RUHR-UNIVERSITÄT BOCHUM

# Cracking bcrypt on FPGAs

## 04. June 2014

**FluxFingers**
**Ruhr University Bochum**

Friedrich Wiemer





# Outline







RUHR-UNIVERSITÄT BOCHUM
Outline
1 Motivation
2 FPGAs
3 bcrypt
4 Design of Cracker
5 Demo
6 Results

# Motivation

Why do we care?

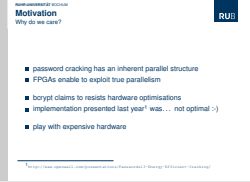
RUB

- password cracking has an inherent parallel structure
- FPGAs enable to exploit true parallelism
- bcrypt claims to resists hardware optimisations
- implementation presented last year[1] was. . . not optimal :-)
- play with expensive hardware

[1] http://www.openwall.com/presentations/Passwords13-Energy-Efficient-Cracking/








- OpenWall bcrypt implementation on different platforms:
  - (mainly) parallela/epiphany board
  - zedboard
  - Xeon Phi
  - Haswell
- Problems with FPGA implementation:
  - only implemented cost loop in fabric
  - wait cycles
  - 14 cores with very unbalanced resource utilization



What are FPGAs?

short: (re-)programmable hardware

long: see next slides

# What are FPGAs?

GPCPUs ↔ FPGAs ↔ ASICs

RUB

Field Programmable Gate Arrays lie between General Purpose CPUs and Application Specific Integrated Circuits. They combine advantages from both.

**Advantages**

- fast to program
- low level hardware design
- re-programmable

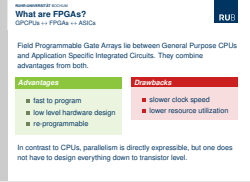
**Drawbacks**

- slower clock speed
- lower resource utilization

In contrast to CPUs, parallelism is directly expressible, but one does not have to design everything down to transistor level.








- CPUs have general functions
- supports wide range of applications

⇒ good in many points but not best in specific applications

- ASICs require huge design effort
- tedious modeling and simulation down to gate-level

⇒ fastes hardware for specific applications

- FPGAs easier to program than ASICS
- re-programmable (loadable with task specific programs)
- allow low level hardware design while still abstracts from gate level

# What are FPGAs?

How does it work?


FPGAs are build mainly out of

- Look Up Tables (LUTs)
- Flip Flops (FFs)
- Block RAMs (BRAMs)
- Digital Signal Processing units (DSPs)

but can contain hard instantiated cores
(e.g. ARM Cortex A9, etc.)



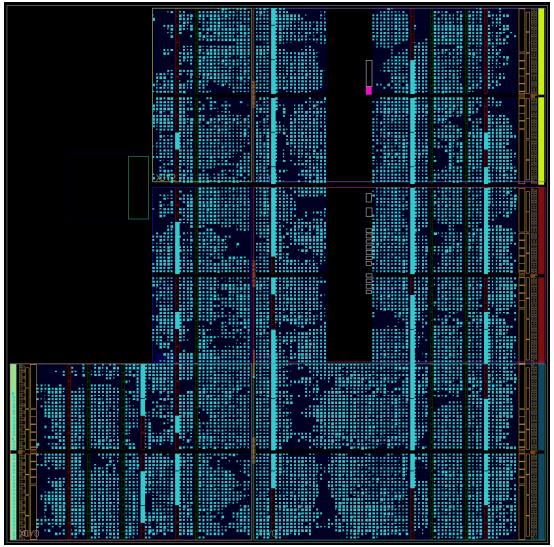


contains different kind of logic blocks

- LUTs are needed for custom logic function
- FFs to register LUT outputs, build local storage, reduce critical path, re-programmable routes
- BRAMs bigger local storage, still accessible within one clock cycle
- DSPs for signal processing (multiply accumulate)
- other hard instantiated cores (e.g. ARM Cortex A9)

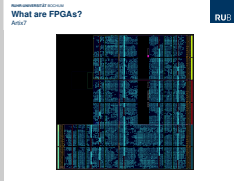
# What are FPGAs?

Artix7





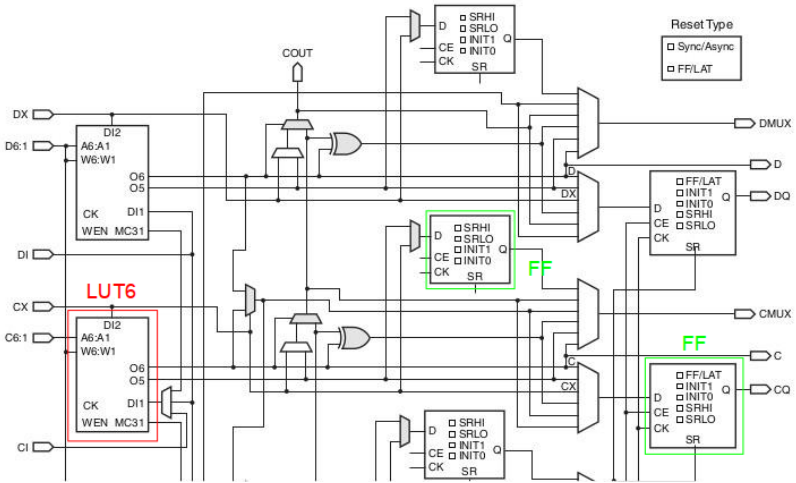




- Fabric is the place were all logic is located
- BRAM and DSPs are arranged column wise
- LUTs and FFs are arranged in slices
  (2 slices per configurable logic block (CLB))

# What are FPGAs

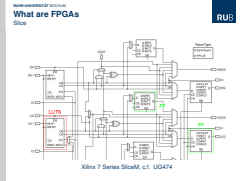
## Slice




Xilinx 7 Series SliceM, c.f. UG474



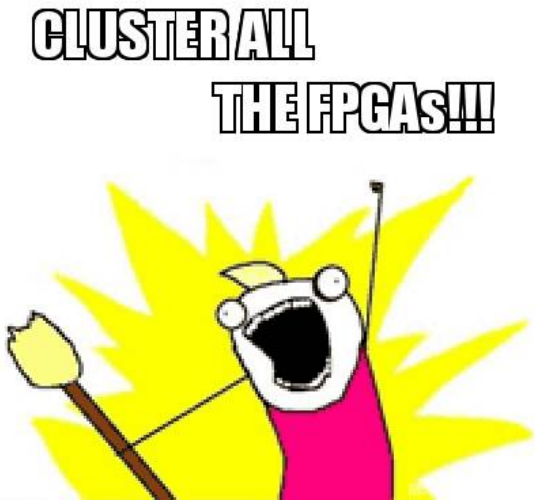




- 4 LUT6, 8 FFs per slice (2 FFs per LUT6)
- additional muxer, fast carry path, etc.

- is this important to know?
- well.. depends :-)
- one has to know how the hardware is build up, to get best performance (respect the hardware)

# FPGA-Cluster







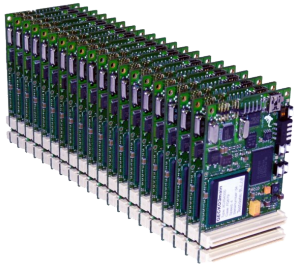

FPGA-Cluster
CLUSTER ALL
THE FPGAs!!!

What are FPGA Cluster?

- similar to GPU cluster
- aim: improve parallel performance

# FPGA-Cluster

FPGA Cluster like COPACOBANA or RIVYERA combine many FPGA chips.

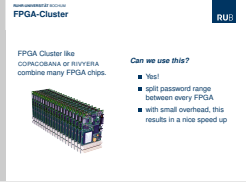


***Can we use this?***

- Yes!
- split password range between every FPGA
- with small overhead, this results in a nice speed up








- example for cluster:
  - COPACOBANA
    - 120 low power Spartan3 FPGAs
  - RIVYERA
    - 64 Spartan6 FPGAs
- Why is this useful for us?
  - Password Range can be split up at synthesis time
  - every FPGA works independently
  - with few dependencies, this results in quasi linear speed up

# What is bcrypt?

Introduced in 1999 by Provos and Mazières.[2]

***EksBlowfish***

- block cipher based on blowfish
- expensive key schedule
- encryption equals blowfish encryption

***bcrypt***

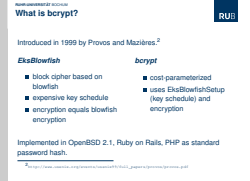
- cost-parameterized
- uses EksBlowfishSetup (key schedule) and encryption

Implemented in OpenBSD 2.1, Ruby on Rails, PHP as standard password hash.

[2] http://www.usenix.org/events/usenix99/full_papers/provos/provos.pdf








- proposed two constructions
- *EksBlowfish* less known, but serves as underlying building block for bcrypt
- itself is based on blowfish, expensive key schedule
- *bcrypt* cost parameterized hash function
- cost factor controlls computational efford of `EksBlowfishSetup`
- practical applications in OpenBSD, Ruby on Rails, PHP
- We start with explaining the key schedule

# What is bcrypt?

EksBlowfishSetup

RUB

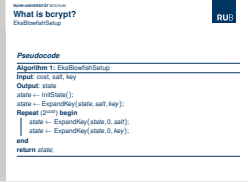
## *Pseudocode*

**Algorithm 1:** EksBlowfishSetup

**Input**: cost, salt, key
**Output**: state
$state \leftarrow \text{InitState}()$;
$state \leftarrow \text{ExpandKey}(state, salt, key)$;
**Repeat** $(2^{cost})$ **begin**
    $state \leftarrow \text{ExpandKey}(state, 0, salt)$;
    $state \leftarrow \text{ExpandKey}(state, 0, key)$;
**end**
**return** *state*;








- state consists of 4 SBoxes, each 256 $\times$ 32 bit and
- 18 32 bit P-values (subkeys) $\Rightarrow$ 4 kB memory
- initialized with digits of $\pi$
- `ExpandKey` updates state (see next slide)
- number of iterations exponential in cost parameter
- $ExpandKey(state, 0, .)$ equals blowfish key schedule

# What is bcrypt?

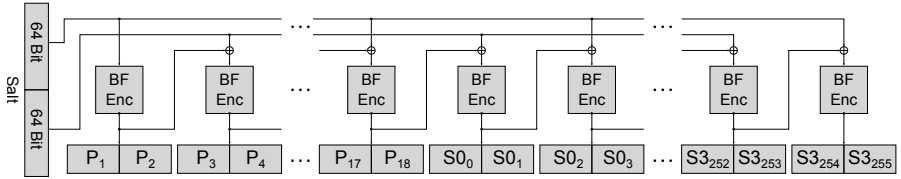
ExpandKey

RUB

## *ExpandKey*

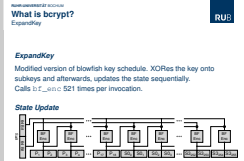
Modified version of blowfish key schedule. XORes the key onto subkeys and afterwards, updates the state sequentially.
Calls `bf_enc` 521 times per invocation.

## *State Update*





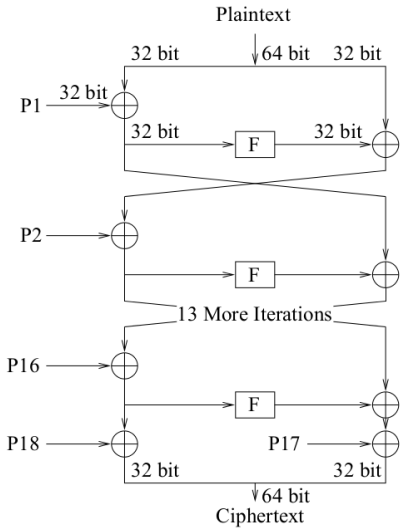


- `ExpandKey` blowfish encrypts lower 64 bit of Salt
- Result substitutes $P_1$, $P_2$ and
- gets xored with higher 64 bit of Salt, encrypted again...
- Alltogether 521 blowfish encryptions per call
- How does blowfish encryption works?

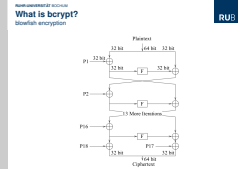
# What is bcrypt?

blowfish encryption







- simple 16 round Feistel network
- in every round: key xor, f-Function xor, swap
- f-Function consists of 4 SBox look ups, 2 additions, 1 xor
- finalize: xor $P_{17}$, $P_{18}$
- 64 bit block size

$\Rightarrow$ nice minimalistic structure

# What is bcrypt?

bcrypt

RUB

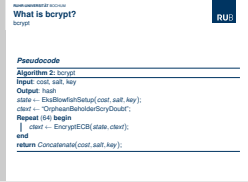
### *Pseudocode*

**Algorithm 2:** bcrypt

**Input**: cost, salt, key
**Output**: hash
*state* ← EksBlowfishSetup(*cost*, *salt*, *key*);
*ctext* ← “OrpheanBeholderScryDoubt”;
**Repeat** (64) **begin**
  *ctext* ← EncryptECB(*state*, *ctext*);
**end**
**return** *Concatenate*(*cost*, *salt*, *key*);







- bcrypt itself is quite simple
- calls `EksBlowfishSetup` followed by
- 64 blowfish encryptions in ECB mode of magic value (3 blocks)

⇒ allmost all work/time is spent during key schedule

# Where are we?

RUB

***bcrypt is***

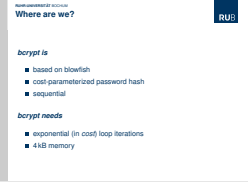
- based on blowfish
- cost-parameterized password hash
- sequential

***bcrypt needs***

- exponential (in *cost*) loop iterations
- 4 kB memory








- bcrypts cost depend only on many sequential blowfish encryption
- authors claim randomly accessed memory is costly

# Design of Cracker






Design of Cracker


- we have no chance to parallelize bcrypts structure
- does the memory consumption hurt?
- depends on target FPGAs

# Target Platforms

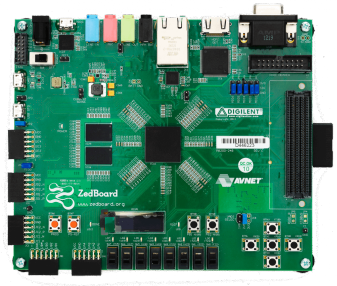
zedboard

RUB

Low cost, low power FPGA

## *Specs*

- Zynq-7000 XC7Z020 FPGA (comarable to Artix7)
- ARM Cortex A9 CPU
- HDMI, VGA, Ethernet, Audio, USB, JTAG, SD Card, Buttons. . .








- OpenWall project also targeted zedboard
- combines ARM Cortex A9 with programmable logic
- contains 140 18 kbit BRAMs
- embedded Linux can be run from SD card

# Target Platforms

RIVYERA

RUB

High Performance FPGA cluster

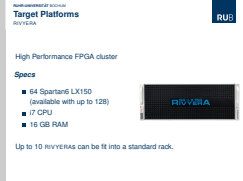
## *Specs*

- 64 Spartan6 LX150 (available with up to 128)
- i7 CPU
- 16 GB RAM


Up to 10 RIVYERAs can be fit into a standard rack.








- RIVYERA is the actual FPGA cluster available @ EmSec
- contains up to 128 Spartan6 (ours: 64)
- each Spartan6 268 18 kbit BRAMs
  (roughly twice as big as Artix7 on zedboard, but slower speed grade)

RUB

# Low Area Footprint





Optimization Goal
RUB
Low Area Footprint

Design Goal?

- simple Feistel structure will allow high clock rates

$\Rightarrow$ keep area footprint as low as possible

$\Rightarrow$ fit as many cores onto on FPGA as possible

# Design of Cracker

RUB

## *API*

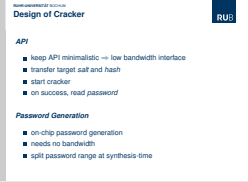
- keep API minimalistic ⇒ low bandwidth interface
- transfer target *salt* and *hash*
- start cracker
- on success, read *password*

## *Password Generation*

- on-chip password generation
- needs no bandwidth
- split password range at synthesis-time







# Design of Cracker

RUB

## *bcrypt Core*

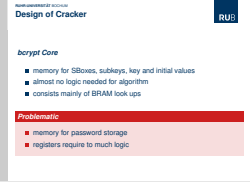
- memory for SBoxes, subkeys, key and initial values
- almost no logic needed for algorithm
- consists mainly of BRAM look ups

## *Problematic*

- memory for password storage
- registers require to much logic








- 2 SBoxes per BRAM, subkeys in 1 extra Bram
- initial values take another 2 BRAMs (can be shared)
- but: how to store passwords?
- registers in logic are to costly

$\Rightarrow$ quad-core design

# Design of Cracker

## *Quad Core*

- bundle four bcrypt cores together
- store four passwords in one BRAM
- during initialization, password generator can access BRAM
- core0, core1 and core2, core3 can access the BRAM alternately

## *Advantages*

- cores need access only during begin of `ExpandKey`
  thus, they only diverge by 36 clock cycles

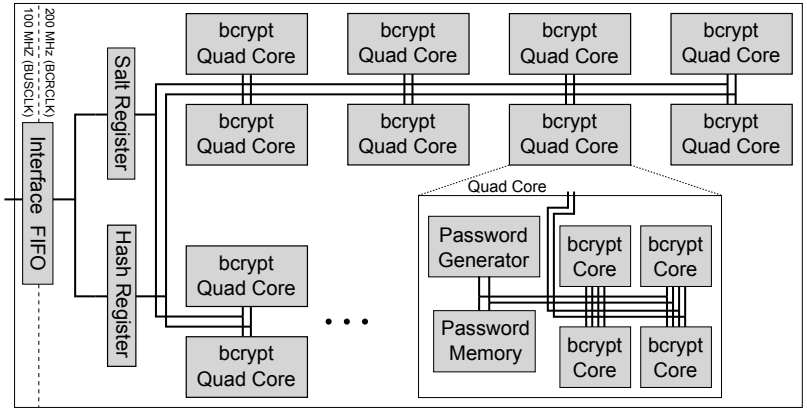




- dual port BRAM allows 2 read/write accesses per clock
- bundle 4 cores, alternate their memory access phases
- password generator can access the memory during 256 initialization cycles
- cores diverge only by 36 clock cycles

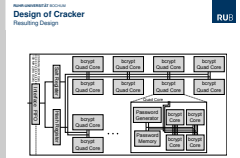
# Design of Cracker

## Resulting Design







- every quad core operates fully independent on his own password subrange

# How to interface the Cracker?

Different possibilities for zedboard:

- AXI4 Interfaces
  - \+ low area footprint
  - – more complex to implement
- Xillybus
  - \+ abstracts from AXI4 Interfaces
  - \+ Linux support ⇒ easy to demonstrate
  - – proprietary
  - – “fat”, i.e. higher resource consumption

Custom interface for RIVYERA needed.





RUHR-UNIVERSITÄT BOCHUM
How to interface the Cracker?
Different possibilities for zedboard:
AXI4 Interfaces
+ low area footprint
– more complex to implement
Xillybus
+ abstracts from AXI4 Interfaces
+ Linux support ⇒ easy to demonstrate
– proprietary
– “fat”, i.e. higher resource consumption
Custom interface for RIVYERA needed.

# How to interface the Cracker?

RUB

## *Xillybus*

- stream access through FIFOs or DMA into fabric logic
- abstraction from AXI4 is quite nice
- Linux driver ⇒ just write to `/dev/xillybus_mem_8`

## *Memory Layout*

```
root@localhost:~# hexdump -C -v -n 64 /dev/xillybus_mem_8
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040
root@localhost:~#  Salt Hash Password Status Register
```






- Linux driver allows DMA into logic
- host side interface consists only of file write/reads.
- Status Register:
  - low nibble used for host-to-FPGA communication (reset, start)
  - high nibble used for FPGA-to-host communication (done, success)

# Design of Cracker

## Resulting Resources

RUB

### *zedboard*

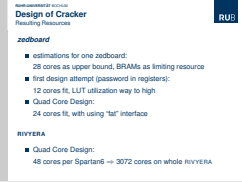
- estimations for one zedboard:
  28 cores as upper bound, BRAMs as limiting resource
- first design attempt (password in registers):
  12 cores fit, LUT utilization way to high
- Quad Core Design:
  24 cores fit, with using “fat” interface

### RIVYERA

- Quad Core Design:
  48 cores per Spartan6 $\Rightarrow$ 3072 cores on whole RIVYERA








- with Xillybus 24 cores @ 200 MHz possible (zedboard)
- with custom interface 48 cores @ 100 MHz possible (RIVYERA)

# Demo







- demonstrate interface :-)

# Results

Compared to GPUs

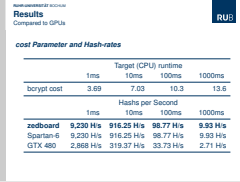
RUB

## *cost Parameter and Hash-rates*

| | Target (CPU) runtime | | | |
|---|---|---|---|---|
| | 1ms | 10ms | 100ms | 1000ms |
| bcrypt cost | 3.69 | 7.03 | 10.3 | 13.6 |
| | Hashs per Second | | | |
| | 1ms | 10ms | 100ms | 1000ms |
| **zedboard** | **9,230 H/s** | **916.25 H/s** | **98.77 H/s** | **9.93 H/s** |
| Spartan-6 | 9,230 H/s | 916.25 H/s | 98.77 H/s | 9.93 H/s |
| GTX 480 | 2,868 H/s | 319.37 H/s | 33.73 H/s | 2.71 H/s |








- measured bcrypt runtime on different CPUs

⇒ averaged cost factor

- Hash-rates compared too previous cost factors
- GPU Hash-rate measured with oclHashcat

⇒ about 3 times faster than GTX 480

# Results

RUB

## *Compared to OpenWall*

- OpenWall achieves 780 H/s with cost 5
- zedboard achieves 3,200 H/s with cost 5

## *Besides pure Hash-rate*

- zedboard consumes 5 W, GTX 480 consumes up to 430 W
- zedboard costs $300–$400, GTX 480 cost $500 in 2010





RUHR-UNIVERSITÄT BOCHUM
Results
RUB
Compared to OpenWall
OpenWall achieves 780 H/s with cost 5
zedboard achieves 3,200 H/s with cost 5
Besides pure Hash-rate
zedboard consumes 5 W, GTX 480 consumes up to 430 W
zedboard costs $300–$400, GTX 480 cost $500 in 2010

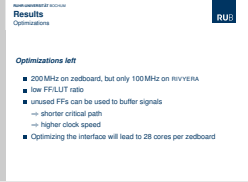
RUB

## *Optimizations left*

- 200 MHz on zedboard, but only 100 MHz on RIVYERA
- low FF/LUT ratio
- unused FFs can be used to buffer signals
  - ⇒ shorter critical path
  - ⇒ higher clock speed
- Optimizing the interface will lead to 28 cores per zedboard







# Questions?

Thanks!





RUHR-UNIVERSITÄT BOCHUM
Questions?
Thanks!