# An ECDSA Processor for RFID Authentication

**Michael Hutter, Martin Feldhofer, and Thomas Plos**

Workshop on RFID Security 2010

07. - 09.06.2010, Istanbul, Turkey

IAIK TU Graz

Institute for Applied Information Processing and Communications (IAIK)

Graz University of Technology

# Outline

- Motivation
- Implementation Requirements
- The ECDSA Processor
  - The System Architecture
  - Memory Unit and Datapath
  - Microcontroller
  - Instruction Set Extensions for ECDSA
- Synthesis Results
- Conclusion

# Motivation

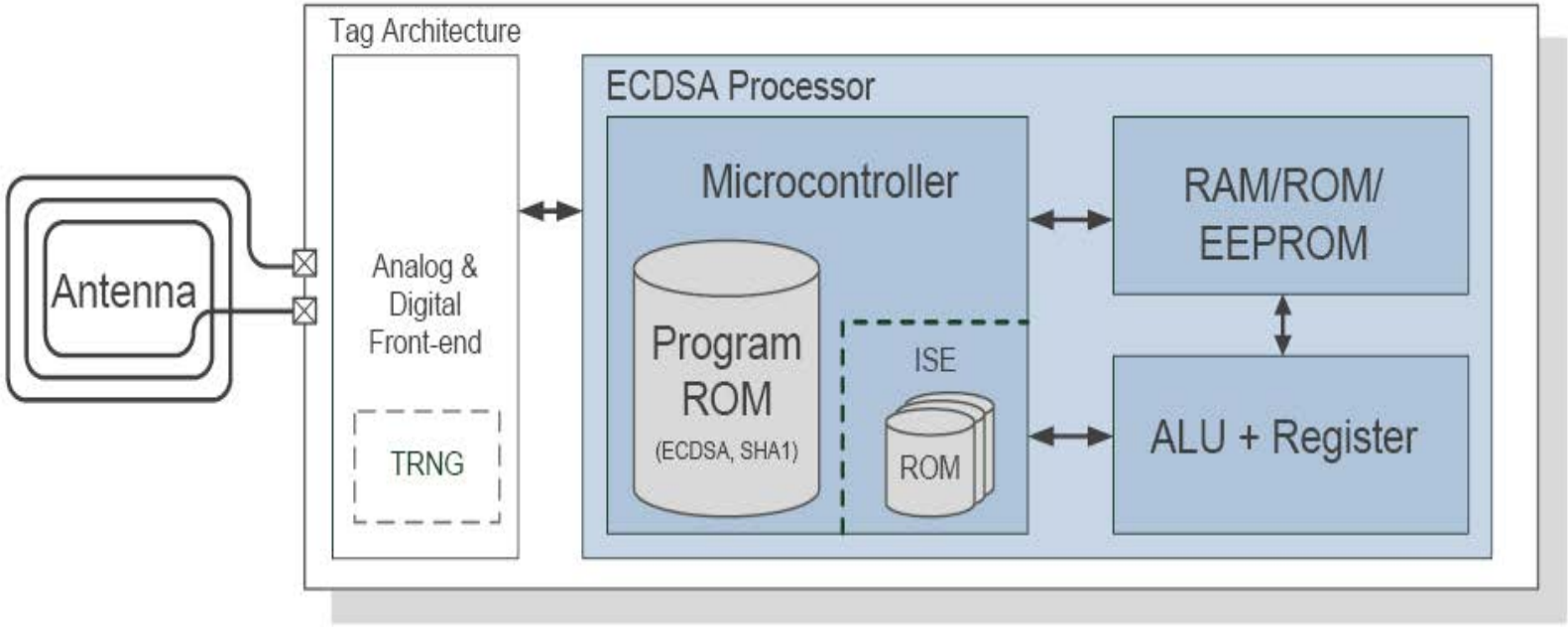
- RFID is one key enabler for the “Internet of Things”
  - Intelligent “smart things/tags” extend the Internet
- Tags are already integrated into many products
- There are still open issues in realizing a “secure Internet of things”

# Requirements

- Digital-signature service
  - To provide a transferable proof of origin
  - Message authentication, non-repudiation, data integrity
- Asymmetric cryptography
  - Large scale deployment
  - Integration in open-loop systems (Internet)
- Standardized algorithms
  - ECDSA has been tested/proved over many years
  - Existing PKI (X.509 certificates using ECDSA)
- Strong authentication
  - Challenge-response protocol (e.g. ISO/IEC 9798-3)
- Low-resource HW design

# What we did?

- Design of an ECDSA processor for RFID
  - Based on NIST recommended elliptic curve GF(p192)

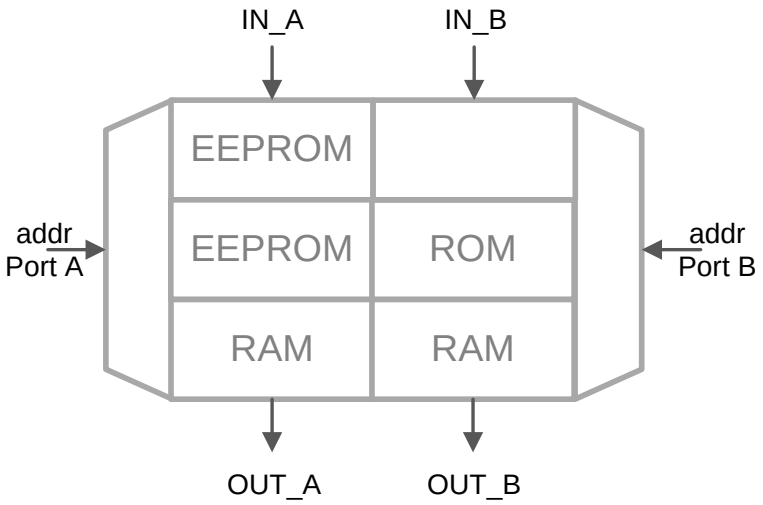


# Tag Authentication using ECDSA

| **Reader** | | **Tag** |
|---|---|---|
| $c_1 \in_R \mathbb{Z}_{2^t}$ | | $k \in_R \mathbb{Z}_n$, $c_2 \in_R \mathbb{Z}_{2^t}$ |
| | $\xleftarrow{cert_{tag}}$ | |
| if verifyCert$(cert_{tag})$ failed, reject | $\xrightarrow{c_1}$ | |
| | | $(x, y) \leftarrow [k]P$, convert $x$ to an integer $\bar{x}$<br>$r \leftarrow \bar{x} \pmod n$. check if $r \neq 0$<br>$e \leftarrow SHA1(c_1, c_2)$<br>$s \leftarrow k^{-1}(e + dr) \pmod n$. check if $s \neq 0$ |
| | $\xleftarrow{(r,s),c_2}$ | |
| if verifySig$((r, s), c_2)$ failed reject else accept | | |

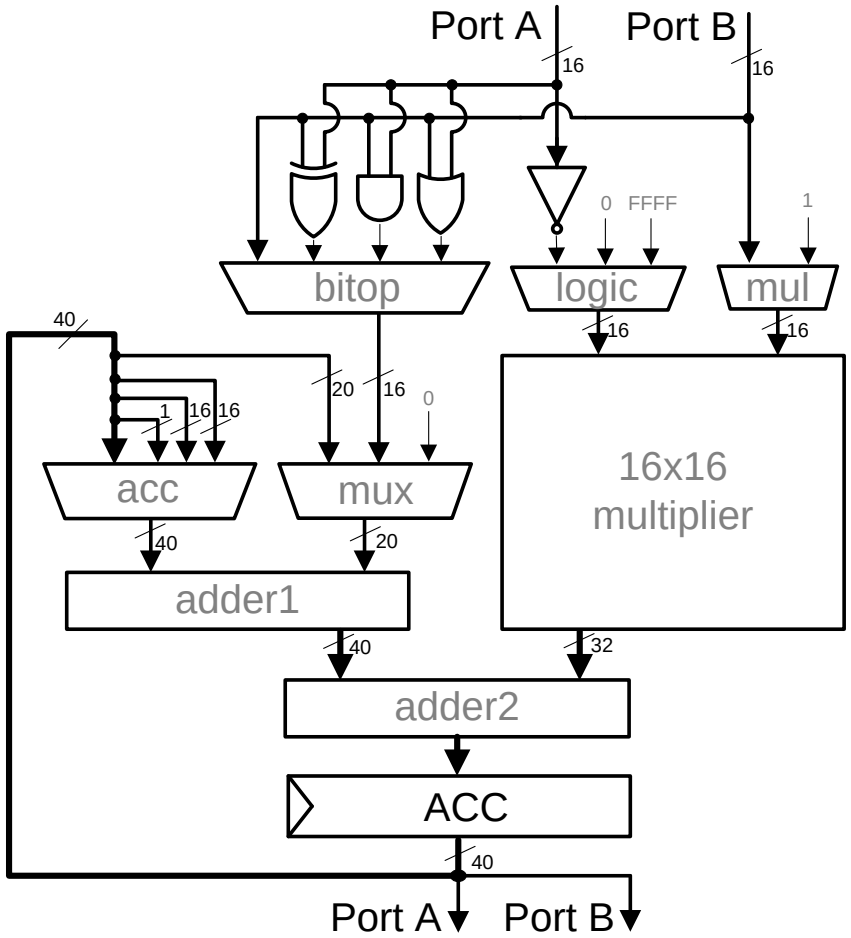
# Memory Unit

- 16-bit dual ported interface
  - Concurrently read/write from/to two ports
- RAM macro (128x16 bit)
- ROM
  - ECC constants (e.g. base point P)
- EEPROM
  - Stores the private key
  - Stores the certificate

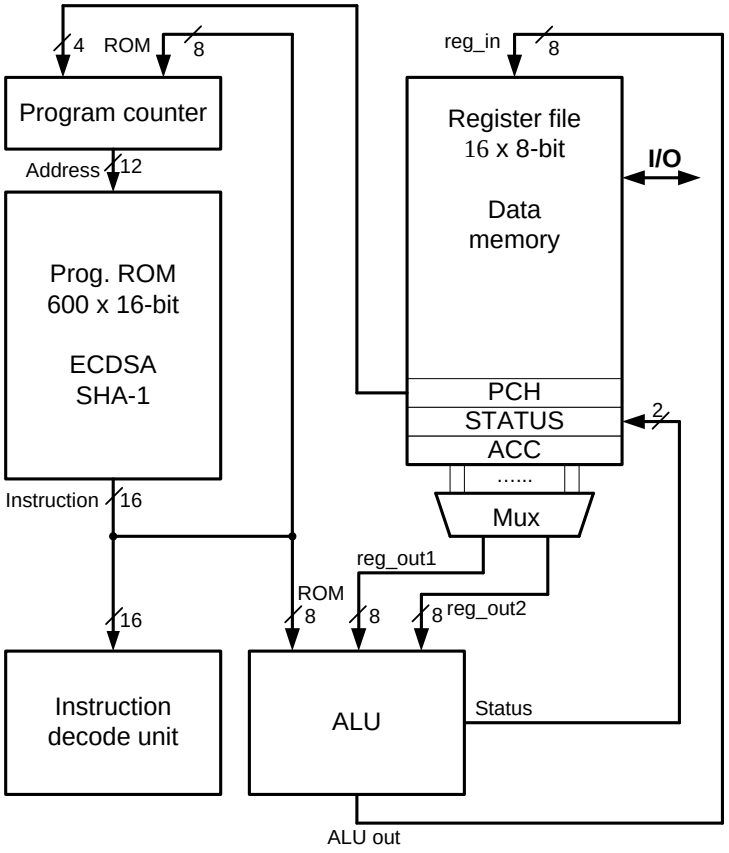


# 16-bit Datapath

- 16x16-bit multiply accumulate (MAC) unit
- 1 cycle 16-bit operations
- Two 40-bit adders
- One 40-bit accumulator
  - Feedback of ACCU signal
- Logic operations for SHA1
  - XOR, AND, OR
- Writing into memory using two 16-bit values concurrently



# 8-bit Microcontroller




- 32 instructions supported
  - Arithmetic operations (ADD, SUB,…)
  - Logical operations (OR, AND,…)
  - Control operations (GOTO, CALL,…)
- Register file and program ROM
- Instruction decoder, ALU, Counter,…
- Two-stage pipeline (fetch and execute)
- Call-stack support (3 recursive subroutines possible)
- Self-written Java compiler

# Instruction Set Extensions

- 55 ISEs for ECDSA and SHA1
  - Can be executed by the microcontroller by a MICRO instruction
- Implemented in 8 ROM tables
  - Area reduction through different table sizes
- Modular arithmetic
  - Addition: 32 cycles
  - Subtraction: 38 cycles
  - Multiplication: 204 cycles
  - NIST reduction applied ($p_{192} \equiv 2^{192} - 2^{64} - 1$)
- Montgomery arithmetic
  - Inversion: 20823 cycles
  - Multiplication: 785 cycles
- SHA1: 3455 cycles

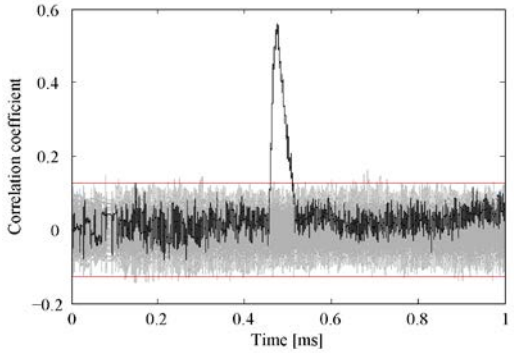
```
...
MovLF(ADDR1_REG, 0x4);
MICRO(INST_ADD, addr_par9, 19);
CALLR("NIST_RED");
...

LABEL("NIST_RED");
  MICRO(INST_RED1, addr_null, 4);
  MICRO(INST_RED2, addr_null, 8);
  BWS(STATUS, CU_NEXT_INSTR);
  BTC(STATUS, CU_CARRY);
  MICRO(INST_SUB, addr_par14, 19);
RET();
```

# Improving ECC Point Multiplication

- Montgomery Ladder
- Use of x-coordinate only formulas (Brier and Joye)
- Combined double-and-add (Izu, Möller, and Takagi)
- Common-Z coordinate representation (Meloni, Lee)
- Total: 12M + 4S + 9add + 7sub
- 7x192-bit RAM used

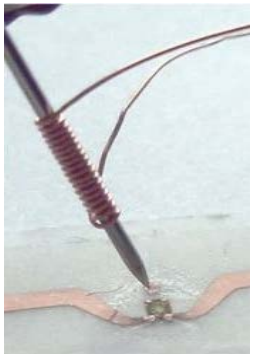
# Implementation Attack Countermeasures



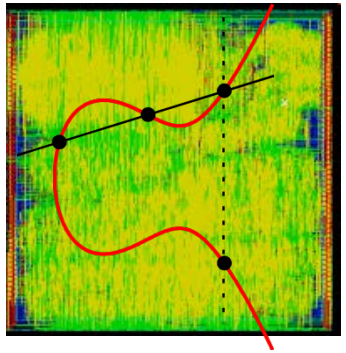

- SPA
  - Montgomery Ladder
- DPA
  - Randomized Projective Coordinates (S. Coron)
  - First-order blinding of the private-key multiplication

    $s = k^{-1}e + (k^{-1}r)d$ instead of $s = k^{-1}(e + dr)$
- Fault Injections
  - Check of curve equation after point multiplication (Ebeid and Lambert)

    $$Z(Y^2 - bZ^2) = X(X^2 + aZ^2)$$
  - Y recovery necessary



# Synthesis Results

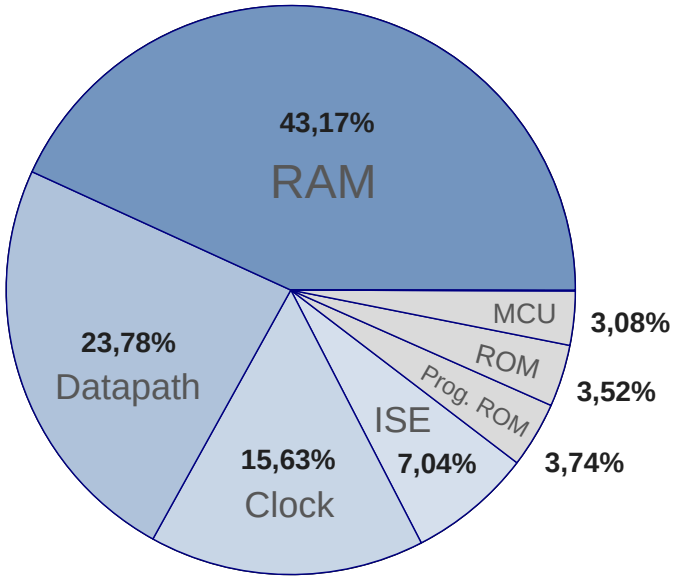


- Cadence RTL Compiler (0.35 µm CMOS)
- Synopsys NanoSim for power simulation
  - 387 µA mean current at 3.3 volt and 847 kHz

## Chip Area

| Component | GE |
|---|---|
| Microcontroller without program ROM | 1 786 |
| Program ROM (ECDSA, SHA1, RNG) | 2 132 |
| ISE control logic (ROM, decoder,...) | 3 310 |
| 2048-bit RAM macro | 8 727 |
| ROM for ECC constants | 789 |
| Datapath (ALU+register) | 2 371 |
| **Total Size** | **19 115** |

## Power Consumption



# Comparison with Related Work

| | Area [GE] | Time [Cycles] | Field | Features |
|---|---|---|---|---|
| **This Work** | **19 115** | **859 188** | $\mathbb{F}_{p192}$ | **ECDSA, SHA1, RNG** |
| Fürbass07 [8] | 23 656 | 502 000 | $\mathbb{F}_{p192}$ | ECDSA(no SHA1,no RNG) |
| Wolkerstorfer05 [27] | 23 800 | 677 000 | $\mathbb{F}_{p192}$ | ECC |
| Öztürk04 [24] | 30 333 | 545 440 | $\mathbb{F}_{(2^{167}+1)/3}$ | ECC |
| Satoh03 [25] | 29 655 | 4 165 000 | $\mathbb{F}_{p192}$ | ECC |
| Hein08 [11] | 11 904 | 296 000 | $\mathbb{F}_{2^{163}}$ | ECC |
| Bock08 [3] | 12 876 | 80 000 | $\mathbb{F}_{2^{163}}$ | ECC, DH, RNG |
| Lee08 [19] | 12 506 | 302 457 | $\mathbb{F}_{2^{163}}$ | ECC, Schnorr |
| Kumar06 [18] | 19 048 | 527 284 | $\mathbb{F}_{2^{193}}$ | ECC |
| Batina06 [2] | 8 104 | 353 000 | $\mathbb{F}_{2^{131}}$ | ECC, without memory |
| Schroeppel02 [26] | 191 000 | 93 000 | $\mathbb{F}_{2^{178}}$ | ECC, ElGamal, PRNG |

# Conclusions

- Improved the state-of-the-art in designing a low-resource ECC hardware processor
  - First ECDSA hardware implementation results
- Fully capable digital signature generating device
  - Allows proof of origin to prevent product counterfeiting
- Sample implementation
  - Processor will be integrated in an NFC-compliant HF tag
  - Fabricated in summer 2010

# Thanks for your attention!

http://www.iaik.tugraz.at/content/research/implementation_attacks/


**Michael Hutter**

IAIK – Graz University of Technology
michael.hutter@iaik.tugraz.at
www.iaik.tugraz.at

# Montgomery Ladder

**Algorithm** Point-multiplication method using the improved Montgomery ladder in common Z projective coordinate system.

**Require:** Base point $P = (P_x, P_y) \in E(\mathbb{F}_{p192}), k \in_R [1, 2^{192} - 1]$, random $\lambda$
**Ensure:** $Q = kP$, where $Q = (x, y) \in E(\mathbb{F}_{p192})$
1: $X_0 \leftarrow \lambda P_x$, $Z_0 \leftarrow \lambda$
2: $(X_1, Z_1) \leftarrow Dbl(P)$
3: $X_0 \leftarrow X_0 \cdot Z_1$, $X_1 \leftarrow X_1 \cdot Z_0$, $Z \leftarrow Z_0 \cdot Z_1$
4: **for** $i = 190$ downto 0 **do**
5: $\quad (X_{k_i \otimes 1}, X_{k_i}, Z) \leftarrow CombinedDoubleAndAdd(X_{k_i}, X_{k_i \otimes 1}, Z)$
6: **end for**
7: $Y_0 \leftarrow Yrecovery(X_0, X_1, Z, P)$
8: **if** $Z'(Y_0'^2 - bZ'^2) \neq X_0'(X_0'^2 + aZ'^2)$ **abort.**
9: $x \leftarrow X_0 \cdot Z^{-1}$
10: Return $(x)$.