MERKLE-HELLMAN-KRYPTOSYSTEM

Algorithmus zur Entschlüsselung von Kryptotexten

Projektarbeit im Rahmen der Vorlesung *Algorithm Engineering*

WiSe 2015/16, Friedrich-Schiller-Universität Jena

Sebastian Brühl

Kryptosystem nach Merkle-Hellman

- 1978 von Ralph C. Merkle und Martin Hellman vorgestellt
- *Hiding Information and Signatures in Trapdoor Knapsacks* [1]
- basiert auf Untersummenproblem -> $O(2^n)$
- Entschlüsselung bedeuted Berechnung eines Skalarproduktes
- empfohlene Schlüssellänge um ein Knacken der Verschlüsselung in angemessener Zeit zu verhindern: $n > 100$
- 1976 – Cray-1 – 80 MFLOPS
- 2016 – raj – 4 TFLOPS
- Gilt die Aussage $n > 100$ heute noch?

Vektorisierung

Public Key

2292	1089	211	1625	1283	599	759	315	2597	2463

Klartext ●

?	?	?	?	?	?	?	?	?	?

=

Kryptotext | 6790 |

Bedingung: ? = {0, 1}

Vektorisierung

Public Key

2292	1089	211	1625	1283	599	759	315	2597	2463

Klartext

0	0	1	1	1	0	1	1	1	0

Kryptotext | 6790 |

Bedingung: ? = {0, 1}

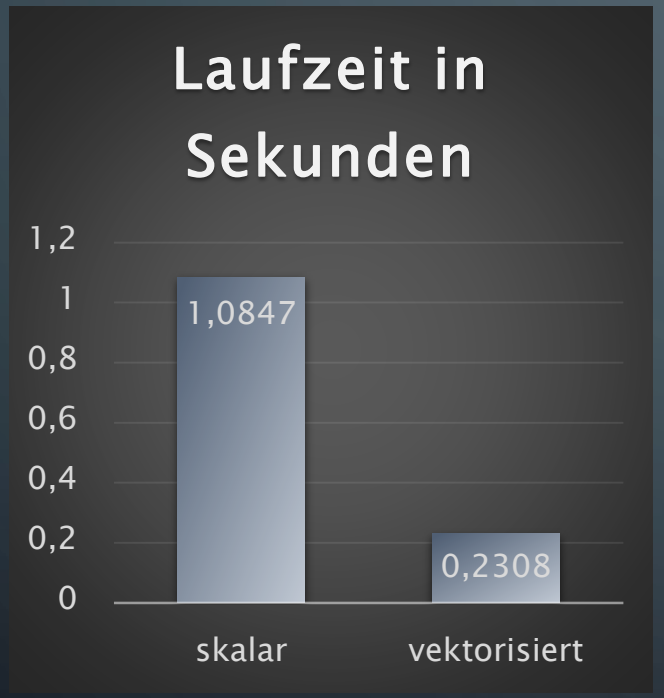
Vektorisierung

Probleme

- Schlüssellängen $n > 100$ benötigen sehr große Zahlen (2^{200})
- Algorithmus in zwei Versionen
- _64: `unsigned long long` (vektorisierbar)
- _BN: `boost::multiprecision::int_512_t` (nicht vektorisierbar)
- Initialisierung/Allozierung des Testvektors
- Padding des Testvektors

Vektorisierung

- skalare Version: GNU Compiler
- vektorisierte Version: Intel Compiler + OpenMP/SIMD



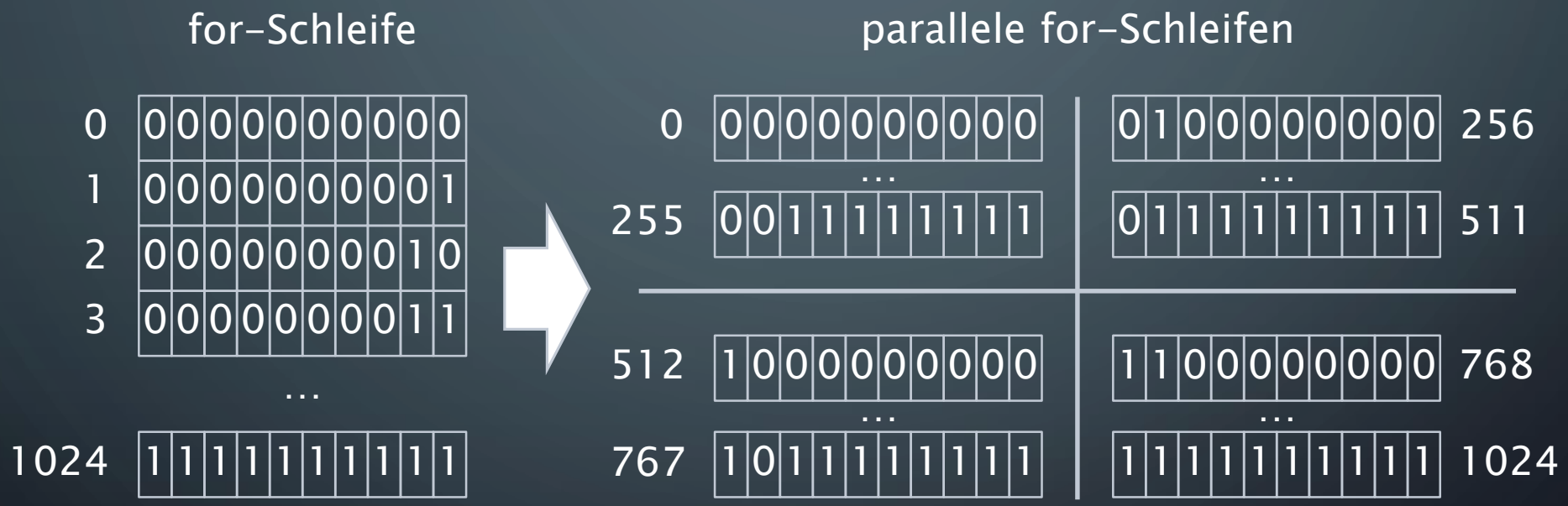

```
#pragma omp for schedule(static)
for(uint64_t n=0; n<n_max; ++n) {
    uint64_t result=0;

    #pragma omp simd aligned(pThreadTestVector,aKeyVector:64) reduction(+:result)
    for(int i=0; i<aKeyLength; ++i) {
        pThreadTestVector[i] = ((n >> i) & 1);
        result += pThreadTestVector[i] * aKeyVector[i];
    }
    if(aDP == result) {
        #pragma omp simd aligned(pThreadTestVector,pThreadPlainVector:64)
        for(int i=0; i<aKeyLength; ++i) {
            pThreadPlainVector[i] = pThreadTestVector[i];
        }
        #pragma omp cancel for
    }
}
```

Parallelisierung

Hauptaufgabe des Algorithmus

- *Testvektor* ● *Public Key* $\stackrel{?}{=}$ *Kryptotext*
- 2^n voneinander unabhängige Iterationen, im Beispiel $2^{10}=1024$



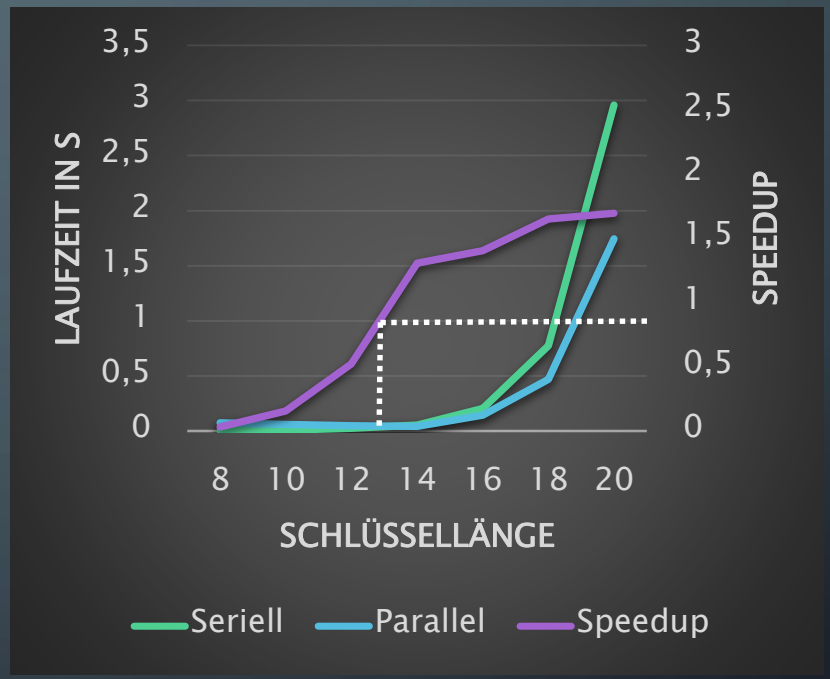
- Problem: Abbruch bei Treffer -> `omp cancel for`

Parallelisierung

Parallelisierung mittels `omp for`

```
#pragma omp for schedule(static)
for(uint64_t n=0; n<n_max; ++n) {
    uint64_t result=0;

    #pragma omp simd aligned(pThreadTestVector,aKeyVector:64) reduction(+:result)
    for(int i=0; i<aKeyLength; ++i) {
        pThreadTestVector[i] = ((n >> i) & 1);
        result += pThreadTestVector[i] * aKeyVector[i];
    }
    if(aDP == result) {
        #pragma omp simd aligned(pThreadTestVector,pThreadPlainVector:64)
        for(int i=0; i<aKeyLength; ++i) {
            pThreadPlainVector[i] = pThreadTestVector[i];
        }
        #pragma omp cancel for
    }
}
```



Verteilte Ausführung

- Merkle-Hellman ist eine Blockchiffre
- jeder Block muss einzeln entschlüsselt werden
- Verteilung der Blöcke auf verschiedene Knoten

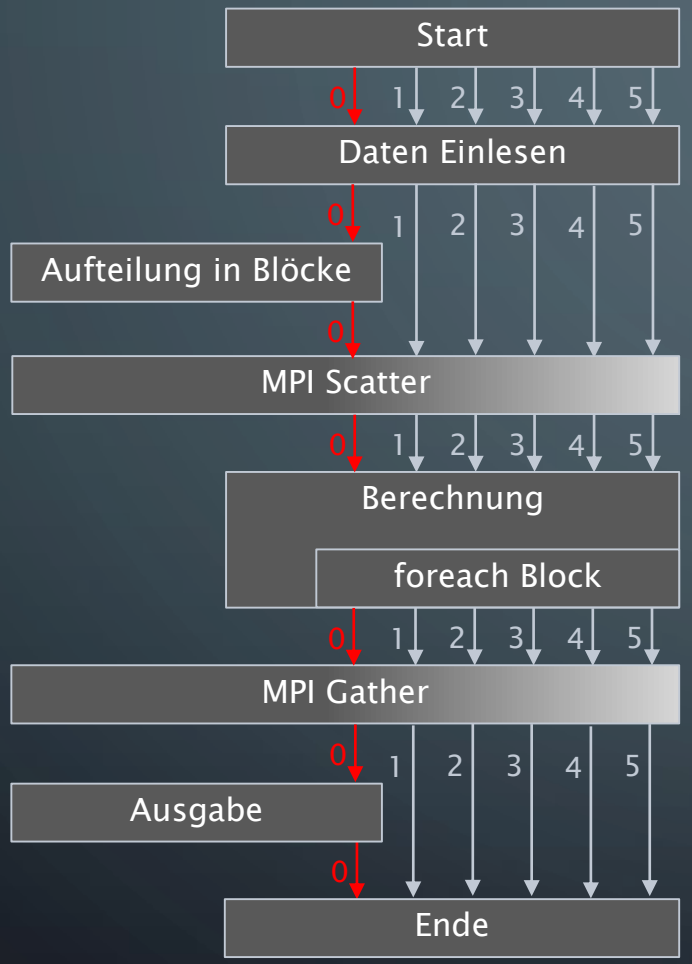
1 Knoten

173827820633687652372624452047340402030882221937893958511734521023872356418939085597090364389627364262981059221135046440247117475990107748665711044593982336475481416529010910714520788414159083012046649554111174443135439315902065428428215154146405706987961414368228039787904434979082798126805867157970945712079109491368153579913062443570815038298210976213795694513811374897094481784855885396337418574317731032706247758097529972562272159782548137519606286546903470925992160413351629756774223911931282503775067036655917747618068886063711916416668715956791072013372980365000799780756056231912669471404272013833375546208052710389895352526885028546982994426354812054399946016359004753116070612675535546961874807733704054194643823103193483850762825

6 Knoten

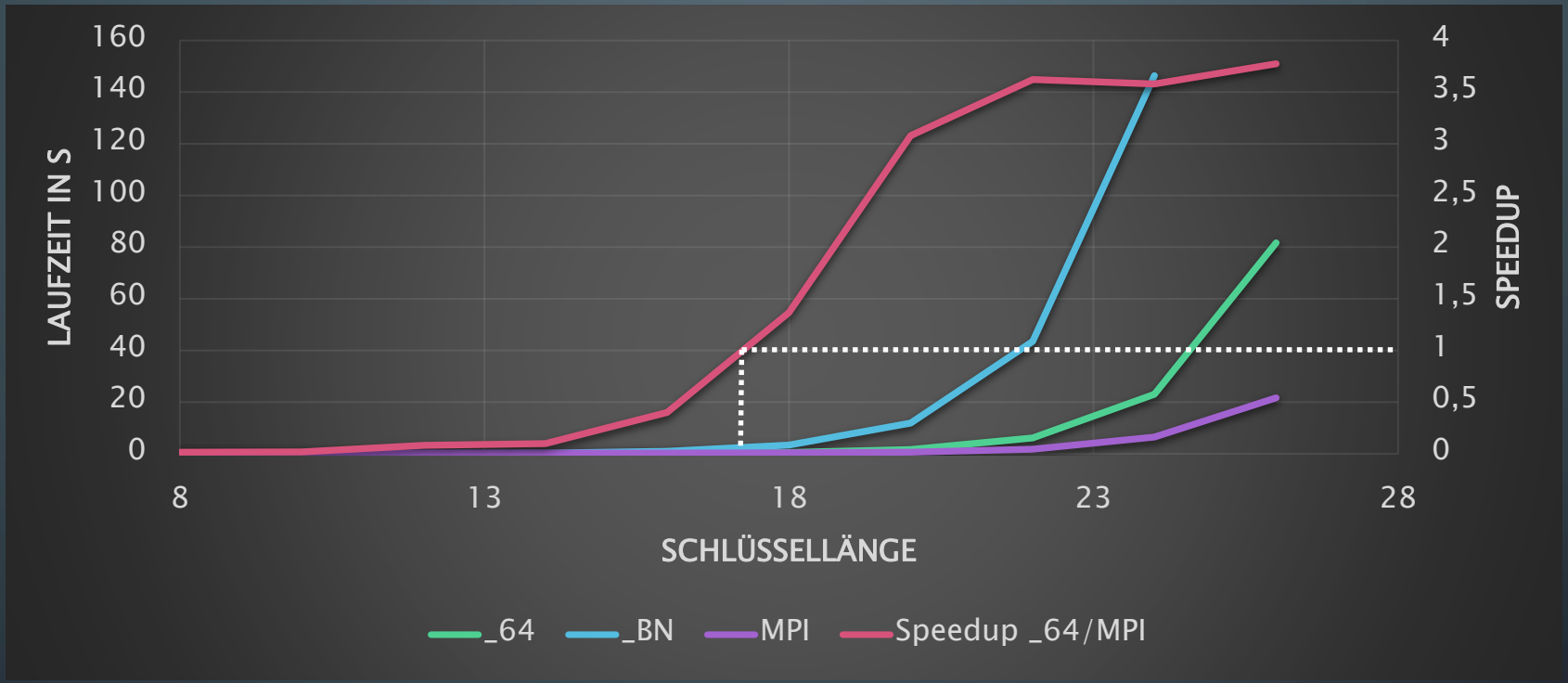
173827820633687652372624452047340402030882221937893958511734521023872356418939085597090364389627364262981059221135046440247117475990107748665711044593982336475481416529010910714520788414159083012046649554111174443135439315902065428428215154146405706987961414368228039787904434979082798126805867157970945712079109491368153579913062443570815038298210976213795694513811374897094481784855885396337418574317731032706247758097529972562272159782548137519606286546903470925992160413351629756774223911931282503775067036655917747618068886063711916416668715956791072013372980365000799780756056231912669471404272013833375546208052710389895352526885028546982994426354812054399946016359004753116070612675535546961874807733704054194643823103193483850762825

Verteilte Ausführung



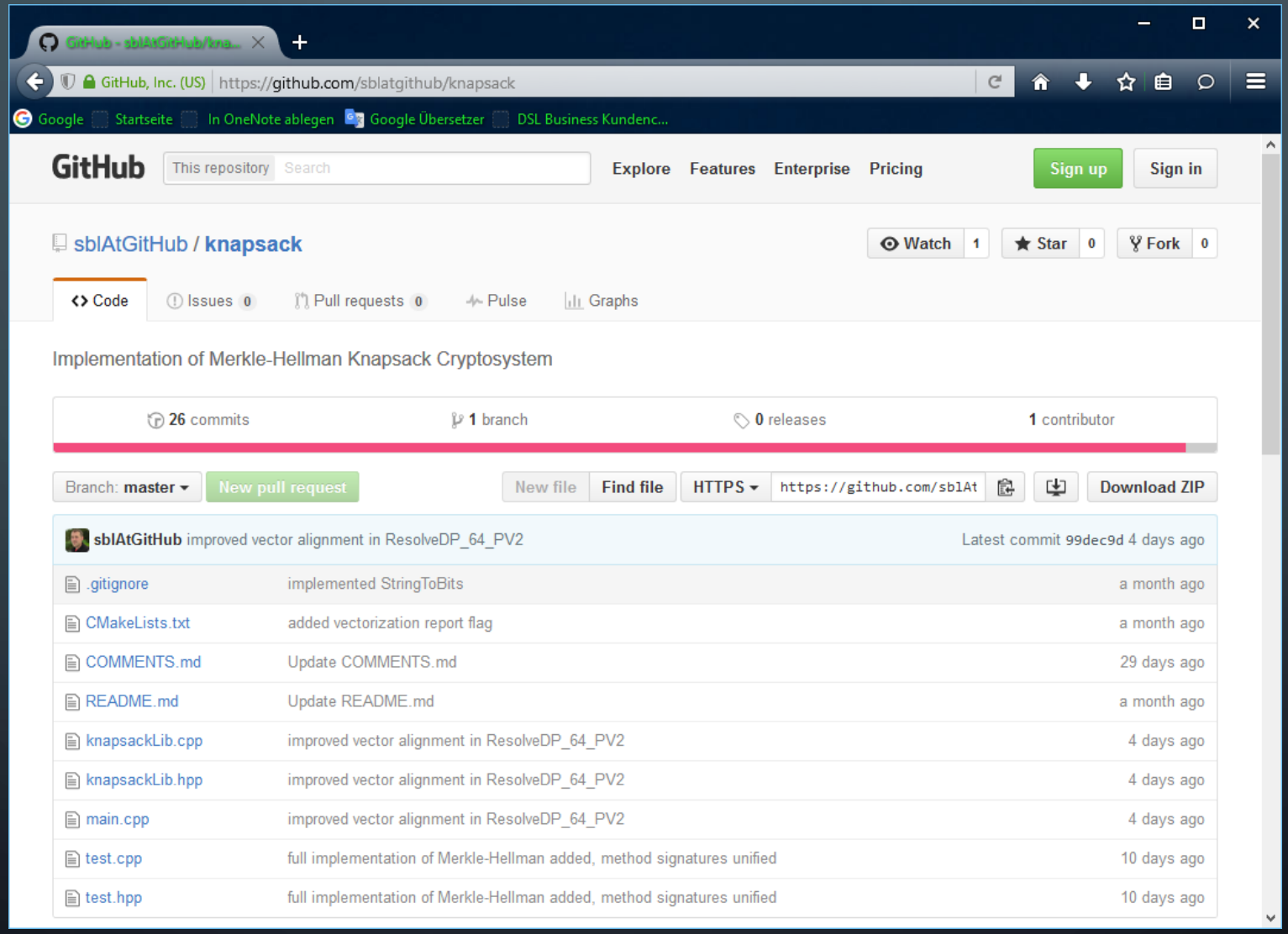
- Verteilung mittels MPI
- beispielsweise 6 Knoten
- Knoten 0 übernimmt Steuerung
- Alle Knoten sind an der Berechnung beteiligt

Benchmark – Speedup



- ca. $22*10^9$ Operationen/s (MPI)
- Laufzeit MPI bei einer Schlüssellänge von 100: $80*10^{12}$ Jahre
- Alter des Universums: $14*10^9$ Jahre

Quellcode: https://github.com/sblatgithub/knapsack



Quellen

[1] Ralph C. Merkle, Martin E. Hellman. “Hiding Information and Signatures in Trapdoor Knapsacks“ In IEEE TRANSACTIONS ON INFORMATION THEORY, VOL: IT-24, NO. 5, SEPTEMBER 1978.