RUHR-UNIVERSITÄT BOCHUM

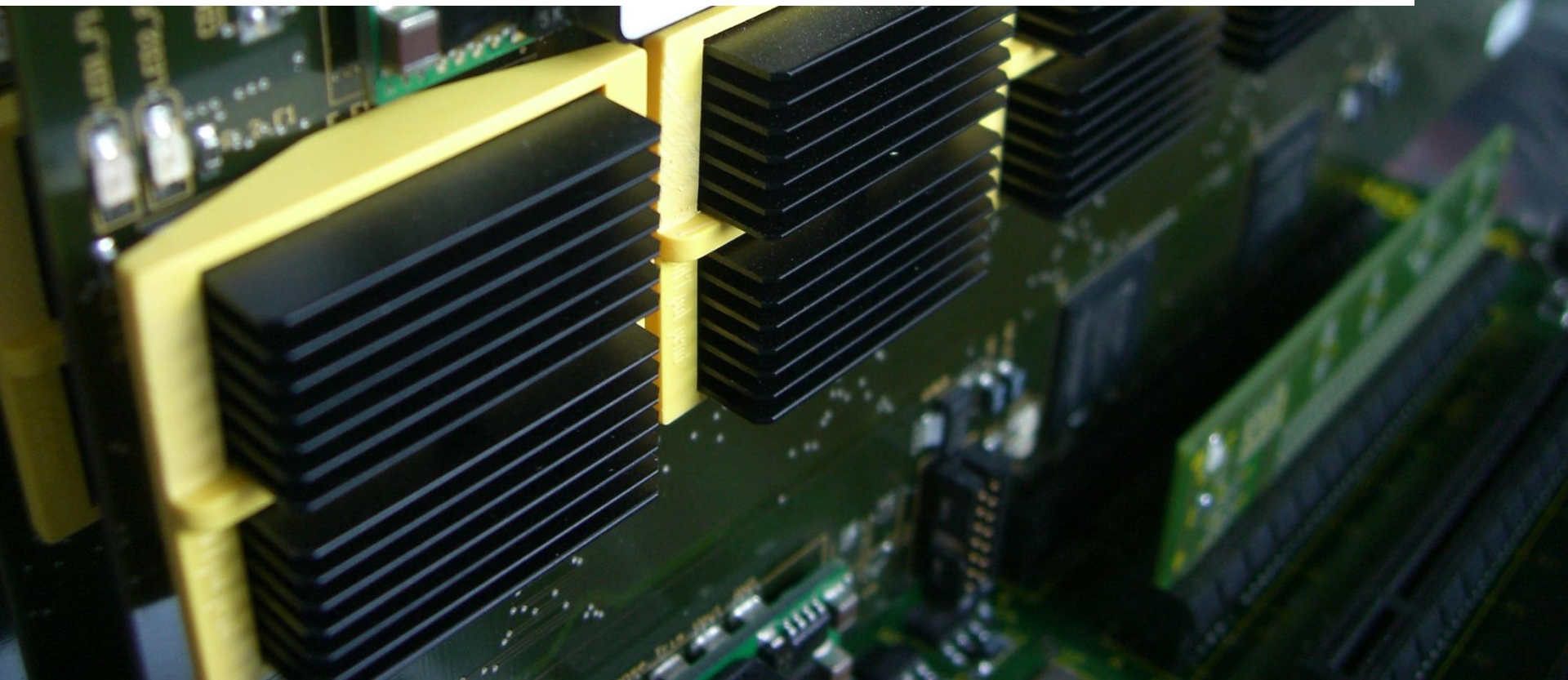
hgi EMSEC

RUB

IMPLEMENTING CURVE25519 FOR SIDE-CHANNEL-PROTECTED ELLIPTIC CURVE CRYPTOGRAPHY

PASCAL SASDRICH, TIM GÜNEYSU

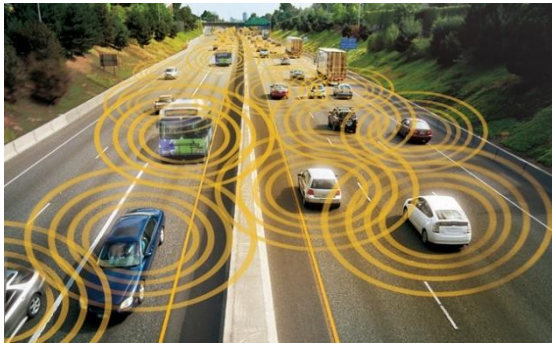
20TH WORKSHOP ON ELLIPTIC CURVE CRYPTOGRAPHY, YAŞAR UNIVERSITY, İZMIR, TURKEY SEPTEMBER 6, 2016



INTRODUCTION

WHY DO WE NEED HIGH-SPEED ASYMMETRIC CRYPTOGRAPHY?

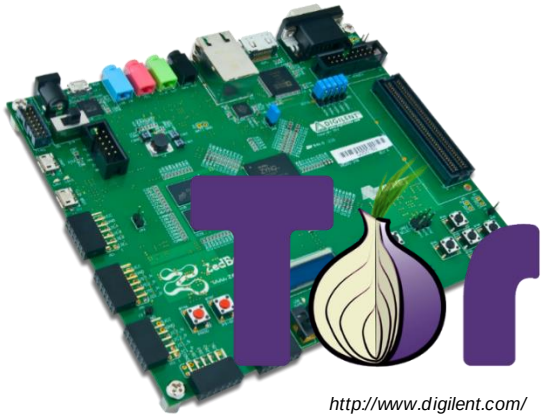
CAR2CAR COMMUNICATION



http://www.extremetech.com/

- Car2Car and Car2Environment communication requires exchange of messages
- messages have to be authenticated and protected against manipulation using **asymmetric cryptography (signatures)**
- time-critical procedure that requires **high-speed processing** of thousands of signatures per second

HARDWARE ROUTER FOR TOR NETWORK



http://www.digilent.com/

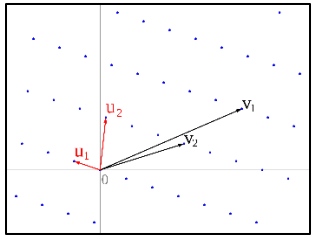
- **high-speed** hardware-based router for Tor network
- establishes thousands of new connections per second
- communication is protected against surveillance and analysis
- **asymmetric cryptography** is used to establish symmetric keys between communication partners **(key agreement)**

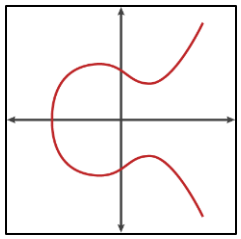
MODERN SYSTEMS REQUIRE HIGH-SPEED CRYPTOGRAPHIC IMPLEMENTATIONS PERFORMING THOUSANDS OF OPERATIONS PER SECOND.

HOW DO WE DESIGN CRYPTOGRAPHIC IMPLEMENTATIONS?

I. Choose cryptographic scheme:

- RSA
- ElGamal
- ✓ Elliptic Curve Cryptography
- Code-based/Lattice-based Cryptography
- …





II. Choose target platform:

- CPU
- Microprocessor
- ✓ Field-Programmable Gate Array (FPGA)
- Application-Specific Integrated Circuit (ASIC)
- …

III. Implement and optimize signature scheme for chosen platform

IV. Include protection against (implementation) attacks

- ✓ Timing Attacks
- ✓ Power / EM Analysis (Simple / Differential)
- Fault Injection
- …

OUR CONTRIBUTION

This work:

Implementing Curve25519 for Side-Channel-Protected Elliptic Curve Cryptography on FPGAs.

CHALLENGE: *Curve25519 is a state-of-the-art elliptic-curve Diffie-Hellman function that was chosen and designed for fast software implementations.*

OUR MAIN CONTRIBUTIONS:

1. We present the first implementation of Curve25519 on (reconfigurable) hardware.
2. Our implementation provides a high-speed scalar multiplication function using Curve25519.
3. It includes inherent protection against Timing and Simple Power Analysis (SPA) attacks.
4. It provides additional protection against Differential Power Analysis (DPA) attacks.

AGENDA OF THIS TALK

CURVE25519

THE ELLIPTIC CURVE CURVE25519

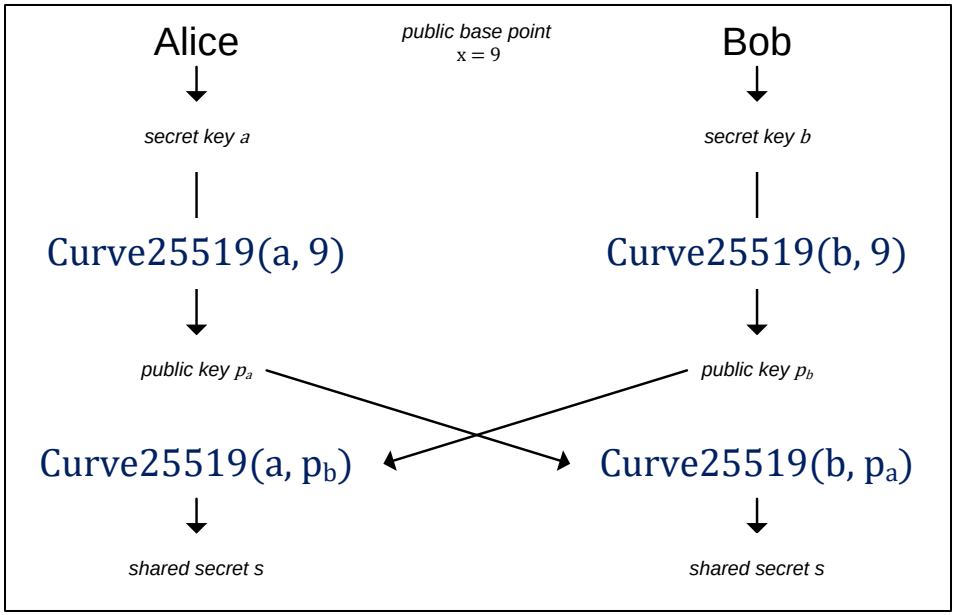
- **Curve25519:**
  - state-of-the-art elliptic curve over prime fields with ~128-bit level of security
  - special prime structure (Pseudo Mersenne prime): $\boldsymbol{p = 2^{255} - 19}$
    - namesake for the elliptic curve function
    - allows efficient modular reduction using multiplication (with small constant $c = 19$)
  - recently proposed and considered in RFC 7748 (along with Curve448) for next generation of TLS

- **X25519 (Bernstein 2006):**
  - Elliptic Curve Diffie-Hellman scheme
  - Montgomery curve: $y^2 = x^3 + 486662x^2 + x \ \ (mod\ p)$
  - public keys and shared secrets are points on the curve

- **Ed25519 (Bernstein, Duif, Lange, Schwabe, and Yang, 2011):**
  - Elliptic Curve Signature scheme
  - Twisted Edwards curve: $-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2 \ \ (mod\ p)$
  - public keys and (part of) the signatures are points on the curve

THE X25519 ECDH SCHEME

X25519 is ECDH key agreement scheme which establishes a shared secret between Alice and Bob.



PROPERTIES:

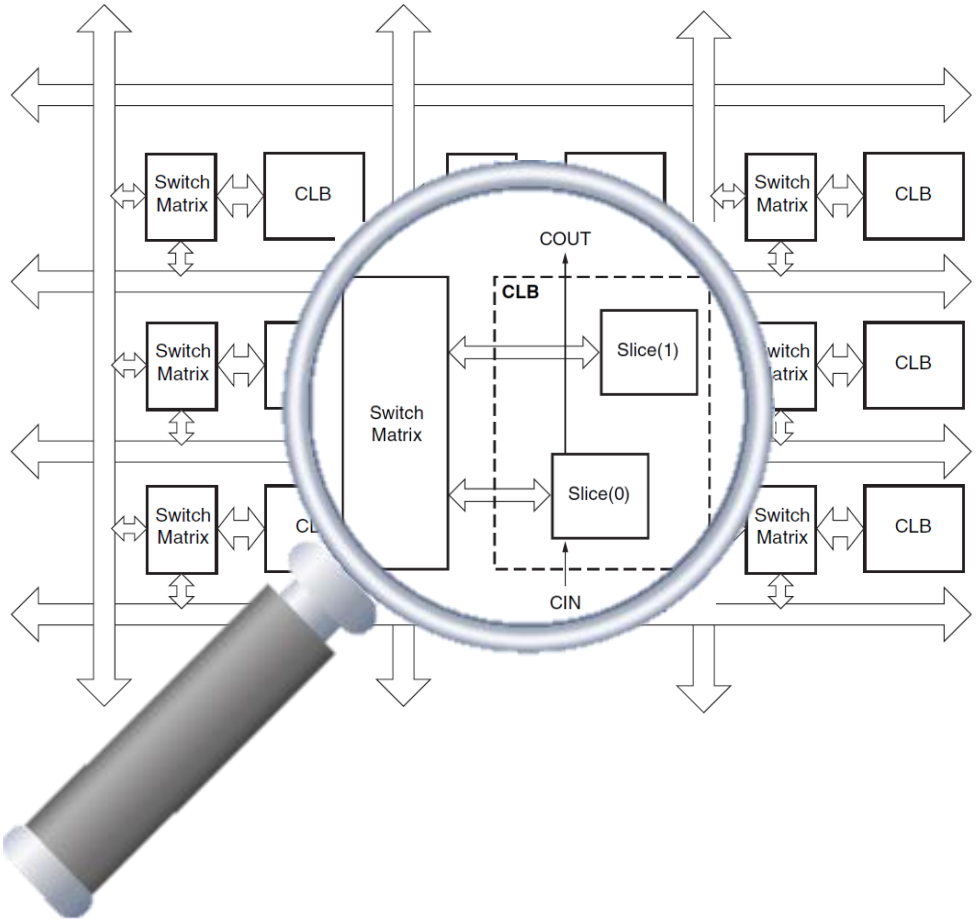
- Alice and Bob both have 255-bit public and private keys
- both derive their public key by a **scalar multiplication** using their private key and the public base point $x = 9$
- shared secrets are derived by a second **scalar multiplication** using public and private keys
- public keys and shared secrets consist only of X instead of (X, Y)

FIELD-PROGRAMMABLE GATE ARRAYS (FPGA)

SCHEMATIC LAYOUT OF MODERN FPGAS

Modern FPGAs are highly regular arrays of freely programmable logic blocks:

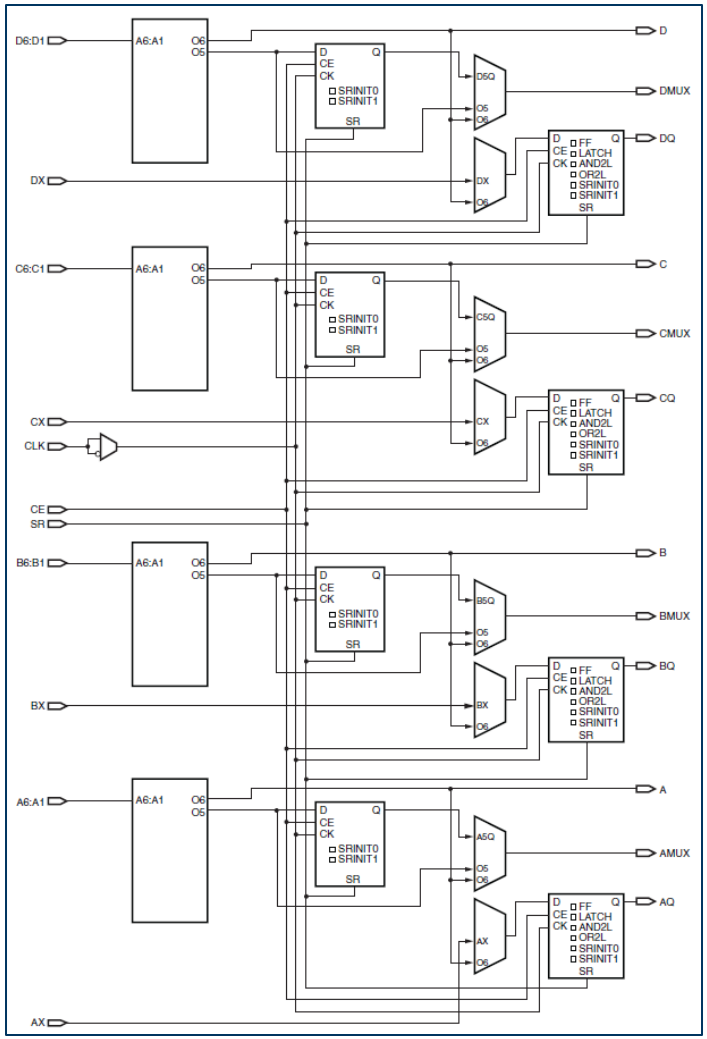
- **General Purpose Logic**
  - Configurable Logic (CLB)
  - Slices (Slice-X, Slice-L, Slice-M)
  - Look-Up Tables (LUTs)
  - Flip-Flops (FFs)
  - …
- **Routing**
  - Programmable Switch Matrix
  - Programmable Interconnections
- **Special Purpose Logic**
  - Digital Signal Processors (DSP)
  - (True Dual-Port) Block Memory (BRAM)
  - I/O blocks
  - …



GENERAL PURPOSE LOGIC

Modern Xilinx FPGAs (7-Series) provide up to three different types of slices for various use-cases:

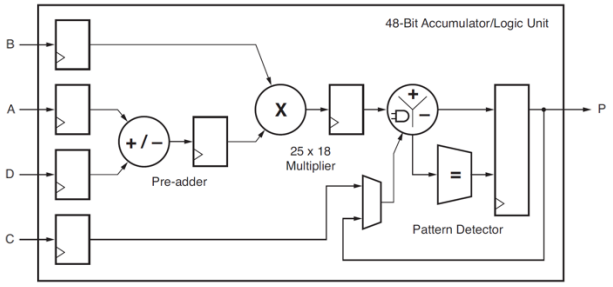
- **SLICE-X:**
  - *basic slice architecture*
  - 4 independent Look-Up Tables (LUT), each as:
    - one 6-to-1 Boolean function
    - two 5-to-2 Boolean functions (shared inputs)
  - 8 Flip-Flops (for clock synchronization of outputs)
- **SLICE-L**
  - *arithmetic and logic slice architecture*
  - additional arithmetic for fast carry-handling
  - wide multiplexers to combine LUTs as 7-to1 or 8-to-1 Boolean function
- **SLICE-M**
  - *arithmetic, logic and memory slice architecture*
  - allows to use LUT configuration memory either as:
    - 256-bit distributed memory (RAM)
    - 128-bit shift registers (SR)



SELECTED SPECIAL PURPOSE LOGIC

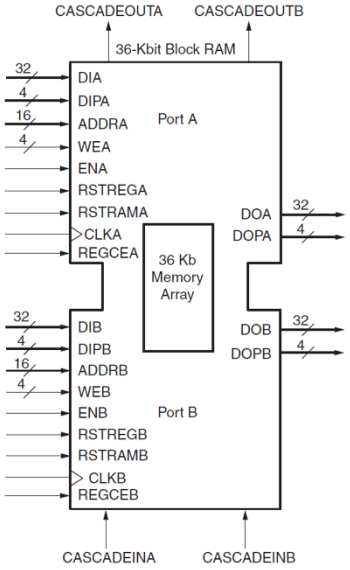
DIGITAL SIGNAL PROCESSOR (DSP)

- *dedicated, full-custom, low-power digital signal processor*
- 25×18 signed multiplier (with dynamic bypass)
- 48-bit accumulator
- optional 24-bit pre-adder
- multiple register stages for high-speed, pipelined operation



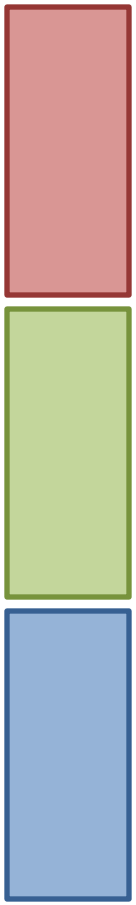
BLOCK MEMORY (BRAM)

- *efficient, low-power data storage, e.g. for large look-up tables*
- each BRAM can store up to 36 Kbits of data
- configurable data input and output width
- true-dual port access capabilities (with independent clocks)
- additional Error Correction Code capabilities



CURVE25519 ON RECONFIGURABLE HARDWARE

IMPLEMENTING PRIME FIELD ELLIPTIC CURVE CRYPTOGRAPHY



SCALAR MULTIPLICATION:

variable scalar-point multiplication as sequence of point additions and doublings

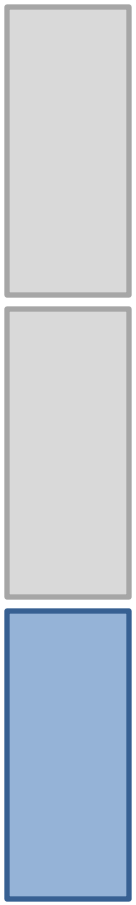
GROUP ARITHMETIC:

point addition and doubling using finite field arithmetic and curve specific formulas

FINITE FIELD ARITHMETIC:

addition, subtraction, multiplication, inversion, reduction in *GF(p)*

IMPLEMENTING PRIME FIELD ELLIPTIC CURVE CRYPTOGRAPHY



SCALAR MULTIPLICATION:

variable scalar-point multiplication as sequence of point additions and doublings

GROUP ARITHMETIC:

point addition and doubling using finite field arithmetic and addition/doubling formulas

FINITE FIELD ARITHMETIC:

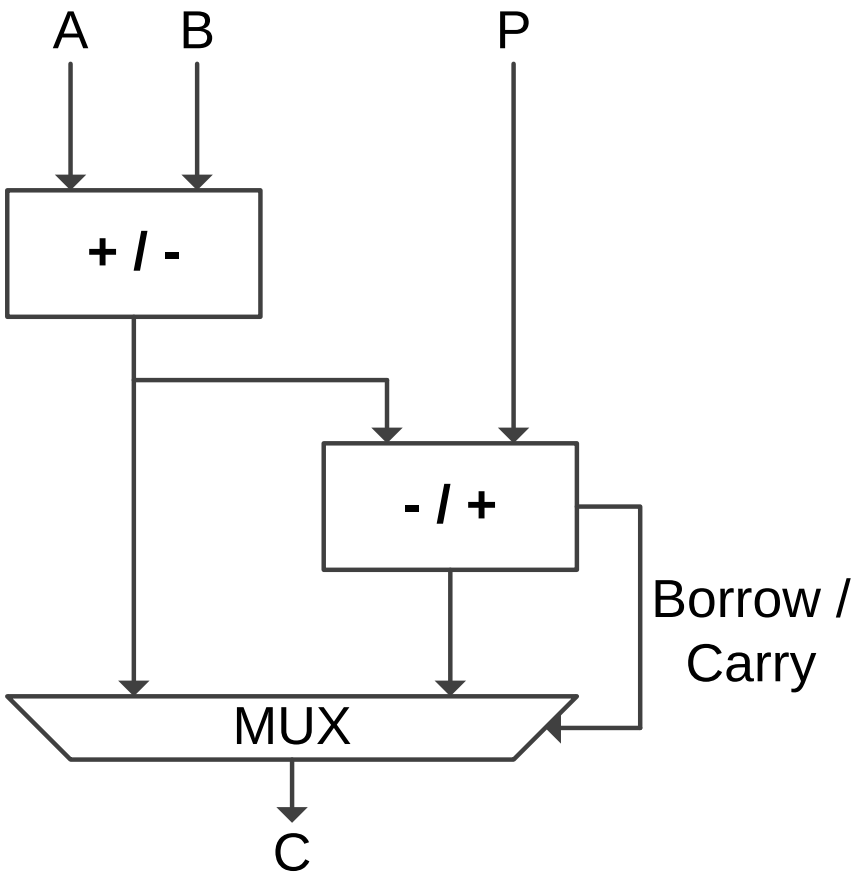
addition, subtraction, multiplication, inversion, reduction in *GF(p)*

THE MODULAR ADDITION AND SUBTRACTION UNIT

MODULAR ADDITION/SUBTRACTION:

$$C = A \pm B \ (mod\ P)$$

- includes reduction modulo p
- simple and elegant design in hardware
- always performs addition/subtraction in combination with reduction
- serial computation using 2 DSP units
  - 1st DSP performs main operation
  - 2nd DSP performs reduction
- output selection depending on final carry/borrow
- 34-bit operands A, B, P and 8 steps for final result



THE MODULAR REDUCTION SCHEME (I)

PSEUDO MERSENNE PRIME:

$$A\ prime\ of\ the\ form\ p = 2^n \pm a, 0 < a < 2^{\frac{n}{2}}, is\ a\ Pseudo\ Mersenne\ Prime.$$

MODULO REDUCTION WITH PSEUDO MERSENNE PRIMES:

I. in general, it holds: $\boldsymbol{2^n \equiv \pm a\ (mod\ p)}$

II. further: $\boldsymbol{A \times B = C = C_H \times 2^n + C_L}$

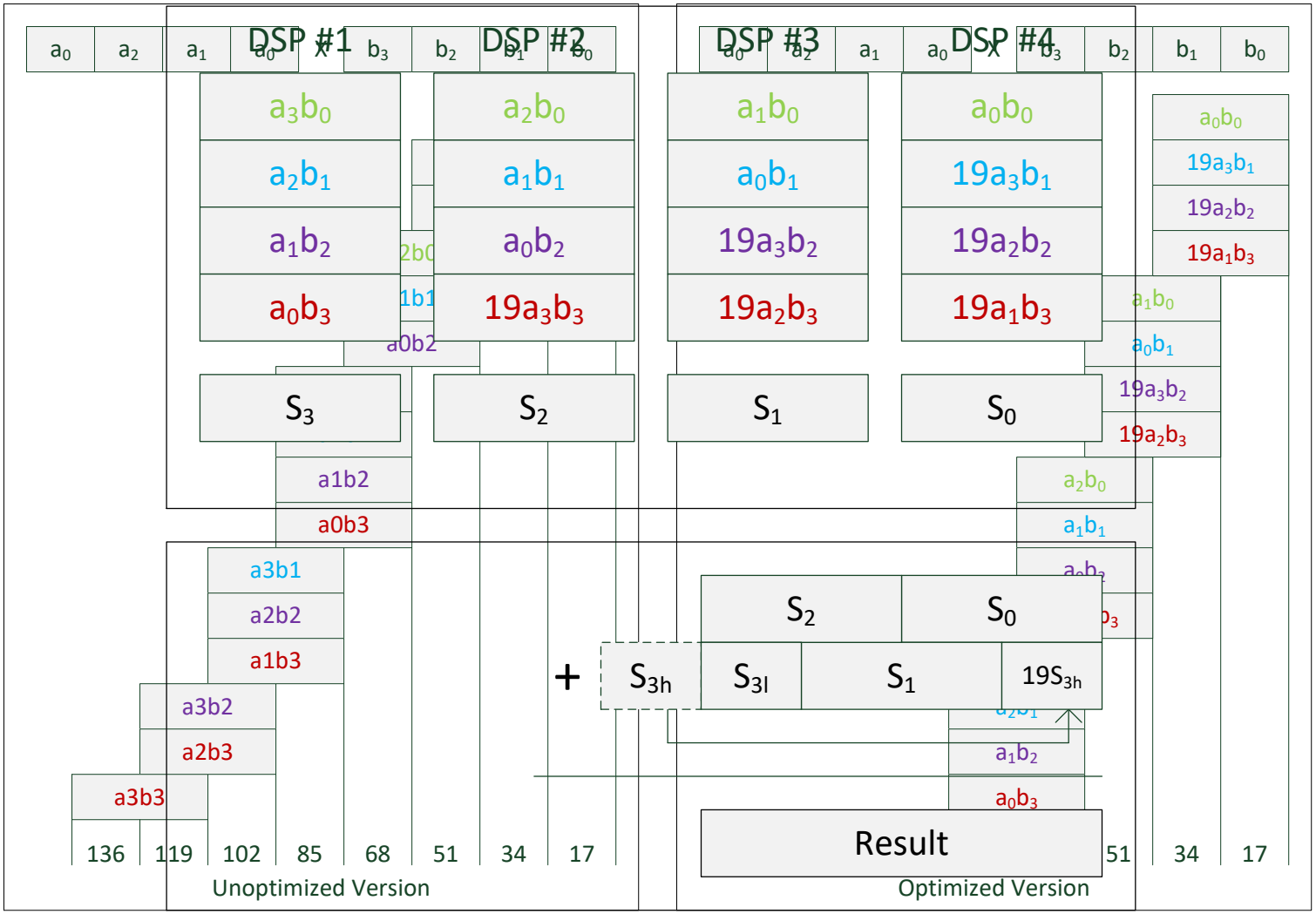
III. reduction results in: $\boldsymbol{C \equiv C_H \times 2^n + C_L \equiv C_L \pm C_H \times a\ (mod\ p)}$

OBSERVATIONS:

- it might be necessary to apply III. multiple times
- reduction uses to multiplication with (small) constant and addition
- can be (partially) interleaved with multiplication

I will present a small example in order to illustrate the basic concept…

THE MODULAR REDUCTION SCHEME (II)

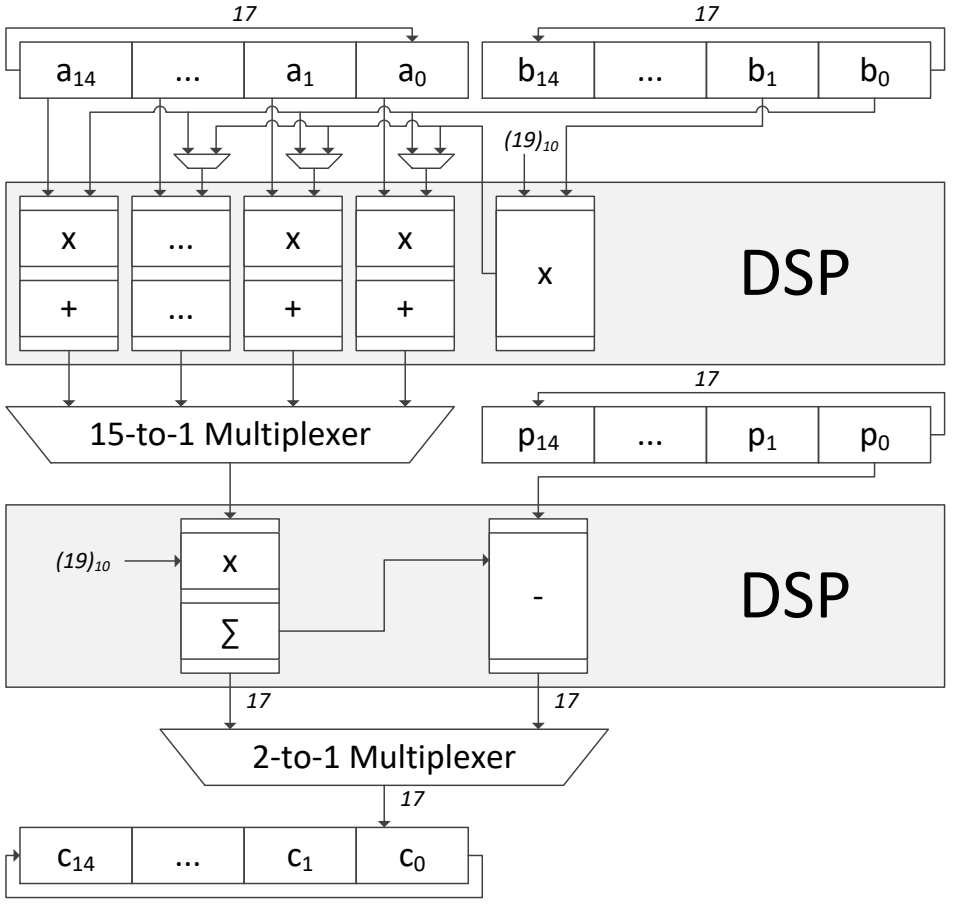


THE MODULAR MULTIPLICATION UNIT

MODULAR MULTIPLICATION/SQUARING:

$$C = A \times B \ (mod\ P)$$

- largest and most sophisticated component of final ECC architecture
- performs multiplication including reduction
- benefits from fact: $255 = 15 \times 17$
- DSPs allow 25×18 signed or 24×17 unsigned multiplications
- uses 18 DSP units in total:
  - 15 DSPs for partial products
  - 1 DSP for pre-reduction
  - 2 DSP for post-reduction
- separated and arranged in 2 stages (allows pipelining to increase throughput):
  - 1st stage: partial products
  - 2nd stage: post-reduction



FERMAT'S LITTLE THEOREM

MODULAR INVERSION:

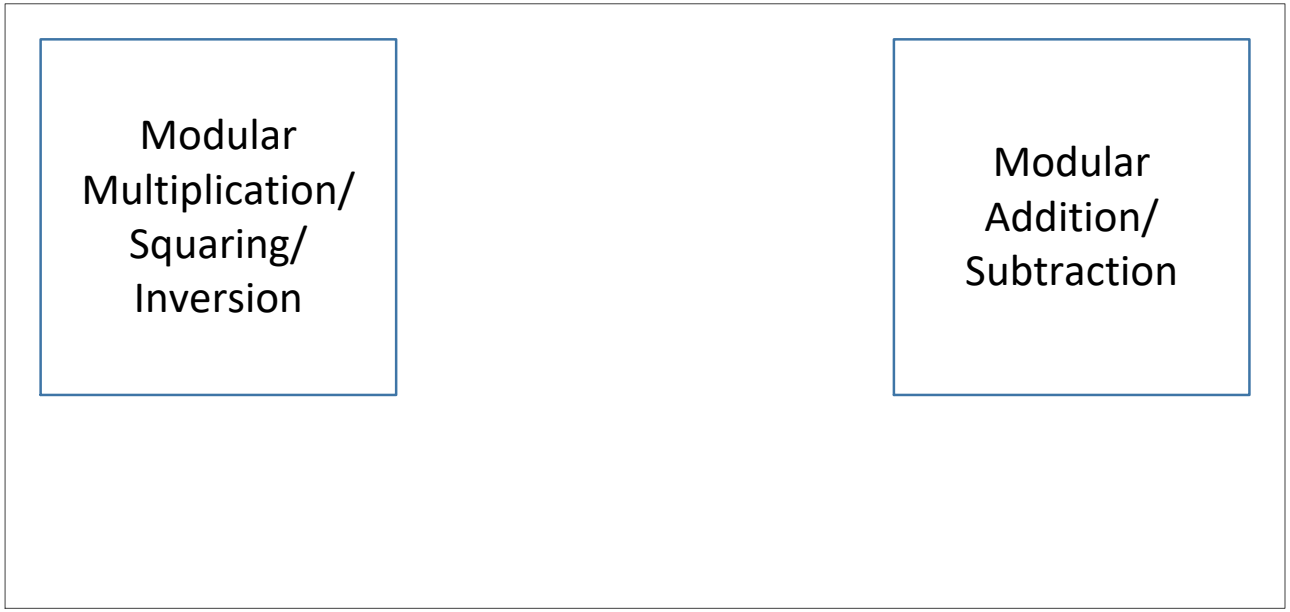
$$a^{p-2} \equiv a^{-1} \ (mod\ p)$$

- computes affine point from projective point: $X_A = X_P \times {Z_P}^{2^{255}-21}$
- performed only once per scalar multiplication in order to transform final result
- implemented using sequence of modular multiplications and squarings
- no additional hardware unit required
- requires 265 multiplications/squaring and 14630 cycles

ALTERNATIVE:

- modular inversion can be implemented as dedicated inversion unit (using Extended Euclidean Algorithm)
- requires additional hardware unit but is faster than FLT
- requires only 1667 cycles but about 2800 FF/3600 LUTs (~ single Curve25519 core)
- additional cost can be abated using resource sharing (e.g. for multi-core architectures)

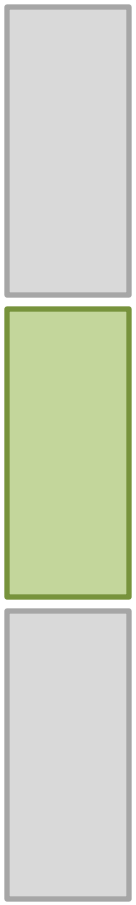
OVERALL ARCHITECTURE



IMPLEMENTING FINITE FIELD ARITHMETIC FUNCTIONALITY:

- independent units for modular squaring/multiplication and addition/subtraction
- allows parallel operation of addition and multiplication unit

IMPLEMENTING PRIME FIELD ELLIPTIC CURVE CRYPTOGRAPHY



SCALAR MULTIPLICATION:

Variable scalar-point multiplication as sequence of point additions and doublings

GROUP ARITHMETIC:

point addition and doubling using finite field arithmetic and curve specific formulas

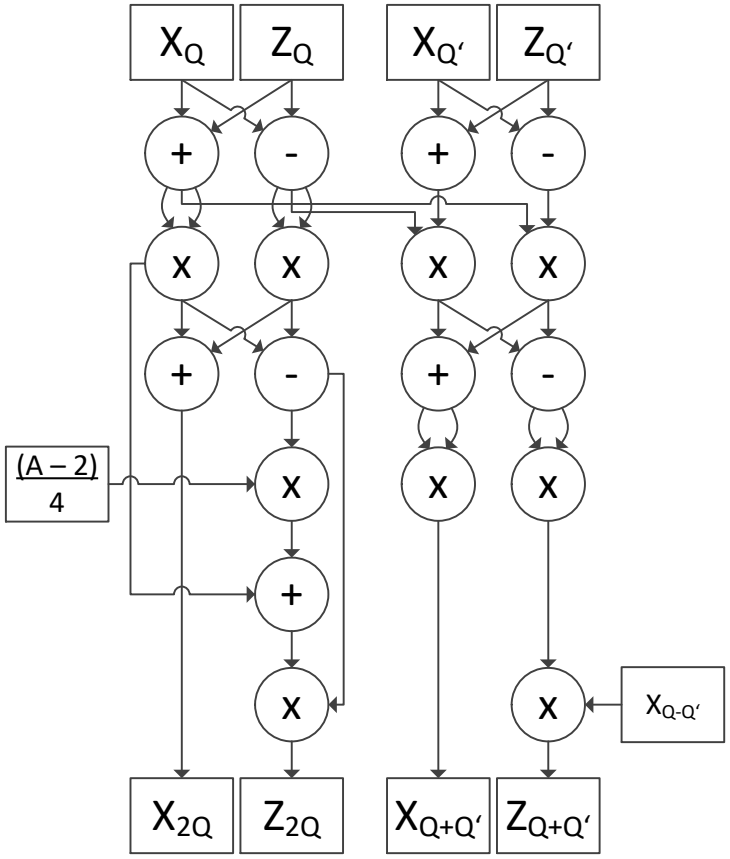
FINITE FIELD ARITHMETIC:

addition, subtraction, multiplication, inversion, reduction in *GF(p)*

THE MONTGOMERY LADDER

MONTGOMERY LADDER:

- time-invariant, combined double-and-add algorithm
- order of inputs depending on current bit of secret scalar
- operates on projective coordinates (x/z)
- implemented as sequence of additions, subtraction, squarings and multiplications
- (almost) each squaring/multiplication is followed by an addition/subtraction



POINT DOUBLING:

$$x_{2Q} = (x - z)^2(x + z)^2$$

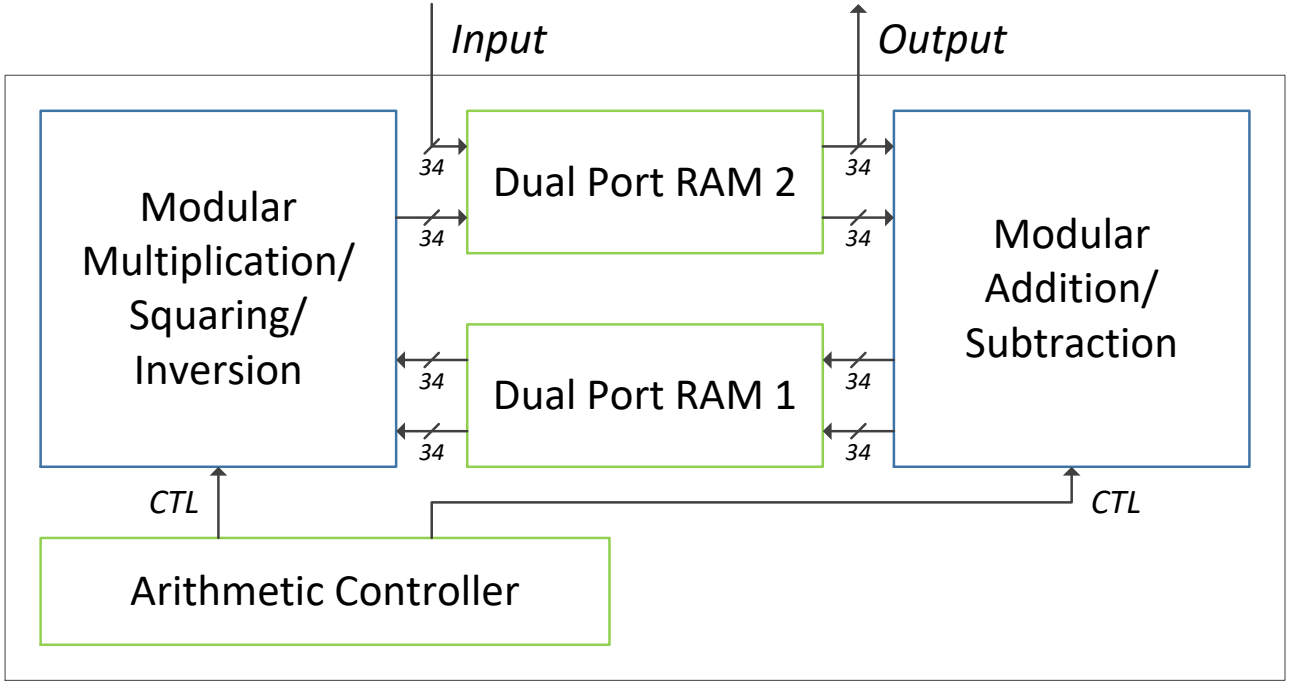
$$z_{2Q} = 4xz(x^2 + Axz + z^2)$$

POINT ADDITION:

$$x_{Q+Q'} = 4(xx' - zz')$$

$$z_{Q+Q'} = 4(xz' - zx')x_1$$

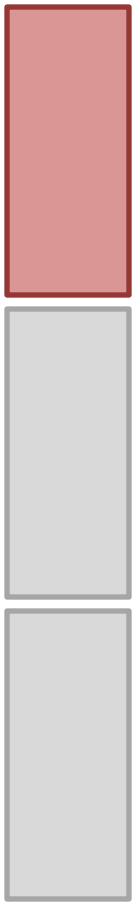
OVERALL ARCHITECTURE



IMPLEMENTING GROUP ARITHMETIC FUNCTIONALITY:

- circular operation flow (every multiplication followed by addition)
- Arithmetic Controller implements sequence of Montgomery ladder and inversion (Fermat's Little Theorem)
- Arithmetic Controller controls order of inputs for Montgomery ladder (depending on secret scalar)

IMPLEMENTING PRIME FIELD ELLIPTIC CURVE CRYPTOGRAPHY



SCALAR MULTIPLICATION:

variable scalar-point multiplication as sequence of point additions and doublings

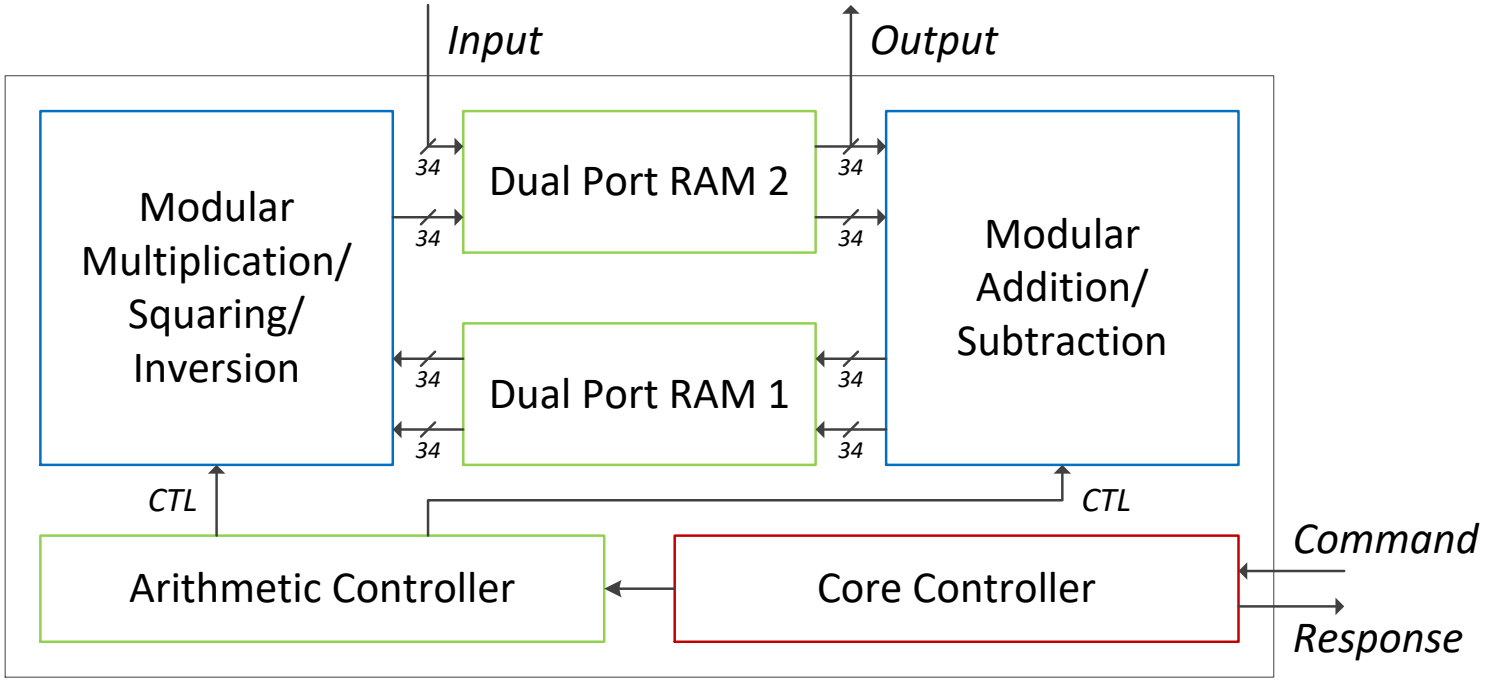
GROUP ARITHMETIC:

point addition and doubling using finite field arithmetic and curve specific formulas

FINITE FIELD ARITHMETIC:

addition, subtraction, multiplication, inversion, reduction in *GF(p)*

OVERALL ARCHITECTURE

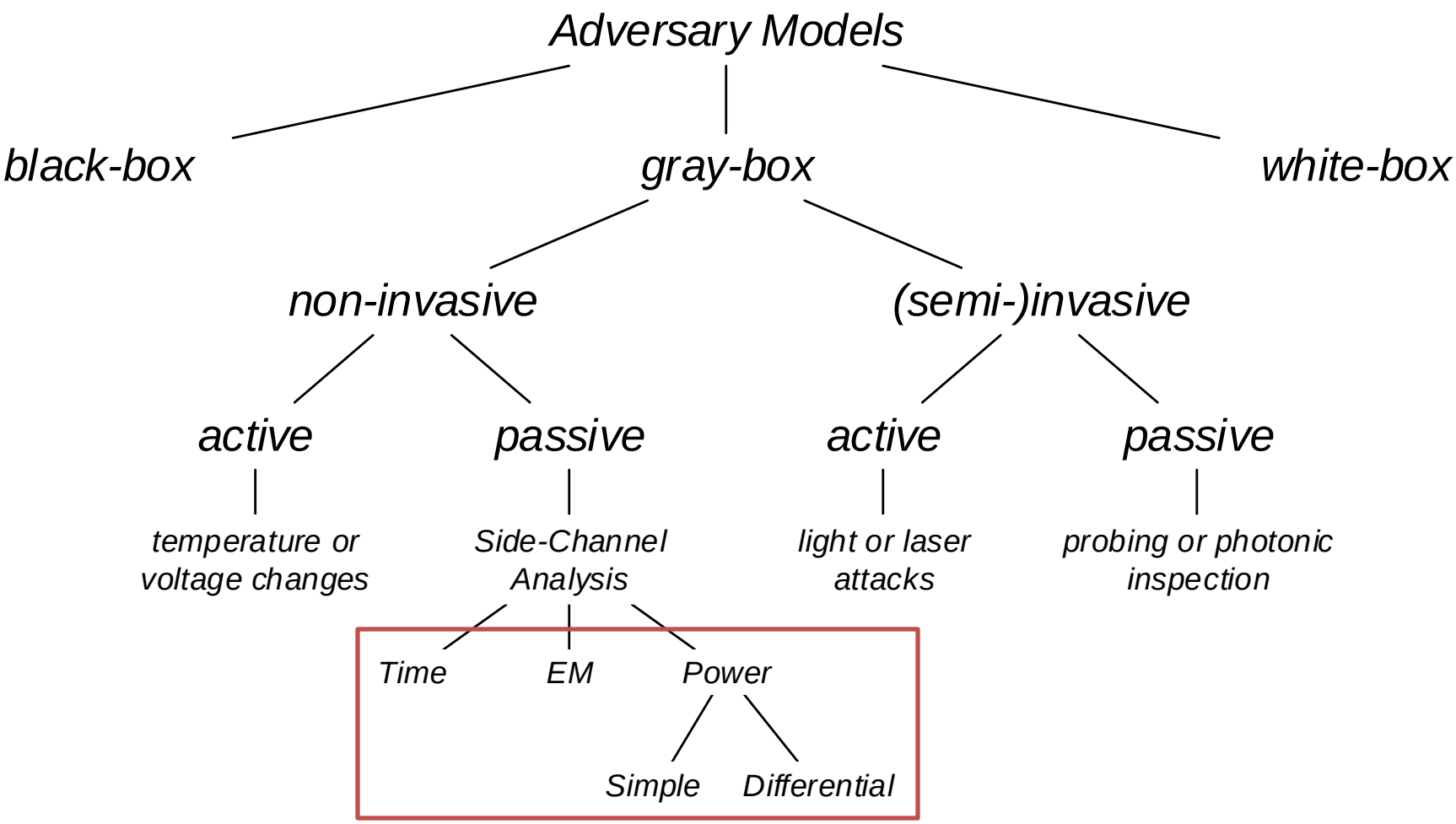


IMPLEMENTING SCALAR MULTIPLICATION FUNCTIONALITY:

- Core Controller manages scalar multiplication by calling Montgomery ladder and final inversion accordingly
- manages external communication using a command and response based protocol

ADDING SIDE-CHANNEL PROTECTION

CLASSIFICATION OF ADVERSARY MODELS

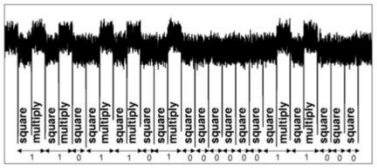


CONSIDERED SIDE-CHANNEL ATTACKS

TIMING ANALYSIS:

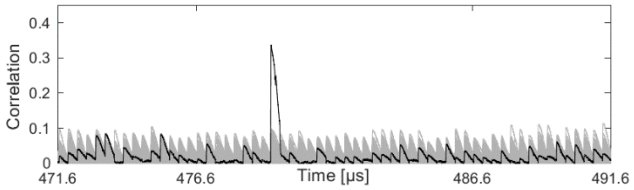
- exploits dependencies of secret data and execution time
- ECC is vulnerable if not implemented carefully:
- implementing constant time algorithms prevents timing analysis

SIMPLE POWER ANALYSIS (SPA):



- measurement and inspection of instantaneous power consumption of a device
- attacker tries to reveal patterns depending on secret scalar (e.g., point addition or point multiplication)
- requires only a few power traces but deep knowledge about implementation
- SPA countermeasures aim to unify patterns (e.g., always double and add points)

DIFFERENTIAL POWER ANALYSIS (DPA):



- exploits data or key dependencies in the power consumption
- predicts power consumption and compares to actual consumption
- uses statistical test rather than visual inspection
- requires larger amounts of power traces
- many different ways to implement countermeasures

SIDE-CHANNEL COUNTERMEASURES AGAINST DPA

MASKING / BLINDING:

Randomly mask or blind intermediate values and possible leakage.

- requires additional source for randomness
- may require changes in algorithmic flow and implementation (components)
- usually applied on inputs

HIDING:

Hide side-channel leakage within (random) noise (Signal-to-Noise Ratio).

- decrease leakage signal (power equalization,…)
- increase random noise (dummy operations, shuffling,…)

RE-KEYING:

Change keys frequently to mitigate key extraction.

- key distribution and synchronization is difficult
- usually comes with significant performance drop
- leakage-resilient algorithms

SCALAR BLINDING (MASKING)

GENERAL IDEA: Re-compute secret scalar in a way that intermediate results differ but final result is still correct.

OBSERVATION: Given the group order $\#E$, computing $\#E \times P = O$ results in the point at infinity.

RANDOMIZATION OF SECRET SCALAR:

- instead of using the original scalar k we compute a new scalar $k' = k + r \times \#E$
- correctness is given by:

$$k' \times P = (k + r \times \#E) \times P = k \times P + r \times O = k \times P$$

- requires a random value r (should be ~ 128 bit – we only implemented 24 bit for proof of concept)
- increases the bit size of scalar and hence runtime of scalar multiplication

COUNTERMEASURE:

- even using same scalars as input will yield in different intermediate values
- power consumption and side-channel leakage is randomized to prevent (or hamper) DPA attacks

RANDOMIZED PROJECTIVE COORDINATES (MASKING)

GENERAL IDEA: Re-compute public point P in a way that intermediate results differ but final result is still correct.

OBSERVATION: Points (X, Y) are represented by projective coordinates $(X/Z, Z)$ and initially: $Z = 1$.

RANDOMIZATION OF PUBLIC POINT:

- introducing projective coordinates relaxes computations and introduces higher degrees of freedom
- instead of choosing $\mathrm{Z} = 1$ we choose a random 255-bit value $\mathrm{Z} = \lambda$
- updating $P = (\lambda X, Z)$ ensures correctness of final result (Y coordinate is omitted during operation)
- countermeasure requires 255-bit randomness and an additional modular multiplication (prior to scalar multiplication)

COUNTERMEASURE:

- even using same points as input will yield in different intermediate values
- power consumption and side-channel leakage is randomized to prevent (or hamper) DPA attacks

MEMORY ADDRESS SCRAMBLING (HIDING)

OBSERVATION: Montgomery ladder prevents timing and SPA attacks but still has dependencies to the secret scalar.

RANDOMIZATION OF BRAM MEMORY ADDRESSES:

- BRAM addressing is dependent on the secret scalar
  - if current scalar bit is 0: first address will be Q, second will be Q'
  - if current scalar bit is 1: first address will be Q', second will be Q
- addresses should be scrambled after every scalar multiplication in order to hide dependencies
- scrambling of 6-bit addresses can be realized using a LFSR and requires 6 bit of randomness

COUNTERMEASURE:

- BRAM addresses for Q and Q' will be different after every execution
- our approach still might have a limited complexity but practically hardens design against DPA attacks

RESULTS AND COMPARISON

IMPLEMENTATION AND PERFORMANCE RESULTS

SINGLE-CORE ARCHITECTURE

Component	Unprotected	Protected	Available	Utilization
Number of Slice Registers	3592	3784	106400	3% / 3%
Number of Slice LUTs	2783	2862	53200	5% / 5%
Number of Occupied Slices	1029	1180	13300	7% / 8%
Number of DSP48E1	20	22	220	9% / 10%
Number of RAMB36E1	2	2	140	1% / 1%
Cycles per Step Function	64770	68880	*at 200MHz*	
Cycles per Inversion	14630	14372	*at 200MHz*	
Total Cycles	79400	83252	*at 200MHz*	

UNPROTECTED VS. PROTECTED:

- only slight increase of area and hardware resources (192 FF, 79 LUT, 2 DSP)
- increased latency due to larger scalar size (total increase of 24-bit due to randomization)
- small optimization in dataflow of inversion/LFT (258 clock cycles)

SIDE-CHANNEL COUNTERMEASURES

Countermeasure	Delay		Area		
	Initialization	*Computation*	*LUTs*	*Registers*	*DSP*
Random Scalar Blinding	17	6150	306	311	2
Random Projective Coordinates	45	-	5	4	-
Random BRAM Addresses	22	-	1	6	-

SCALAR BLINDING:

- only uses small random r with 24 bits
  - might be problematic in practice
  - can be increased easily at cost of additional latency
- requires two additional DSPs and $|r|$ additional Montgomery ladder calls

RANDOM COORDINATES:

- requires 255-bit random and small modification in arithmetic controller
- coordinate multiplication can be realized using finite field multiplication

ADDRESS SCRAMBLING:

- requires 6-bit random seed for LFSR
- executed prior to every scalar multiplication

COMPARISON TO OTHER WORK

A fair comparison to hardware implementations of Curve25519 is hardly possible, but related work:

NIST P-256 (Güneysu and Paar, 2008):

- standardized elliptic curve (NIST) offering same security level as Curve25519
- uses Generalized Mersenne Prime for fast reduction using additions/subtractions
- 20% slower using 45% more hardware resources (logic and DSPs)

FOUR$\mathbb{Q}$ (Järvinen et al., 2016):

- recently proposed high-performance elliptic curve with 128-bit of security
- uses four dimensional decomposition on a $\mathbb{Q}$-curve
- arithmetic is performed using Mersenne prime $p = 2^{127} - 1$
- 265% faster using 45% more logic and 20% more DSP resources

CURVE448 (Sasdrich and Güneysu, 2016):

- Curve448: second candidate of RFC7748 (along with Curve25519) offering 224-bit of security
- 85% slower using only 45% more DSPs (but same logic resources)

CONCLUSION

WHAT YOU SHOULD TAKE HOME OF THIS TALK…

HIGH-PERFORMANCE ASYMMETRIC CRYPTOGRAPHY…

…allows modern systems and devices to process thousands of signatures per second.

FIELD-PROGRAMMABLE GATE ARRAYS…

…provide many dedicated logic resources that allow efficient cryptographic implementations.

CURVE25519…

… was originally design for fast software implementations but even supports fast hardware implementations (on FPGAs).

HIGHLY OPTIMIZED FINITE FIELD ARITHMETIC…

…is the foundation of high-performance ECC architectures and requires thorough engineering.

SIDE-CHANNEL ANALYSIS ATTACKS…

…are a big threat to modern embedded and constrained devices, ***but:***

- *carefully chosen ECC parameters inherently provide protection against Timing and SPA attacks*
- *Masking and Hiding countermeasures can be implemented easily atop of ECC architectures*

RUHR-UNIVERSITÄT BOCHUM

hgi EMSEC

RUB

IMPLEMENTING CURVE25519 FOR SIDE-CHANNEL-PROTECTED ELLIPTIC CURVE CRYPTOGRAPHY

pascal.sasdrich@rub.de

20TH WORKSHOP ON ELLIPTIC CURVE CRYPTOGRAPHY, YAŞAR UNIVERSITY, İZMIR, TURKEY SEPTEMBER 6, 2016

Thank you for your attention!
Any questions?

BACKUP SLIDES

XILINX FPGA DESIGN FLOW

HDL

HARDWARE DESCRIPTION LANGUAGE:

Formal description of the hardware design using a HDL, e.g., Verilog or VHDL.

SYN

DESIGN SYNTHESIS:

Conversion of the HDL description into a netlist, i.e., a formally written digital circuit schematic.

MAP

MAP DESIGN ELEMENTS TO DEVICE RESOURCES:

Mapping of the netlist onto particular device internal structures and elements.

PAR

PLACE AND ROUTE DESIGN RESOURCES:

Layout and allocate FPGA resources (logic and routing) in order to implement hardware design.

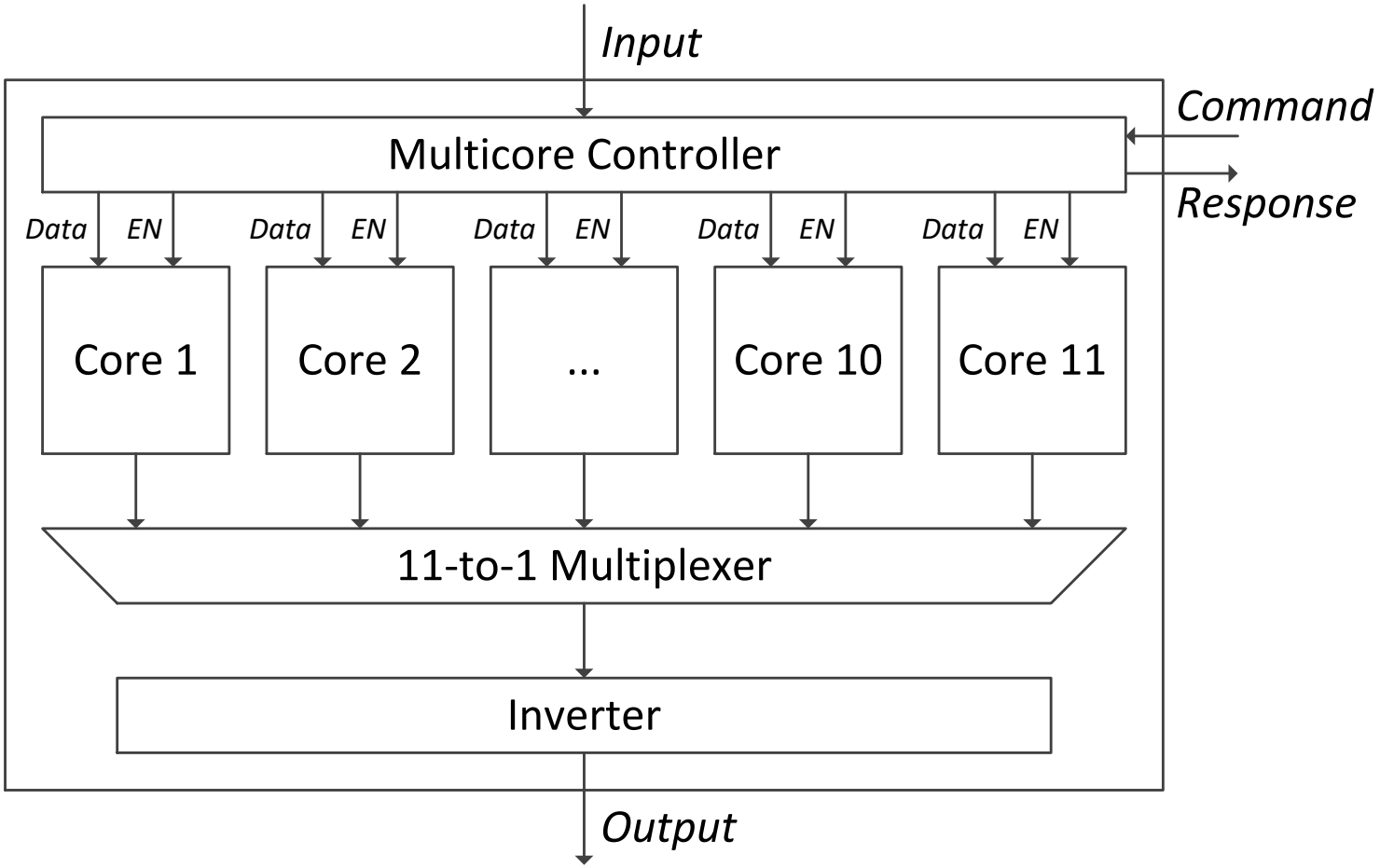
BIT

GENERATE BITSTREAM FILE:

Generate bitstream file that contains final FPGA configuration and which can be loaded to the device.

HIGH-PERFORMANCE MULTI-CORE ARCHITECTURE

MULTI-CORE ARCHITECTURE (I)



MULTI-CORE ARCHITECTURE (II)

Component	Unprotected	Protected	Available	Utilization
Number of Slice Registers	43875	39916	106400	41% / 37%
Number of Slice LUTs	34009	30582	53200	63% / 57%
Number of occupied Slices	11277	10777	13300	84% / 81%
Number of DSP48E1	220	220	220	100% / 100%
Number of RAMB36E1	22	20	140	15% / 14%
Cycles per Step Function	64770 @11 cores	68880 @10 cores	*at 200MHz*	
Cycles per Inversion	1667	1667	*at 100MHz*	
Total Cycles	34052	36107	*at 100MHz*	

COMPARISON TO OTHER WORK

Scheme	Device	Implementation	Logic	Clock	OP/s
Single-Core[1]	XC7Z020	255-bit Curve25519	1029 LS/20 DSP	200 MHz	2519
Multi-Core[1]	XC7Z020	255-bit Curve25519	11277 LS/220 DSP	11×100 MHz	32304
Single-Core[2]	XC7Z020	255-bit Curve25519	1169 LS/22 DSP	200 MHz	2402
Multi-Core[2]	XC7Z020	255-bit Curve25519	11690 LS/220 DSP	10×100 MHz	27695
Single-Core[3]	XC7Z020	256-bit Four$\mathbb{Q}$	565 LS/16 DSP	190 MHz	3222
Single-Core[3]	XC7Z020	256-bit Four$\mathbb{Q}$	1691 LS/27 DSP	190 MHz	6389
Multi-Core[3]	XC7Z020	256-bit Four$\mathbb{Q}$	5697 LS/187 DSP	11×175 MHz	64730
Single-Core[1,4]	XC7Z020	448-bit Curve448	963 LS/30 DSP	100 MHz	400
Single-Core[2,4]	XC7Z020	448-bit Curve448	1146 LS/32 DSP	100 MHz	322
ECC[5]	XC4VFX12-12	256-bit GF(p), NIST	1715 LS/32 DSP	490 MHz	2020

[1] unprotected, [2] protected, [3] [Järvinen et al. 2016], [4] [Sasdrich and Güneysu 2016], [5] [Güneysu and Paar 2008],