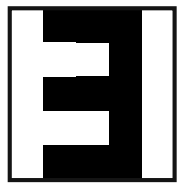
HUMBOLDT-UNIVERSITÄT ZU BERLIN

Implementierung eines MPFSS Algorithmus

Leonie Reichert · April 15, 2019

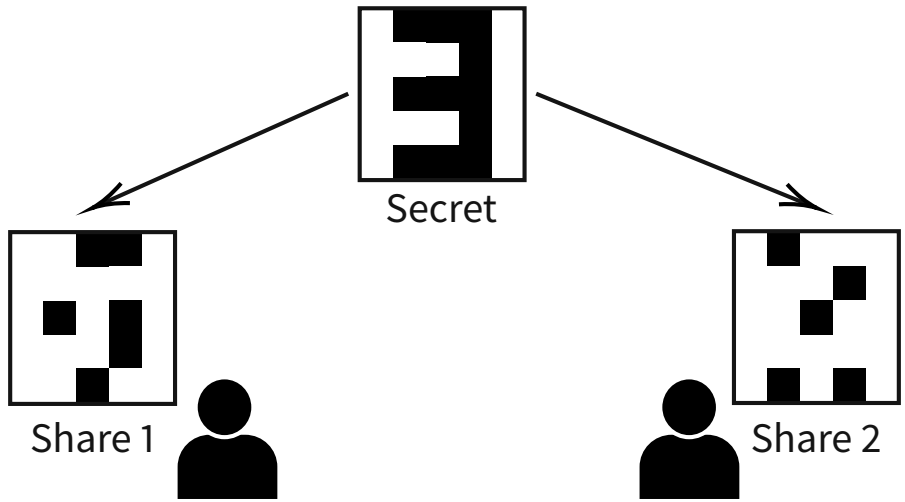
Was ist Multi-Point Function Secret Sharing?

Was ist Secret Sharing?

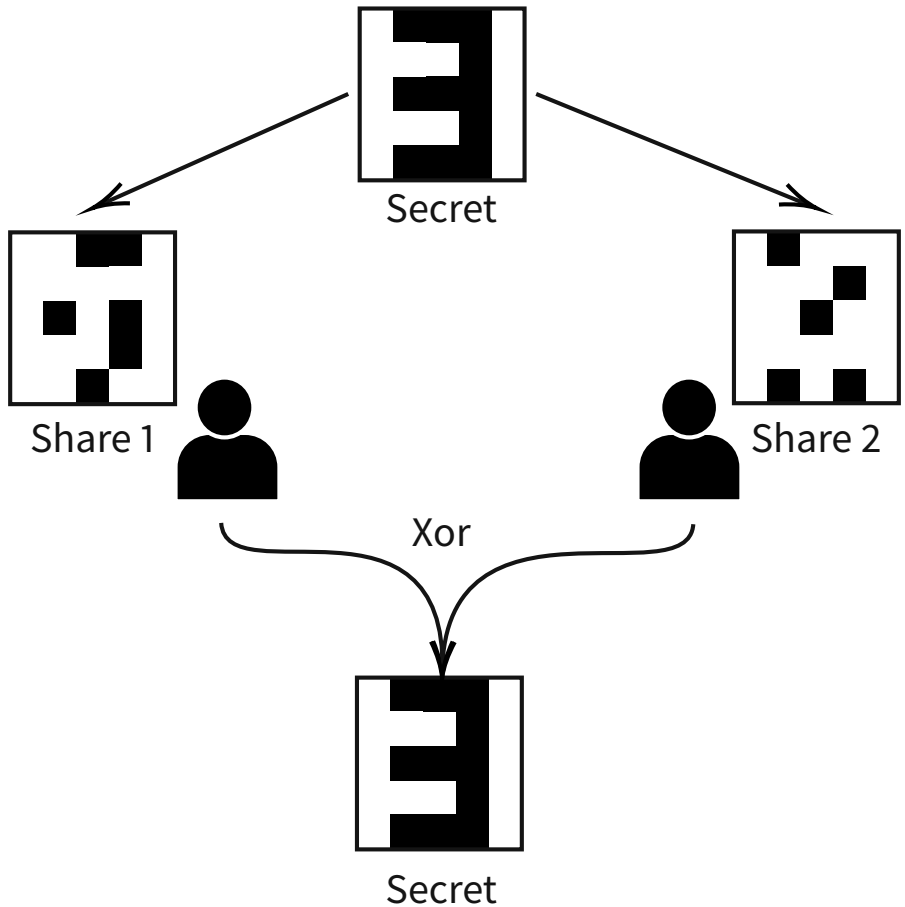


Secret

Was ist Secret Sharing?



Was ist Secret Sharing?



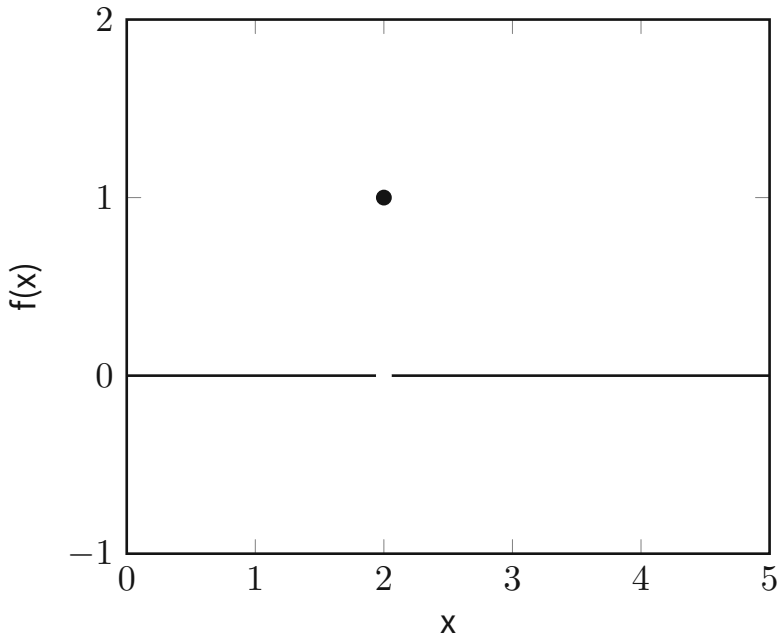
Was ist Function Secret Sharing?

- Grundidee: Function Secret Sharing
  - Zerlege $f(x)$ so dass : $f(x) = f_0(x) + f_1(x)$

Was ist Function Secret Sharing?

- Grundidee: Function Secret Sharing
  - Zerlege $f(x)$ so dass : $f(x) = f_0(x) + f_1(x)$
  - Teilfunktionen verschleiern $f(x)$
  - $f_0(b)$ und $f_1(b)$ für beliebiges b berechenbar
    - $\Rightarrow$ Keine zusätzliche Kommunikation
- Zentral oder verteilt berechenbar

Was ist eine Multi-Point Function?



$$f(x) = \begin{cases} 1 & \text{wenn } x = 2 \\ 0 & \text{sonst} \end{cases}$$

Was ist MPFSS?

- $f(x) \neq 0$ an t Stellen ("Indices")
- Eine oder keine Partei kennt die Indices

Was ist MPFSS?

- $f(x) \neq 0$ an t Stellen ("Indices")
- Eine oder keine Partei kennt die Indices
- Nehme Intervall aus den natürlichen Zahlen

  $\Rightarrow \; t_i \in [0, n]$

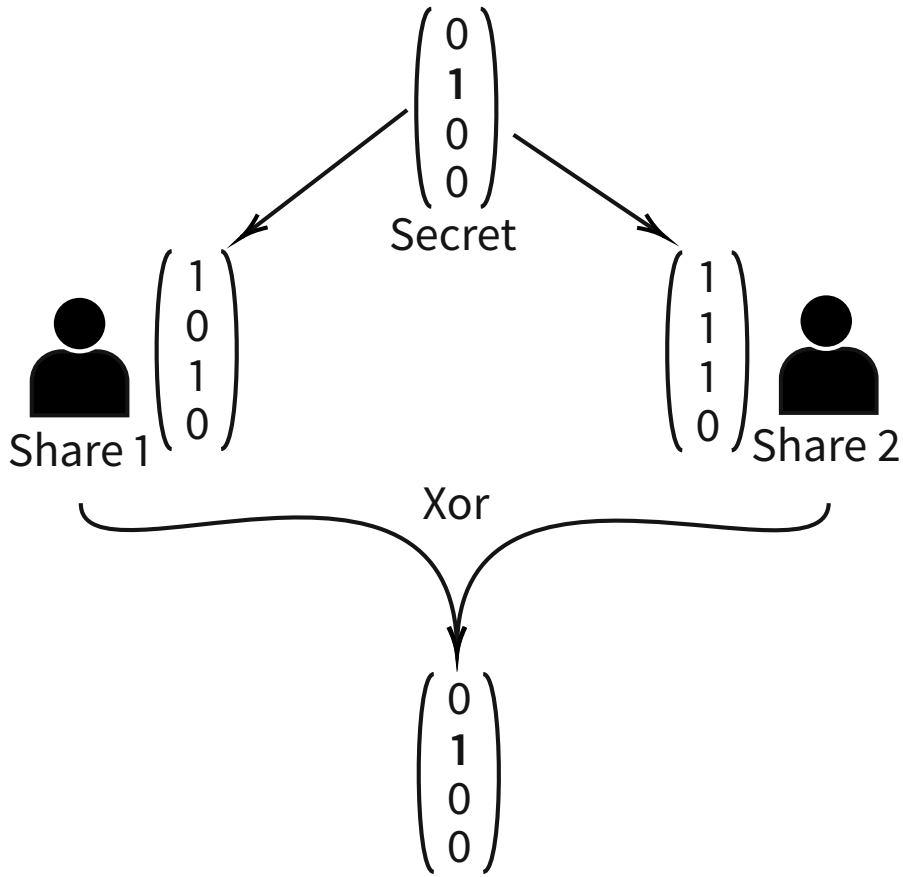
Was ist MPFSS?

- $f(x) \neq 0$ an t Stellen ("Indices")
- Eine oder keine Partei kennt die Indices
- Nehme Intervall aus den natürlichen Zahlen

  $\Rightarrow$ $t_i \in [0, n]$
- Ergebnis: Zwei Vektoren ("Shares") $\vec{v_0}$, $\vec{v_1}$
  - $\vec{v_0} \text{ xor } \vec{v_1} = \vec{v}$
  - $\vec{v} \neq 0$ an den t Indices

Distributed Point Funtion

- Distributed Point Function (DPF)
  - Implementierung von FSS für Point Functions
- Zwei Parteien Protokoll



Existierende Implementierungen

- Implementierung DPF existiert [1]
  - In Obliv-C geschrieben

  $\Rightarrow$ Darauf aufbauen

[1] Doerner and Shelat: "Scaling ORAM for Secure Computation"

[2] Zahur and Evans: "Obliv-C: A Language for Extensible Data-Oblivious Computation"

Existierende Implementierungen

- Implementierung DPF existiert [1]
  - In Obliv-C geschrieben
  - $\Rightarrow$ Darauf aufbauen
- Obliv-C [2]
  - Framework für *Secure Multi-Party Computation*
  - Abstrahiert und Vereinfacht Kommunikation zwischen Parteien
  - Übersetzt C-Code in *Yao Garbled Circuits*

[1] Doerner and Shelat: "Scaling ORAM for Secure Computation"

[2] Zahur and Evans: "Obliv-C: A Language for Extensible Data-Oblivious Computation"

Naives MPFSS

Naives MPFSS

- Single-Point Function zu Multi-Point Function?
  $\Rightarrow$ Führe DPF t mal aus
- Jede Partei verxodert die entstehenden t Vektoren

Naives MPFSS

- Single-Point Function zu Multi-Point Function?
  $\Rightarrow$ Führe DPF t mal aus
- Jede Partei verxodert die entstehenden t Vektoren
- Problem: Jede DPF geht einmal über gesamtes Inputintervall
  $\Rightarrow$ Kosten: $\mathcal{O}(t \cdot n)$

Naives MPFSS

- Single-Point Function zu Multi-Point Function?
  $\Rightarrow$ Führe DPF t mal aus
- Jede Partei verxodert die entstehenden t Vektoren
- Problem: Jede DPF geht einmal über gesamtes Inputintervall
  $\Rightarrow$ Kosten: $\mathcal{O}(t \cdot n)$
- Vorteil: Verschleierte Indices möglich

MPFSS mit Batch Codes

MPFSS mit Batch Codes

- Ziel: Laufzeit verbessern

[3] Boyle et al.: "Compressing Vector OLE"

MPFSS mit Batch Codes

- Ziel: Laufzeit verbessern
- Idee: Mache Intervall für DPFs kleiner
- Verwende Grundidee von *Combinatorial Batch Codes* [3]

  $\Rightarrow$ Zerlegen des Inputintervalls

[3] Boyle et al.: "Compressing Vector OLE"

Batch Codes: Schritte

1. Nehme Inputintervall, teile auf m Buckets auf
   - Jeder Wert wird d mal zufällig eingefügt

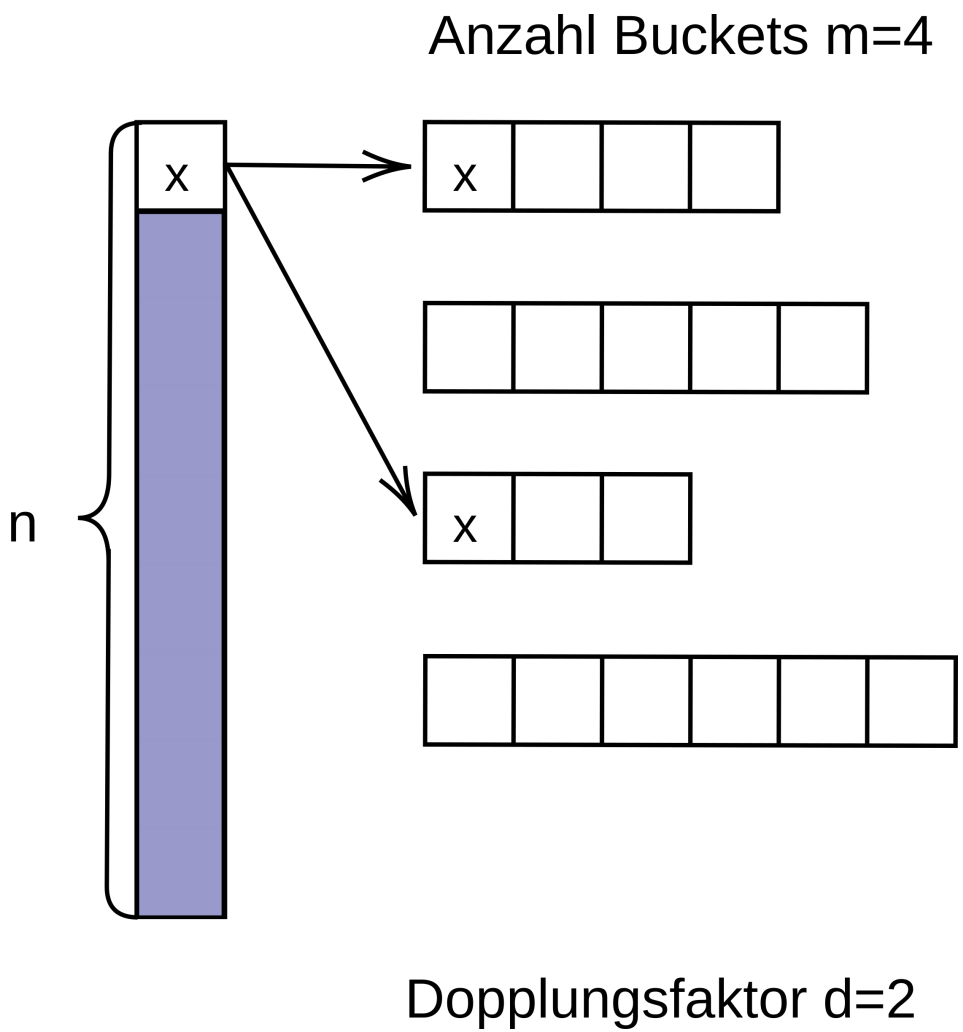
Batch Codes: Schritte

1. Nehme Inputintervall, teile auf m Buckets auf
   - Jeder Wert wird d mal zufällig eingefügt
2. Ordne Indices zu Buckets

Batch Codes: Schritte

1. Nehme Inputintervall, teile auf m Buckets auf
   - Jeder Wert wird d mal zufällig eingefügt
2. Ordne Indices zu Buckets
3. Führe pro Bucket einmal DPF aus
   - Input für DPF deutlich kürzer
   - Durchschnittliche Länge: $\frac{d \cdot n}{m}$

Batch Codes: Buckets erstellen



Batch Codes: Zuordnung der Indices

- Ordne die t Indices Buckets zu
  - $\Rightarrow$ Index muss in Bucket vorkommen

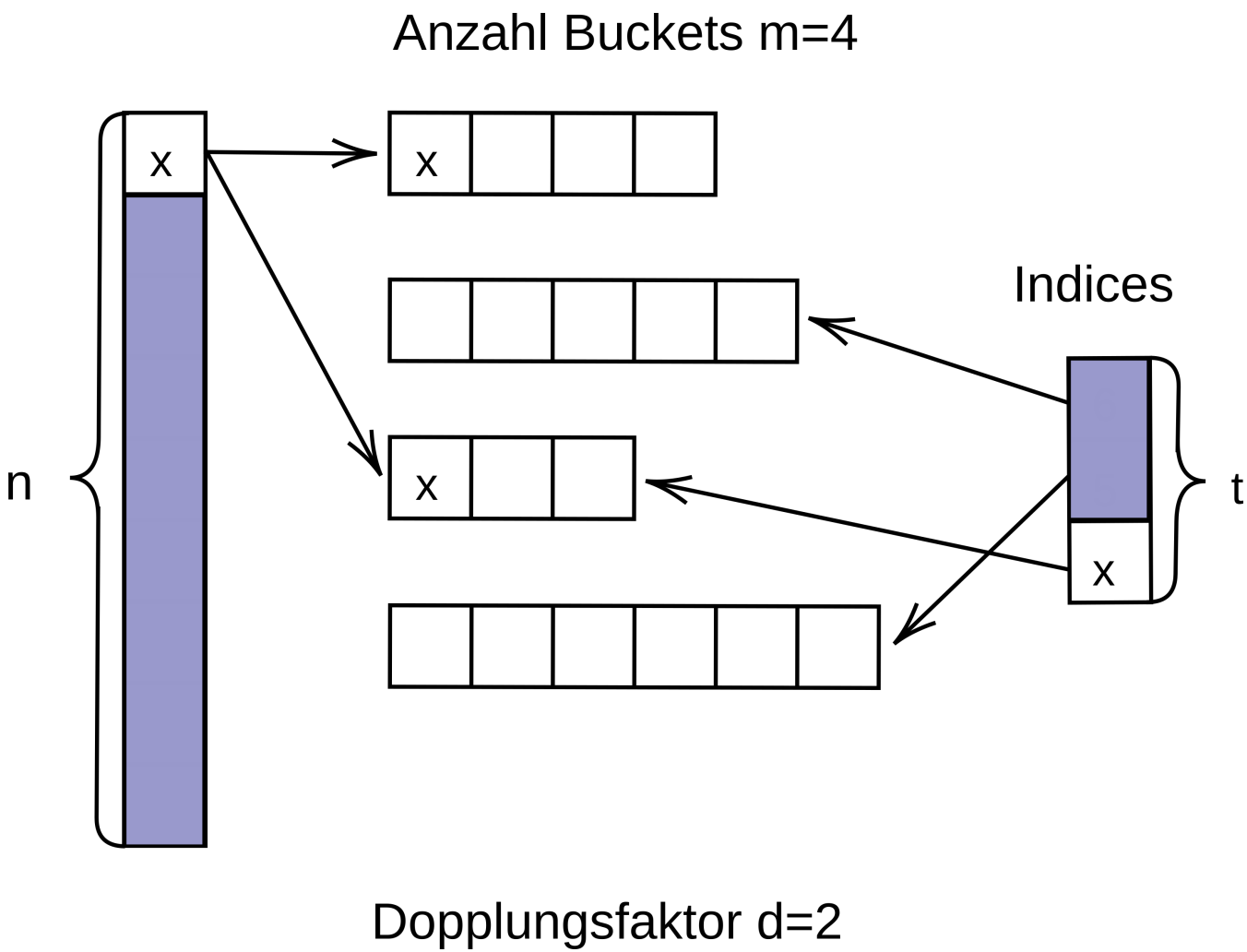
Batch Codes: Zuordnung der Indices

- Ordne die t Indices Buckets zu
  - $\Rightarrow$ Index muss in Bucket vorkommen
- Problem: Einer muss die Zuordnung machen

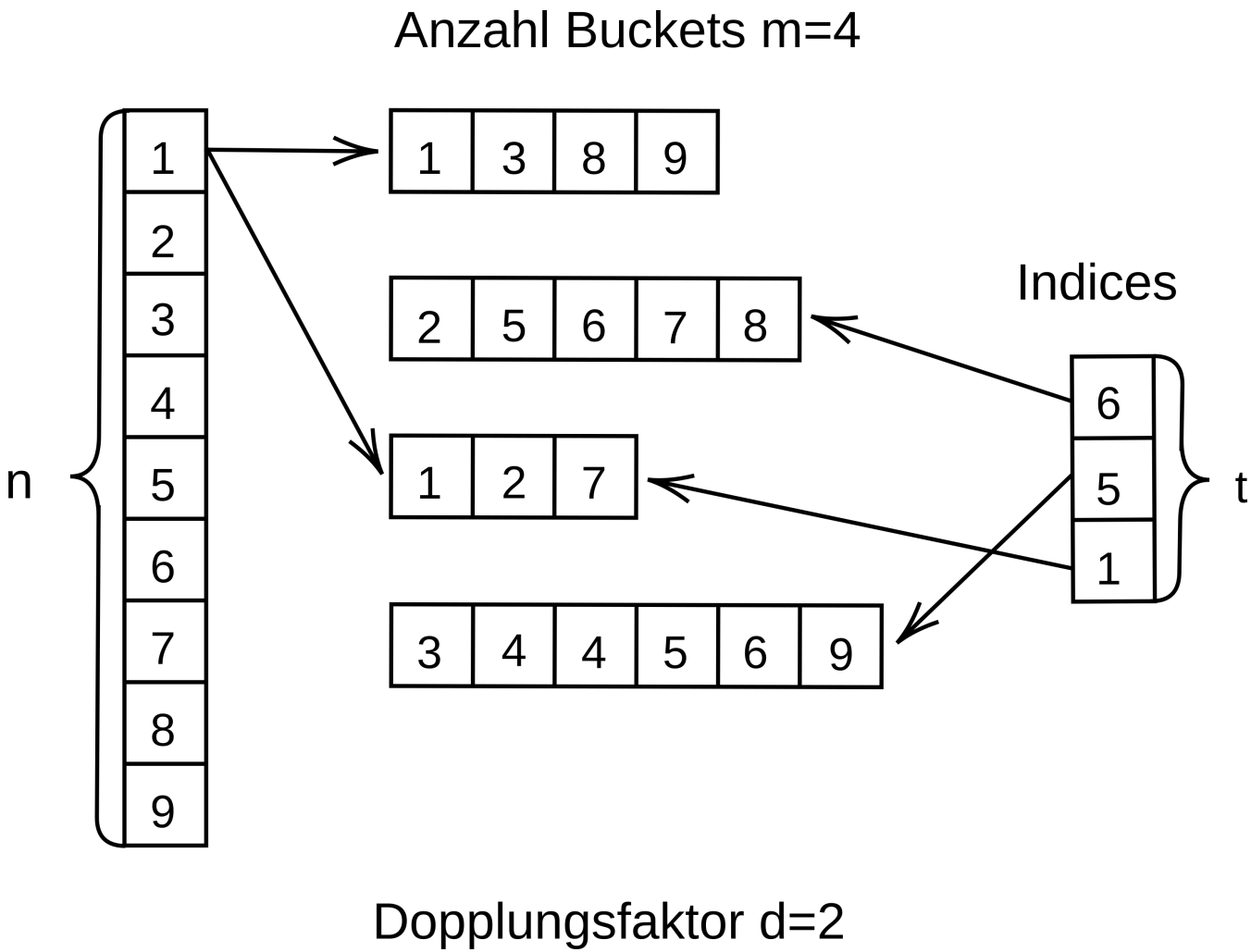
Batch Codes: Zuordnung der Indices

- Ordne die t Indices Buckets zu
  - $\Rightarrow$ Index muss in Bucket vorkommen
- Problem: Einer muss die Zuordnung machen
  - Eine Partei muss die Indices kennen
  - Abschwächung des allgemeinen MPFSS
  - Für viele Zwecke ausreichend

Batch Codes: Zuordnung der Indices



Batch Codes: Zuordnung der Indices



Batch Codes: Zuordnung der Indices

- Ziel: Zuordnung finden, wenn möglich

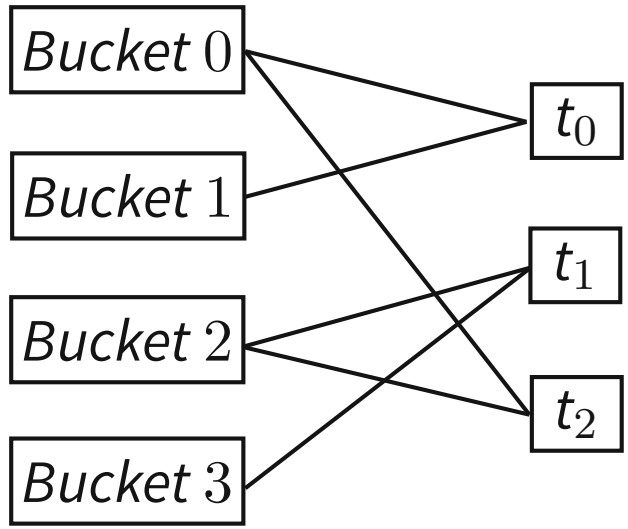
Batch Codes: Zuordnung der Indices

- Ziel: Zuordnung finden, wenn möglich
- Erfolgswahrscheinlichkeit nach Boyle et al. (2018):

  $\Rightarrow \; P = 1 - t^{-2d+2}$

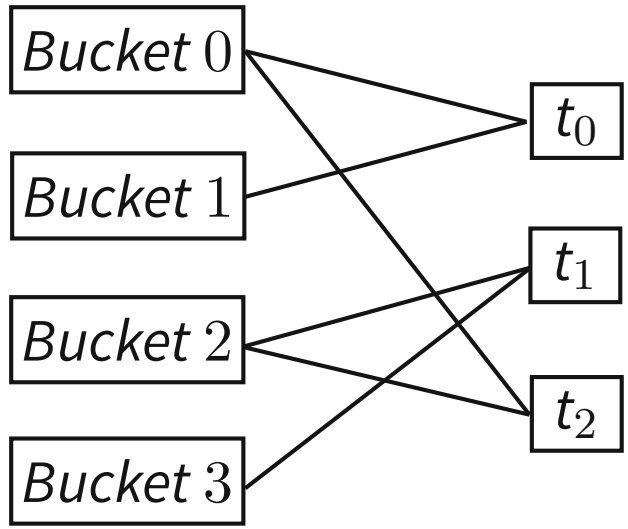
Batch Codes: Zuordnung der Indices

- Zuordnung als bipartites Matching betrachten



Batch Codes: Zuordnung der Indices

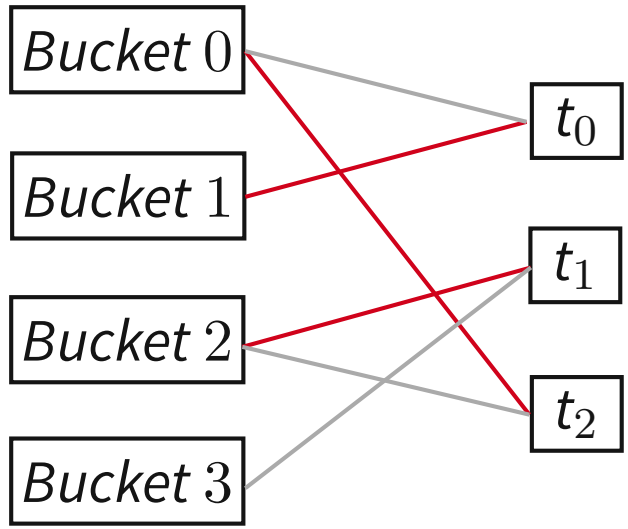
- Zuordnung als bipartites Matching betrachten



- Finde maximales Matching M von Buckets zu Indices
- Wenn $|M| < t$, dann gibt es keine Lösung

Batch Codes: Zuordnung der Indices

- Zuordnung als bipartites Matching betrachten



- Finde maximales Matching M von Buckets zu Indices
- Wenn $|M| < t$, dann gibt es keine Lösung

Batch Codes: Zuordnung der Indices

- Greedy Ansatz
  - Schlägt oft fehl, auch wenn Zuordnung theoretisch möglich

Batch Codes: Zuordnung der Indices

- Greedy Ansatz
  - Schlägt oft fehl, auch wenn Zuordnung theoretisch möglich
- Flow Graphen
  - Füge Quelle und Senke zu bipartiten Graphen
  - Finde maximalen Fluss

Batch Codes: Zuordnung der Indices

- Greedy Ansatz
  - Schlägt oft fehl, auch wenn Zuordnung theoretisch möglich
- Flow Graphen
  - Füge Quelle und Senke zu bipartiten Graphen
  - Finde maximalen Fluss
  - Hopcroft-Karp

    $\Rightarrow \mathcal{O}((E+V)\cdot\sqrt{V}) = \mathcal{O}((m+t+t\cdot d)\cdot\sqrt{m+t})$

Batch Codes: DPF ausführen

- Pro Bucket einmal DPF ausführen
  - Benötige Position von Index t_i in Bucket i (verschleiert!)

Batch Codes: DPF ausführen

- Pro Bucket einmal DPF ausführen
  - Benötige Position von Index t_i in Bucket i (verschleiert!)
- Erhalte m DPF Output Vektoren
  - Benötigt $\mathcal{O}(\frac{n \cdot d}{m} m)$
- Xor über alle Output Vektoren ergibt MPFSS Vektor mit Länge n
  - Benötigt $\mathcal{O}(n \cdot d)$

Batch Codes: Parameterwahl

- Frei wählbare Parameter
  - Anzahl Buckets m
  - Dopplungsfaktor d

Batch Codes: Parameterwahl

- Frei wählbare Parameter
  - Anzahl Buckets m
  - Dopplungsfaktor d
- Boyle et al.(2018) schlagen Parameter vor

Batch Codes: Parameterwahl

- Frei wählbare Parameter
  - Anzahl Buckets m
  - Dopplungsfaktor d
- Boyle et al.(2018) schlagen Parameter vor
- Erfolgswahrscheinlichkeit $P = 1 - t^{-2d+2}$

  $\Rightarrow$ Parameter so, dass P maximal
- Probleme
  - Paper nicht eindeutig
  - Versteckte Konstanten

MPFSS mit Batch Codes: Parameterwahl

- Berechnung für $n = 1.000.000$ und $P = 1 - 10^{-25}$

Indices t	100	200	400	800	1600	5000
Dopplungsfaktor d	7,25	6,43	5,80	5,31	4,9	4,40
Buckets m (Wert $\cdot 10^6$)	0,13	0,58	2,41	9,53	35,94	292,54

MPFSS mit Batch Codes: Parameterwahl

- Berechnung für $n = 1.000.000$ und $P = 1 - 10^{-25}$
- Für jede DPF: Setup Kosten
  - $\Rightarrow$ Beachtlicher Overhead (wollen wir eigentlich minimieren)

Indices t	100	200	400	800	1600	5000
Dopplungsfaktor d	7,25	6,43	5,80	5,31	4,9	4,40
Buckets m (Wert $\cdot 10^6$)	0,13	0,58	2,41	9,53	35,94	292,54

Vergleich der Algorithmen

Vergleich

- MPFSS naive
  - DPF ausführen: $\mathcal{O}(n \cdot t)$
  - MPFSS Vektor erstellen

Vergleich

- MPFSS naive
  - DPF ausführen: $\mathcal{O}(n \cdot t)$
  - MPFSS Vektor erstellen
- MPFSS Batch Codes
  - Buckets erstellen
    - $\Rightarrow$ Vernachlässigbar

Vergleich

- MPFSS naive
  - DPF ausführen: $\mathcal{O}(n \cdot t)$
  - MPFSS Vektor erstellen
- MPFSS Batch Codes
  - Buckets erstellen
    - $\Rightarrow$ Vernachlässigbar
  - Zuordnung finden

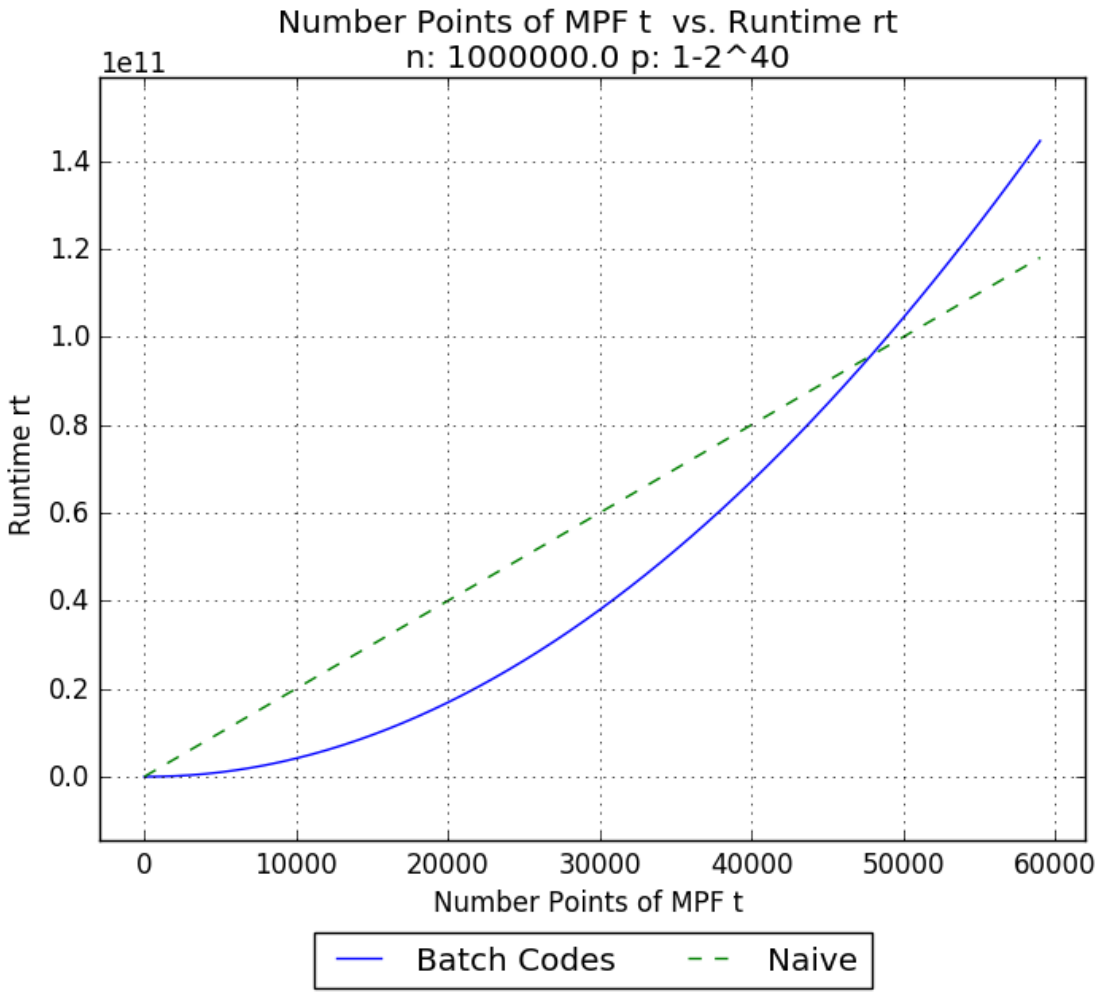
Vergleich

- MPFSS naive
  - DPF ausführen: $\mathcal{O}(n \cdot t)$
  - MPFSS Vektor erstellen
- MPFSS Batch Codes
  - Buckets erstellen
    - $\Rightarrow$ Vernachlässigbar
  - Zuordnung finden
  - DPF ausführen: $\mathcal{O}(\frac{n \cdot d}{m} m)$

Vergleich

- MPFSS naive
  - DPF ausführen: $\mathcal{O}(n \cdot t)$
  - MPFSS Vektor erstellen
- MPFSS Batch Codes
  - Buckets erstellen
    - $\Rightarrow$ Vernachlässigbar
  - Zuordnung finden
  - DPF ausführen: $\mathcal{O}(\frac{n \cdot d}{m} m)$
  - MPFSS Vektor erstellen

Vergleich


Number Points of MPF t vs. Runtime rt
n: 1000000.0 p: 1-2^40
1e11
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Runtime rt
0
10000
20000
30000
40000
50000
60000
Number Points of MPF t
Batch Codes
Naive

Ergebnis und Aussicht

- Einige Kosten und Schritte in Paper nicht erwähnt
- Versteckte Konstanten können Laufzeitabschätzung kaputt machen
- Man müsste bessere Parameter finden
- Laufzeitmessungen wären notwendig

Ergebnis und Aussicht

- Einige Kosten und Schritte in Paper nicht erwähnt
- Versteckte Konstanten können Laufzeitabschätzung kaputt machen
- Man müsste bessere Parameter finden
- Laufzeitmessungen wären notwendig
- Bessere Ansätze?
  - $\Rightarrow$ Hashing zum Erstellen der Buckets und für Zuordnung

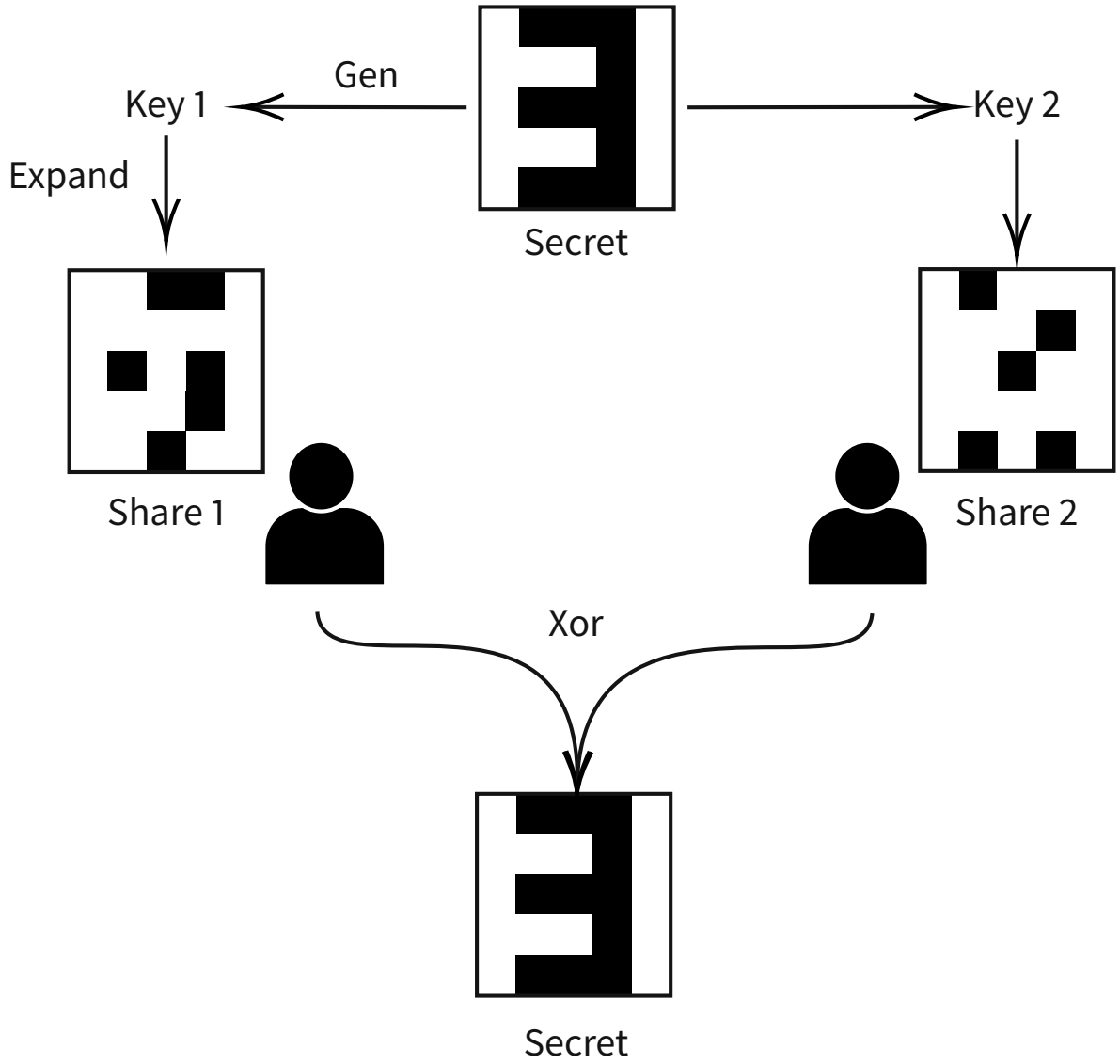
Danke für die Aufmerksamkeit.
Fragen?

Quellen

1. Boyle et al. Compressing vector OLE.*In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 896912. ACM, 2018.
2. Boyle et al. Function secret sharing: Improvements and extensions. *In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 12921303. ACM, 2016.
3. Doerner and Shelat. Scaling ORAM for secure computation. *In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 523535. ACM, 2017.
4. Zahur and Evans. Obliv-C: A Language for Extensible Data-Oblivious Computation. *IACR Cryptology ePrint Archive*, 2015.
5. Gilboa and Ishai. Distributed point functions and their applications. *In Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 640658. Springer, 2014.
6. Hopcroft and M Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225231, 1973.

Backup Slides

Was ist Secret Sharing?



Vergleich

- MPFSS naive
  - DPF ausführen: $\mathcal{O}(n \cdot t)$
  - MPFSS Vektor erstellen: $\mathcal{O}(n \cdot t)$
- MPFSS Batch Codes
  - Buckets erstellen: $\mathcal{O}(n \cdot t)$

    $\Rightarrow$ Vernachlässigbar
  - Zuordnung finden: $\mathcal{O}((m + 2t) \cdot \sqrt{m + t})$
  - DPF ausführen: $\mathcal{O}(\frac{n \cdot d}{m} m)$
  - MPFSS Vektor erstellen: $\mathcal{O}(n \cdot d)$