Versatile Hardware Framework for Elliptic Curve Cryptography

Vít Mašek*† and Martin Novotný*
* *Czech Technical University in Prague*, Prague, Czech Republic
{masekvit|novotnym}@fit.cvut.cz
† *TropicSquare s. r. o.*, Prague, Czech Republic
vit.masek@tropicsquare.cz

***Abstract*—We propose versatile hardware framework for ECC. The framework supports arithmetic operations over P-256, Ed25519 and Curve25519 curves, enabling easy implementation of various ECC algorithms. Framework finds its application area e.g. in FIDO2 attestation or in nowadays rapidly expanding field of hardware wallets. As the design is intended to be ASIC-ready, we designed it to be area efficient. Hardware units are reused for calculations in several finite fields, and some of them are superior to previously designed circuits in terms of time-area product. The framework implements several attack countermeasures. It enables implementation of certain countermeasures even in later stages of design. The design was validated on SoC FPGA.**

***Index Terms*—Elliptic curve cryptography, Public key cryptography, Side channel hardening, ECDH, EdDSA, ECDSA, FPGA**

I. Introduction

Elliptic curve cryptography [1] is nowadays used in many cryptographic algorithms, such as digital signature [2] or Diffie-Hellman key exchange [3]. It takes advantage over RSA [4] in shorter keys while keeping the same security level which often leads to more modest design. ECC-based algorithms include e.g. ECDSA (over NIST curves P-256, P-224 or P-384) [5], EdDSA [6] (over Edwards curves Ed25519 and Ed448 [7, 8]) or ECDH (based on Curve25519 and X25519 [9]).

When implementing cryptographic algorithms, the designer must pay special attention to side-channel attacks, such as DPA [10], CPA [11], SPA or timing attacks [12], and to fault-injection attacks [13]. Many countermeasures against side-channel attacks are based on hiding or masking principle [14]. Hiding typically randomizes the power consumption (hiding in amplitude) or time of execution (hiding in time). Masking randomizes processed data to make it difficult for an attacker to predict any intermediate values.

In this work we focus on design of hardware framework that provides:

- Arithmetic over general finite fields with up to 256 bit primes.
- Optimized arithmetic over P-256, Ed25519 and Curve25519 to support ECDSA, EdDSA and ECDH.
- Sharing of hardware resources to be area efficient.
- Set of side channel countermeasures.
- Set of instructions allowing freedom in the choice of algorithms and countermeasures in later stages of design.
- Standardized AHB interface and special handshake interface to obtain and provide secret data, e.g. random number, private key or computed secret result.
- ASIC-ready design.

Such a hardware framework finds its application e.g. in FIDO2 attestation [15] or in nowadays rapidly expanding field of hardware wallets. Critical parts (flag registers) utilize triple-module redundancy (TMR). Functionality of the framework was validated on Xilinx Zybo Zynq-7000 platform equipped with Artix-7 FPGA, however, as the framework is ASIC-ready, we refrain from implicitly using FPGA specific resources.

This paper is structured as follows: Chapter II brings necessary background and discusses side-channel countermeasures applicable to ECC. In Chapter III we discuss our design strategy and describe the arithmetic unit in more detail. Chapter IV shows synthesis results on the target platform. In Chapter V we discuss our future work on this topic. Chapter VI summarizes results of this work.

II. Background

To reach the goal of implementing ECC algorithms mentioned above, the design should support modular arithmetic over following four primes:

- $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- $p_{25519} = 2^{255} - 19$
- $\#E_{256}$, the order of P-256.
- $\#E_{25519}$, the order of Ed25519 and Curve25519.

The arithmetic in $GF(p_{256})$ and $GF(p_{25519})$ should be optimized for performance, since it will be used thousands times per signature.

A. Related work

There are many works related to ECC implementation over prime field $GF(p)$. One of the first implementations of ECC over $GF(p)$ in reconfigurable hardware were proposed in [16, 17]. Recent works on implementation of P-256 include [18] and [19], while [20] focuses on Ed25519 together with Curve25519 for Diffie–Hellman key exchange. The 100 microseconds barrier of X25519 was then broken in [21].

This research has been supported by the grant VJ02010010 of the Ministry of the Interior of the Czech Republic, "Tools for AI-enhanced Security Verification of Cryptographic Devices" in the program Impakt1 (2022-2025).

B. Side-channel countermeasures

Besides *constant time of computation* preventing timing attacks [12], the designer must implement several other countermeasures to further protect the design against other types of attacks. Here we discuss three masking techniques applicable to ECC [22], namely Z-coordinate randomization, scalar blinding, and point blinding.

With *Z-coordinate randomization*, we use projective coordinates (X, Y, Z) to represent point on elliptic curve, s.t. $(x, y) = (xZ, yZ, Z)$. Random value Z is generated when converting to projective coordinates. The point can be re-randomized at any time by simply generating another random value r and multiplying all three coordinates, $(xZ, yZ, Z) = (xZr, yZr, Zr)$.

When calculating scalar point multiple $Q = s \cdot P$, one can mask both scalar s and point on curve P. *Scalar blinding* masks the secret scalar s with some random value r. We derive two new scalars as $s_1 = r$, $s_2 = s - r$, i.e. $s_1 + s_2 = s$. Result Q is then calculated as $Q = s_1 \cdot P + s_2 \cdot P$. Alternatively, we can blind the scalar as $s' = s + r \cdot \#E$.

Point blinding works on similar principle. We generate random point R on the curve and then derive two new points as $P_1 = R, P_2 = P - R$, i.e. $P_1 + P_2 = P$. Result Q is then calculated as $Q = s \cdot P_1 + s \cdot P_2$.

III. Design

In this section we describe our design strategy, the way we implement efficient non-modular and modular arithmetic and how we prepare the design to accommodate certain side-channel countermeasures.

A. Design Strategy

Our design is versatile, i.e. when one decides to change algorithms (e.g. point addition or doubling, scalar multiplication, ...) or even to add a new curve to the mix, it can be done easily with no need to touch much of the design. To achieve this goal we implement the framework as a micro-architecture with instruction decoder and custom set of instructions. Algorithms are then implemented in firmware. The micro-architecture is composed of datapath, controller and memory subsystem, as seen in Fig. 1.

Let define following terms: *Firmware* (microcode) consists of instructions that are decoded by instruction decoder. *Instruction* is composed of one or several operations. *Operation* refers to calculation performed by HW (sub)unit. It can last one or several clock cycles.

B. Instruction Set

Instruction set consists of four classic types of instruction – R, I, M, J. Besides instructions for program flow control such as branches, 32 bit arithmetic, subroutine calls etc., our custom ISA contains also special instructions for modular arithmetic, such as addition, subtraction, reduction and multiplication in $GF(p)$ and special instruction for fast multiplication in $GF(p_{25519})$ and $GF(p_{256})$. Instructions for obtaining random value from external TRNG and obtaining private keys are also

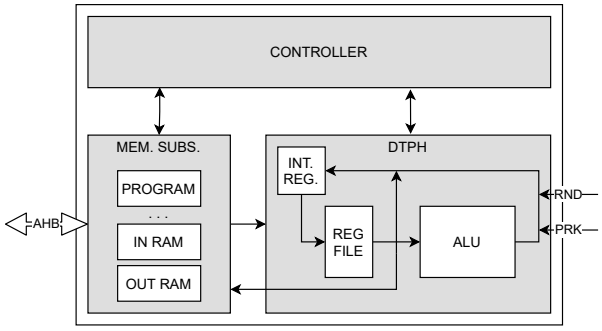

Fig. 1. High-Level Design

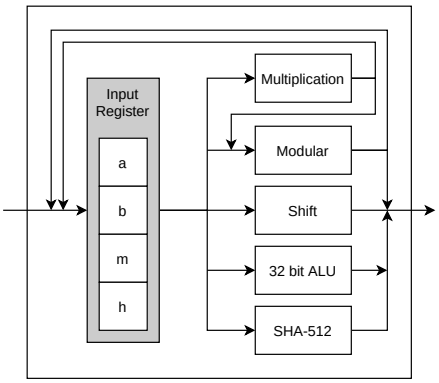

Fig. 2. Internal structure of arithmetic unit

present, as well as instructions for SHA-512. Each instruction has 32 bits, with bit 31 reserved as a parity bit to prevent fault-injection attacks against the microcode. The program memory has space for 2048 instructions.

C. Datapath

Datapath consists of an arithmetic unit, a register file and an intermediate register.

Register file can hold 32 values, each being 256 bits wide. The file is implemented as RAM, since implementation by flip-flops would be high area demanding. Although RAM allows fetching of only one operand in one cycle, slight performance lost is negligible.

D. Arithmetic Unit

The arithmetic unit contains input registers a, b, m and h, and subunits implementing certain subsets of operations. The two main subunits are *Modular* subunit and *Multiplication* subunit, as seen in Fig. 2.

Modular subunit performs modular addition, subtraction and reduction by up to 256 bit general modulus m. It is composed of 257 bit adder, 257 bit intermediate register c, and control logic. *Addition* and *subtraction* is done by adding/subtracting the operands a and b followed by trial subtraction/addition of m. *Reduction* is available for values of up to 512 bits wide. The value to be reduced is stored into registers a and b, s.t., $ab = a \cdot 2^{256} + b$. Reduction is then done via modular multiplication by 1, using double-and-add algorithm

with interleaved reduction. As we are multiplying by 1, the double-and-add part is simplified to logical left shift of c and shifting in the actual bit of ab. After each step, trial subtraction of modulus m is done. This design approach allows us to reuse the resources for addition/subtraction. Time ineffectiveness of *reduction* is negligible since it will be used only a few times per ECC algorithm. Besides these operations, *Modular* subunit can also serve as a *modular accumulator* (applicable e.g. for reduction in P-256), as *non-modular adder* with ability to hold carry bit (i.e., addition of operands wider than 256 bits can be done as well; applicable e.g. when computing $s' = s + r \cdot \#E$), or as simple *trial subtractor* of m.

Multiplication subunit performs non-modular multiplication of two 256 bit operands resulting in 512 bit result, and fast multiplication in $GF(p_{25519})$. This subunit is composed of four 16×16 one-cycle multipliers, intermediate register p, result register c, and control logic. The 256×256 *non-modular multiplication* is done in 64 cycles using schoolbook algorithm with 16 bit words. When implementing *fast multiplication in* $GF(p_{25519})$, we first considered adopting the design presented in [20]. This design is FPGA-optimized, exploiting 15 (otherwise unused) DSP48E units to reach the maximum speed. When transferred to ASIC, such a design would occupy high area. Even scaling-down the design to just four DSP48E units (i.e. four multipliers) and sharing resources with already designed four multipliers (see above) is not advantageous, as the multipliers in [20] are of different size 21×17 and much additional logic would be necessary to share the resources. For that reasons we decided to use already designed 256×256 multiplier, add one extra 16 bit register and some additional control logic, do non-modular multiplication first and then perform fast reduction of the 512 bit c result by p_{25519}. The reduction is done in four steps:

1) $h = c/2^{255}$, $l = c \pmod{2^{255}}$
2) $h = h \cdot 19$
3) $h = (h \pmod{2^{255}}) + (h/2^{255}) \cdot 19$
4) $c = h + l$

After these four steps, c holds the result. As the result may be $p_{25519} \leq c \leq 2p_{25519} - 1$, one more trial subtraction is needed. This is then done by the *Modular* subunit described above. By this approach, multiplication in $GF(p_{25519})$ is done in 76 cycles with just 4 multipliers 16×16 (equivalent to 1192 LUTs). This yields in twice better time-area product compared to [20], that performs multiplication in 33 cycles, but with 15 multipliers 21×17 (equivalent to 5505 LUTs). The amount of FFs and additional control logic is roughly the same.

Multiplication in $GF(p_{256})$ is done the same way as in $GF(p_{25519})$ – non-modular multiplication is followed by fast reduction. The fast reduction is done with algorithm described in [19] and the Modular subunit is used as an accumulator modulo p_{256}. Note that design in [20] is hardwired for $GF(p_{25519})$, which limits its potential reuse for $GF(p_{256})$.

E. Side channel countermeasures

Timing attacks are prevented on every level of calculation. Operations in HW units are performed in constant time (independently on processed data), as well as all instructions. Scalar point multiplication will be implemented in firmware using Montgomery ladder algorithm [23].

For SPA countermeasures, the Montgomery algorithm can be implemented either with branches or using conditional swap of the two points Q_0 and Q_1. For this, special instruction *CSWAP* is prepared.

Critical parts of the design use TMR to prevent faults. E.g., the flag register is implemented as three registers, where one holds inverted values. Instruction code contains one parity bit to check if a fault was injected into FW.

Certain masking countermeasures will be implemented in firmware, e.g., Z-coordinate randomization, or point blinding. Scalar blinding is be implemented via group scalar randomization [22]. We will use 256 bit random value r, so the randomized scalar results in 512 bit value. The formula $s' = s + r \cdot \#E$ is implemented as one operation in the arithmetic unit with dedicated instruction, to maintain less work with private keys.

Randomness for above mentioned countermeasures is provided from an external TRNG. Special instruction implements handshake protocol with TRNG, and stores generated value to the register file.

Private keys can be obtained directly from external system, via same handshake as for random value, in case of it is considered insecure to load private keys via the AHB bus to the memory subsystem.

IV. Synthesis Results

Table I provides results of synthesis done with Vivado 2020.02. The synthesis was set to not infer DSP48E units and use LUTs instead. ASIC results (equivalent gates) were obtained by Synopsys CAD-tools.

unit	LUT	LUTRAM	FF	gates
DTPH	10462	300	4180	74793
ALU	9021	128	3667	69750
HASH	1839	128	1564	24715
MULT+MOD	4052	0	1079	25622

TABLE I
SYNTHESIS RESULTS

V. Future Work

Implementation of firmware is to be done. Then, we will analyze side-channel resistance of the proposed framework. Based on the results, additional countermeasures will be designed and implemented. Alternatively, some countermeasures may show as superfluous and could be removed.

VI. Conclusion

We discussed, proposed and designed versatile hardware framework for ECC. The framework supports arithmetic operations over P-256, Ed25519 and Curve25519 curves, enabling easy implementation of ECDSA, EdDSA and ECDH algorithms. Other curves and/or algorithms can be added at low cost. Hardware framework finds its application area e.g. in

FIDO2 attestation or in nowadays rapidly expanding field of hardware wallets. To make the design area efficient, hardware units are designed to support arithmetic operations in several finite fields. Moreover, our design of $GF(p_{25519})$ multiplier is superior to that of [20] in terms of time-area product, and can be easily reused for multiplication in $GF(p_{256})$.

To prevent timing attacks, all operations are executed in constant time. Several countermeasures will be implemented also in firmware. The design is ASIC-ready. It was validated on SoC FPGA platform.

Side-channel resistance evaluation is a subject of future work and will be provided later.

VII. Acknowledgement

This work was supported by the Student Summer Research Program 2021 of FIT CTU in Prague.

References

[1] Victor S Miller. "Use of elliptic curves in cryptography". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1985, pp. 417–426.

[2] Michael J Ganley. "Digital signatures and their uses". In: *Computers Security* 13.5 (1994), pp. 385–391.

[3] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[4] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126.

[5] National Institute of Standards and Technology. *Digital Signature Standard*. Tech. rep. Federal Information Processing Standards Publications (FIPS PUB) 186-4. Gaithersburg: National Institute of Standards and Technology, 2013.

[6] National Institute of Standards and Technology. *Digital Signature Standard*. Tech. rep. Federal Information Processing Standards Publications (FIPS PUB) 186-5 (draft). Gaithersburg: National Institute of Standards and Technology, 2019.

[7] Daniel J. Bernstein et al. "High-Speed High-Security Signatures". In: *CHES*. Vol. 6917. Lecture Notes in Computer Science. Springer, 2011, pp. 124–142.

[8] Mike Hamburg. *Ed448-Goldilocks, a new elliptic curve*. Cryptology ePrint Archive, Report 2015/625. https://ia.cr/2015/625. 2015.

[9] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006*. Ed. by Moti Yung et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228.

[10] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[11] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation Power Analysis with a Leakage Model". In: *Cryptographic Hardware and Embedded Systems - CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29.

[12] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology — CRYPTO '96*. Ed. by Neal Koblitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113.

[13] Alessandro Barenghi et al. "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures". In: *Proceedings of the IEEE* 100.11 (2012), pp. 3056–3076. DOI: 10.1109/JPROC.2012.2188769.

[14] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Berlin, Heidelberg: Springer-Verlag, 2007.

[15] FIDO Alliance. *FIDO2: Web Authentication (WebAuthn)*. URL: https://fidoalliance.org/fido2/fido2-web-authentication-webauthn/.

[16] Gerardo Orlando and Christof Paar. "A Scalable GF(p) Elliptic Curve Processor Architecture for Programmable Hardware". In: *Cryptographic Hardware and Embedded Systems — CHES 2001*. Ed. by Çetin K. Koç, David Naccache, and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 348–363.

[17] A. Satoh and K. Takano. "A scalable dual-field elliptic curve cryptographic processor". In: *IEEE Transactions on Computers* 52.4 (2003), pp. 449–460.

[18] Di Matteo Stefano. *Design of an ECC hardware accelerator for ECDSA applications compliant to the WAVE standard (master thesis)*. Feb. 2019.

[19] Rashmi Agrawal, Ji Yang, and Haris Javaid. "Efficient FPGA-based ECDSA Verification Engine for Permissioned Blockchains". In: *CoRR* abs/2112.02229 (2021). arXiv: 2112.02229. URL: https://arxiv.org/abs/2112.02229.

[20] Furkan Turan and Ingrid Verbauwhede. "Compact and Flexible FPGA Implementation of Ed25519 and X25519". In: *ACM Trans. Embed. Comput. Syst.* 18.3 (Apr. 2019).

[21] Philipp Koppermann et al. "Low-latency X25519 hardware implementation: breaking the 100 microseconds barrier". In: *Microprocessors and Microsystems* 52 (2017), pp. 491–497.

[22] Jean-Luc Danger et al. "A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards". In: *Journal of Cryptographic Engineering* 3.4 (2013), pp. 241–265.

[23] Peter L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of computation* 48 (1987), pp. 243–264.