that of Güneysu and Paar [9] (CHES08) but we

focused much more on optimizing the heaviest operation, finite field multiplication and we introduced a novel approach of SAU.

Organization of the Paper. We start by addressing the related work in Section 2 and outlining some preliminaries in Section 3. The whole design process and design space exploration is discussed in Section 4. The implementation details and results are given in Section 5 and we finally conclude in Section 6.

2 Related Work

Only very few low-latency implementations of full ECDSA signature verification have been reported in literature. Many more address the performance issues of the most common operation in ECC; the point multiplication. In order to have a meaningful overview and for easier comparison with our work, we outline only those studies that discuss performance of either point multiplication or full ECDSA signature verification/generation for elliptic curves defined over a 256-bit prime field. In what comes next, we assume that the elliptic curve arithmetic is always done in that finite field.

A very extensive study, covering many aspects of efficient software implementation of the NIST elliptic curves over prime fields is provided by Brown et al. [2]. By implementing the finite field arithmetic in assembly language, the authors achieved very competitive results, for that time, on a Pentium II 400 MHz workstation. A multiple point multiplication on the NIST P-256 curve took their implementation 6.4 ms.

McIvor et al. [13] presented an FPGA architecture for performing the arithmetic functions needed in elliptic curve cryptographic primitives over GF(p). Using the embedded 18×18-bit multipliers and fast carry look-ahead logic available on the Xilinx Virtex2 Pro family of FPGA (XC2VP125-7) they were able to tailor the architecture to perform a single point multiplication in 3.84 ms. Their implementation runs at 39.5 MHz and uses 15,755 slices and 256 multipliers in total.

In her dissertation, Mentens [14] reports an extension of the ECC coprocessor which was previously described by Sakiyama et al. [18]. Running at a frequency of 67 MHz and using 3,529 slices and 36 embedded multipliers of the XC2VP30 development board, their implementation achieves a latency of 2.27 ms for a single point multiplication. The design builds upon a pipelined architecture for the classical multi-precision algorithm making use of long word additions and carry-lookahead addition, as well as on the use of non-adjacent form (NAF) recoding.

Güneysu and Paar [9] describe an ultra fast ECC implementation over two NIST primes on commercial FPGAs. Their design benefits from intensive use of the DSP blocks available in modern FPGAs and which are especially well suited for implementation of the field operations. The paper reports a latency of 749 μs for a multiple point multiplication over the NIST P-256 curve. Furthermore, the authors claim even faster multiplication is possible when a windowing method is used. Their estimate of 495 μs comes at the price of an additional 1,715 logic slices (LS) and 32 digital signal processing (DSP) units being used on the XC4VFX12-12 Xilinx FPGA board.

Schinianakis et al. [19] take a different approach and explore the use of residue number systems (RNS) for building efficient finite field arithmetic operations. Occupying 36,000 lookup tables (LUT) on the XCV1000E-8 Xilinx FPGA board and running at the frequency of 39.7 MHz, their implementation requires 3.95 ms for a single point multiplication.

The work of Guillermin [8] is the fastest among the public hardware designs to compute scalar multiplication for elliptic curves defined over a general prime ground field. The design is also based on RNS, guaranteeing carry-free arithmetic and easy parallelism. Consuming 9,117 adaptive logic modules (ALM) on the EP2S30F484C3 Altera FPGA device and running at the frequency of 157 MHz, their implementation takes 680 μs for a single point multiplication.

Very detailed reports on software performance of many different cryptographic algorithms can be found at eBACS – the ECRYPT benchmarking platform of cryptographic systems [4]. Currently, the fastest implementation of a complete ECDSA signature generation over the NIST P-256 prime achieves a latency

of only 427 μs. It comes as a surprise that this implementation, running at a frequency of 3.5 GHz on four 64-bit cores of Intel Xeon E3 processor, is able to achieve a result that is better than any previously reported implementation in hardware.

3 Preliminaries

ECDSA Signature Verification. Algorithm 1 outlines basic steps of the ECDSA signature verification. The finite field is defined over the prime p and the order of the elliptic curve is denoted with n. With $P(x_P, y_P)$ we define the base point and its coordinates and with ∞ we define the point at infinity. The most computationally expensive operation of the algorithm is the computation of $X = u_1P + u_2Q$, very often referred to as a *multiple point multiplication*.

Algorithm 1 ECDSA Signature Verification.

INPUT: Curve Parameters $D = (p, n, P(x_P, y_P))$, Public Key $Q(x_Q, y_Q)$, Message Hash e, Signature (r, s).
OUTPUT: ACCEPT or REJECT the signature.

1: **if** $r, s \notin [1, n-1]$ **then** REJECT.
2: $w = s^{-1} \bmod n$.
3: $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
4: $X = u_1P + u_2Q$.
5: **if** $X(x, y) = \infty$ **then** REJECT.
6: **if** $x \bmod n = r$ **then** ACCEPT **else** REJECT.

Shift-and-Add Finite Field Multiplication. A classical shift-and-add multiplication in finite field is outlined in Alg. 2. Step 4 of the algorithm involves an integer division and is the most computationally demanding part of it. In case M is of a special form this step can be performed rather efficiently. Next, we demonstrate this by using an example of the NIST recommended prime. Inputs are the two l-bit integers A and B, and their product $C = AB \bmod M$ is the output.

Algorithm 2 Shift-and-Add Finite Field Multiplication.

INPUT: $A = (A_{l_w-1} \dots A_0)_r$, $B = (B_{l_w-1} \dots B_0)_r$, $M = (M_{l_w-1} \dots M_0)_r$ where $0 \leq A, B < M$, $2^{l-1} \leq M < 2^l$, $r = 2^w$ and $l_w = \lceil l/w \rceil$.

OUTPUT: $C = AB \bmod M$.
1: $C = 0$
2: **for** $i = l_w - 1$ downto 0 **do**
3: $\quad C = Cr + AB_i$
4: $\quad q_C = \lfloor C/M \rfloor$
5: $\quad C = C - q_C M$
6: **end for**
7: Return C.

NIST Modular Reduction with P-256. Due to a computationally complex operation of integer division, modular reduction is considered to be an expensive task in hardware. Advanced algorithms such

as the ones of Montgomery [15] and Barrett [1] are typically used for improving the efficiency of many consecutive reductions with any general modulus. However, when the modulus is of a special form, the modular reduction becomes rather simplified. NIST has therefore selected a set of special primes that permit a very fast reduction [17]. The prime we are interested in is a 256-bit prime denoted as P-256 and equal to $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. Algorithm 3 outlines the reduction procedure with this special modulus.

Algorithm 3 Modular reduction with P-256 $= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

INPUT: Integer $A = (A_{15}, \ldots, A_2, A_1, A_0)$ where $0 \leq A_i < 2^{32}$.
OUTPUT: $B = A \bmod$ P-256.

$C_1 = (A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$, $C_2 = (A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0)$,
$C_3 = (0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0)$, $C_4 = (A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8)$,
$C_5 = (A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9)$, $C_6 = (A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11})$,
$C_7 = (A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12})$, $C_8 = (A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13})$,
$C_9 = (A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14})$.

$B = C_1 + 2C_2 + 2C_3 + C_4 + C_5 – C_6 – C_7 – C_8 – C_9 \bmod$ P-256.

Karatsuba Multiplication. One of the very famous multiplication algorithms that efficiently improves the speed of classical multiplication was discovered by Karatsuba [12] and published back in 1962. Let x and y be the two l-bit numbers and let $k = \lceil l/2 \rceil$. One-level deep Karatsuba algorithm is given as follows:

$$\begin{aligned} xy &= (x_H 2^k + x_L)(y_H 2^k + y_L) \\ &= x_H y_H 2^{2k} + \big((x_H + x_L)(y_H + y_L) - x_H y_H - x_L y_L\big)2^k + x_L y_L \ . \end{aligned}$$

The two-level deep Karatsuba is obtained when the algorithm is applied recursively to the intermediate products of one-level deep Karatsuba ($x_H y_H$, $(x_H + x_L)(y_H + y_L)$, and $x_L y_L$). If the algorithm is applied recursively too many times, the increasing number of additions becomes the main bottleneck of the implementation. Additionally, the control logic of the Karatsuba multiplier becomes more complex and, in practice, the benefit ceases compared to the classical multiplication.

4 ECDSA Pyramid

Designing a highly efficient ECDSA verification core can be illustrated with a pyramid, where different layers of abstraction are represented as illustrated in Fig. 2. The selection of a highly efficient finite field multiplier, for instance, will allow faster Point Addition and Point Doubling and therefore make the whole ECDSA verification process faster. This is indeed the reason why finite field arithmetic occupies the basis of the pyramid and, depending on the choice of the underlying algorithms, a path to the higher layers can be more or less efficient. Therefore, optimizations regarding both bottom-up and top-down approaches need to be addressed simultaneously.

4.1 Climbing Up the Pyramid

Multiplicaion in $\mathbb{F}_p$. Forming the basis of the pyramid, it is crucial for the finite field arithmetic to be optimized for low latency. We therefore explore a large set of possible implementations, carefully selecting the most promising architectures for evaluation within the initial phase. Since finite field multiplication is by far the most time-consuming operation, our experiment starts with the synthesis of 3 different architectures,

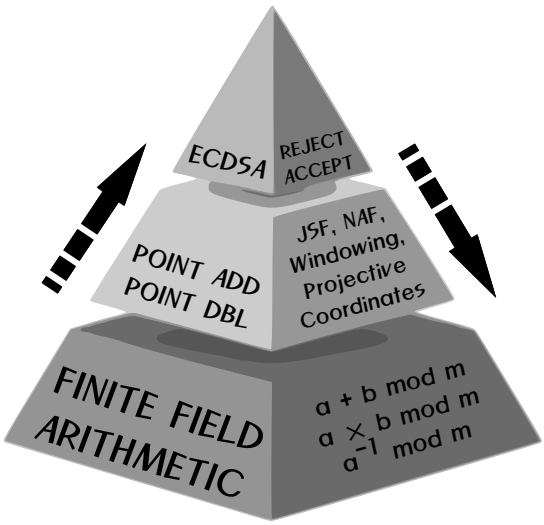

Fig. 2: ECDSA pyramid.

each of which consists of 9 different flavors, depending on the input word size. More precisely, we evaluate different input word sizes for the underlying basic[1] multiplier: 16×32, 16×64, 16×128, 32×32, 32×64, 32×128, 64×64, 64×128, and ultimately 256×256.

Based on each of these basic multipliers, a finite field multiplier is constructed using one of the following 3 architectures:

- MULTIPLY-THEN-REDUCE: The whole product of 512 bits is first computed by utilizing the basic multiplier and a simple shift-and-add algorithm, followed by a reduction as described in Section 3.
- INTERLEAVED: The reduction is interleaved with the multiplication. In other words, a partial result is reduced after each shift-and-add operation. Compared to the MULTIPLY-THEN-REDUCE architecture, the execution of the INTERLEAVED architecture requires a single clock cycle less, but at the same time, the INTERLEAVED architecture has a longer critical path.
- KARATSUBA: We use the trick of Karatsuba to compute the whole product of 512 bits. With this architecture we minimize the number of clock cycles at the expense of a longer critical path. We explore one- and two-level deep Karatsuba multipliers (KARATSUBA-I and KARATSUBA-II).

In order to explore the design space thoroughly, we synthesize every instance[2] of the multiplier at five different target frequencies (50 MHz, 62.5 MHz, 83.3 MHz, 125 MHz, and 250 MHz). Figure 3 illustrates the achieved results. A table containing the complete overview of implementation results contains 640 entries (160 rows) and would not not fit the space requirements. We will therefore make it available upon request. Similarly we synthesized six different architectures for the inverter and Figure 6 illustrates the achieved results.

To achieve the requirement of $1,000$ verifications per second, we estimate the maximum allowed latency of the finite field multiplier to be at most 250 ns for a single multiplication (a detailed calculation is provided in Appendix A). All implementations that achieve a latency under this threshold are marked in gray in Fig. 3. As our final choice, we select the smallest of all candidates, making sure the operating frequency remains as low as possible in order to limit the power consumption. It is the MULTIPLY-THEN-REDUCE architecture with a 64×64 basic multiplier, running at 83.3 MHz, that consumes the minimum area (indicated with the green arrow in Fig. 3). Surprisingly, the best result comes from the most straightforward

[1] Basic means an ordinary integer multiplier that inputs two operands of size m and n bits respectively and outputs the product of $m + n$ bits.

[2] An instance is a version of the finite field multiplier with a specific combination of the basic multiplier, meaning specific input word sizes, and architecture.

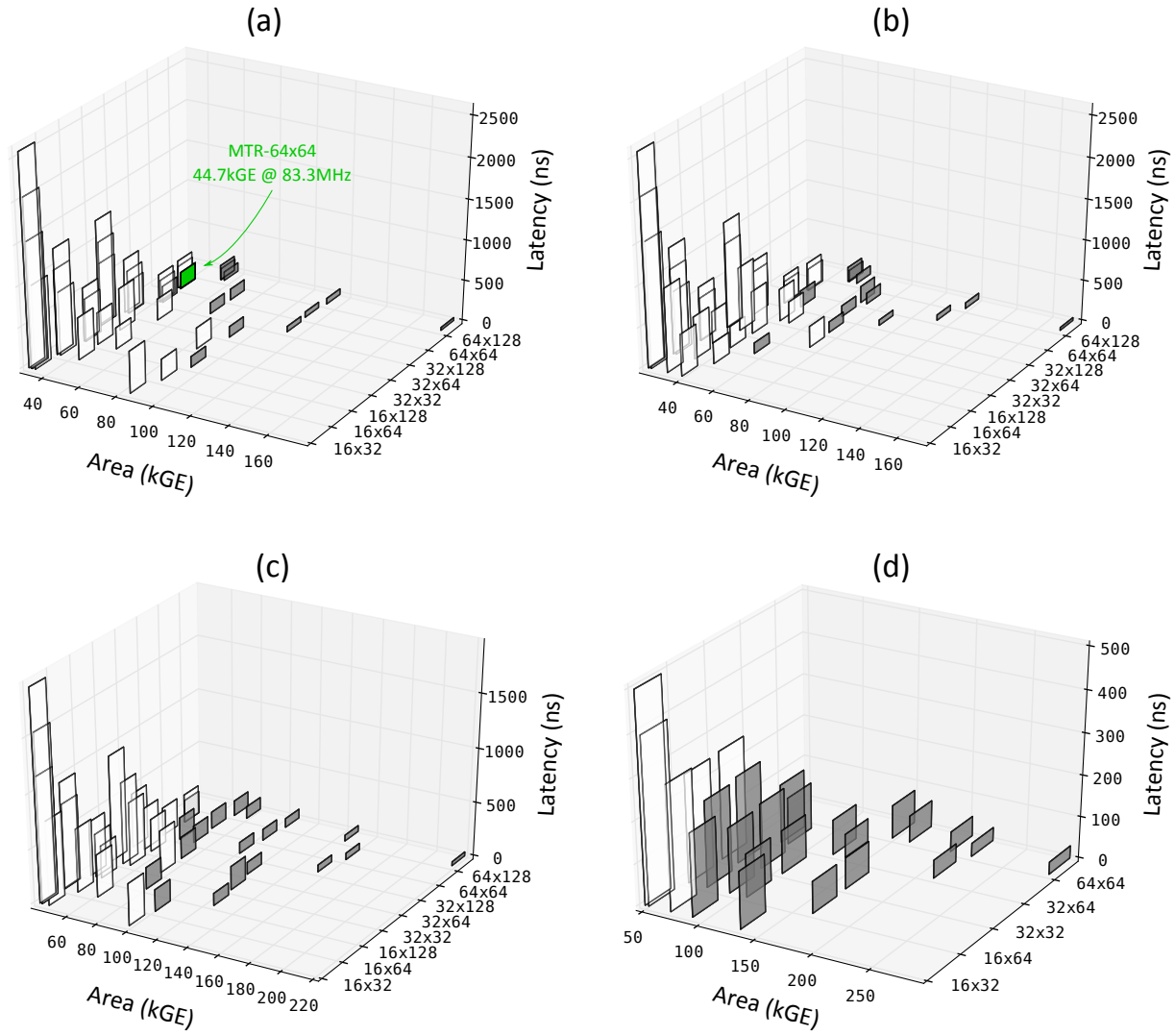

Fig. 3: Four different architectures of the finite field multiplier – synthesis results. (a) MULTIPLY-THEN-REDUCE. (b) INTERLEAVED. (c) KARATSUBA-I one-level deep. (d) KARATSUBA-II two-level deep.

architecture where the multiplication and reduction parts are simply performed consecutively (MULTIPLY-THEN-REDUCE). One would expect that more advanced techniques, such as interleaved or Karatsuba multiplication, would result in better performance. There is however an easy explanation.

We first observe all of the synthesized designs and compare them to the MULTIPLY-THEN-REDUCE architecture. Out of 40 synthesized instances of INTERLEAVED, 33 perform better than corresponding MULTIPLY-THEN-REDUCE instances (for the same latency, area is smaller). The reason for this is simple. When both architectures are synthesized for frequencies that are considerably lower than their inverted shortest critical path, the INTERLEAVED architecture provides better figures due to the smaller reduction part. Recall that the reduction part of MULTIPLY-THEN-REDUCE reduces 512-bit integers to 256-bit integers in a single clock cycle, while the reduction of INTERLEAVED reduces an $m + n + 256$-bit integer to a 256-bit one (m and n are the operand sizes of the basic multiplier as indicated previously). When the operating frequency is increased, however, the INTERLEAVED architecture reaches its shortest critical path faster than MULTIPLY-THEN-REDUCE. This is obvious since, due to the merged multiplication and reduction part, the critical path of INTERLEAVED is longer than the critical path of MULTIPLY-THEN-REDUCE. Synthesizing for a frequency

of 83.3 MHz (i.e. 12 ns) obviously causes INTERLEAVED to increase its size considerably more than the architecture of MULTIPLY-THEN-REDUCE.

Although achieving lower latency, the architectures based on Karatsuba algorithm have a larger circuit area, mainly due to the complex structure of datapath and control logic. Note that the same basic multipliers are used for all the explored architectures and, thus, the overhead comes from the circuitry for correct scheduling, the reduction part, and the temporary storage.

4.2 Skiing Down the Pyramid

Point Addition & Point Doubling. We start the top-down approach by identifying an appropriate coordinate system. Our goal is to avoid area-consuming storage, while still providing maximal speed up for the signature verification process. Examination of a set of possible projective coordinates lead us to the selection of Jacobian coordinates for point doubling (DBL) and mixed coordinates (affine-Jacobian) for point addition (ADD). In order to facilitate implementation we introduce a *simple arithmetic unit* (SAU) that prepares operands before they enter the finite field multiplier.The SAU supports very simple modular operations such as addition, subtraction, multiplication with 2 (left shift) and division by 2 (right shift). It is a purely combinational block without any storage. This way we are able to reduce the number of cycles in both ADD and DBL algorithms which is crucial for lowering the latency. Moreover, we save two temporary registers, each of 256 bits, compared to the originally proposed algorithms [10]. This way we gain back the area initially invested in the SAU. After a slight modification of both ADD and DBL algorithms, we propose the following two formulas (Table 1).

Table 1: Point Addition & Point Doubling.

ADD	DBL
$T1 = Z1^2$	$T2 = X1 - Z1^2$
$T2 = T1Z1$	$T2 = 3T2(2X1 - T2)$
$T2 = T2Y2 - Y1$	$X3 = T2^2$
$T1 = T1X2 - X1$	$Z3 = 2Y1Z1$
$Z3 = Z1T1$	$Y3 = (2Y1)^2$
$Y3 = T1^2$	$X3 = X3 - 2Y3X1$
$X3 = T2^2$	$T1 = (Y3X1^* - X3)T2$
$T1 = Y3T1$	$Y3 = T1 - (Y3^2)/2$
$X3 = X3 - T1 - 2Y3X1$	
$Y3 = (Y3X1^* - X3)T2$	
$Y3 = Y3 - T1Y1$	

$Y3X1^*$ is fed directly as a product from the previous step.

Each step of the two algorithms consists of at most one single finite field multiplication and a few basic operations. Steps denoted with asterisk (*) may seem to consist of two multiplications, however what occurs in fact is that the product from the previous step ($Y3X1$) is directly fed back into the finite field multiplier after subtracting $X3$ from it. With this approach save one temporary register.

Based on the selection of these algorithms, we further explore the possibility of using one or more of the multiplier cores of the previous section. Due to the dependencies in both the ADD and DBL algorithms, it is apparent that the multi-core approach brings no significant advantages. In other words, for the price of an additional multiplier core, the improvement in speed is only 27.3 % for ADD and 25 % for DBL, while depending on the multiplier architecture, its circuitry consumes at least one third of the overall core size. Table 2 shows the analysis.

Table 2: Multi-core scheduling.

ADD		DBL	
core 1	core 2	core 1	core 2
$T1 = Z1^2$ $T2 = T1Z1$ $T2 = T2Y2 - Y1$	$T1 = T1X2 - X1$ $Z3 = Z1T1$ $Y3 = T1^2$ $T1 = Y3T1$ $X3 = X3 - T1 - 2Y3X1$ $Y3 = (Y3X1 - X3)T2$ $Y3 = Y3 - T1Y1$	$T2 = X1 - Z1^2$ $T2 = 3T2(2X1 - T2)$ $X3 = T2^2$ $X3 = X3 - 2Y3X1$ $T1 = (Y3X1 - X3)T2$ $Y3 = T1 - (Y3^2)/2$	$Z3 = 2Y1Z1$ $Y3 = (2Y1)^2$

Finite Field Inversion. During the precomputational phase of the ECDSA signature verification, an inverse ($s^{-1} \bmod n$) is needed in order to proceed (inversion in $\mathbb{F}_n$). Also, after $X = u_1P + u_2Q$ has been evaluated, a conversion from projective Jacobian back to affine coordinates is needed. This operation involves inversion in $\mathbb{F}_p$.

These two inversions are done either at the beginning or at the end of the whole ECDSA verification, consuming very little time compared to the main operation $X = u_1P + u_2Q$. Rather than implementing a very fast and area-consuming inversion, we therefore opt for a slower, yet very compact implementation. Based on the work of Chen and Qin [3], we implement the compact finite field inversion in 742 clock cycles. Its simple structure consists of two 256-bit adders, five 256-bit registers and control logic.

Multiplication in $\mathbb{F}_n$. In order to compute u_1 and u_2 we need to perform finite field multiplication in $\mathbb{F}_n$ ($u_1 = ew \bmod n$ and $u_2 = rw \bmod n$). Similar to the finite field inversion, this computation is only performed during the precomputational phase and consumes negligible time compared to the main part of the ECDSA algorithm. For that reason, we implement a compact, bit-serial multiplier, that takes 256 clock cycles for the whole operation. It uses in fact a simple shift-and-add algorithm whose implementation consists of three 256-bit adders, three 256-bit registers, and some control logic.

Joint Sparse Form (JSF). In order to speed up the most computationally intensive part of the ECDSA algorithm, we turn our attention to optimizing the computation of $X = u_1P+u_2Q$. As shown by Shamir [5], this is easily performed by precomputing $P + Q$, then scanning the binary expansions of both u_1 and u_2 concurrently and, depending on the value of the pair of currently selected bits, adding P, Q, $P + Q$ or nothing, before doubling the current value. More specifically, both u_1 and u_2 are scanned bit by bit, starting from their most significant side (let b_{u1} be the current bit of u_1 and b_{u2} the bit of u_2). If $(b_{u1}, b_{u2}) = (0, 0)$, only point doubling is performed; for $(b_{u1}, b_{u2}) = (0, 1)$ we add Q and double the result; for $(b_{u1}, b_{u2}) = (1, 0)$ we add P and double the result; finally, if pair $(b_{u1}, b_{u2}) = (1, 1)$ is detected, we add $P + Q$ and double the result.

Based on the work of Solinas [20], it is possible to further speed up the multiple point multiplication using a low-weight signed binary representation for a pair of integers. Both u_1 and u_2 are decoded and represented such that at least one of any three consecutive (b_{u1}, b_{u2}) pairs is a zero pair. This representation has minimal weight among all joint signed binary expansions, where the weight is defined to be the number of nonzero (b_{u1}, b_{u2}) pairs. Due to this property, the expected number of point additions of $u_1P + u_2Q$ is $k/2$, where $k = 256$ is the length of u_1 and u_2. Compared to the approach of Shamir, JSF reduces the number of point additions by 33 %. Implementation of the JSF recoding mainly consists of two 512-bit and two 256-bit registers, while the rest is a tiny control logic.

A further reduction of the latency is possible by applying the windowing method for fast scalar multiplication [10]. We have, however, not chosen to take this approach in order to keep the area consumption low.

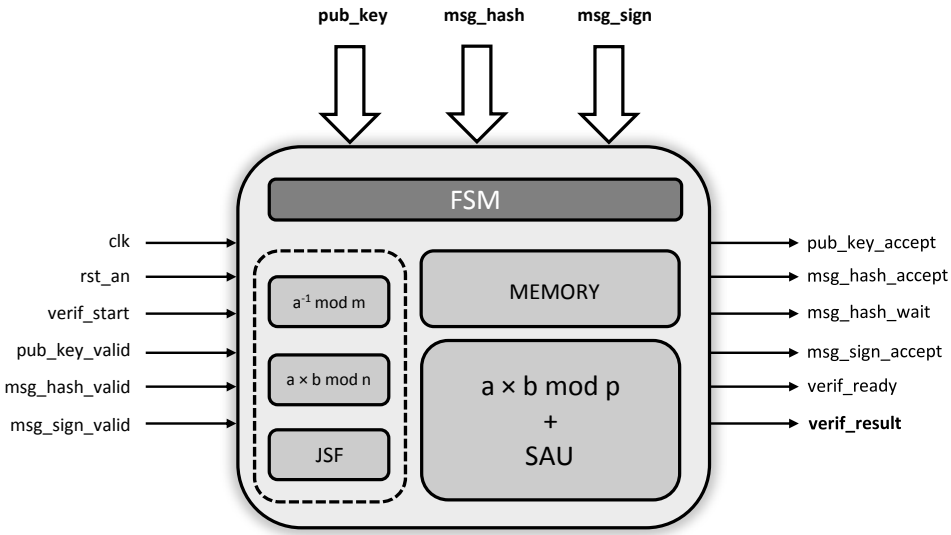

Fig. 4: Block scheme of the ECDSA verification core.

The application of JSF already requires the precomputation and storage of two additional points ($P+Q$, $P-Q$). If one would, for instance, combine JSF with the windowing method (shortest window width of 2), the precomputation and storage of 22 extra points would be needed ($2Q$, $3Q$, $P+Q$, $P+2Q$, $P+3Q$, $P-Q$, $P-2Q$, $P-3Q$, $2P$, $2P+Q$, $2P+2Q$, $2P+3Q$, $2P-Q$, $2P-2Q$, $2P-3Q$, $3P$, $3P+Q$, $3P+2Q$, $3P+3Q$, $3P-Q$, $3P-2Q$, $3P-3Q$). Since each precomputed point consists of two coordinates and consumes 512 bits of memory, this approach would introduce a significant area overhead.

Storage reduction. The three previously described blocks – finite field inversion, multiplication in $\mathbb{F}_n$, and JSF – consume $3,584$ bits of memory in total, of which the most memory hungry block JSF consumes $1,536$ bits. We therefore merge the three blocks into a single one where the memory is efficiently shared amongst them. This comes at the expense of a more complex control logic but certainly saves overall on area due to a 57 % of storage reduction.

5 Implementation and Results

Figure 4 represents the block scheme of our system. Next to the standard clock input, asynchronous reset, and the start switch, our core has three main inputs for transferring the public key Q (`pub_key`, 512-bit), a message hash e (`msg_hash`, 256-bit), and the signature of the message (r, s) (`msg_sign`, 512-bit). In addition, there are three input flags to indicate when the main inputs are ready to be fetched (`pub_key_valid`, `msg_hash_valid`, and `msg_sign_valid`). The communication to the outside world is done via six output signals. We provide three acknowledgment lines that are activated after the public key, message hash, or message signature have been received, and these are `pub_key_accept`, `msg_hash_accept`, and `msg_sign_accept`, respectively. Since the message hash is not immediately used at the start of the computation[3], we provide the output signal `msg_hash_wait` which indicates that the core has reached the stage where the hash is needed in order to proceed, i.e. to compute $u_1 = ew \bmod n$. This approach provides additional time for the external hash engine to finalize its computation and may improve the overall performance. Finally, once the verification has been completed, the `verif_ready` signal activates and result of the verification is available at `verif_result`: 0 for REJECT and 1 for ACCEPT.

[3] After being initialized, the core starts computing $w = s^{-1} \bmod n$ and $u_2 = rw \bmod n$.

The core consists of several independent modules, which are controlled by a finite state machine (FSM). The largest of all, a finite field multiplier together with the SAU, represents the heart of the core and consumes most of the total computation time. In order to share the memory resources, finite field inversion (both in $\mathbb{F}_p$ and $\mathbb{F}_n$), multiplication in $\mathbb{F}_n$, and JSF blocks are merged into one and denoted with the dashed rectangle in Fig. 4. Memory, except of a few very dedicated registers inside the big multiplier, is shared between all other resources. Different SAU architectures are given in Fig. 5

Next, we provide the implementation results of the ECDSA core. The RTL code has been written in Verilog and tested against more than 100,000 test vectors, covering all classes of corner cases[4]. The synthesis has been carried out in Cadence RTL Compiler version 11.10-p005 and we used two different libraries for that purpose. The library which is used for the final prototype is 65 nm TSMC CMOS process with the worst case PVT conditions (125° C temperature and 0.8 V supply). In order to allow future comparisons to our core, we additionally provide the synthesis results using an open core NANGATE library, release PDKv1.3_2010_12 [16]. This is a 45 nm CMOS process with typical case PVT conditions (25° C temperature and 1.1 V supply).

Although we aim for a single solution that satisfies our requirements, yet consuming the smallest area, we still provide a range of solutions that we find valuable with respect to low latency. We believe that this approach outlines the importance of having the design space exploration done thoroughly. The first column in Table 3 indicates the architecture of the large finite field multiplier ($a \times b \bmod p$) as this is the only difference between all the reported solutions (FFM type).

Table 3: Hardware performance of the implemented ECDSA cores (synthesis results). Latency is defined as the time needed for a single ECDSA signature verification.

FFM type	TSMC 65 nm CMOS process, worst case PVT (125° C, 0.8 V)			
	Area (kGE)	Frequency (MHz)	Latency (μs)	Verifications/s
MTR-64 × 64	154.5	83.3	840	1,190
INT-64 × 128	174.6	83.3	456	2,193
KB2-32 × 64	265.8	125	368	2,717
KB2-64 × 64	373.0	135	185	5,405
MTR-256 × 256	1,111.3	107	103	**9,708**
FFM type	NANGATE 45 nm CMOS process, typical case PVT (25° C, 1.1 V)			
	Area (kGE)	Frequency (MHz)	Latency (μs)	Verifications/s
MTR-64 × 64	223.1	330	212	4,717
INT-64 × 128	253.4	330	115	8,696
INT-64 × 128	322.1	500	76	13,158
KB2-64 × 64	373.1	333	75	13,333
MTR-256 × 256	1,033.9	295	37	**27,027**

MTR – MULTIPLY-THEN-REDUCE
INT – INTERLEAVED
KB2 – KARATSUBA-II (two-level deep KARATSUBA)
FFM – Finite Field Multiplier

An architecture that fulfills our initial requirements operates on a frequency of 83.3 MHz and consumes 154.5 kGE, while achieving a throughput of 1,190 signature verifications per second. Its latency, being only a fraction of a millisecond, is short enough for any safety-critical operations inside the vehicle. This, indeed, is the solution that has been chosen for the final prototype. Next, we highlight two additional architectures which show remarkable results with respect to throughput, latency, and time-area product.

[4] A corner case occurs when during the verification process, the point at infinity occurs at any stage of the computation.

Although it comes at the price of 1.1 million gate equivalences, our core with the MTR-256×256 multiplier achieves more than 27,000 signature verifications per second. To the best of the authors knowledge, this is the fastest reported implementation of P-256 ECDSA signature verification on a single core. In [9], the authors report a performance of 16,352 multiple points multiplications per second over the P-256 elliptic curve on a single chip[5] (Virtex-4 FPGA device). Their implementation, however, consists of 16 independent cores running in parallel, which translates the overall throughput to 1,022 multiple points multiplications per second per single core.

When observing the latency, our fastest architecture is able to verify a single signature in only 37 μs while previously the fastest implementation, which is in fact a software implementation of an ECDSA signature generation, comes from [4] and reports a latency of 427 μs. The previously fastest hardware implementation comes from [8] with 680 μs for a single point multiplication. Another notable result is reported in [9], where the multiple point multiplication on a single core takes 749 μs. The authors further estimate that in case the windowing method is used, the latency can be improved to 495 μs. We have to note here that a fair comparison between these designs is very difficult since the technology used is different in these three cases (ASIC versus FPGA versus software), although in a system design all three options could be weighed against each other.

Finally, by observing the efficiency of proposed architectures, we highlight the INT-64 × 128 core which delivers the highest throughput per area. Its time-area product equals only 24.5 sGE.

In order to compare our designs with state-of-the-art implementations of ECDSA signature verification, we provide Table 4. While comparing latency is easy, it is much more difficult to compare the implementation price at which the latency comes. Naturally, this is due to the variety of platforms in the reported results. Hence, our attempt to make a comparison in this form does not only serve to illustrate the benefits of our approach, but also provides a better understanding of what is feasible in different technologies.

Table 4: High-performance implementations of the ECDSA signature verification over P-256 elliptic curve.

Reference	Technology	Logic Consumption	Frequency (MHz)	Latency♣ (μs)
This work	NANGATE 45 nm CMOS	1,033.9 kGE	295	37
		373.1 kGE	333	75
		322.1 kGE	500	76
		253.4 kGE	330	115
		223.1 kGE	330	212
2013 [4]♠	Intel Xeon E3	4×64-bit cores	3,500	427
2010 [8]♦	EP2S30F484C3	9,117 ALM	157.2	680
2008 [9]♥	XC4VFX12-12	1,715 LS + 32 DSP	490	749★
2006 [14]♦	XC2VP30	3,529 slices + 36 MUL	67	2,270
2004 [13]♦	XC2VP125-7	15,755 LS + 256 MUL	39.5	3,840
2009 [19]♦	XCV1000E-8	36,000 LUT	39.7	3,950

♣ Latency is measured for performing a single operation (verification, generation, or point multiplication).
♠ The performance of the ECDSA signature generation is reported.
♦ The performance of the single point multiplication is reported.
♥ The performance of the multiple point multiplication is reported.
★ The authors report an estimated latency of 495 μs in case the windowing method would be used.

[5] Multiple point multiplication is the most expensive, but not the only operation of the ECDSA signature verification. The authors further report an optimistic estimate of 22,700 multiple point multiplications per second, assuming the usage of the windowing method.

6 Conclusion

We have presented the fastest core for performing a digital signature verification based on elliptic curves over the NIST P-256 prime. The results achieved in this work, including amongst others a latency of 37 μs for a single signature verification and an efficiency of 24.5 sGE, significantly outperform any previous implementation known by the authors.

We have further shown that it is possible to meet the requirements for Car2X communication security in intelligent transportation systems – signature verification in less than a millisecond – with a core of only 155 kGE, providing a much more cost effective solution for mass production compared to any software or FPGA based solution.

References

1. P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1987.
2. M. Brown, D. Hankerson, J. López, and A. Menezes. Software Implementation of the NIST Elliptic Curves Over Prime Fields. *Lecture Notes in Computer Science*, 2020, 2001.
3. C. Chen and Z. Qin. Fast Algorithm and Hardware Architecture for Modular Inversion in GF(p). In *Intelligent Networks and Intelligent Systems, 2009. ICINIS '09. Second International Conference on*, pages 43–45, 2009.
4. D. J. Bernstein and T. Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems. Available at http://bench.cr.yp.to/ebats.html, accessed 14 Janruary 2014.
5. T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology — CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1986.
6. ETSI TS 102 637-2 V1.2.1 (2011-03). Technical Specification; Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Co-operative Awareness Basic Service.
7. ETSI TS 103 097 V0.0.13 (2013-02). Technical Specification; Intelligent Transport Systems (ITS); Security; Security Services and Architecture.
8. N. Guillermin. A High Speed Coprocessor for Elliptic Curve Scalar Multiplications over Fp. In *Cryptographic Hardware and Embedded Systems — CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 48–64. Springer, 2010.
9. T. Güneysu and C. Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In *Cryptographic Hardware and Embedded Systems — CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 2008.
10. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.
11. IEEE Std 1609.2-2006. IEEE Trial-Use Standard for Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages.
12. A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Soviet Physics - Doklady*, 7:595–596, 1962. Translation from Doklady Akademii Nauk SSSR, 145:2, 293–294, 1962.
13. C. McIvor, M. McLoone, and J. McCanny. An FPGA Elliptic Curve Cryptographic Accelerator Over GF(p). In *Irish Signals and Systems Conference (ISSC)*, pages 589–594, 2004.
14. N. Mentens. *Secure and Effcient Coprocessor Design for Cryptographic Applications on FPGAs*. PhD thesis, Katholieke Universiteit Leuven, 2007.
15. P. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.
16. NANGATE. The NanGate 45nm Open Cell Library. Available at http://www.nangate.com.
17. National Institute of Standards and Technology (NIST). FIPS 186-3: Digital Signature Standard, 2009. Available at http://csrc.nist.gov/.
18. K. Sakiyama, E. De Mulder, B. Preneel, and I. Verbauwhede. A Parallel Processing Hardware Architecture for Elliptic Curve Cryptosystems. *IEEE ICASSP*, pages 904–907, 2006.
19. D. Schinianakis, A. Fournaris, H. Michail, A. Kakarountas, and T. Stouraitis. An RNS Implementation of an Elliptic Curve Point Multiplier. *IEEE Transactions on Circuits and Systems I*, 56(6):1202–1213, 2009.
20. J. A. Solinas. Low-Weight Binary Representations for Pairs of Integers. Technical report, National Security Agency, USA, 2001.
21. Traffic Control. Encyclopedia Britannica. Available at http://www.britannica.com/.
22. T. Zhou, X. Wu, G. Bai, and H. Chen. Fast GF(p) Modular Inversion Algorithm Suitable for VLSI Implementation. 38(14):706–707, July 2002.

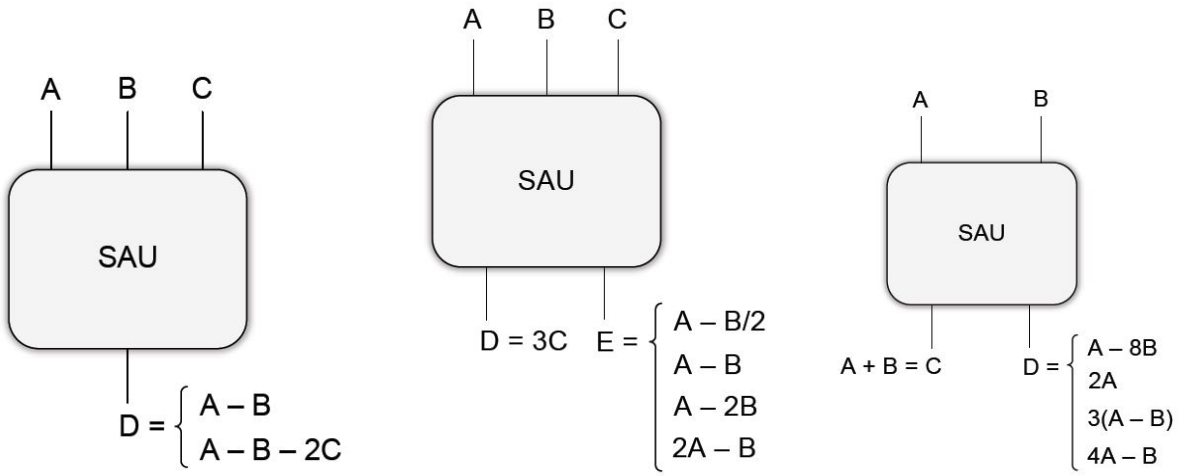

Fig. 5: Different SAU architectures.

A Performance Requirements

Let T be the time needed for one multiple point multiplication $(u_1P + u_2Q)$ and t the time needed for one multiplication in $\mathbb{F}_p$. Let l be the bit length of u_1 and u_2. Due to the property of the JSF recoding, our core will perform l DBL and $\lceil l/2 \rceil$ ADD operations for one signature verification on average. Table 1 shows that DBL consists of 8 and ADD of 11 finite field multiplications. Since $l = 256$, it turns out that the computation of $u_1P + u_2Q$ takes $3,456$ finite field multiplications on average. We round up the total number of multiplications to $4,000$, which takes into account other operations of the ECDSA algorithm (see Alg. 1) and provides a safe margin. Since $1,000$ signature verifications need to be done per second, it follows that one finite field multiplication can take at most 250 ns.

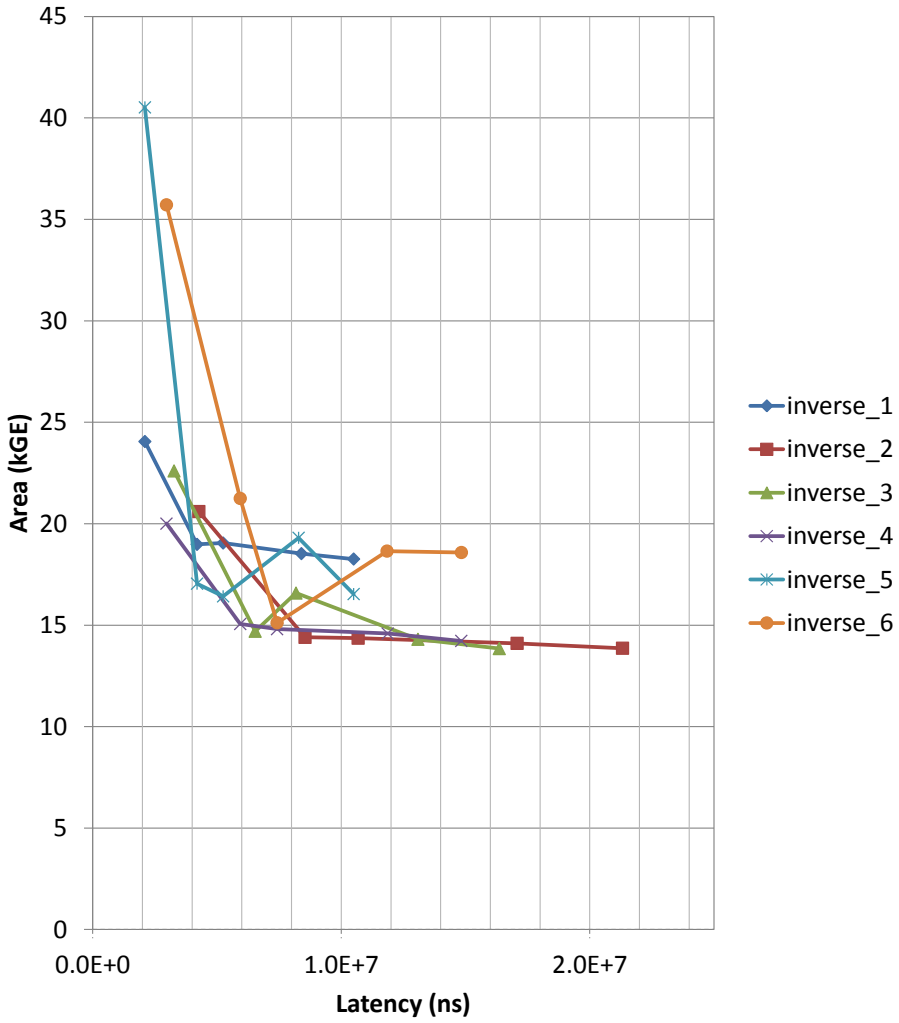

Fig. 6: Six different architectures of the finite field inversion – synthesis results. (v1) The Extended Binary GCD Algorithm. (v2) Based on [22]. (v3) Based on [3]. (v4) Based on [3] RTL optimization for speed. (v5) The Extended Binary GCD Algorithm, RTL optimization for area. (v6) Based on [3] RTL optimization for speed and area.