RUB

Side-Channel Attacks on the Bitstream Encryption Mechanism of Altera Stratix II

Amir Moradi, **David Oswald**,
Christof Paar, **Pawel Swierczynski**

Chair for Embedded Security

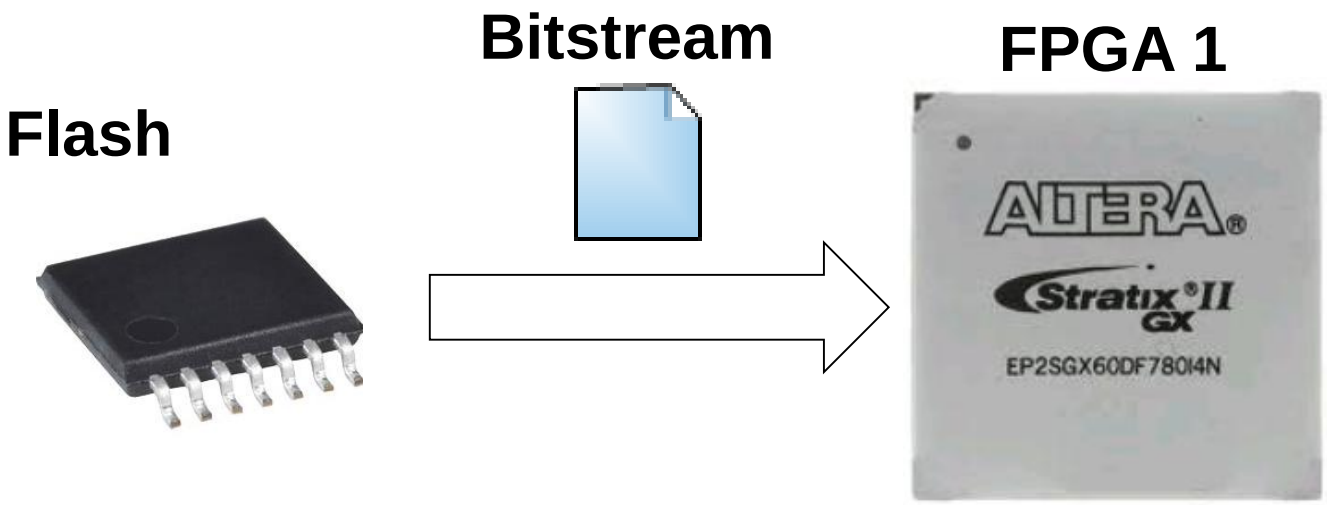
Horst Görtz Institute for IT-Security
Ruhr-University Bochum

FPGAs widely used in

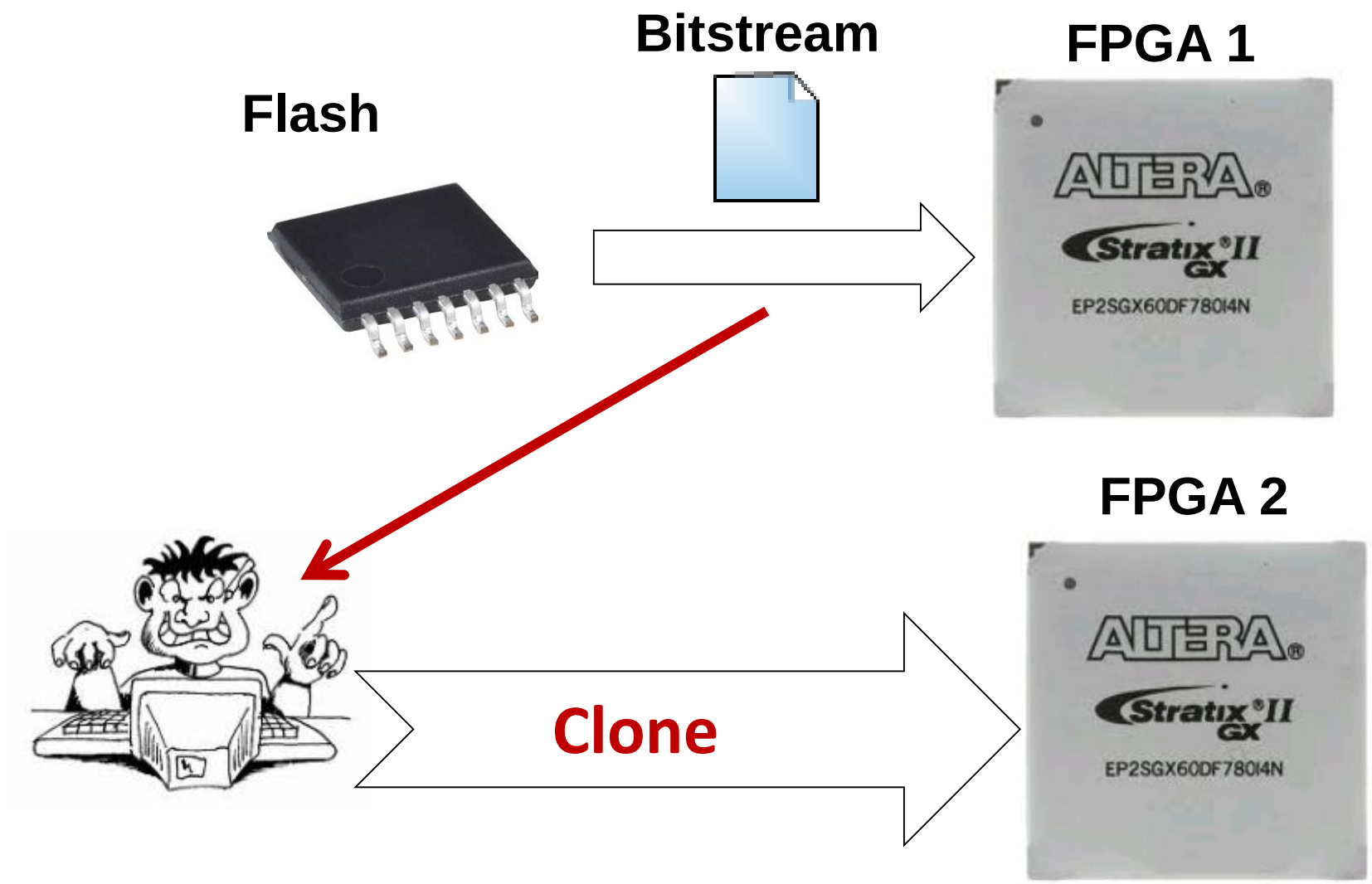
- Routers
- Consumer products
- Cars
- Military

Problem:
FPGA design (bitstream) can be easily copied

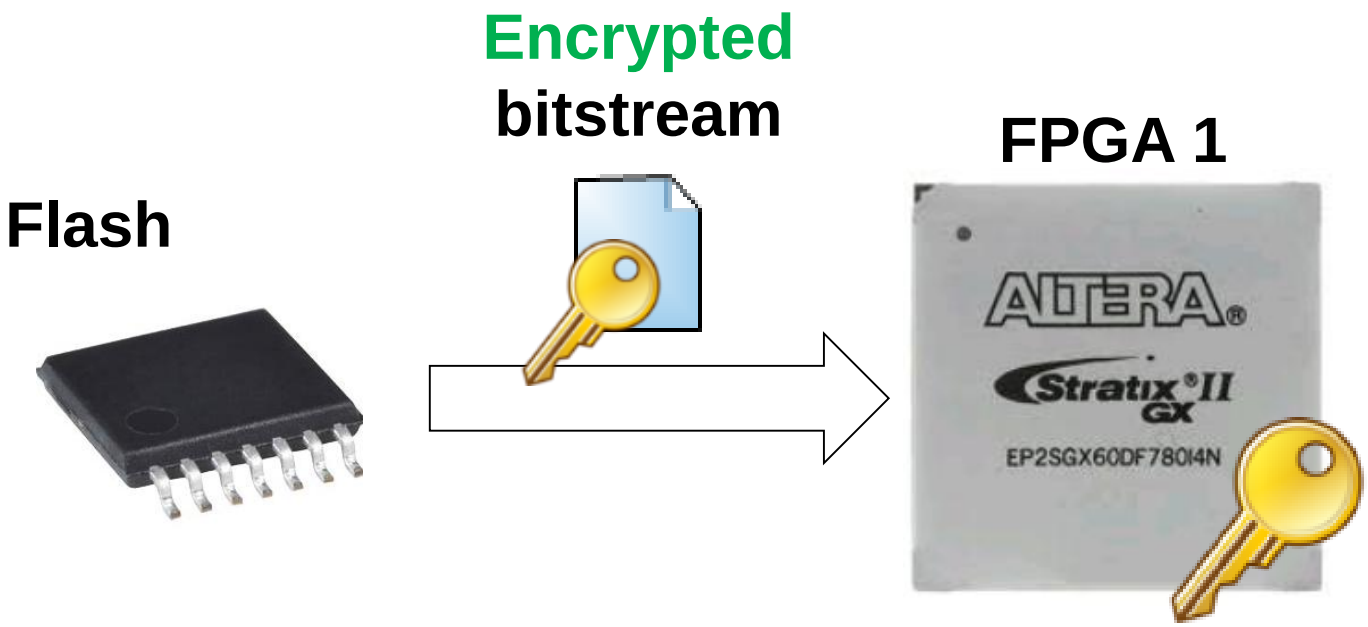
On each power-up



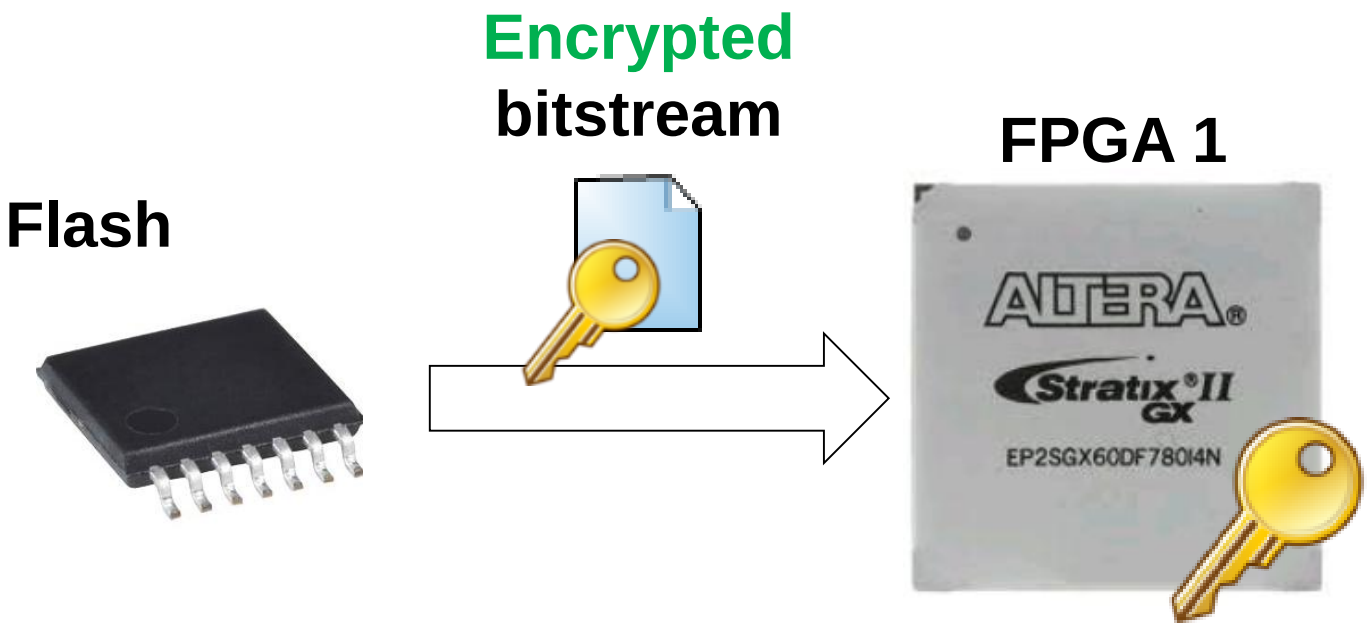
Problem: Cloning

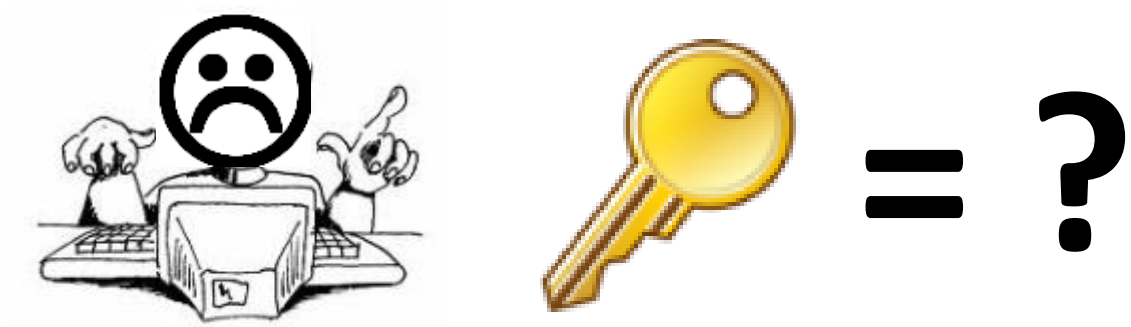


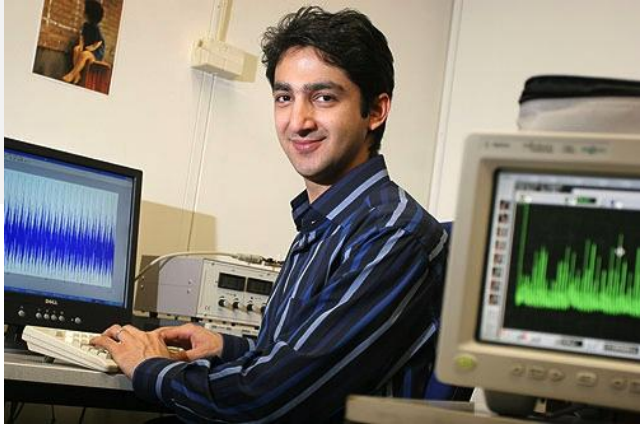
Industry‘s solution



Industry‘s solution







Previous Work

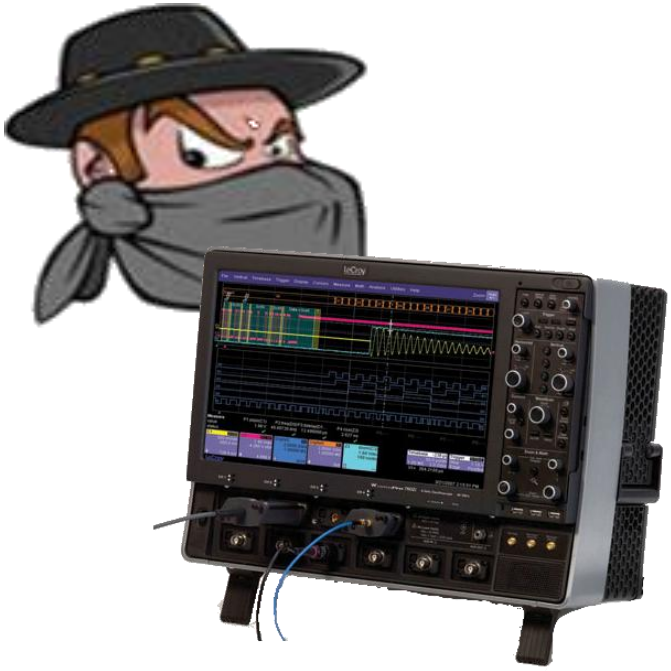
- Bitstream encryption scheme of several Xilinx product lines broken
  - Virtex 2 (3DES)
  - Virtex 4 & 5 (AES256)
  - Spartan 6 (AES256)
- **Method: Side-Channel Analysis (SCA)**

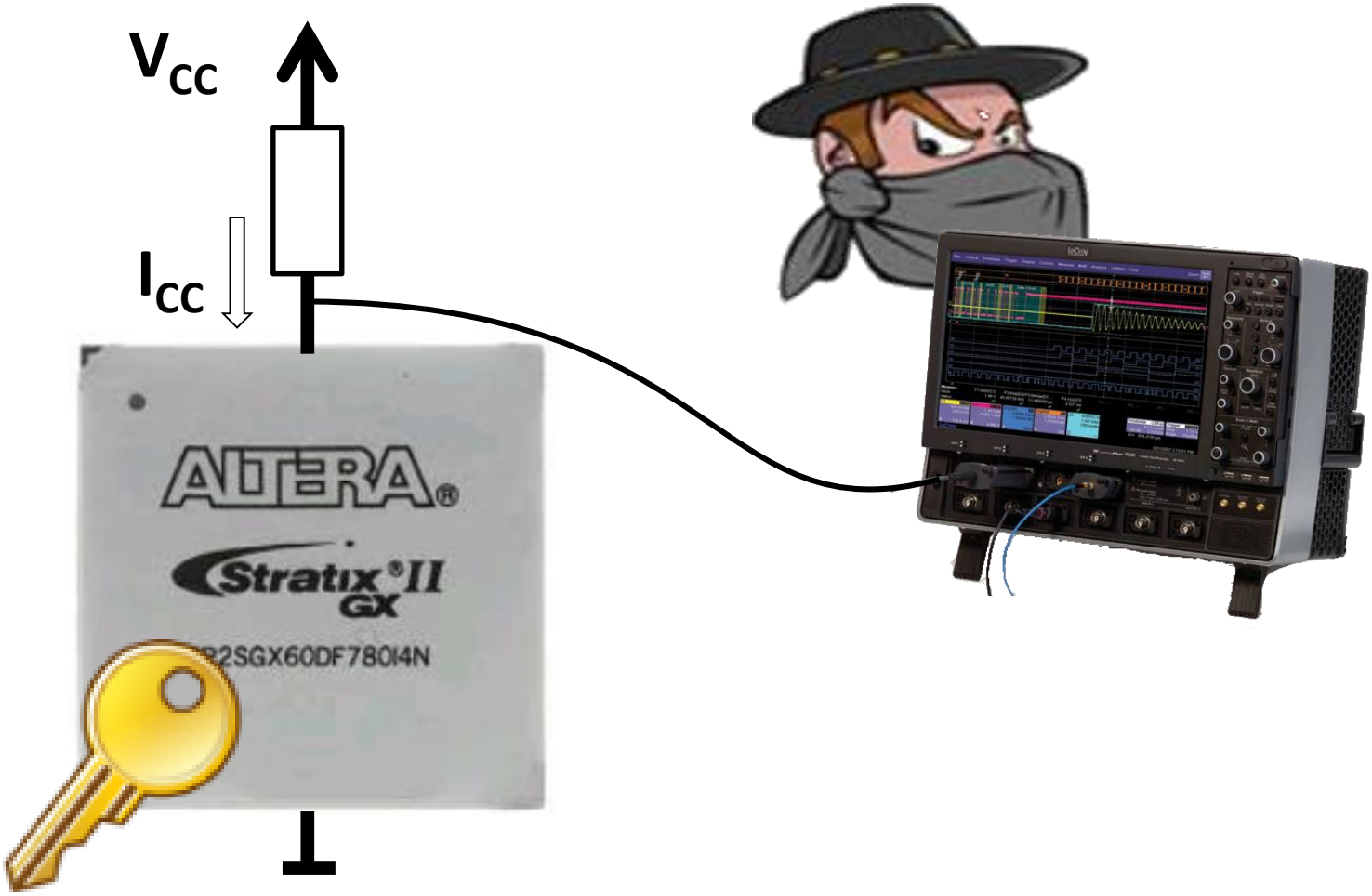
Side-Channel Analysis?


DIEBOLD SAFE & LOCK C.
CANTON.O.


ALTERA
Stratix® II
GX
2SGX60DF780I4N


V_{CC}
ALTERA
Stratix II GX
2SGX60DF780I4N




V_{CC}
I_{CC}
ALTERA
Stratix II GX
2SGX60DF780I4N

What about Altera?

- **Target:** Stratix II
- Bitstream encryption („design security“) uses AES w/ 128-bit key
- Side-Channel Analysis possible?
- **Problem:** Proprietary and undocumented mechanisms for key derivation and for encryption

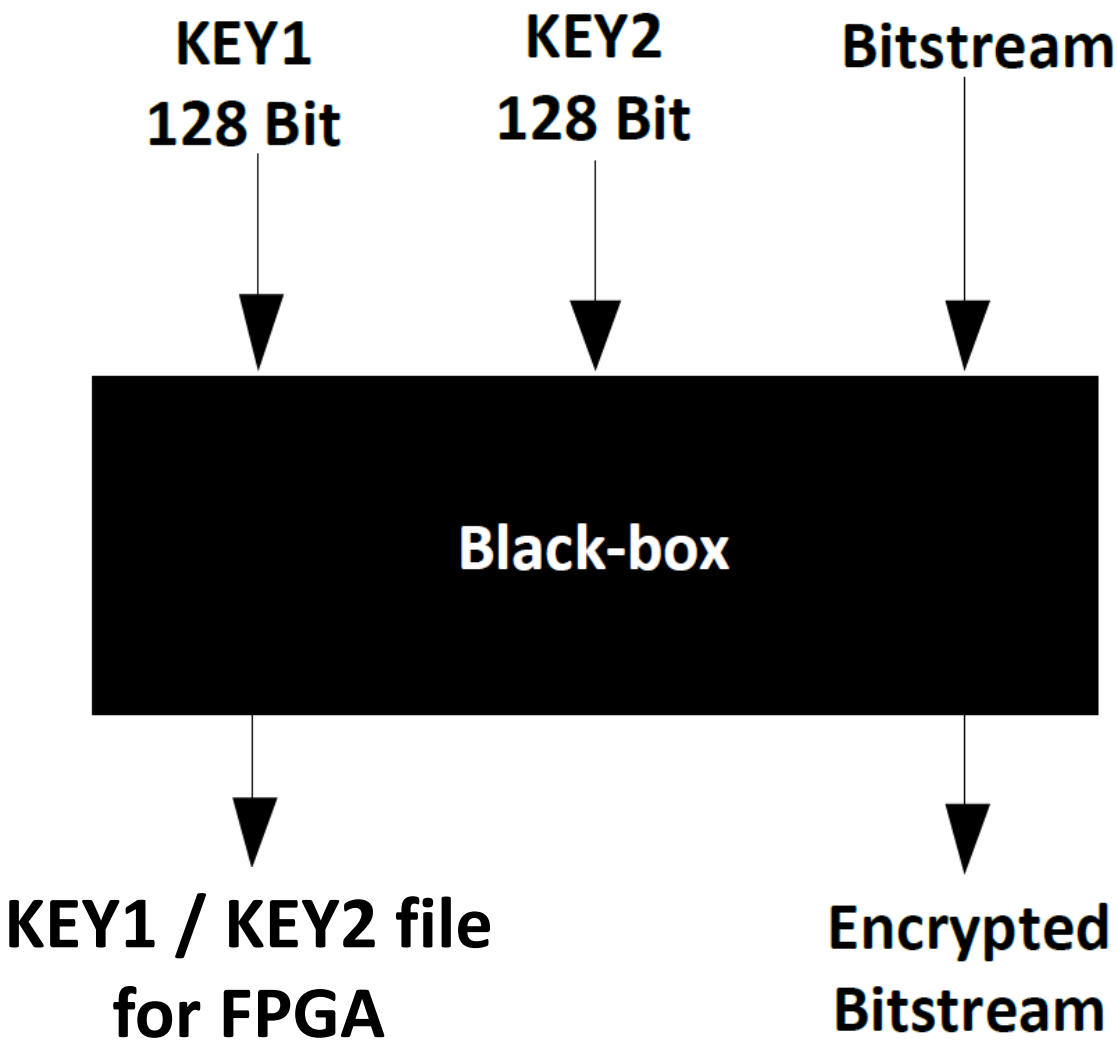
Let‘s have a look at the Quartus II Software ...

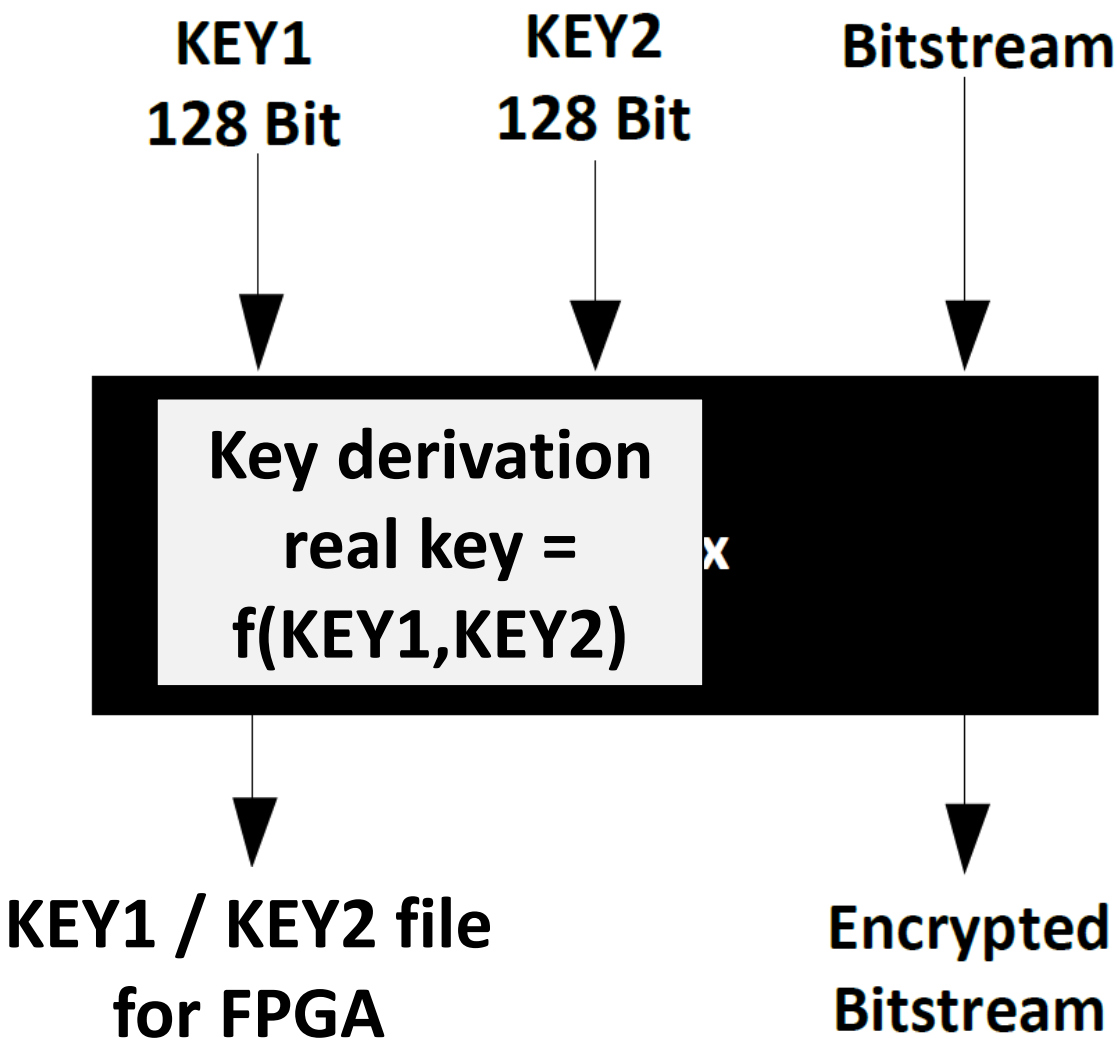
Our approach

- **Reverse-engineer** proprietary mechanisms from Quartus II software
- **IDA Pro** (disassembler / debugger)

```
pgm_pgmio_nv_aes.dll:0A431170 pgm_pgmio_nv_aes_?key_init@PGM_AES@@AAEHQAIQBEH@Z:
pgm_pgmio_nv_aes.dll:0A431170 mov     eax, [esp+8]
pgm_pgmio_nv_aes.dll:0A431174 movzx   edx, byte ptr [eax]
pgm_pgmio_nv_aes.dll:0A431177 movzx   ecx, byte ptr [eb
pgm_pgmio_nv_aes.dll:0A43117B shl     edx, 8
pgm_pgmio_nv_aes.dll:0A43117E xor     edx, ecx
pgm_pgmio_nv_aes.dll:0A431180 movzx   ecx, byte ptr [ea
pgm_pgmio_nv_aes.dll:0A431184 shl     edx, 8
pgm_pgmio_nv_aes.dll:0A431187 xor     edx, ecx
pgm_pgmio_nv_aes.dll:0A431189 movzx   ecx, byte ptr [ea
pgm_pgmio_nv_aes.dll:0A43118D shl     edx, 8
```

Rename N
Jump to operand Enter
Jump in a new window Alt+Enter
Jump in a new hex window
Use standard symbolic constant


KEY1
128 Bit
KEY2
128 Bit
Bitstream
Black-box
KEY1 / KEY2 file
for FPGA
Encrypted
Bitstream


KEY1
128 Bit
KEY2
128 Bit
Bitstream
Key derivation
real key =
f(KEY1,KEY2)
x
KEY1 / KEY2 file
for FPGA
Encrypted
Bitstream

Why this key derivation?

- Real key cannot be set directly
- Key derivation is performed **once** when programming the FPGA
- **Idea:** When real key is extracted, KEY1 and KEY2 cannot be found

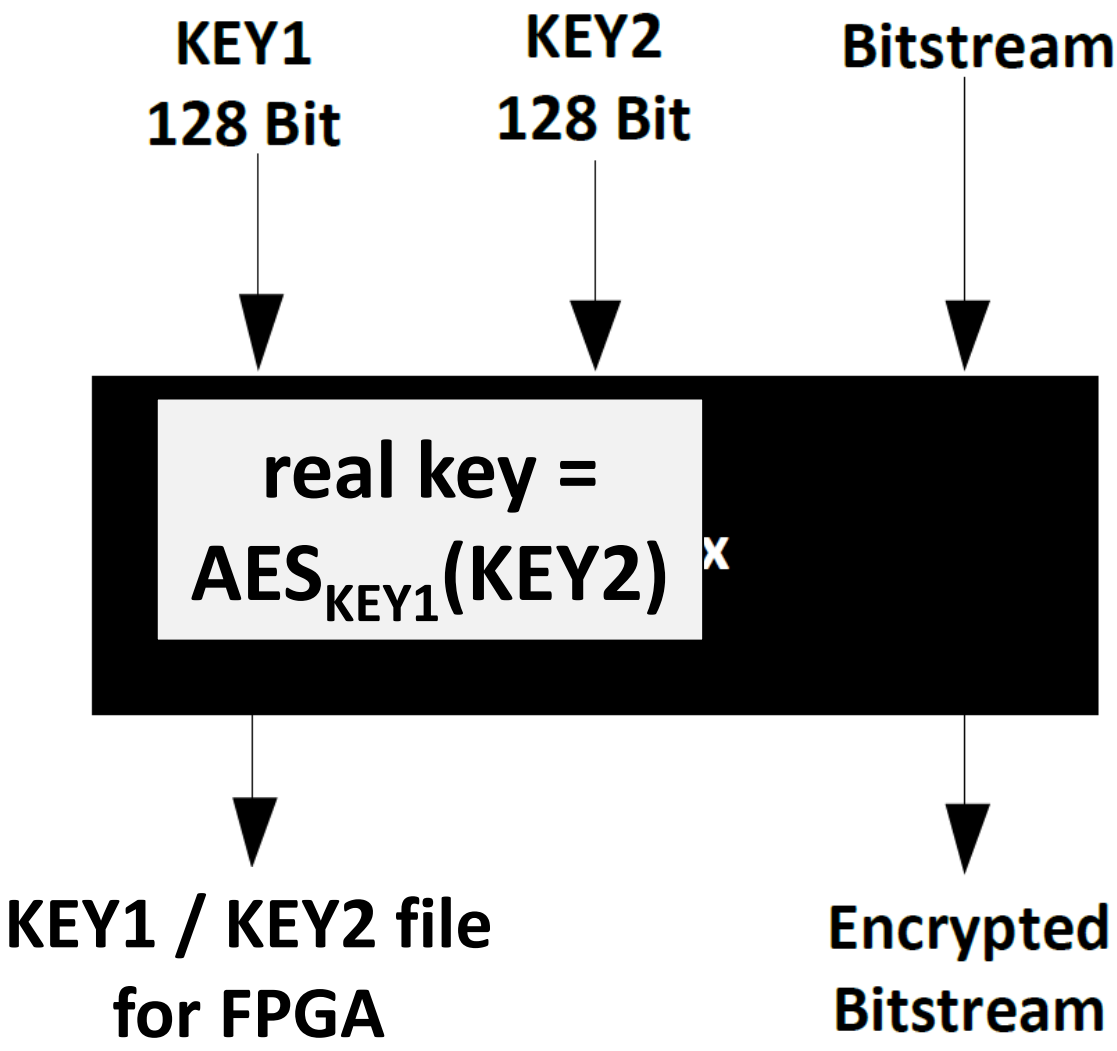
  → **Prevent cloning**: real key of blank FPGA cannot be set

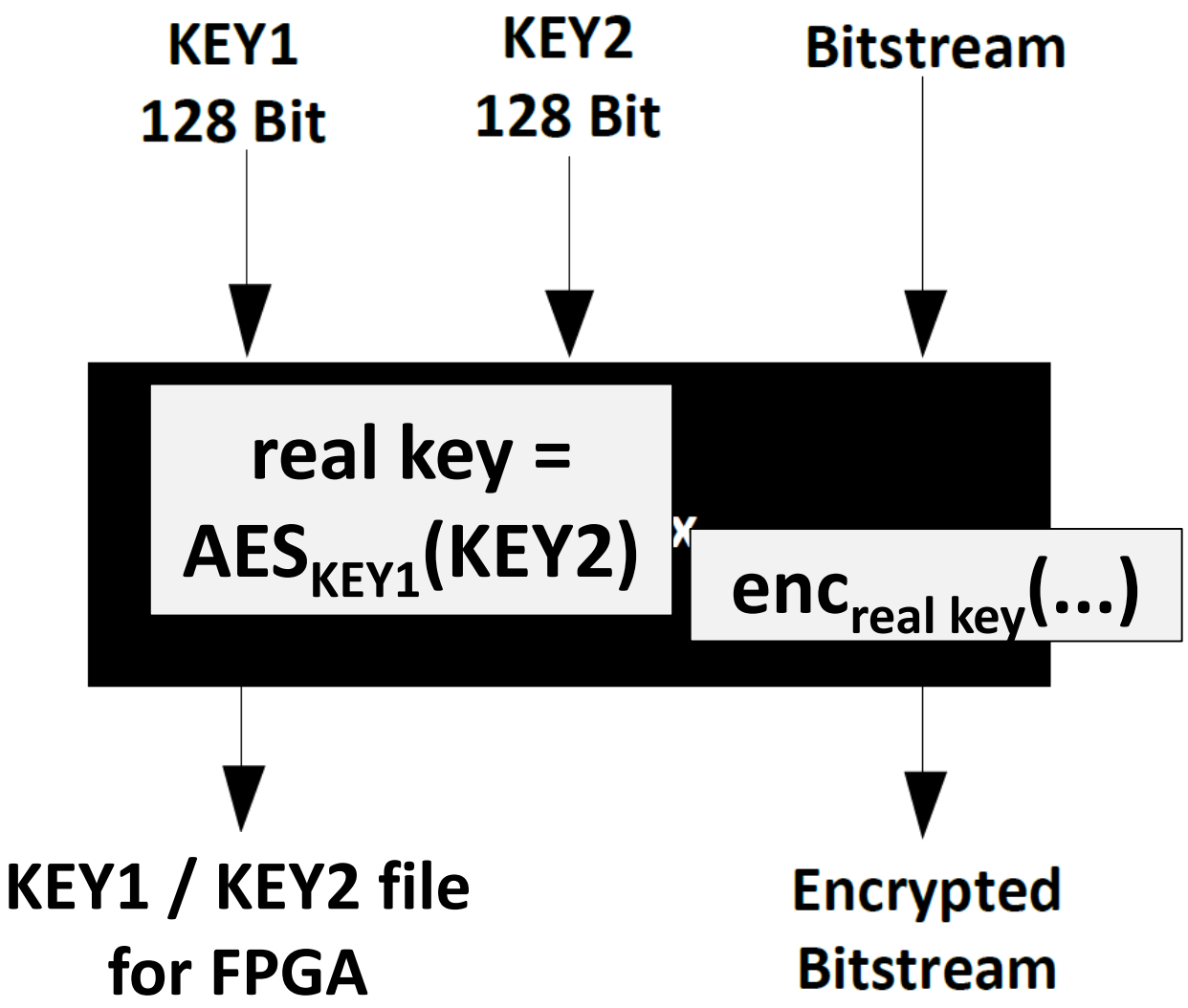
„real key“ = AES_{KEY1}(KEY2)

Is f (KEY1,KEY2) „good“?

Good idea?

- In principle: **Yes**
- **But:** AES (in this form) is **not** one-way:
- Pick any KEY1*
- KEY2* = AES^{-1}_{KEY1*}(real key)
- This (KEY1*, KEY2*) leads to same real key


KEY1
128 Bit
KEY2
128 Bit
Bitstream
real key =
AES$_{KEY1}$(KEY2)
x
KEY1 / KEY2 file
for FPGA
Encrypted
Bitstream


KEY1
128 Bit
KEY2
128 Bit
Bitstream
real key =
AES$_{KEY1}$(KEY2)
enc$_{real\ key}$(...)
KEY1 / KEY2 file
for FPGA
Encrypted
Bitstream

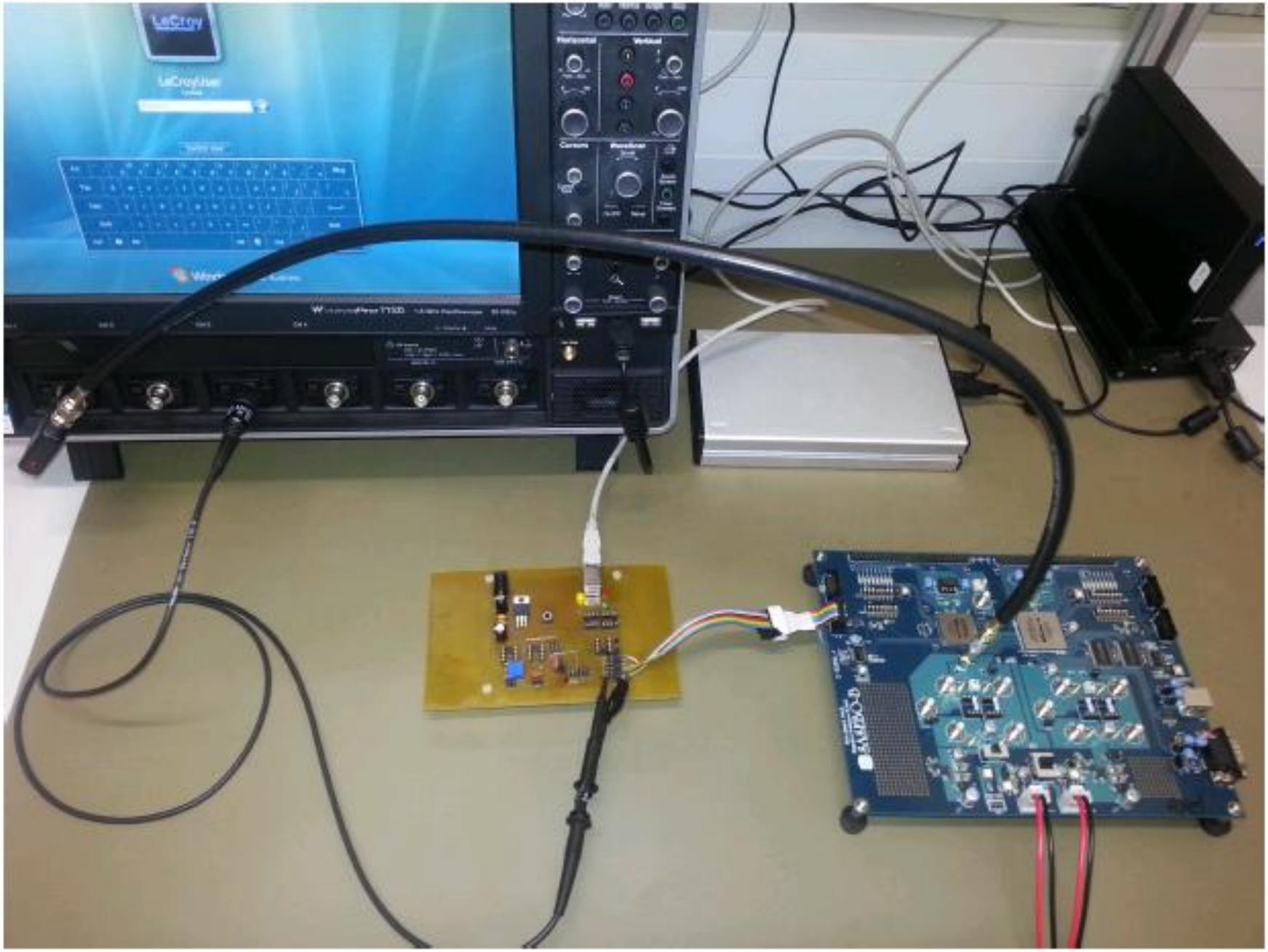
Encrypted block i =
$AES128_{real\ key}(IV_i) \oplus$ plain block i

Encryption method:
AES in Counter mode

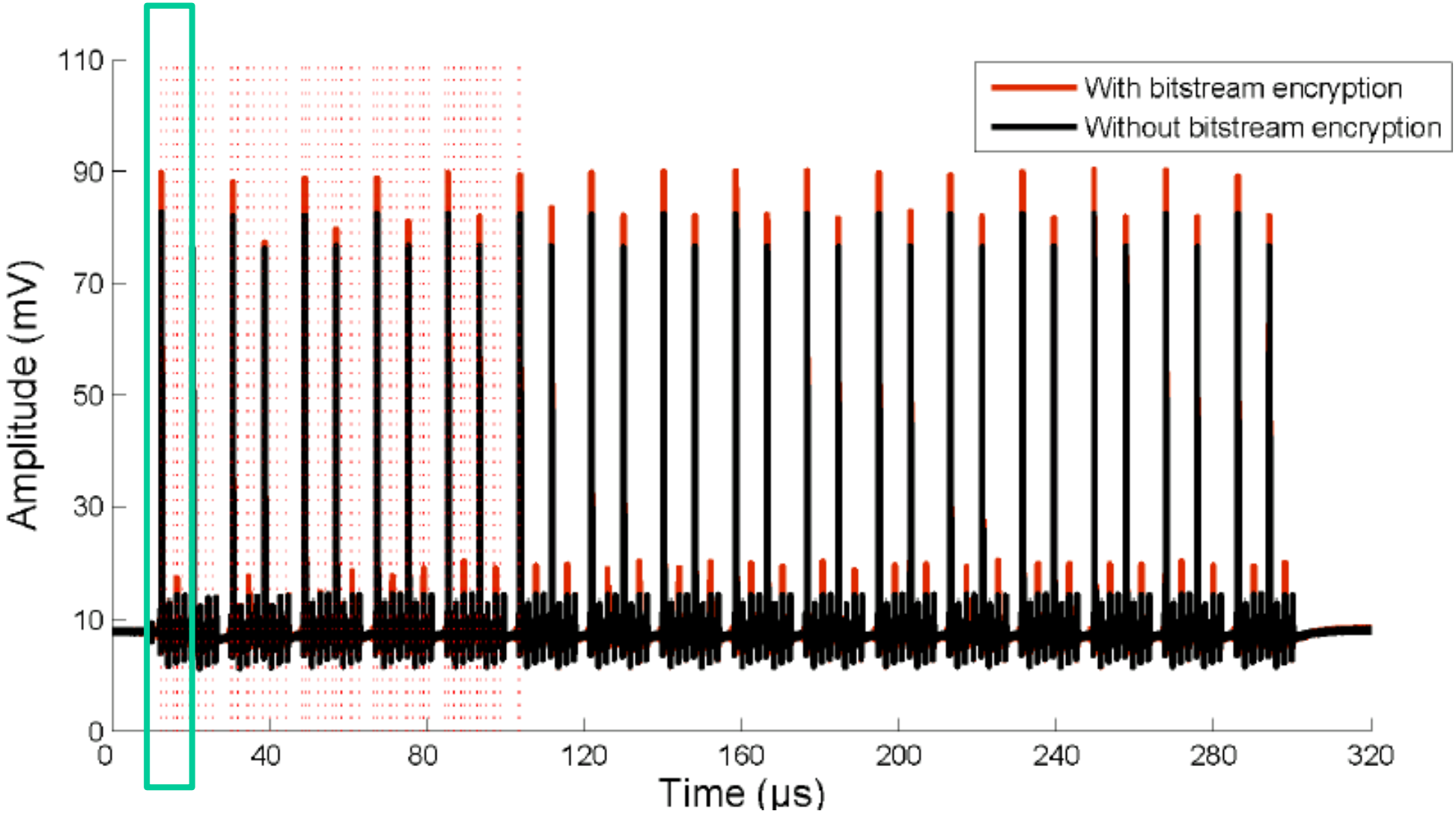
Reverse-Engineering: Summary

- All „obscurity features“ reverse-engineered
- Further details: file format, coding, ...
- **Black-box → white box**
- **Side-channel analysis possible**
  (target: 128-bit real key)

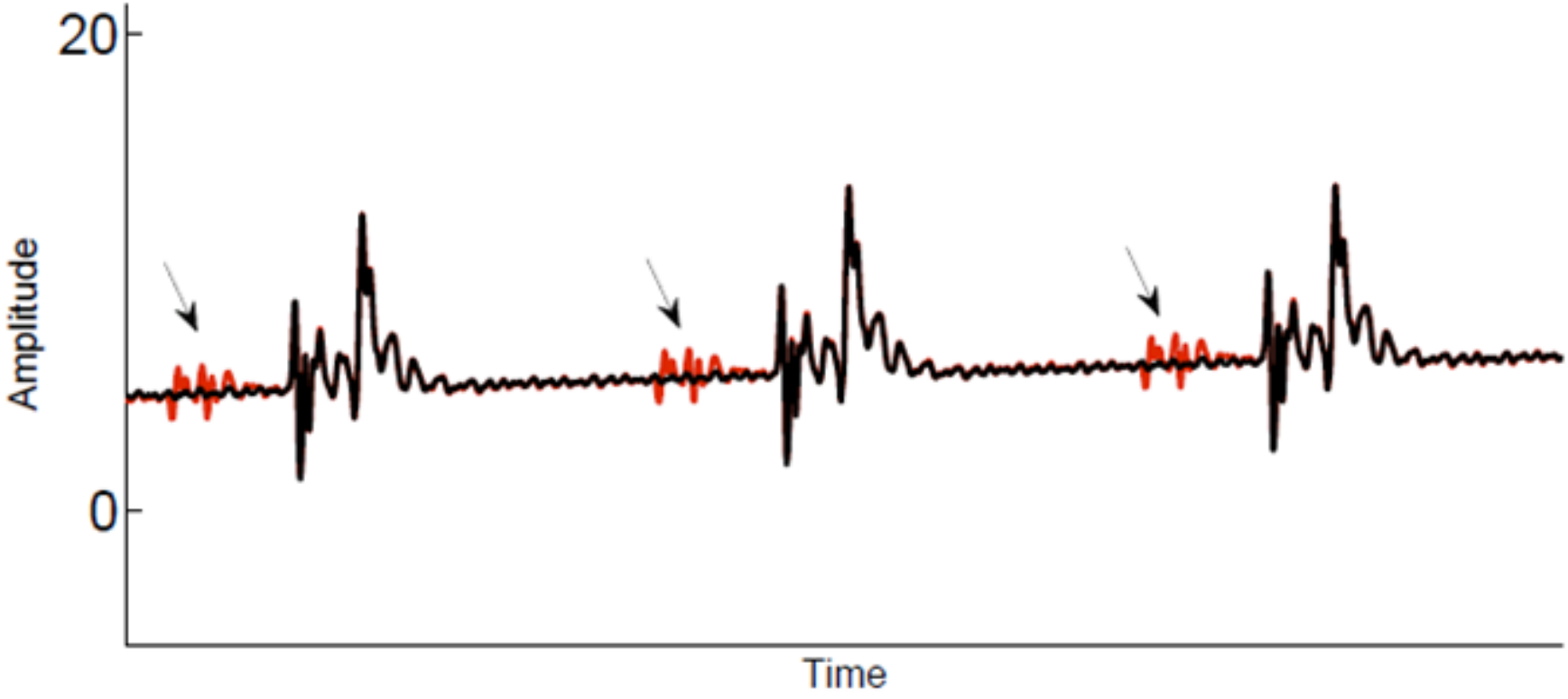
Side-Channel Analysis of Stratix II


LeCroy

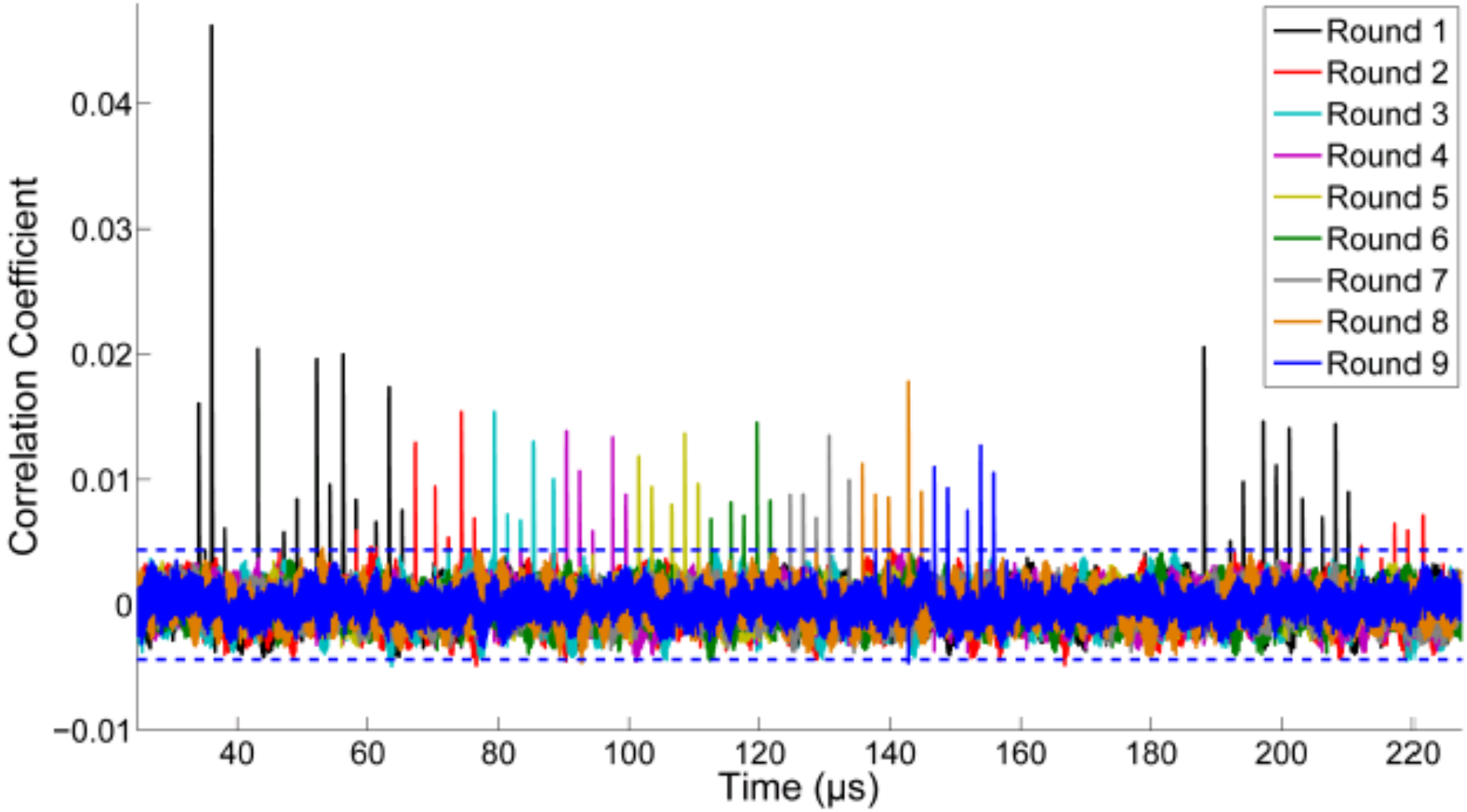
Mean trace for unencrypted and encrypted bitstream



Mean trace for unencrypted and encrypted bitstream

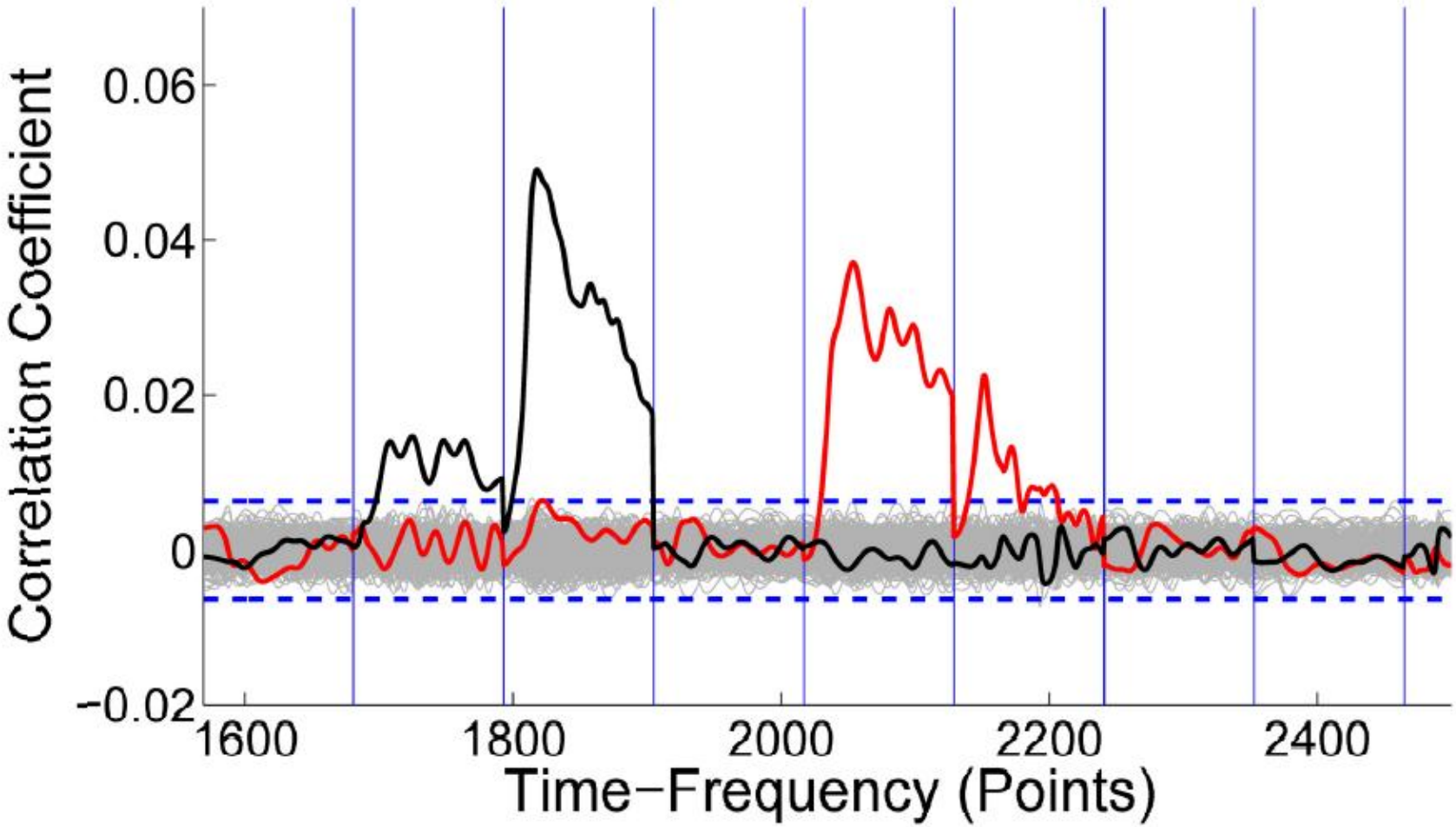


Side-channel leakage found: HW of state after AddRoundKey for AES round 1 – round 9

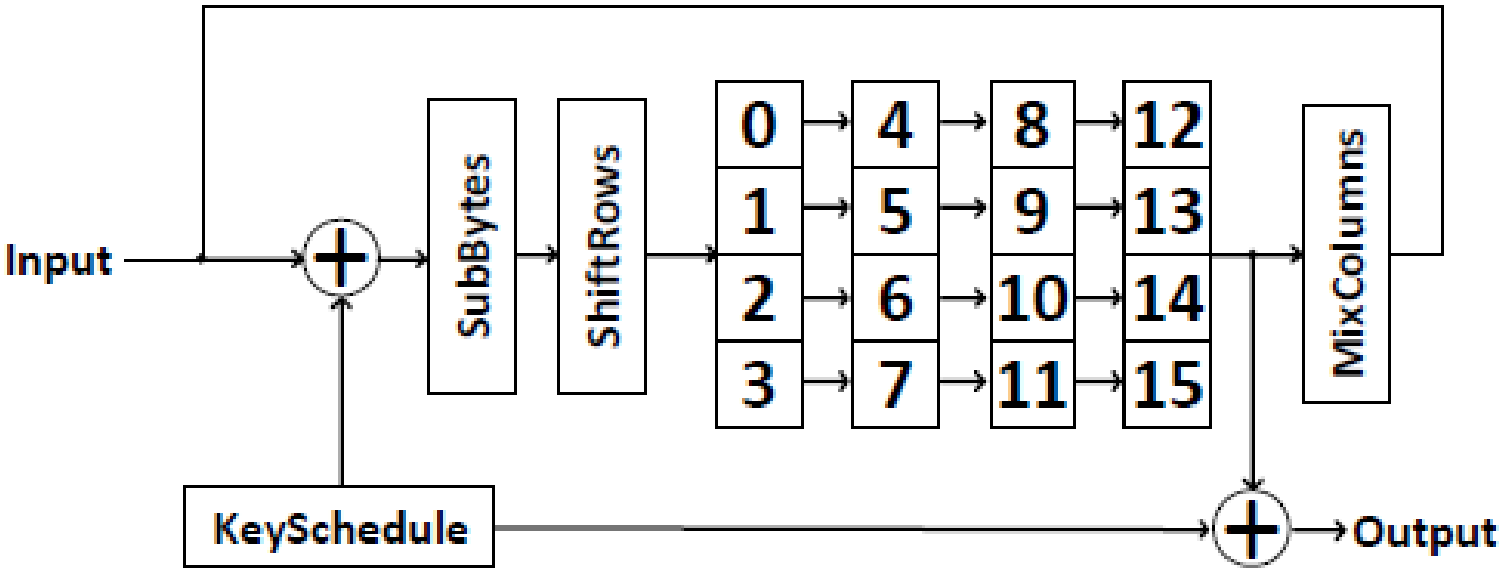


With further experiments and signal processing ...

... we recovered the 128-bit AES key with 30,000 traces (~ 3 hours of measurement)



... and came up with a hypothetical architecture of the AES engine



Conclusion

- **Full 128-bit AES key** of Stratix II can be extracted using 30,000 traces (3 hours)
- **Proprietary security mechanisms** can be reverse-engineered from software
- Security by obscurity?
- Key derivation **does not** prevent cloning

Future Work

- Other Altera product lines?
- Understand bitstream itself
- Countermeasures?

Questions now?

... or later:

pawel.swierczynski@rub.de
david.oswald@rub.de
amir.moradi@rub.de