NaCl's `crypto_box` in hardware

Michael Hutter, Jürgen Schilling, Peter Schwabe, and Wolfgang Wieser

Cryptography Research, TU Graz (IAIK), Radboud University Nijmegen

September 14, 2015

CHES 2015, Saint-Malo, France

NaCl and `crypto_box`

- Networking and Cryptography library - NaCl
- Easy-to-use and fast

NaCl and `crypto_box`

- Networking and Cryptography library - NaCl
- Easy-to-use and fast
- `crypto_box` offers *public-key authenticated encryption*
  - X25519 Diffie-Hellman key exchange (using Curve25519),
  - Salsa20 stream cipher, and
  - Poly1305 message-authentication code.

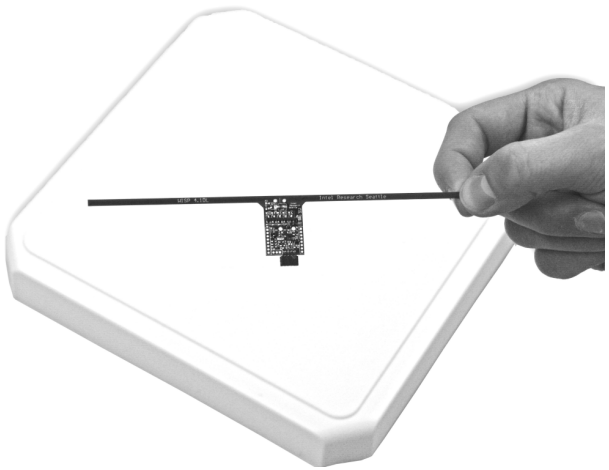
NaCl and `crypto_box`

- Networking and Cryptography library - NaCl
- Easy-to-use and fast
- `crypto_box` offers *public-key authenticated encryption*
  - X25519 Diffie-Hellman key exchange (using Curve25519),
  - Salsa20 stream cipher, and
  - Poly1305 message-authentication code.
- Allows fast and secure end-to-end communication via the Internet
- 128-bit security
- See also http://nacl.cr.yp.to

...but how does it perform in hardware?

...but how does it perform in hardware?

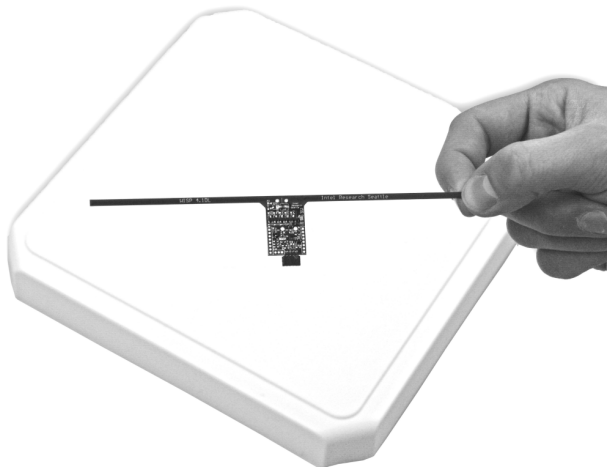
- `crypto_box` suitable for IoT?
- Wireless Identification and Sensing Platforms (WISPs)



...but how does it perform in hardware?

- `crypto_box` suitable for IoT?
- Wireless Identification and Sensing Platforms (WISPs)

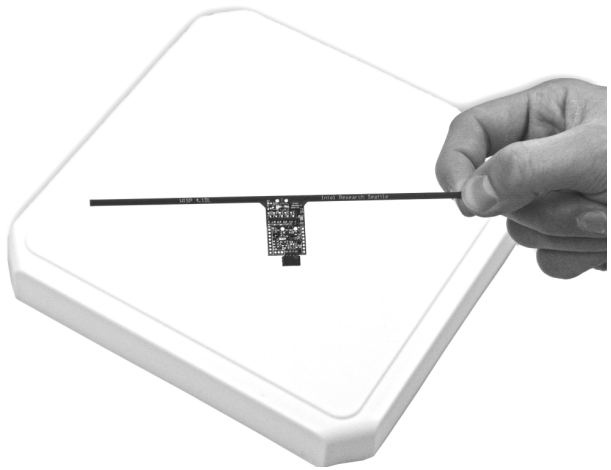
- So why not using SSL or IPSec?
  - Proposal from Gross et al. [1] at last year's RFIDsec
  - Chosen set of IPSec primitives: AES-128 and ECDH using NIST P-192
  - Still may require too much resources (52 kGEs)...



...but how does it perform in hardware?

- `crypto_box` suitable for IoT?
- Wireless Identification and Sensing Platforms (WISPs)

- So why not using SSL or IPSec?
  - Proposal from Gross et al. [1] at last year's RFIDsec
  - Chosen set of IPSec primitives: AES-128 and ECDH using NIST P-192
  - Still may require too much resources (52 kGEs)...



. . . can we do better?

What we did...

- We present a carefully optimized hardware architecture of the basic primitives of NaCl

What we did...

- We present a carefully optimized hardware architecture of the basic primitives of NaCl
- 128-bit public-key authenticated encryption

What we did...

- We present a carefully optimized hardware architecture of the basic primitives of NaCl
- 128-bit public-key authenticated encryption
- Compatibility with existing NaCl interfaces

What we did...

- We present a carefully optimized hardware architecture of the basic primitives of NaCl
- 128-bit public-key authenticated encryption
- Compatibility with existing NaCl interfaces
- No need for signatures

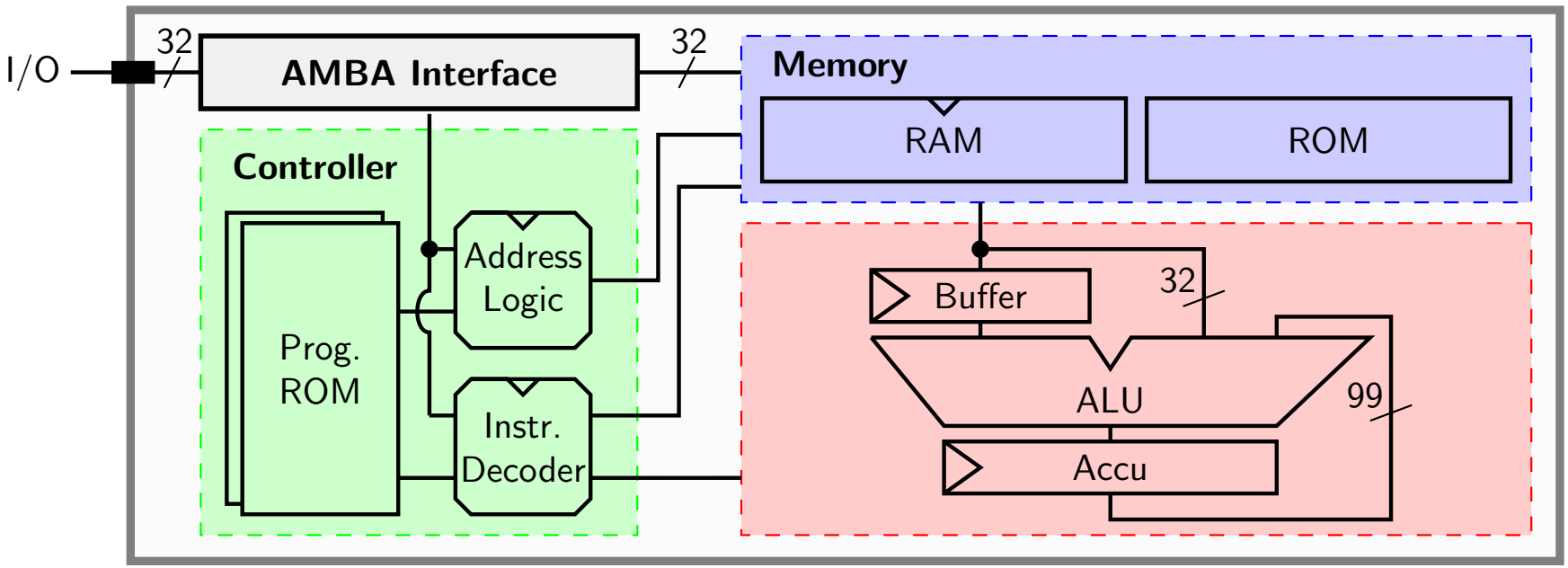
What we did...

- We present a carefully optimized hardware architecture of the basic primitives of NaCl
- 128-bit public-key authenticated encryption
- Compatibility with existing NaCl interfaces
- No need for signatures
- Low power, not low energy

What we did...

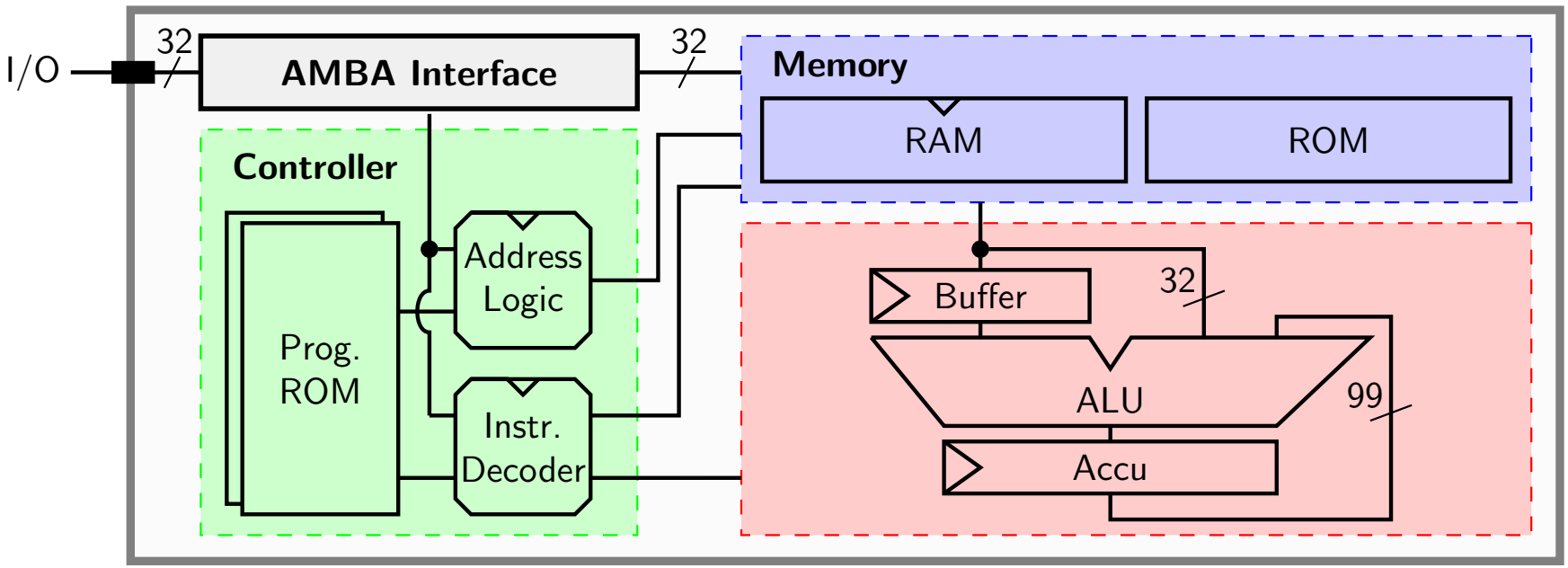
- We present a carefully optimized hardware architecture of the basic primitives of NaCl
- 128-bit public-key authenticated encryption
- Compatibility with existing NaCl interfaces
- No need for signatures
- Low power, not low energy
- Constant-runtime implementation

Hardware architecture overview



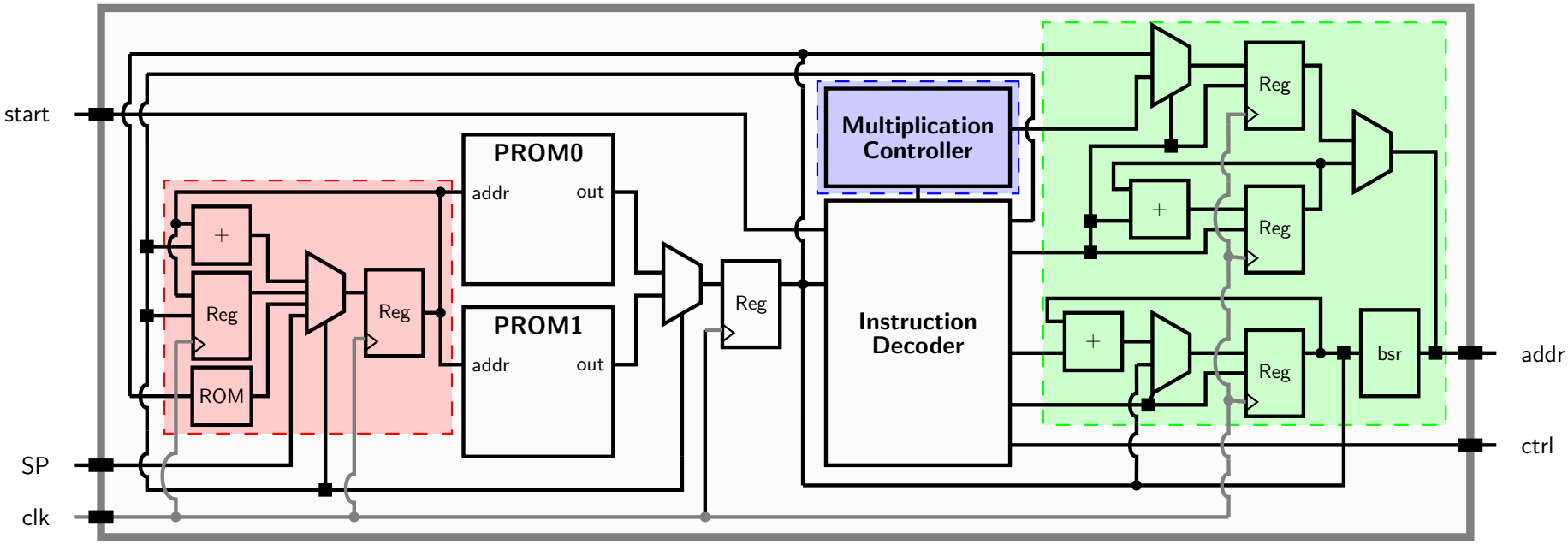
- 32-bit architecture with single-port memory

Hardware architecture overview



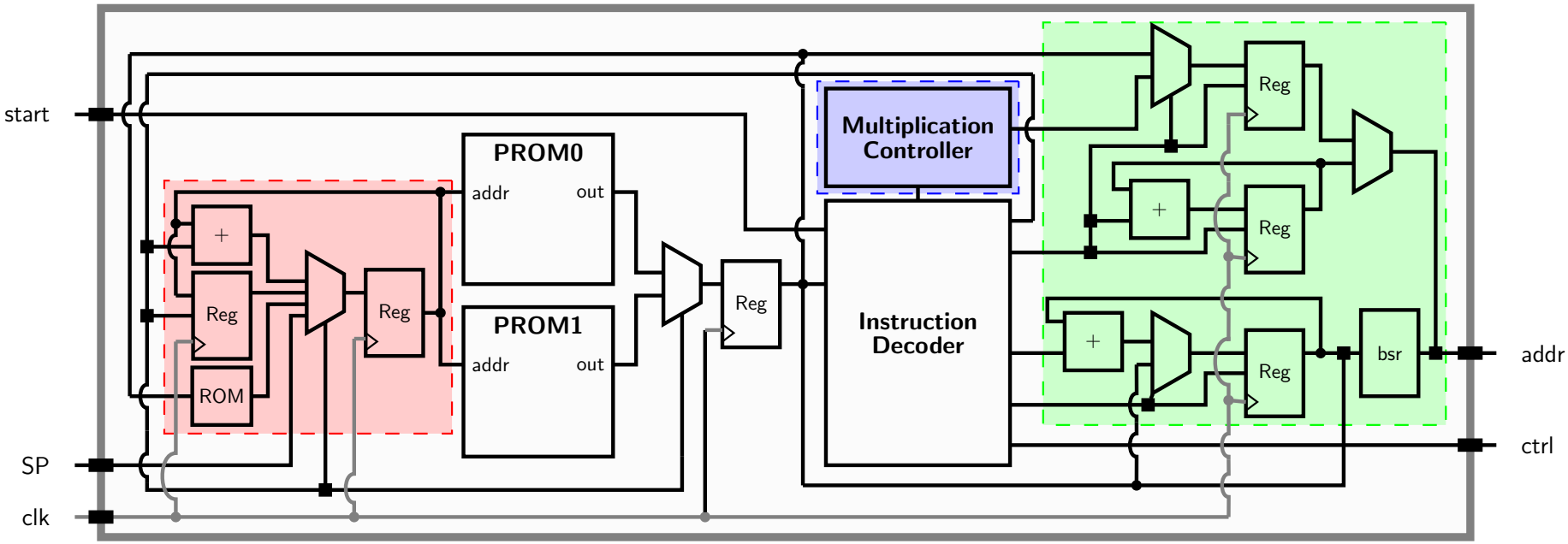
- 32-bit architecture with single-port memory
- ASIP tailored for `crypto_box` using microcode-control
  - Self-written "compiler" (written in Java) that generates machinecode
  - Automatically outputs RTL of the program ROM (ready to integrate)
  - Easy to use and to add functionality

The controller



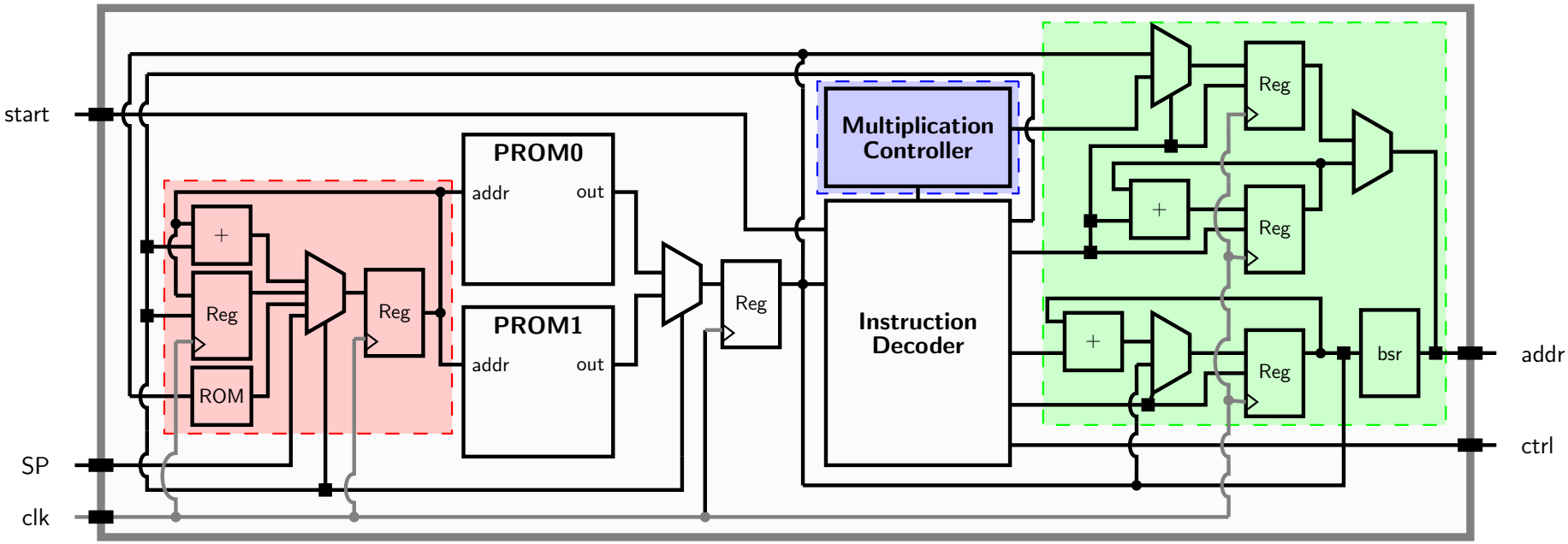
- 2 microcode program ROMs: Curve25519 and Salsa20/Poly1305
  - Splitting allows isolating ROMs to reduce power consumption
  - Area reduction if microcodes have different opcode lengths

The controller



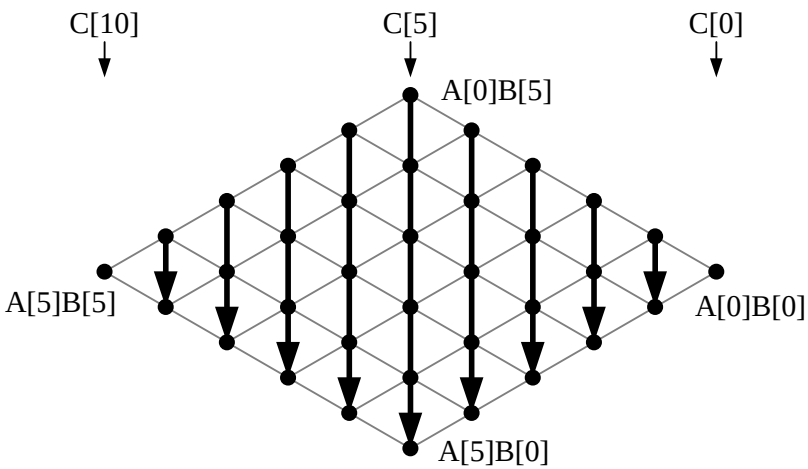
- 2 microcode program ROMs: Curve25519 and Salsa20/Poly1305
  - Splitting allows isolating ROMs to reduce power consumption
  - Area reduction if microcodes have different opcode lengths
- Support for single-level subroutines
  - 11-bit register stores return address, program counter update
  - Subroutine addressing: decoder using a look-up table (ROM)

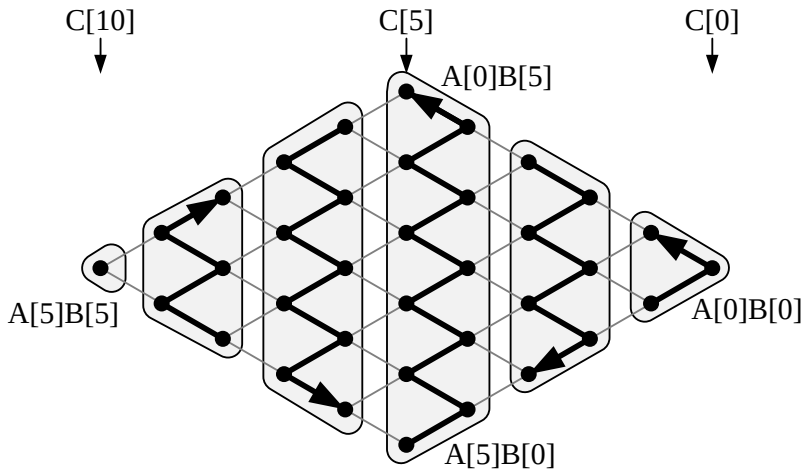
The controller



- 2 microcode program ROMs: Curve25519 and Salsa20/Poly1305
  - Splitting allows isolating ROMs to reduce power consumption
  - Area reduction if microcodes have different opcode lengths
- Support for single-level subroutines
  - 11-bit register stores return address, program counter update
  - Subroutine addressing: decoder using a look-up table (ROM)
- 256-bit multiplication controller (optional)

2-column product-scanning multiply control



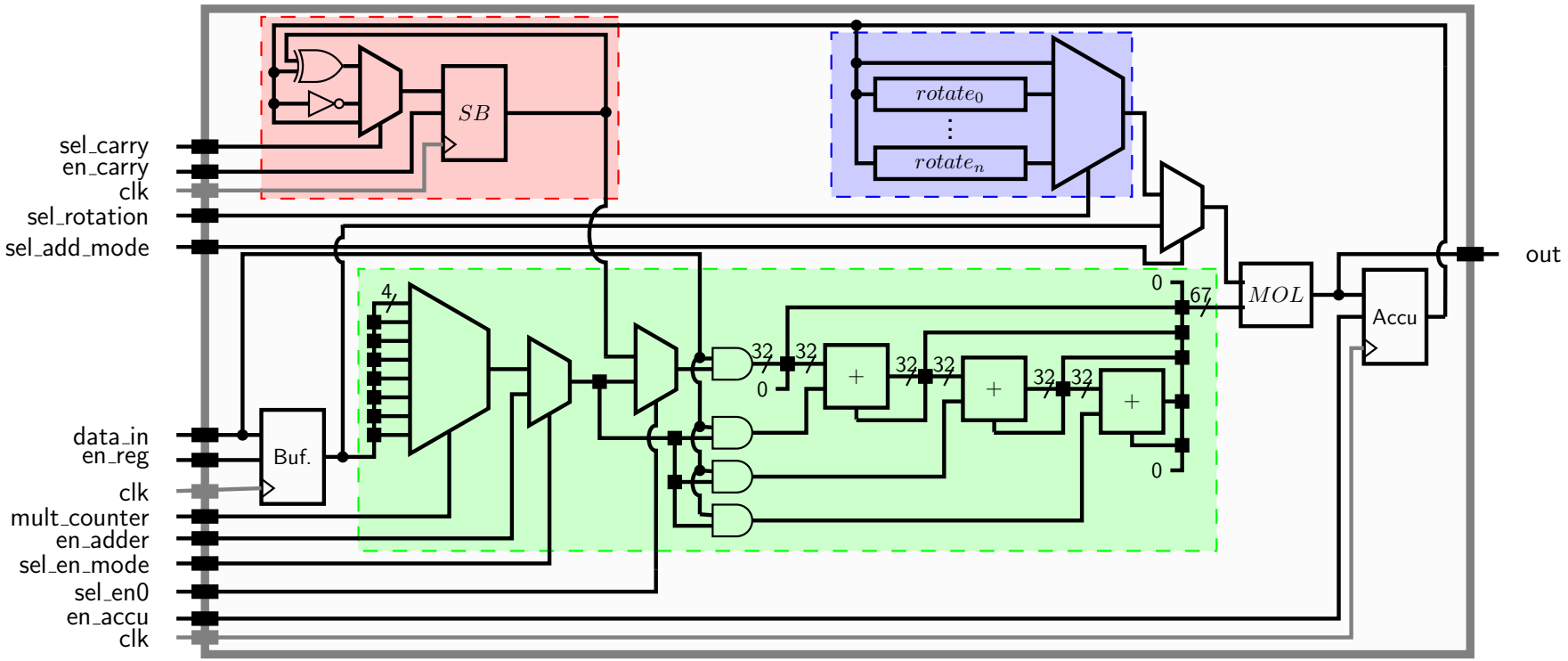


- We implemented product-scanning multiplication and process two columns in parallel
  - Column-wise product-scanning multiplication (left)
  - 2-column parallel product-scanning multiplication (right).
- Allows to hold one operand in a register while next operand is pre-fetched from memory

Memory paging

- Most of the time, `crypto_box` primitives require access to a limited number of RAM locations only
- Reduce length of address bits in opcode
  - Divide memory into virtual memory pages
  - One memory page consists of 4×256 bits of RAM
- Special instructions:
  - Memory Page Select (`MPS`)
  - Memory Page Increment (`MPI`)
  - Memory Page Decrement (`MPD`)
- Savings
  - Only 5 opcode bits are required
  - 2 bits to address a single 256-bit row of the currently selected page
  - 3 bits to address a single 32-bit word

ALU



- 32-bit digit-serial multiplier
  - Parameterizable digit width $w = 2, 4, 8, 12, 16$ bits
  - Also re-used for addition and subtraction
- Pre-fetch buffer used to store one 32-bit operand
- 32-bit logic operations: AND, OR, XOR
- 99-bit accumulator register with rotation unit

Crypto services

1. X25519 Diffie-Hellman key agreement
2. Authenticated encryption using a streaming API
   - Message is processed in chunks of 64 bytes
   - Support for authenticated decryption of a 32-byte message

Command	Hex	Description
`DH-1`	0x00	X25519 Diffie-Hellman key exchange: computes public key
`DH-2`	0x01	X25519 Diffie-Hellman key exchange: computes session key
`INIT`	0x02	HSalsa20: computes extended session key
`FIRST`	0x03	XSalsa20: computes first cipher block
`UPDATE`	0x04	XSalsa20: computes next cipher block
`FINALIZE`	0x05	Poly1305: computes authentication tag
`DECRYPT`	0x06	XSalsa20/Poly1305: decrypts and authenticates a single block

Subroutines

- Addition, subtraction, and multiplication
- Modular reduction in $\mathbb{F}_{2^{255}-19}$ (iterative approach)
- Modular inversion based on Fermat's little theorem ($11\mathrm{M} + 254\mathrm{S}$)

Subroutines

- Addition, subtraction, and multiplication
- Modular reduction in $\mathbb{F}_{2^{255}-19}$ (iterative approach)
- Modular inversion based on Fermat's little theorem ($11\mathsf{M} + 254\mathsf{S}$)

ECC scalar multiplication:

- Differential addition-and-doubling using Montgomery ladder
- Costs: $5\mathsf{M} + 4\mathsf{S} + 8\mathsf{add} + 1\mathsf{M}_{a24}$
- 6 working registers (plus the register to store the base point x_D)
- Variable $a24 = (a+2)/4$ is stored in ROM

Tools and macros

- Cadence Encounter RTL Compiler v08.10
- UMC 130nm LL logic CMOS process (1 GE equals 5.12 μm^2)
- Target frequency set to 1 MHz
- Results are for post-synthesis not considering overhead of P&R

- Cadence Encounter Power System v08.10 used for power estimations *after* P&R

- We used a synchronous 2 304-bit RAM block implemented as either
  - standard-cell based RAM (∼18.3 kGEs) or
  - register-file RAM macro (∼3.7 kGEs).

Performance of crypto_box

w	Speed [Cycles]					Area [GEs]			
	DH-1	DH-2	FIRST	UPDATE	DECRYPT	Ctrl +ALU	ROM	Total incl. RAM std-cells	macro
2	3 455 394	3 455 428	8 117	9 291	9 085	10 555	307	29 319	14 648
4	1 957 282	1 957 316	7 705	8 465	8 049	10 761	308	29 526	14 855
8	1 151 906	1 151 940	7 685	8 427	7 513	11 484	311	30 252	15 581
12	971 682	971 716	7 557	8 171	7 385	11 794	313	30 564	15 893
16	811 170	811 184	7 443	7 943	7 271	13 869	311	32 637	17 966

- INIT takes 6 641 cycles and FINALIZE needs 62 cycles for all multiplier digit-sizes w.
- Controller (incl. program ROMs) requires 6.3-6.9 kGEs
- Power: 40-70 μW (half of power is spent for RAM)
- Critical path: 53.4-82.6 ns (adder structure in multiplier)

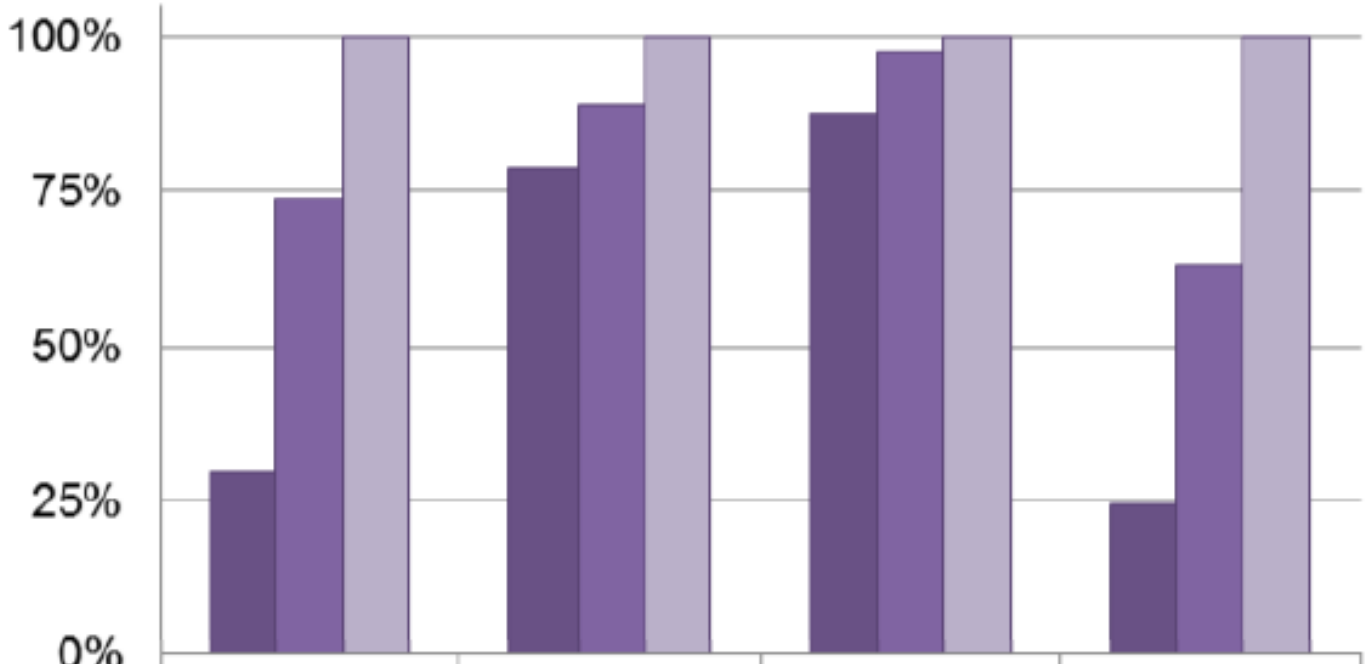
Comparison with Related Work

	Features of the (Co-)processor	Size [bits]	Time [Cycles]	Area [GEs] std-cells	Area [GEs] macro
Wolkerstorfer [7]	Weierstraß $\mathbb{F}_p/\mathbb{F}_{2^m}$	256	1 175 451	37 200	n.a.
Lai et al. [3]	Weierstraß $\mathbb{F}_p/\mathbb{F}_{2^m}$	256	252 067	197 028	n.a.
Satoh et al. [5]	Weierstraß $\mathbb{F}_p/\mathbb{F}_{2^m}$	256	880 000	55 647	n.a.
Liu et al. [4]	Twisted Edwards $\mathbb{F}_p = 2^{207} - 5131$	207	182 653	n.a.[a]	n.a.
Hutter et al. [2]	NIST P192, AES, SHA1	192	753 393	n.a.	21 502
Wenger [6]	NIST P256	256	3 367 000	n.a.	27 244
Ours (smallest)	Curve25519, Salsa20, Poly1305	255	3 455 394	29 319	14 648
Ours (fastest)			811 170	32 637	17 966

[a] Authors reported 5 821 GEs for the size of their ALU. Memory is not included.

Curve25519 vs. NIST P-256 vs. Brainpool P256r1

Curve25519 vs. NIST P-256 vs. Brainpool P256r1



	Speed	Area [GE]	Power [µW]	At
Curve25519	817167	5192	7,9	7,3
NIST	2024179	5883	8,8	19,1
Brainpool	2751664	6623	9,1	30,2

- Area and power values do not include RAM

More online...

- Hardware implementation:
  http://mhutter.org/research/vlsi/#naclhw or
  http://cryptojedi.org/crypto/#naclhw.
- NaCl website: http://nacl.cr.yp.to
- NaCl for microcontrollers (AVR, MSP430, ARM):
  http://munacl.cryptojedi.org

References I

H. Groß, E. Wenger, H. M. Gonzalez, and M. Hutter.

PIONEER—a Prototype for the Internet of Things based on an Extendable EPC Gen2 RFID Tag.

In N. Saxena and A.-R. Sadeghi, editors, *Workshop on RFID Security - RFIDsec 2014, 10th Workshop, Oxford, UK, July 21 -23.*, LNCS. Springer, Heidelberg, 2014.

M. Hutter, M. Feldhofer, and J. Wolkerstorfer.

A cryptographic processor for low-resource devices: Canning ECDSA and AES like sardines.

In C. A. Ardagna and J. Zhou, editors, *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, volume 6633 of *LNCS*, pages 144–159. Springer, 2011.

http://mhutter.org/papers/Hutter2011ACryptographicProcessor.pdf.

J.-Y. Lai and C.-T. Huang.

A highly efficient cipher processor for dual-field elliptic curve cryptography.

IEEE Transactions on Circuits and Systems II: Express Briefs, 56(5):394–398, 2009.

References II

Z. Liu, H. Wang, J. Großschädl, Z. Hu, and I. Verbauwhede.

VLSI implementation of double-base scalar multiplication on a twisted edwards curve with an efficiently computable endomorphism.

Cryptology ePrint Archive: Report 2015/421, 2015.

http://eprint.iacr.org/2015/421.pdf.

A. Satoh and K. Takano.

A scalable dual-field elliptic curve cryptographic processor.

IEEE Transactions on Computers, 52(4):449–460, 2003.

E. Wenger.

A lightweight ATmega-based application-specific instruction-set processor for elliptic curve cryptography.

In G. Avoine and O. Kara, editors, *LightSec 2013*, volume 8162 of *LNCS*, pages 1–15. Springer, 2013.

https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=70640.

J. Wolkerstorfer.

Is elliptic-curve cryptography suitable for small devices?

In E. Oswald, editor, *Workshop on RFID and Lightweight Crypto – RFIDsec 2005*, 2005.

The instruction set

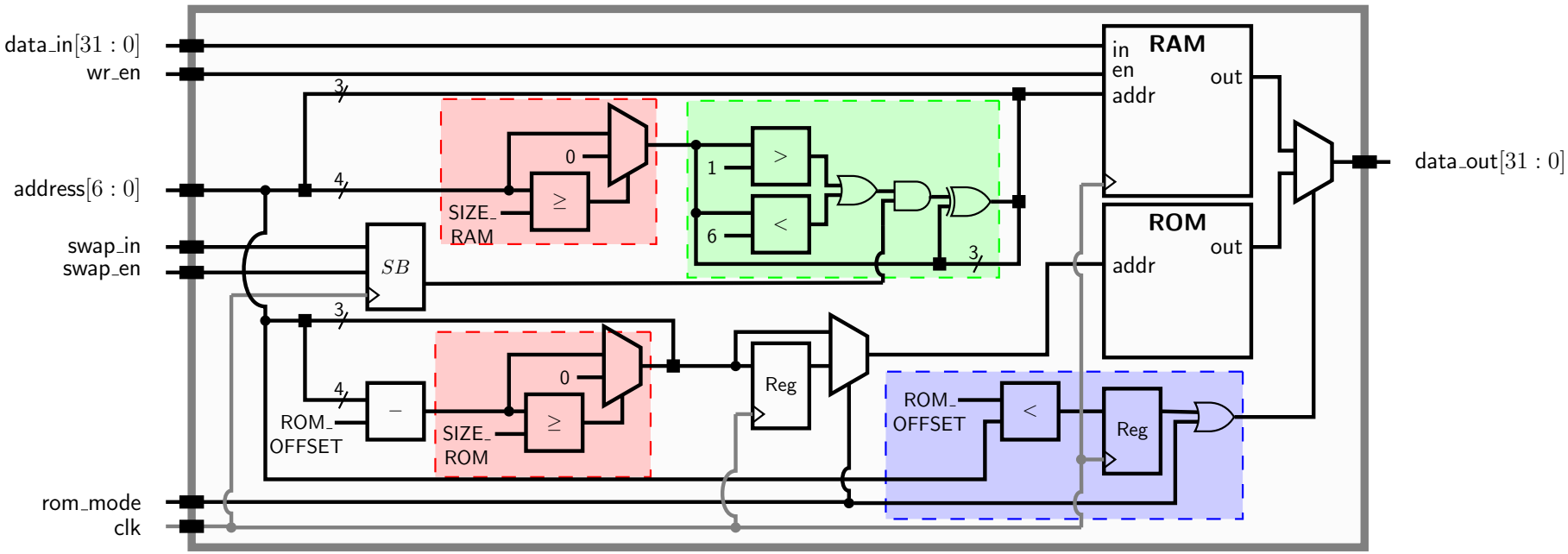
- Special-purpose instruction set with 46 instructions
- Opcode of all instructions has 9 bits only
  - 26 general purpose instructions
  - 20 special `crypto_box` instructions
  - 6 program-flow instructions

Mnemonic	Description
NOP	No operation.
FLC	Fetch word depending on loop counter LC
LD	Load from memory
CLR	Clear ACCU
CLRL	Clear ACCU$_L$
LDC	Load constant from ROM
ST	Store to memory
STR	Rotate and store to memory
MULADD	Multiply-add without carry
MULADC	Multiply-add with carry
MULSUB	Multiply-subtract without carry
MULSBC	Multiply-subtract with carry
MULACC	Multiply-accumulate
MUL	Accumulative multiplication
MUL256	256-bit multiplication
AND	Logical AND
OR	Logical OR
EOR	Logical XOR
STC	Store carry bit
STI	Store carry bit inverted
STX	Update carry bit
RORX	4 rotate right instructions
ROLX	7 rotate left instructions
JMP	Jump to address
RET	Subroutine return
SLCI	Skip if $PB_{LC} = s$, increment LC
SLCD	Skip if $PB_{LC} = s$, decrement LC
SFID	Skip if $FID = f$
INCLC	Increment LC
DECLC	Decrement LC
SW0	Set memory-paging state to 0
SW1	Set memory-paging state to 1
SWLC	Update memory-paging state
MPS	Select memory page
MPI	Increment memory-page index
MPD	Decrement memory-page index
HLT	Halt execution

Speed-Area Trade-Offs

- One can trade speed for lower area: implement 256-bit finite-field multiplication in microcode instead of a dedicated multiply control
- Classical product-scanning multiplication requires 209 instructions (for $w = 2$)
- Total area reduced to 13.2 kGEs
- Run-time for `DH-1` is increased by 10.3 % to 3.8 MCycles
- Authenticated encryption needs 12.3-19.7 % longer for a single 64-byte message block

The memory



- RAM and ROM are logically divided into 256-bit memory pages
  - RAM: 1×256 bits for x-coordinate of base point, 1×256 bits for the X25519 private key, and 7×256 bits for ECC scalar multiplication
  - ROM: 6×256 bits for constants: modular reduction in $\mathbb{F}_{2^{255}-19}$ and $\mathbb{F}_{2^{130}-5}$, 2 logic masks, curve parameter $a24$, and σ for XSalsa20