



Fakultät für  
**Mathematik und**  
**Informatik**

# Balloon Hashing: Eine gute Wahl für Passwort Hashing?

Henry Weckermann

## Agenda

- Einführung + Motivation
- Grundlagen
- Stand der Technik
- Balloon Hashing
- Diskussion
- Zusammenfassung und Ausblick
- Literatur

# Einführung

- Was ist Hashing
- Taxonomie und Abgrenzung
- Warum wird Hashing verwendet?

## Was ist Hashing?

- Hashing ist ein sehr breites Feld (siehe Taxonomie<sup>[1]</sup>)
- Gemeinsamkeiten aller Hashfunktionen <sup>[2]</sup>
  - **Kompressionseigenschaft:** Eingabe mit variabler Länge erzeugt eine feste Ausgabelänge
  - **„Ease of Computation“:** Unter jeder Eingabe  $x$ , soll die Berechnung „einfach“ sein
- Die Ausgabe einer Hashfunktion wird Hash, Hash-Wert, Digest, Fingerprint (kontextspez.) genannt<sup>[3]</sup>
- Eine Hashfunktion ist also eine mathematische **Funktion**  $h: X \rightarrow Y$  ( $|X| > |Y|, Y = \{0,1\}^n$ )

[1] Chi, Lianhua; Zhu, Xingquan (2018): Hashing Techniques. In ACM Comput. Surv. 50 (1), pp. 1–36. DOI: 10.1145/3047307.

[2] Menezes, Alfred J. (1996): Handbook of Applied Cryptography. Hoboken: CRC Press (Discrete Mathematics and Its Applications). Available online at <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=1605633>.

[3] van Tilborg, Henk C. A.; Jajodia, Sushil (Eds.) (2011): Encyclopedia of Cryptography and Security. 2. ed. Boston, MA: Springer US (SpringerLink Bücher).

## Was ist Hashing?

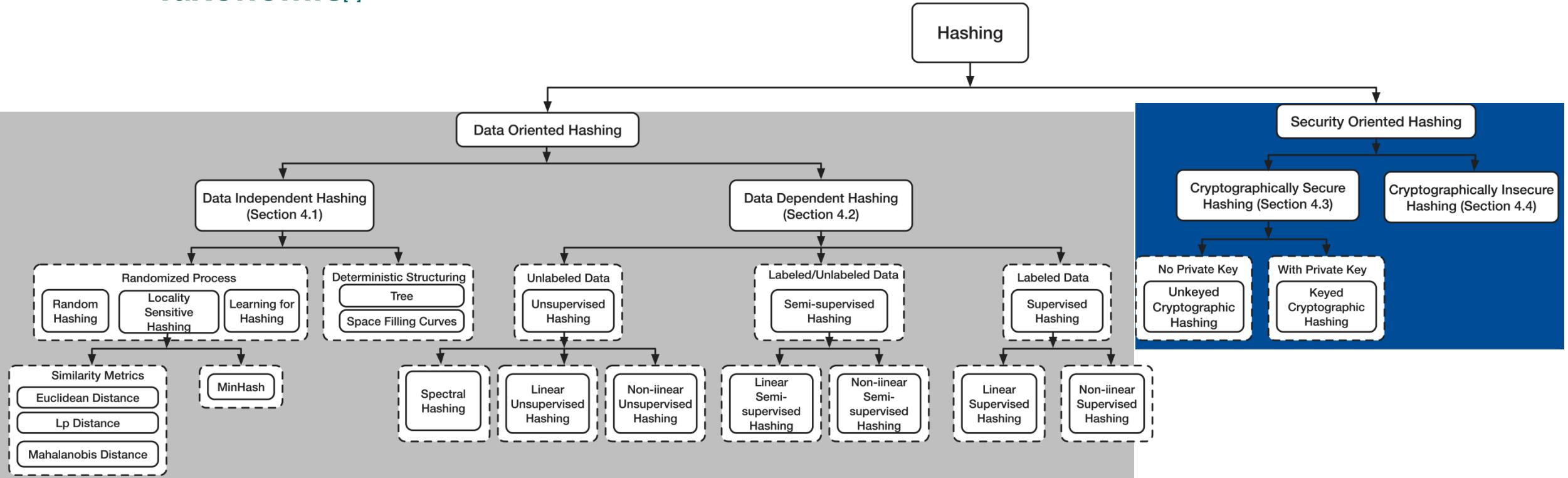
- **Anwendungsgebiete** [1][2]
  - **Kryptographie:** Digitale Signaturen, Public-Key Verfahren, Integritätsverifikation (Prüfsummen), Message Authentication (MACs), Passwortschutz
  - **Datenspeicherung:** Datenduplikate erkennen, Änderungen an Daten erkennen
  - **IDS:** Datenänderungen erkennen / Hashes von bekannter Malware erkennen
  - **Digitale Forensik:** Beweisen / Sicherstellen, dass Daten nicht verändert wurden (z.B. EnCase Format bzw. Expert Witness format<sup>[3]</sup>)
  - **Digitale Währung** und Blockchain
  - **Datenbanken:** Indexe, Datenstrukturen (Hashmap, Hash Tabelle)

[1] Aumasson, Jean-Philippe (2018): Serious Cryptography. A Practical Introduction to Modern Encryption. San Francisco, CA: No Starch Press Incorporated.

[2] Chi, Lianhua; Zhu, Xingquan (2018): Hashing Techniques. In ACM Comput. Surv. 50 (1), pp. 1–36. DOI: 10.1145/3047307.

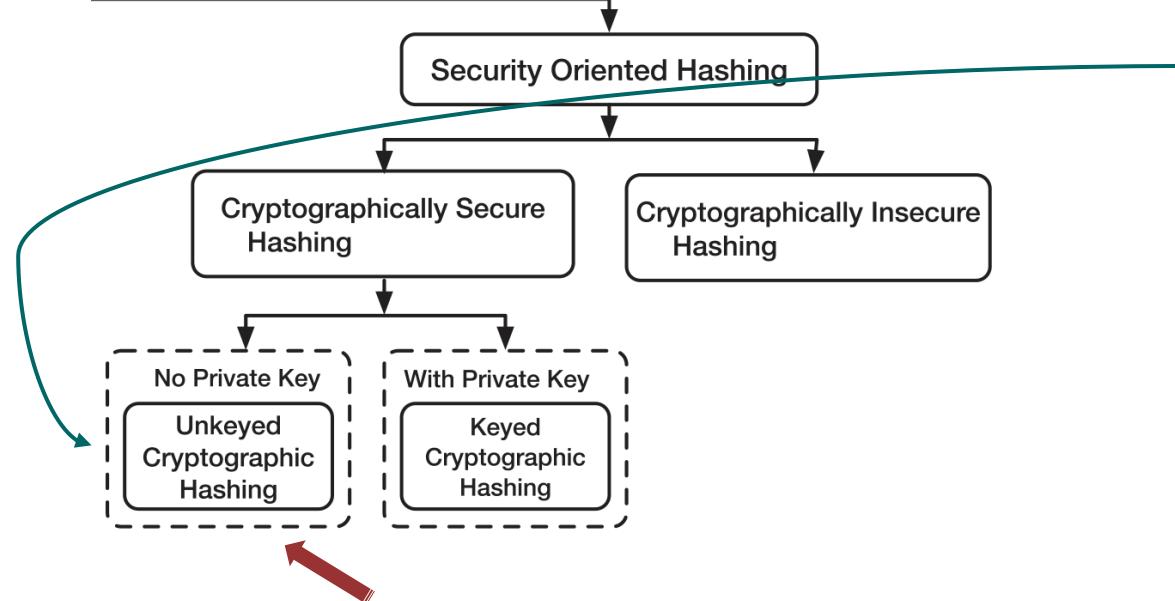
[3] [https://github.com/libyal/libewf/blob/main/documentation/Expert%20Witness%20Compression%20Format%20\(EWF2\).asciidoc](https://github.com/libyal/libewf/blob/main/documentation/Expert%20Witness%20Compression%20Format%20(EWF2).asciidoc)

## Taxonomie<sup>[1]</sup>

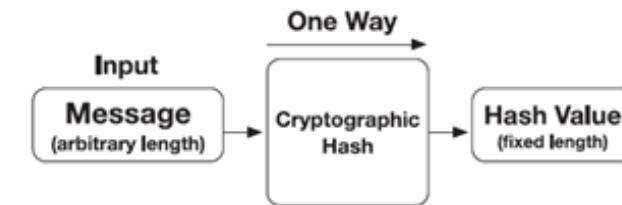


[1] Chi, Lianhua; Zhu, Xingquan (2018): Hashing Techniques. In ACM Comput. Surv. 50 (1), pp. 1–36. DOI: 10.1145/3047307.

## Taxonomie<sup>[1]</sup>



Im weiteren Verlauf meine ich mit „Hashfunktionen“ diese Gruppe



## Beispiele<sup>[1]</sup>

MD2/4/5 (message digest),  
 SHA 1/3/244/256/384/512 (Secure Hash Algorithms),  
 FSB (fast syndrome-based),  
 ECOH (elliptic curve only hash),  
 RIPEMD/-128/-160/-320,  
 HAVAL  
 BLAKE2s, BLAKE2b

[1] Chi, Lianhua; Zhu, Xingquan (2018): Hashing Techniques. In ACM Comput. Surv. 50 (1), pp. 1–36. DOI: 10.1145/3047307.

## Warum Hashing?

- Hashing ist Teil vieler (täglicher) digitaler Aufgaben

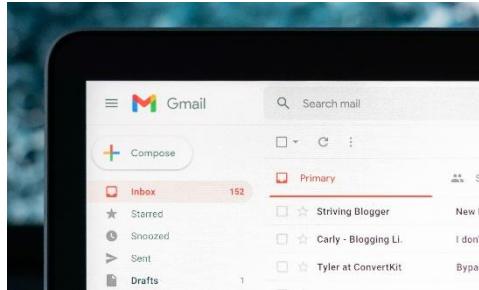


Image for your Device	sha256sum
8dev_carambola2-initramfs-kernel.bin	a93da93fd7601566519c1a236dae87020b6c747862add085f37659b38b569488
8dev_carambola2-squashfs-sysupgrade.bin	138450edc9a69d24ae122967c4b991f9af99c6cf85279767e612c4a3f8bfce30
8dev_lima-initramfs-kernel.bin	fb7d1621404010c650fd951612360663fb68f936df5efbf54aa97e4ce7795c7
8dev_lima-squashfs-sysupgrade.bin	08dd03435b3bb9cec77e89263e8803c3e3b5bc4c77c2088eeb33ad1f08466a28
adtran_bsap1800-v2-initramfs-kernel.bin	5cf29c5c4e7ce6a1fddbaee69d9b84b6c53f71bfd06b8f6c7e0430a42968a223
adtran_bsap1800-v2-squashfs-kernel.bin	b285deb4852a03f2653bbf1d884a97d4ded2e0d486a2fbe15e1199b8544fe133
adtran_bsap1800-v2-squashfs-rootfs.bin	0dda9306057348cf946ae63030e4daeb2fcfd8ea6b026a6891ec2789a9c6f95e

[1] Bildnachweis: unsplash.com [https://unsplash.com/de/fotos/D2TZ-ashGzc] und [https://unsplash.com/de/fotos/Uae7ouMw91A] (lizenzfrei)

[2] Bildnachweis: freepik.com [https://de.freepik.com/download-file/4393734] (lizenzfrei)

## Warum Hashing?

- Menschen können sich nicht gut einzigartige (und sichere) Passwörter ausdenken
  - Einige Passwörter werden daher sehr häufig verwendet (z.B. „123456“ oder „password“)<sup>[1]</sup>
- Wenn eine Nutzerdatenbank gehackt wird, ist also nicht nur der Account des Nutzers auf der gehackten Seite betroffen<sup>[2]</sup>
- **Aufgabe:** Nutzer sicher authentisieren - basierend auf einem Geheimnis, welches sie kennen

Weit verbreitete **Lösung**: Passwort Hashing

[1] <https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt> [Abgerufen: 16.01.2023]

[2] Das, Anupam; Bonneau, Joseph; Caesar, Matthew; Borisov, Nikita; Wang, XiaoFeng (2014): The Tangled Web of Password Reuse. In : Proceedings 2014 Network and Distributed System Security Symposium. Reston, VA. Reston, VA: Internet Society.

## Warum Hashing?

### **Art. 32 (Sicherheit der Verarbeitung) Abs. 1 DS-GVO**

Unter Berücksichtigung des Stands der Technik, der Implementierungskosten und der Art, des Umfangs, der Umstände und der Zwecke der Verarbeitung sowie der unterschiedlichen Eintrittswahrscheinlichkeit und Schwere des Risikos für die Rechte und Freiheiten natürlicher Personen **treffen der Verantwortliche und der Auftragsverarbeiter geeignete technische und organisatorische Maßnahmen**, um ein dem Risiko **angemessenes Schutzniveau zu gewährleisten**; diese Maßnahmen schließen gegebenenfalls unter anderem Folgendes ein:

- a) die **Pseudonymisierung** und **Verschlüsselung personenbezogener Daten**;
- b) die Fähigkeit, die **Vertraulichkeit, Integrität**, Verfügbarkeit und Belastbarkeit der Systeme und Dienste im Zusammenhang mit der Verarbeitung auf Dauer sicherzustellen;
- c) ...

[1] Europäischen Union: Datenschutz-Grundverordnung, DSGVO, revised 2021. In : Amtsblatt der Europäischen Union (ABl. L 119, 04.05.2016). Available online at <https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32016R0679> [Abgerufen: 08.01.2023]

## Warum Hashing?

### Art. 32 (Sicherheit der Verarbeitung) Abs. 2 DS-GVO

Bei der Beurteilung des angemessenen Schutzniveaus sind insbesondere die Risiken zu berücksichtigen, die mit der Verarbeitung verbunden sind, insbesondere durch – ob unbeabsichtigt oder unrechtmäßig – Vernichtung, Verlust, Veränderung oder **unbefugte Offenlegung** von beziehungsweise **unbefugten Zugang** zu personenbezogenen Daten, die **übermittelt, gespeichert** oder auf **andere Weise verarbeitet** wurden.

[1] Europäischen Union: Datenschutz-Grundverordnung, DSGVO, revised 2021. In : Amtsblatt der Europäischen Union (ABl. L 119, 04.05.2016). Available online at <https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32016R0679> [Abgerufen: 08.01.2023]

# Grundlagen

- Was ist kryptographisch-sicheres-Hashing?
- Salt
- Anforderungen an moderne Passwort-Hashing-Algorithmen

# Was ist Kryptographisch-sicheres-Hashing?

## Klassische Anforderungen<sup>[1][2][3]</sup>

### Preimage Resistance

- „Einweg Funktion“
- Wenn man einen Hashwert  $y$  hat, soll es rechnerisch nicht möglich sein, die Eingabe  $x$  zu berechnen bzw. eindeutig zu bestimmen

### Second Preimage Resistance

- Es soll unmöglich sein, eine zweite Eingabe zu finden, die den gleichen Hashwert wie eine andere beliebige Eingabe liefert

### Collision resistance

- Es soll nicht möglich sein zwei ungleiche Eingaben zu finden, welche den gleichen Hashwert ergeben

folgt nicht aus

folgt aus

[1] Rogaway, Phillip; Shrimpton, Thomas (2004): Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor et al. (Eds.): Fast Software Encryption, vol. 3017. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 371–388

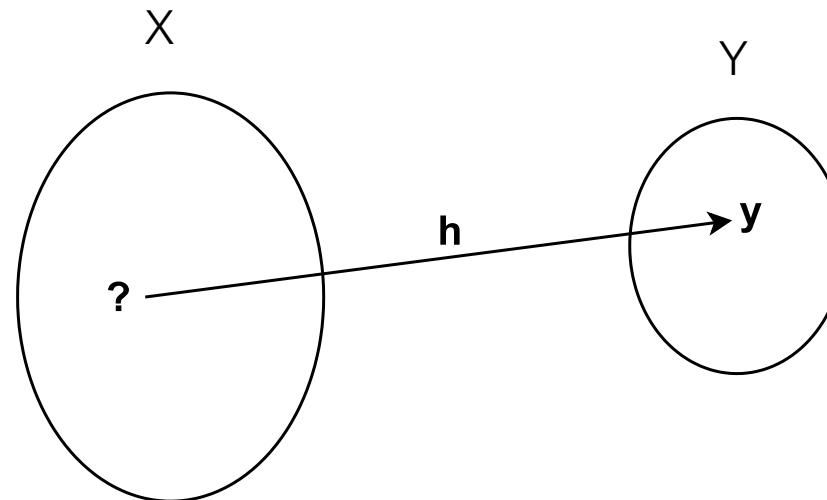
[2] van Tilborg, Henk C. A.; Jajodia, Sushil (Eds.) (2011): Encyclopedia of Cryptography and Security. 2. ed. Boston, MA: Springer US (SpringerLink Bücher).

[3] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

## Was ist Kryptographisch-sicheres-Hashing?

### Preimage Resistenz<sup>[1][2]</sup>

Für einen gegebenen Hashwert  $y \in Y$  soll ein Urbild  $x \in X$  praktisch nicht berechenbar sein



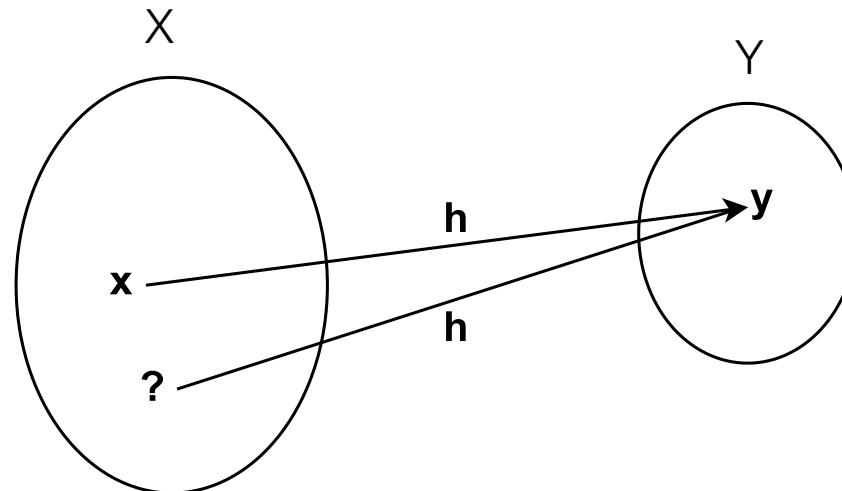
[1] Rogaway, Phillip; Shrimpton, Thomas (2004): Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor et al. (Eds.): Fast Software Encryption, vol. 3017. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 371–388

[2] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

## Was ist Kryptographisch-sicheres-Hashing?

### Second Preimage Resistenz<sup>[1][2]</sup>

Für eine gegebene Nachricht  $x \in X$  soll eine zweite Nachricht  $x' \neq x$  mit  $h(x') = h(x)$  praktisch nicht berechenbar sein



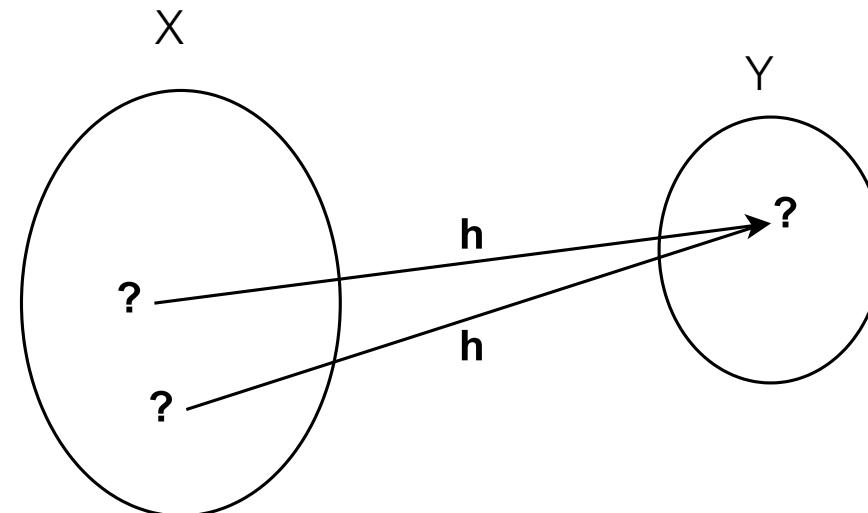
[1] Rogaway, Phillip; Shrimpton, Thomas (2004): Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor et al. (Eds.): Fast Software Encryption, vol. 3017. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 371–388

[2] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

## Was ist Kryptographisch-sicheres-Hashing?

### Kollisions-Resistenz<sup>[1][2]</sup>

Es soll praktisch nicht möglich sein, zwei Nachrichten  $x, x'$  zu finden mit  $x' \neq x$  und  $h(x') = h(x)$



[1] Rogaway, Phillip; Shrimpton, Thomas (2004): Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor et al. (Eds.): Fast Software Encryption, vol. 3017. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 371–388

[2] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

## Was ist Kryptographisch-sicheres-Hashing?

### Am Rande

Für jede Hashfunktion gibt es

- Einen Preimage Angriff in  $O(2^{n-1})$  [Bruteforce]
- Einen Second Preimage Angriff in  $O(2^{n-1})$  [Bruteforce]
- Einen Kollisionsangriff in  $O(2^{n/2})$  [Geburtstagsparadoxon]

der jeweils mit Wahrscheinlichkeit 0,5 erfolgreich ist [1][2]

[1] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

[2] Smart, Nigel P. (2003): Cryptography. An introduction. London: McGraw-Hill (McGraw-Hill education).

## Salt<sub>[1][2]</sub>

- Ein zufälliger (Bit-)String, der zusammen mit den eigentlichen Daten gehashed wird

Gegeben einer Nachricht  $m$ , einem Salt  $s$  und einer Hashfunktion  $h: X \rightarrow Y$  berechnet man statt

$$h(m) \text{ nun } h(m \parallel s)$$

- Das Salt sollte für jede Nachricht **einzigartig** sein und wird zusammen mit dem Hash gespeichert
  - Kann öffentlich oder privat sein

[1] van Tilborg, Henk C. A.; Jajodia, Sushil (Eds.) (2011): Encyclopedia of Cryptography and Security. 2. ed. Boston, MA: Springer US (SpringerLink Bücher).

[2] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

## Salt<sub>[1][2]</sub>

### Vorteile:

- Erschwert sog. **Rainbow-Table Angriffe**
- Der akute Arbeitsaufwand für den Angreifer wird nicht erhöht → **aber:** der Angreifer kann **keine Vorberechnungen** durchführen
- Der Hash von zwei Nutzern mit dem gleichen Passwort ist nicht mehr gleich → dies ist also nicht mehr trivial zu erkennen

[1] van Tilborg, Henk C. A.; Jajodia, Sushil (Eds.) (2011): Encyclopedia of Cryptography and Security. 2. ed. Boston, MA: Springer US (SpringerLink Bücher).

[2] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

# Anforderungen an moderne Passwort-Hashing-Algorithmen

## Memory Hardness

- Ein Angreifer soll es nicht substanzial einfacher haben einen Hash zu berechnen, als ein Authentifikationsserver (z.B. einer Webanwendung bei der sich ein Nutzer anmelden möchte) [1][2][3]
- **Aber:** Angreifer haben Zugriff auf spezielle Hardware → ASICs (Application-specific integrated circuit), ASIP, FPGA etc. [4][5]

[1] Alwen, Joel; Gazi, Peter; Kamath, Chethan; Klein, Karen; Osang, Georg; Pietrzak, Krzysztof et al. (2018): On the Memory-Hardness of Data-Independent Password-Hashing Functions. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier Lopez, Taesoo Kim (Eds.): Proceedings of the 2018 on Asia Conference on Computer and Communications Security. ASIA CCS '18: ACM Asia Conference on Computer and Communications Security. Incheon Republic of Korea, 04 06 2018 04 06 2018. New York, NY, USA: ACM, pp. 51–65.

[2] Niels Kornerup (2022): Memory hard functions and persistent memory hardness. Available online at <https://www.cs.utexas.edu/~dwu4/courses/sp22/static/projects/Kornerup.pdf>.

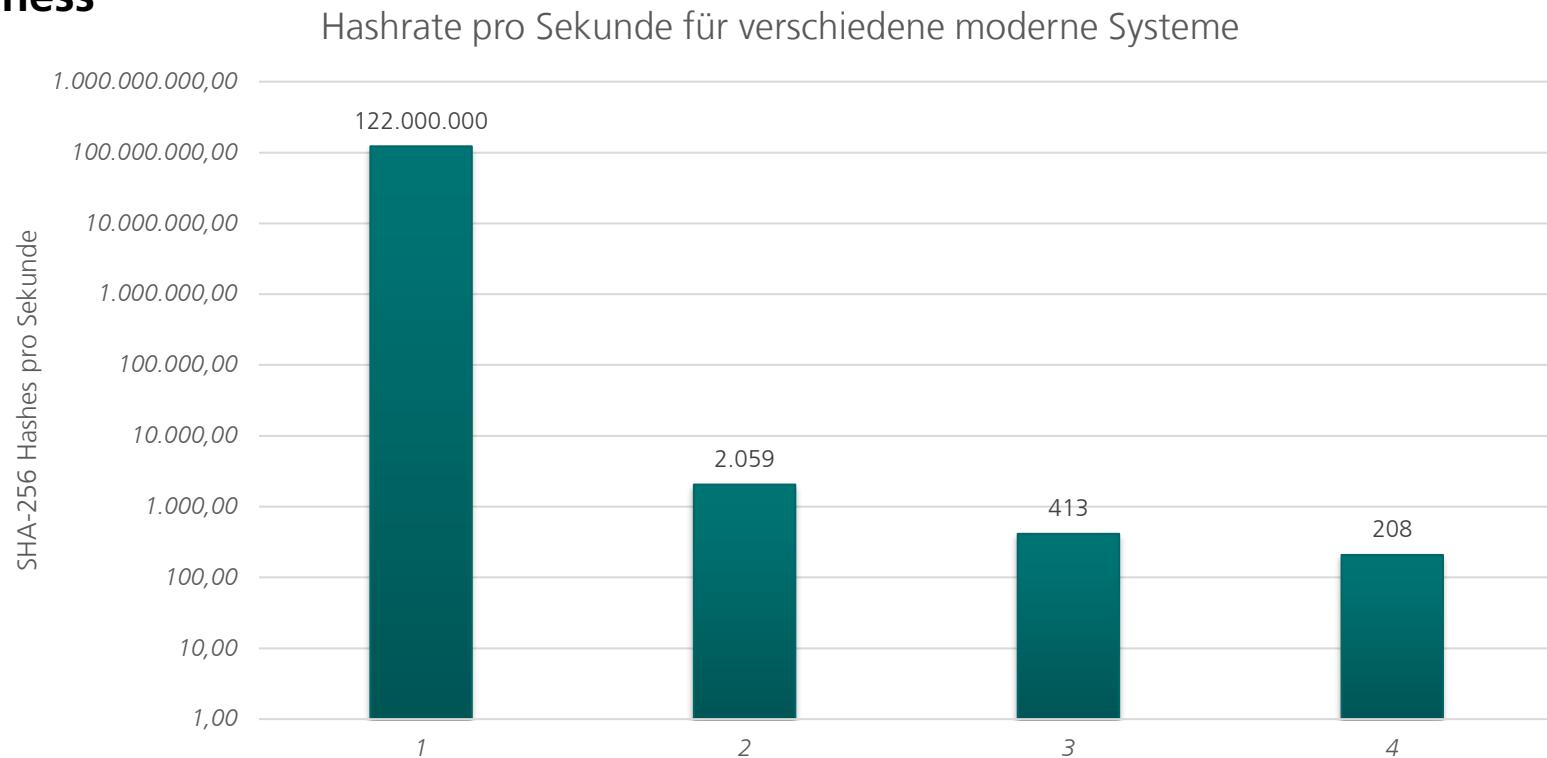
[3] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

[4] Electronic Frontier Foundation (1998): Cracking DES. Secrets of encryption research, wiretap politics & chip design. 1. ed. Sebastopol, Calif.: O'Reilly.

[5] Einspruch, Norman G.; Hilbert, Jeffrey L. (2012): Application specific integrated circuit (ASIC) technology: Academic Press (VLSI electronics, v. 23).

## Anforderungen an moderne Passwort-Hashing-Algorithmen

### Memory Hardness

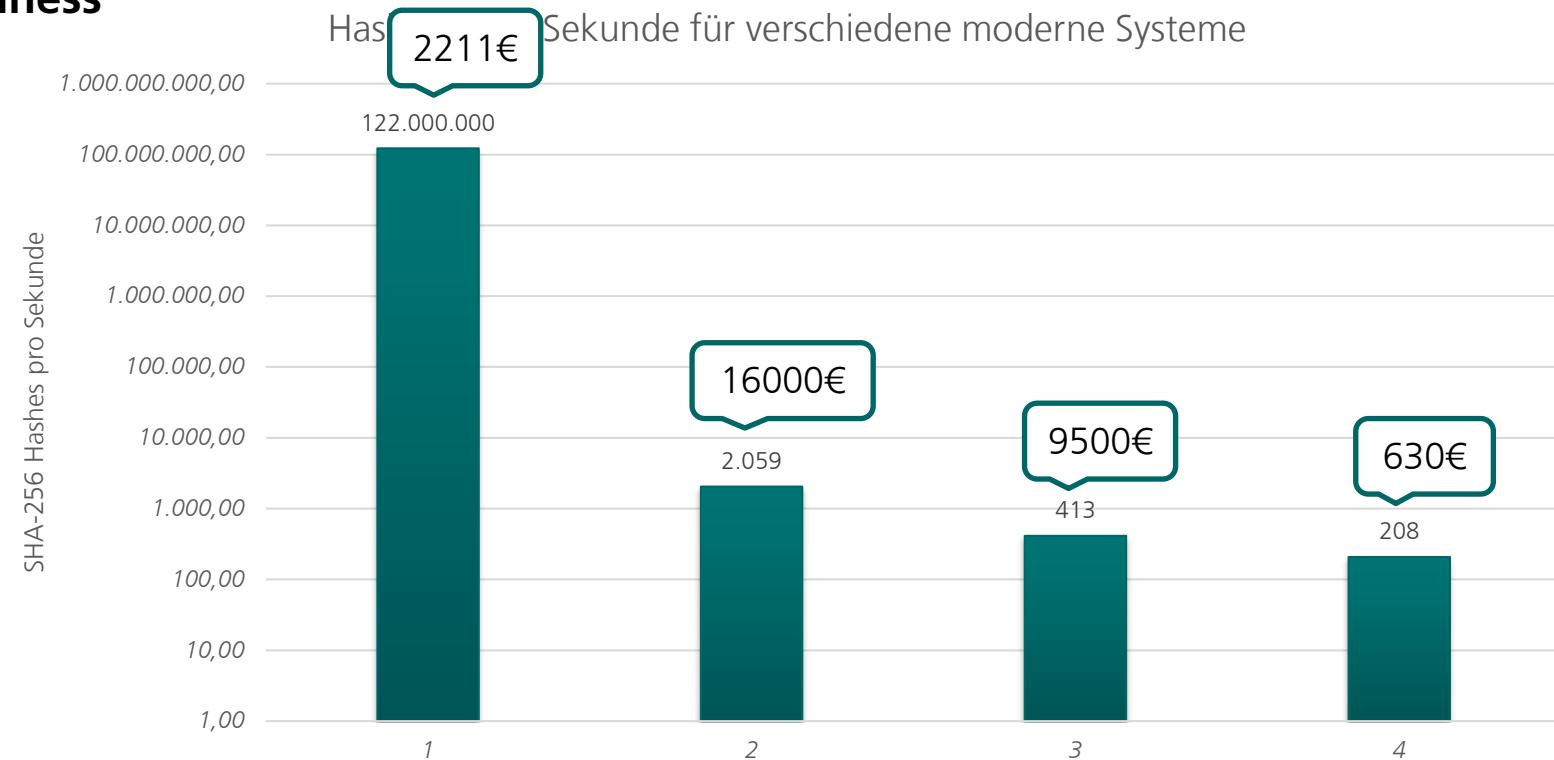


[1] Daten Antminer: <https://m.bitmain.com/product/detail?pid=00020230108213609854b369SGwl0654> [Abgerufen: 11.01.2023]

[2] Daten CPUs: <https://openbenchmarking.org/test/pts/cpuminer-opt&eval=3b4d3232d3f686b1f35d9706f47604c124cb47e6> [Abgerufen: 11.01.2023]

## Anforderungen an moderne Passwort-Hashing-Algorithmen

### Memory Hardness



[1] Daten Antminer: <https://m.bitmain.com/product/detail?pid=00020230108213609854b369SGwl0654> [Abgerufen: 11.01.2023]

[2] Daten CPUs: <https://openbenchmarking.org/test/pts/cpuminer-opt&eval=3b4d3232d3f686b1f35d9706f47604c124cb47e6> [Abgerufen: 11.01.2023]

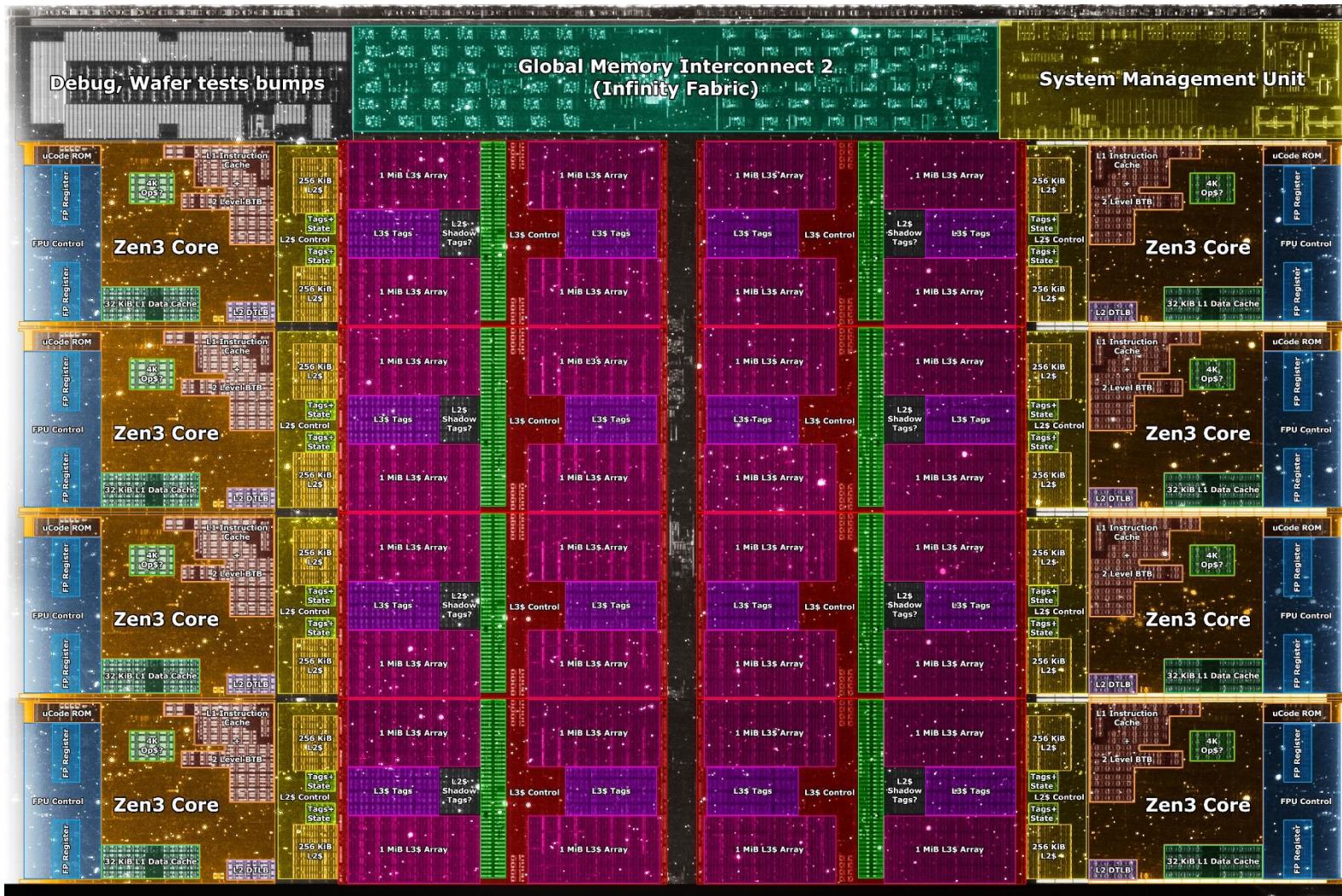
# Anforderungen an moderne Passwort-Hashing-Algorithmen

## Memory Hardness

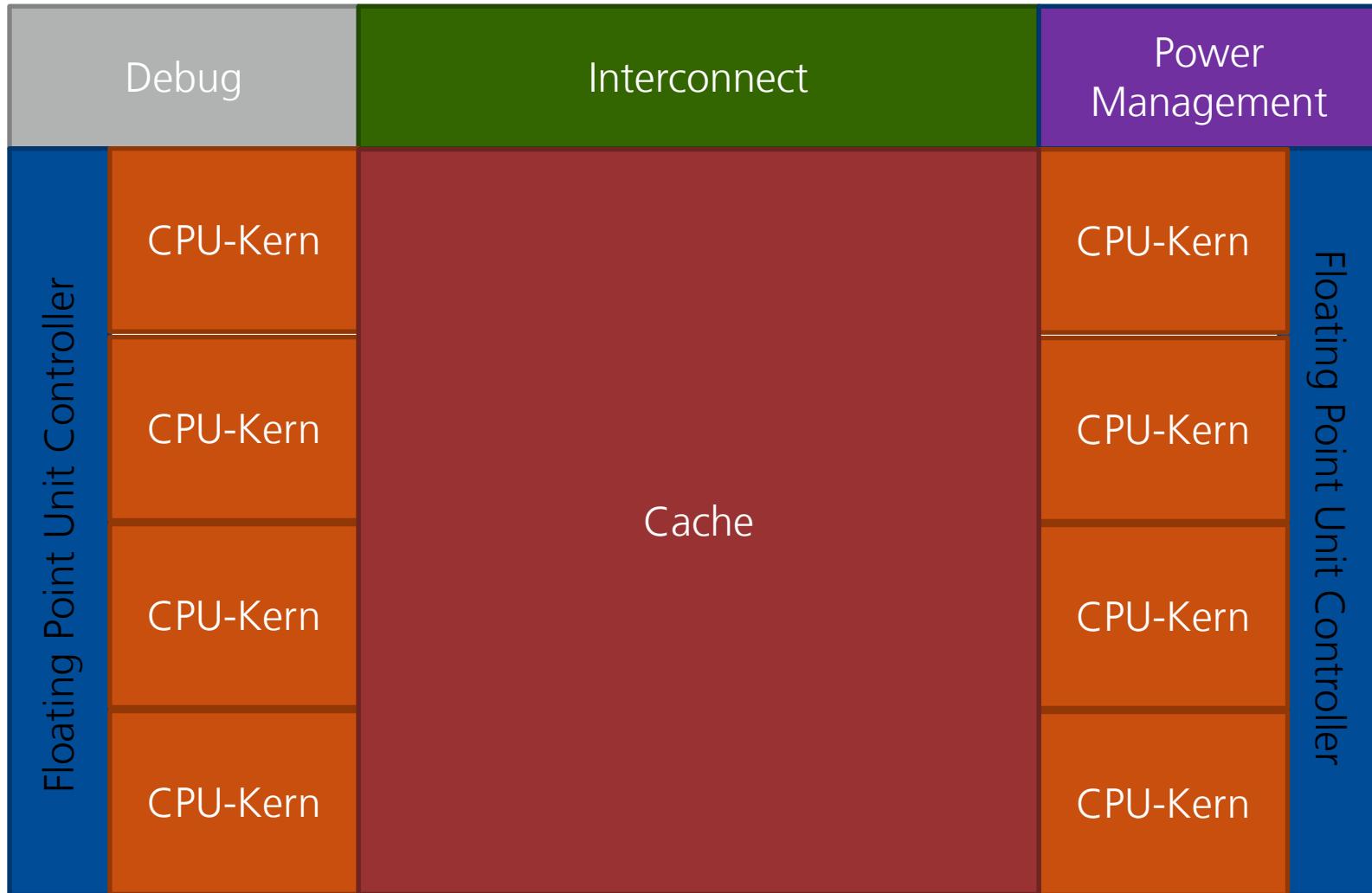
- Ein Angreifer soll es nicht substanzial einfacher haben einen Hash zu berechnen, als ein Authentifikationsserver (z.B. einer Webanwendung bei der sich ein Nutzer anmelden möchte)
  - Angreifer haben Zugriff auf spezielle Hardware → ASICs (Application-specific integrated circuit), ASIP, FPGA etc.
- Handelsübliche CPUs müssen viele verschiedene Aufgaben übernehmen<sup>[1]</sup>
- ASICs sind genau für **eine** Aufgabe gebaut<sup>[2]</sup>

[1] Hennessy, John L.; Patterson, David A. (2003): Computer architecture. A quantitative approach. 3rd ed. San Francisco, CA: Morgan Kaufmann Publishers (Morgan Kaufmann series in computer architecture and design).

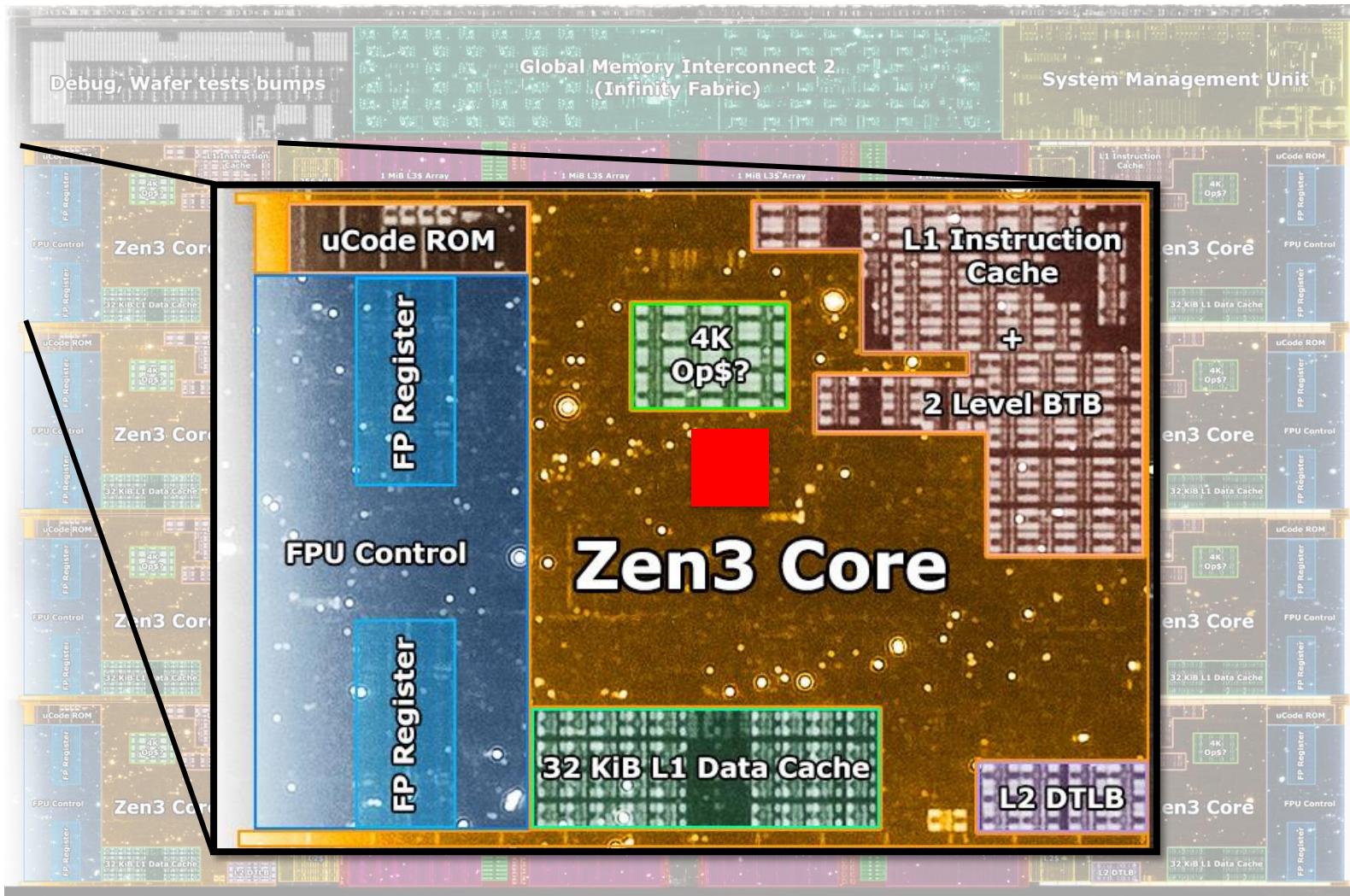
[2] Einspruch, Norman G.; Hilbert, Jeffrey L. (2012): Application specific integrated circuit (ASIC) technology: Academic Press (VLSI electronics, v. 23).



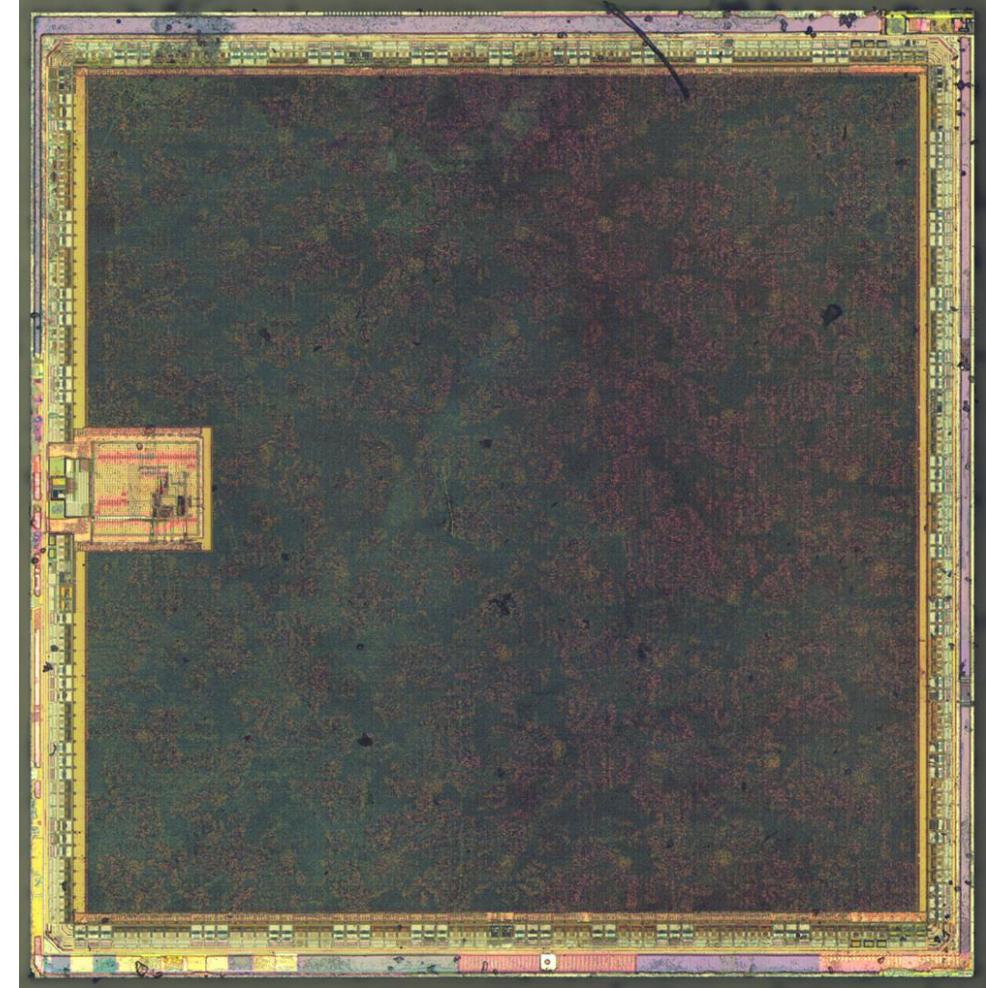
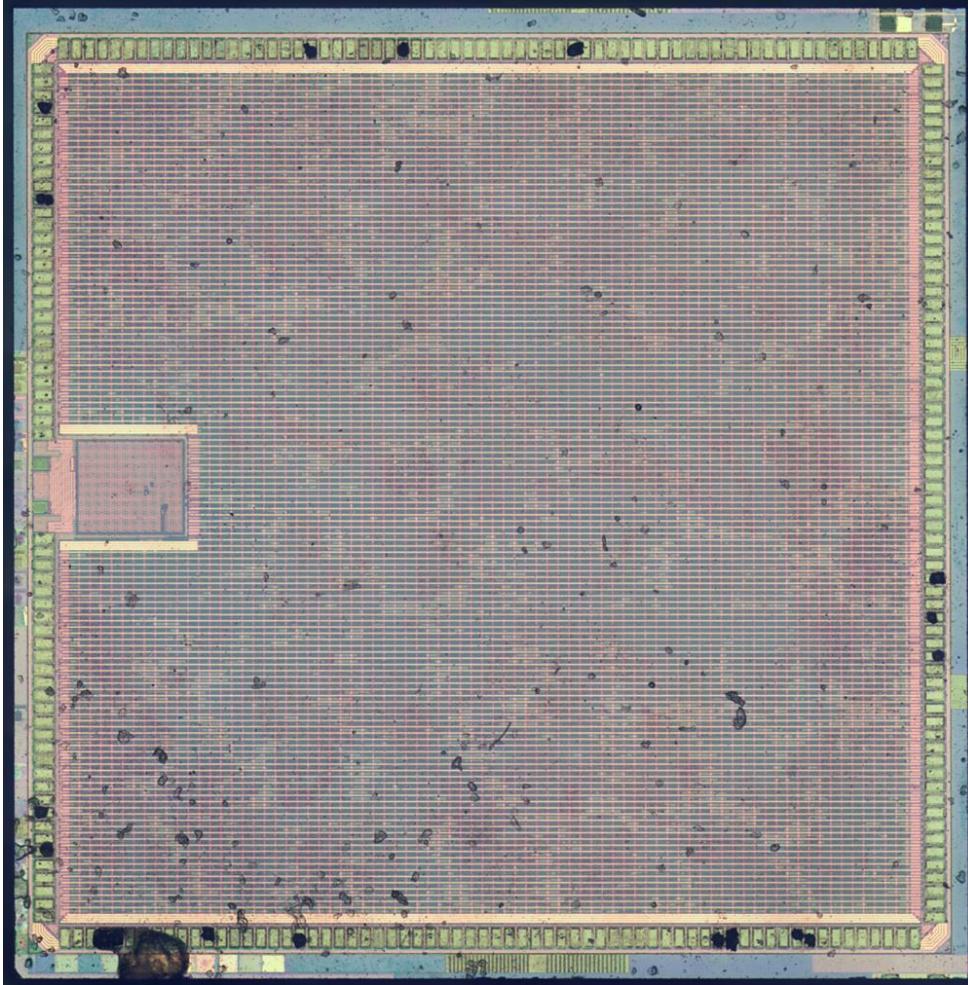
[1] [https://www.reddit.com/r/AmI/comments/jqjg8e/quick\\_zen3\\_die\\_shot\\_annotations\\_die\\_shot\\_from/](https://www.reddit.com/r/AmI/comments/jqjg8e/quick_zen3_die_shot_annotations_die_shot_from/) [Abgerufen: 11.01.2023]



[1] [https://www.reddit.com/r/Amd/comments/jqjg8e/quick\\_zen3\\_die\\_shot\\_annotations\\_die\\_shot\\_from/](https://www.reddit.com/r/Amd/comments/jqjg8e/quick_zen3_die_shot_annotations_die_shot_from/) [Abgerufen: 11.01.2023]



[1] [https://www.reddit.com/r/AMD/comments/jqjg8e/quick\\_zen3\\_die\\_shot\\_annotations\\_die\\_shot\\_from/](https://www.reddit.com/r/AMD/comments/jqjg8e/quick_zen3_die_shot_annotations_die_shot_from/) [Abgerufen: 11.01.2023]



[1] <https://zeptobars.com/ru/read/avalon-bitcoin-mining-unit-rig> [Abgerufen: 11.01.2023]

## Anforderungen an moderne Passwort-Hashing-Algorithmen

### Memory Hardness

- Ein Angreifer soll es nicht substanzial einfacher haben einen Hash zu berechnen, als ein Authentifikationsserver (z.B. einer Webanwendung bei der sich ein Nutzer anmelden möchte)
  - Angreifer haben Zugriff auf spezielle Hardware → ASICs (Application-specific integrated circuit), ASIP, FPGA etc.
  - Handelsübliche CPUs müssen viele verschiedene Aufgaben übernehmen
  - ASICs sind genau für **eine** Aufgabe gebaut
  - Kryptographische Hashfunktionen (SHA-2 etc.) zu iterieren (wie bei bcrypt<sub>[1]</sub> oder PBKDF2<sub>[2]</sub>) erhöht die Kosten für Verteidiger und Angreifer, aber der relative Vorteil bleibt stehts bestehen!

[1] N Provos, D. Mazieres (1999): Bcrypt algorithm: USENIX. Available online at [https://www.usenix.org/events/usenix99/provos/provos\\_html/node5.html](https://www.usenix.org/events/usenix99/provos/provos_html/node5.html).

[2] Kaliski, B. (2000): PKCS #5: Password-Based Cryptography Specification Version 2.0.

## Anforderungen an moderne Passwort-Hashing-Algorithmen

### Memory Hard Hashfunktion

- Nutzen viel Speicherplatz während der Ausführung = Der Chip muss über Caches verfügen
- Angreifer mit spezifischer Hardware haben weniger Vorteile

### Vorgehen<sup>[1]</sup>

- 1. Fill** Den Puffer mit pseudo-zufälligen Zahlen füllen
- 2. Mix** Pseudozufällige Blöcke aus dem Puffer lesen und in diesen schreiben
- 3. Extract** Die Ausgabe der Funktion aus dem Inhalt des Puffers extrahieren

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Anforderungen an moderne Passwort-Hashing-Algorithmen

### Memory Hard Hashfunktion (im Random-Oracle Model)<sup>[1][2]</sup>

$$S \cdot T \in \Omega(n^2)$$

*n: Security parameter*

*S: Space parameter (Buffer size)*

*T: Time parameter (Number of rounds)*

**Erinnerung Landau Symbole:**  
asymptotische untere Schranke in  
der Komplexitätstheorie

- Die Formel soll unabhängig einer bestimmten Berechnungsstrategie gelten
- Ein Angreifer muss also entweder viel Platz (Cache / Speicher) oder viel Zeit verwenden  
→ ASICs lohnen sich nicht

**Achtung:** Parallelisierte Angriffe bzw. Angriffe mit mehreren Instanzen (Geräten) werden nicht betrachtet

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

[2] Colin Percival (2009): STRONGER KEY DERIVATION VIA SEQUENTIAL MEMORY-HARD FUNCTIONS. Available online at <https://www.semanticscholar.org/paper/STRONGER-KEY-DERIVATION-VIA-SEQUENTIAL-MEMORY-HARD-Percival/7c74956d21f0466c9771bb583e2fdf854c2aedbf>.

## Anforderungen an moderne Passwort-Hashing-Algorithmen

### Random Oracle Model<sup>[1][2]</sup>

- Stellt eine Brücke zwischen Theorie und Praxis für formale Beweise dar
- Im Random Oracle Model wird eine Hashfunktion durch eine **Blackbox** ersetzt
  - In der Blackbox wird aus einem arbiträren Input eine Folge aus echten Zufallszahlen erstellt
    - Diese sollte natürlich so lang sein wie die Ausgabe des eigentlichen Hashing-Algorithmus
  - In der Blackbox wird in einer Tabelle die Eingabe und die Ausgabe gespeichert
    - Wenn eine Eingabe erneut vorkommt, wird der gespeicherte Ausgabewert zurückgegeben
- Alle Parteien (normale Nutzer und Angreifer) müssen das Orakel nutzen

[1] van Tilborg, Henk C. A.; Jajodia, Sushil (Eds.) (2011): Encyclopedia of Cryptography and Security. 2. ed. Boston, MA: Springer US (SpringerLink Bücher).

[2] Bellare, Mihir; Rogaway, Phillip (1993): Random oracles are practical. In Dorothy Denning, Ray Pyle, Ravi Ganesan, Ravi Sandhu, Victoria Ashby (Eds.): Proceedings of the 1st ACM conference on Computer and communications security - CCS '93. the 1st ACM conference. Fairfax, Virginia, United States, 03.11.1993 - 05.11.1993. New York, New York, USA: ACM Press, pp. 62–73.

# Anforderungen an moderne Passwort-Hashing-Algorithmen

## Password Independent Access Pattern

- **Definition:** Das Speicherzugriffsmuster eines Hashing-Algorithmus ist unabhängig von der Eingabe<sup>[1]</sup>
  
- Resistenz gegen Seitenkanalangriffe
- Es sollen keine Informationen über das Urbild (z.B. ein Passwort) durch den Cache oder andere Seitenkanäle öffentlich werden <sup>[1][2]</sup>
  - z.B. sehr wichtig im Cloud Computing Umfeld <sup>[3]</sup>

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

[2] Bonneau, Joseph; Mironov, Ilya (2006): Cache-Collision Timing Attacks Against AES. In Louis Goubin, Mitsuru Matsui (Eds.): Cryptographic hardware and embedded systems - CHES 2006. 8th international workshop, Yokohama, Japan, October 10-13, 2006 ; proceedings, vol. 4249. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 4249), pp. 201–215.

[3] Tromer, Eran; Osvik, Dag Arne; Shamir, Adi (2010): Efficient Cache Attacks on AES, and Countermeasures. In J Cryptol 23 (1), pp. 37–71. DOI: 10.1007/s00145-009-9049-y.

[4] Ristenpart, Thomas; Tromer, Eran; Shacham, Hovav; Savage, Stefan (2009): Hey, you, get off of my cloud. In Ehab Al-Shaer, Somesh Jha, Angelos D. Keromytis (Eds.): Proceedings of the 16th ACM conference on Computer and communications security. CCS '09: 16th ACM Conference on Computer and Communications Security 2009. Chicago Illinois USA, 09 11 2009 13 11 2009. New York, NY, USA: ACM, pp. 199–212.

# Balloon Hashing

- Einführung
- Verwendete Syntax
- Implementierung und Funktionsweise
- Parallelisierte Anwendung
- Kollisionsresistenz, Preimage, und 2nd-Preimage Resistenz von Balloon
- Erfüllung der Sicherheitsziele

## Einführung

- Eine „Key Derivation Function“ von Dan Boneh, Henry Corrigan-Gibbs und Stuart Schechter (2016)<sup>[1]</sup>
- „Balloon“ → weil der Speicher „aufgeblasen“ wird (Schritt 1: Fill)
- Der Algorithmus beschreibt im Prinzip einen Betriebsmodus für eine darunterliegende Hashfunktion wie SHA-3
  - Vergleichbar mit den Modi bei Verschlüsselungsalgorithmen → AES-ECB / AES-CBC / AES-CFB<sup>[2]</sup> etc.
- **Ziele**<sup>[1]</sup>
  - Beweisbar Memory Hard (im Random Oracle Model)
  - Passwort-unabhängiges Speicherzugriffsmuster
  - Zu anderen modernen Passwort-Hashing-Funktionen vergleichbare Performance

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

[2] Menezes, Alfred J. (1996): Handbook of Applied Cryptography. Hoboken: CRC Press (Discrete Mathematics and Its Applications). Available online at <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=1605633>.

## Verwendete Syntax<sup>[1]</sup>

### Eingabeparameter:

	Kurzbeschreibung	Variable im Beweis   Pseudocode		Beschreibung
<b>Passwort</b>		<i>passwd</i>   <i>password</i>		Ein Passwort bzw. Eingabedaten
<b>Salt</b>		<i>salt</i>		Ein mit ausreichender Zufälligkeit generierter Salt
<b>Time Parameter</b>	Anzahl der Iterationen	<i>r</i>	<i>R</i>	Je größer dieser Parameter desto länger dauert die Berechnung. Passt die „Kosten“ der Berechnung an, ohne den Speicherbedarf zu erhöhen
<b>Space Parameter</b>	Puffergröße	<i>n</i>	<i>N</i>	Anzahl von Speicherblöcken die die Funktion während der Berechnung benötigt. Mit <i>n</i> oder mehr Speicher sollte die Funktion einfach zu berechnen sein, sonst schwer

**Ausgabe:** Bitstring mit fester Länge → abhängig vom verwendeten Hashing-Algorithmus

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

```
● ● ●  
  
Balloon(password, salt, N = space_cost, R = num_rounds):  
  
    delta = 3      // A security parameter  
    B = Buffer[N] // A buffer of N blocks  
  
    // Step 1: Fill the buffer  
    B[1] ← Hash(password, salt)  
    for i = 2, ..., N:  
        B[i] ← Hash(B[i-1])  
  
    // Step 2: Mix buffer  
    for r = 1, ..., R:  
        for i = 1, ..., N:  
            // Chosen pseudorandomly from salt  
            (v_1, ..., v_delta) ← Hash(salt, r, i)  
            B[i] = Hash(B[i-1 mod N], B[i], B[v_1], ..., B[v_delta])  
  
    // Step 3: Extract  
    return B[N]
```

Mit Hash(...) ist hier eine konventionelle Hash-Funktion gemeint (z.B. SHA-256)

# Schritt 1: Den Puffer füllen

Salt      Passwort

```
Balloon(password, salt, N = space_cost, R = num_rounds):  
    delta = 3      // A security parameter  
    B = Buffer[N] // A buffer of N blocks  
  
    // Step 1: Fill the buffer  
    B[1] ← Hash(password, salt)  
    for i = 2, ..., N:  
        B[i] ← Hash(B[i-1])  
    :  
    :
```

## Implementierung<sup>[1]</sup>

- 1. Expand** → Der Puffer wird mit pseudo-zufälligen Bytes gefüllt, die aus dem wiederholten Hashen von Passwort und Salt gewonnen werden
- 2. Mix** → der Puffer wird  $R$ -Mal gemischt. Das Mischen wird pseudo-zufällig anhand des Salts bestimmt. In jeder Runde wird für jeden Block  $i$  der Wert des Blocks anhand des vorherigen  $((i - 1) \bmod n)$ , der derzeitigen Blocks  $i$  und  $\delta$  pseudo-zufällig gewählten Blöcken bestimmt
- 3. Extract** → Der letzte Block wird als Ausgabe zurückgegeben

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Parallelisierte Ausführung<sup>[1]</sup>

- Intuitiv erlaubt Balloon Hashing keine Parallelisierung, da der Wert eines Blocks immer vom vorherigen abhängt
- **Aber:** man kann eine Variante definieren, sodass der Speicher einer  $M$ –Kern Maschine schneller gefüllt wird

$$\begin{aligned} \textit{Balloon}_M(p, s) &:= \textit{Balloon}(p, s||'1') \oplus \dots \oplus \textit{Balloon}(p, s||'M') \\ p &= \textit{Passwort} \\ s &= \textit{Salt} \end{aligned}$$

- Jeder Aufruf von  $\textit{Balloon}(\dots)$  kann parallel stattfinden und die Ergebnisse werden per XOR verknüpft

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Klassische Anforderungen (Preimage-, 2nd-Preimage-, Collision-Resistance)<sup>[1]</sup>

- Kann durch folgende Konstruktion erreicht werden

$$H_B(\text{password}, \text{salt}) := H(\text{password}, \text{salt}, B(\text{password}, \text{salt}))$$

*B = Balloon Hashing Funktion*

*H = Kryptographische Hashing Funktion (z.B. SHA256)*

**Theorem:** Wenn  $H$  kollisions- (preimage- / 2nd-preimage-) resistent ist, dann ist es auch  $H_B$

Jede Eingabe  $(x_p, x_s) \neq (y_p, y_s)$  die eine Kollision in  $B$  ist (also  $B(x_p, x_s) = B(y_p, y_s)$ ) kann keine Kollision in  $H$  auslösen, da die Kollisionsresistenz für  $H$  bewiesen ist.

$$H(x_p, x_s, B(x_p, x_s)) \neq H(y_p, y_s, B(y_p, y_s))$$

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Ziele von Balloon Hashing<sup>[1]</sup>

- **Passwort-unabhängiges Speicherzugriffsmuster**
  - Muster für Speicherzugriffe ist pseudo-zufällig durch den Salt bestimmt, also unabhängig vom Passwort

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Ziele von Balloon Hashing<sup>[1]</sup>

- **Passwort-unabhängiges Speicherzugriffsmuster**

→ Muster für Speicherzugriffe ist pseudo-zufällig durch den Salt bestimmt, also unabhängig vom Passwort



```
Balloon(password, salt, N = space_cost, R = num_rounds):

    delta = 3      // A security parameter
    B = Buffer[N] // A buffer of N blocks

    :

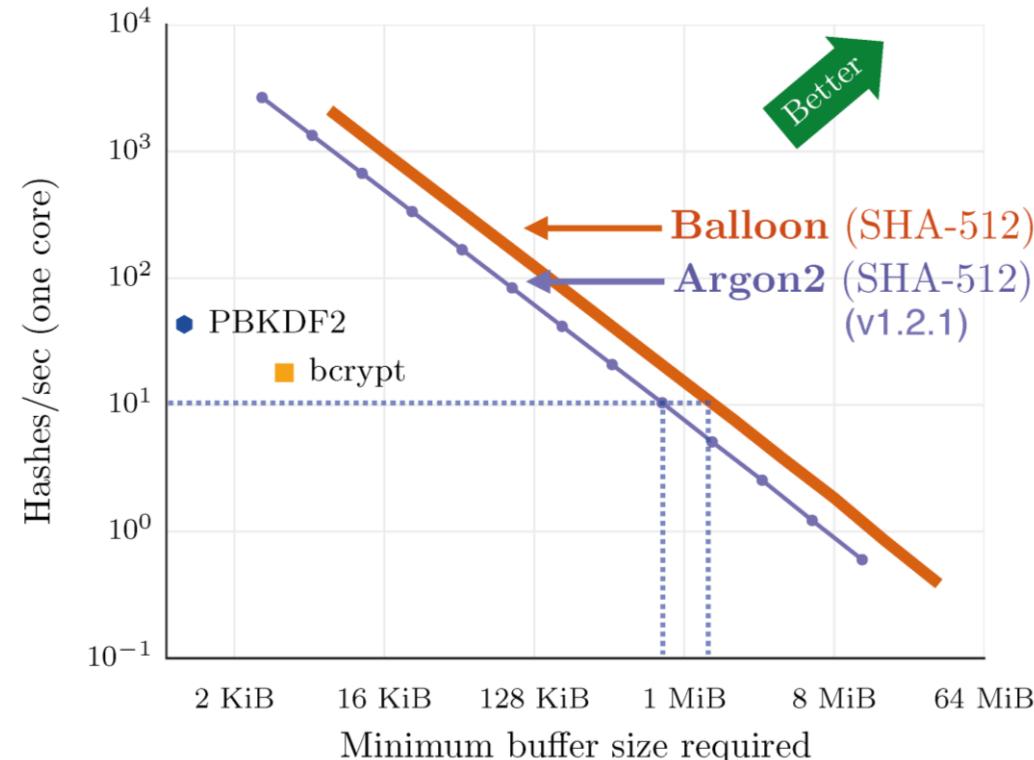
    // Step 2: Mix buffer
    for r = 1, ..., R:
        for i = 1, ..., N:
            // Chosen pseudorandomly from salt
            (v_1, ..., v_delta) ← Hash(salt, r, i)
            B[i] = Hash(B[i-1 mod N], B[i], B[v_1], ..., B[v_delta])
```

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

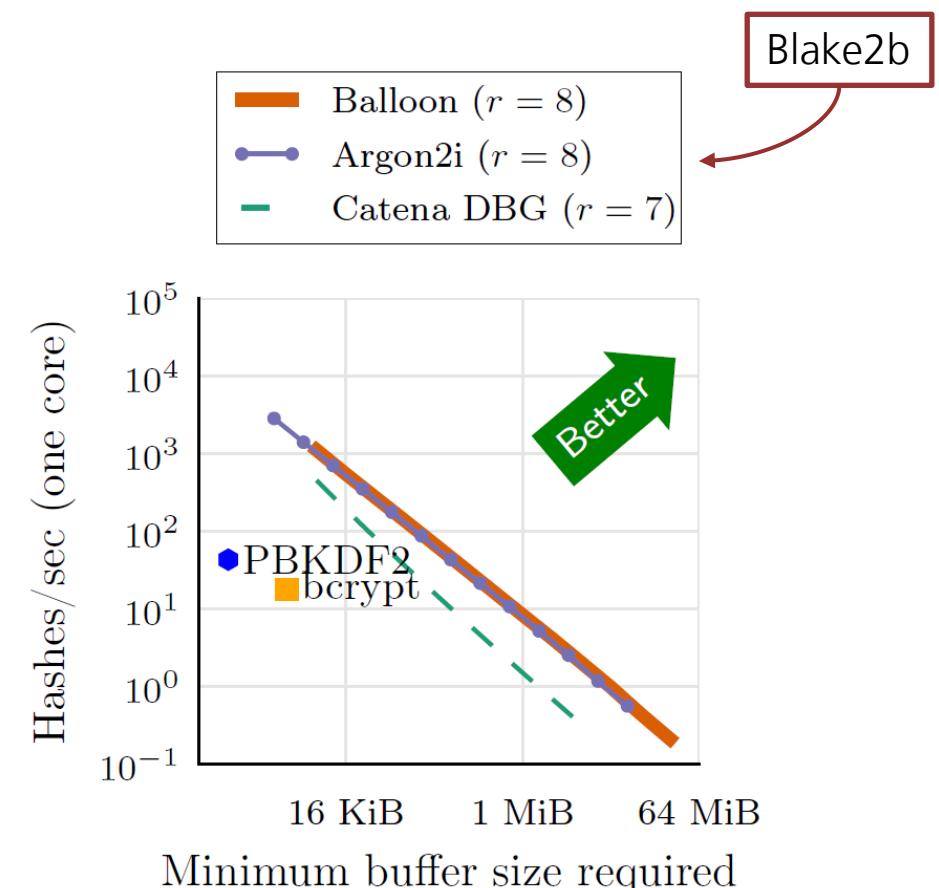
## Ziele von Balloon Hashing

- **Passwort-unabhängiges Speicherzugriffsmuster** 
  - Muster für Speicherzugriffe ist pseudo-zufällig durch den Salt bestimmt, also unabhängig vom Passwort
- **Geschwindigkeit**, die mit anderen modernen Passwort-Hashing Algorithmen mithalten kann

## Performancevergleich



Using Balloon ( $\delta = 3$ ). Both algorithms take four passes over memory.



[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Ziele von Balloon Hashing

- **Passwort-unabhängiges Speicherzugriffsmuster** 
  - Muster für Speicherzugriffe ist pseudo-zufällig durch den Salt bestimmt, also unabhängig vom Passwort
- **Geschwindigkeit**, die mit anderen modernen Passwort-Hashing Algorithmen mithalten kann 
  - Siehe vorherige Folie
- Beweisbare **Memory Hardness** Eigenschaft

## Security Proof

- **Disclaimer:** Der vollständige Beweis ist sehr lang und erfordert viele Grundlagen. Hier ist nur eine Kurzfassung dargestellt

**Theorem:**<sup>[1]</sup>

$A$  ist ein Algorithmus, der eine  $n$ -Block,  $r$ -Runden Balloon Hashing Funktion mit Sicherheitsparameter  $\delta = 3$  berechnet. Die kryptographische Hash-Funktion  $H$  ist dabei ein Zufallsorakel.

Wenn  $A$  nun maximal  $S$  Blöcke Puffer nutzt dann wird die Berechnung (mit sehr hoher Wahrscheinlichkeit) so viel Zeit  $T$  kosten, dass

$$S \cdot T \geq \frac{r \cdot n^2}{32}$$

erfüllt ist. Bei  $\delta = 7$  und  $S < \frac{n}{64}$  erhält man

$$S \cdot T \geq \frac{(2^r - 1) \cdot n^2}{32}$$

- Ein Angreifer bezahlt also mit sehr viel Berechnungszeit, wenn er Platz spart

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Ziele von Balloon Hashing

- **Passwort-unabhängiges Speicherzugriffsmuster** 
  - Muster für Speicherzugriffe ist pseudo-zufällig durch den Salt bestimmt, also unabhängig vom Passwort
- **Geschwindigkeit**, die mit anderen modernen Passwort-Hashing Algorithmen mithalten kann 
  - Siehe vorherige Folie
- Beweisbare **Memory Hardness** Eigenschaft 

# Diskussion

- Parallelisierte Angriffe im Random Oracle Model
- Unterspezifiziert und nicht aktiv unterstützt

## Parallelisierte Angriffe

- Balloon ist Memory-Hard (im Random-Oracle Model)
  - **Problem:** Das Model ist auf eine sequentielle Berechnung beschränkt → d.h. ein Angreifer kann zu jedem Zeitpunkt genau eine Anfrage an das Orakel stellen<sup>[1]</sup>
    - Wörterbuchangriffe nutzen jedoch meist sehr viele Prozessoren bzw. Kerne, da sich diese Aufgabe trivial parallelisieren lässt <sup>[2][3]</sup>

[1] Alwen, Joël; Serbinenko, Vladimir (2015): High Parallel Complexity Graphs and Memory-Hard Functions. In Rocco Servedio, Ronitt Rubinfeld (Eds.): Proceedings of the forty-seventh annual ACM symposium on Theory of Computing. STOC '15: Symposium on Theory of Computing. Portland Oregon USA, 14 06 2015 17 06 2015. New York, NY, USA: ACM, pp. 595–603.

[2] Electronic Frontier Foundation (1998): Cracking DES. Secrets of encryption research, wiretap politics & chip design. 1. ed. Sebastopol, Calif.: O'Reilly.

[3] Jeremi Gosney. Password Cracking HPC. Presented at Password'12 in Oslo, Norway, 2012. Verfügbar unter:

[http://passwords12.project.ifi.uio.no/Jeremi\\_Gosney\\_Password\\_Cracking\\_HPC/Jeremi\\_Gosney\\_Password\\_Cracking\\_HPC\\_Passwords12.pdf](http://passwords12.project.ifi.uio.no/Jeremi_Gosney_Password_Cracking_HPC/Jeremi_Gosney_Password_Cracking_HPC_Passwords12.pdf) [Abgerufen: 21.01.2023]

## Parallelisierte Angriffe

- Es gibt das (verbesserte) **parallel random-oracle model (pROM)**<sup>[1]</sup>
  - Alwen und Blocki zeigten, dass in diesem Modell keine Funktion mit einem passwort-unabhängigen Speicherzugriffsmuster „perfekt“ Memory Hard sein kann → man kann bestenfalls  $\Omega(\frac{n^2}{\log n})$  erreichen<sup>[2]</sup>
  - Ebenfalls zeigen sie einen „Space-saving“ Angriff auf Balloon (und andere Hashfunktionen)<sup>[2]</sup>
  - Alwen et al. zeigten ebenfalls, dass sich Angriffe auf Balloon nicht unendlich parallelisieren lassen<sup>[3]</sup>

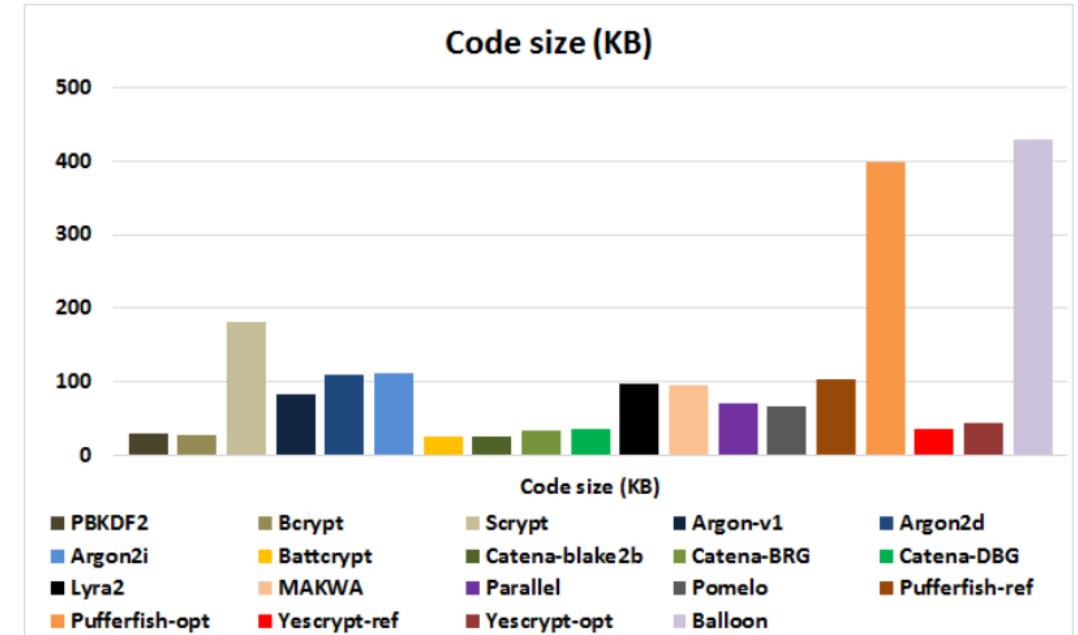
[1] Alwen, Joël; Serbinenko, Vladimir (2015): High Parallel Complexity Graphs and Memory-Hard Functions. In Rocco Servedio, Ronitt Rubinfeld (Eds.): Proceedings of the forty-seventh annual ACM symposium on Theory of Computing. STOC '15: Symposium on Theory of Computing. Portland Oregon USA, 14 06 2015 17 06 2015. New York, NY, USA: ACM, pp. 595–603.

[2] Alwen, Joel; Blocki, Jeremiah (2017): Towards Practical Attacks on Argon2i and Balloon Hashing. In : 2017 IEEE European Symposium on Security and Privacy (EuroS&P): IEEE.

[3] Alwen, Joël; Blocki, Jeremiah (2016): Efficiently Computing Data-Independent Memory-Hard Functions. In Matthew Robshaw, Jonathan Katz (Eds.): Advances in Cryptology – CRYPTO 2016, vol. 9815. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 241–271.

## Unterspezifiziert und nicht mehr unterstützt

- Die „Spezifikation“ von Balloon ist vergleichsweise kurz und verschachtelt<sup>[1]</sup>
- Keine (gute) Referenzimplementierung<sup>[2]</sup>
- Keine Teilnahme an einem Hashing Wettbewerb  
o.Ä. → keine weitere Untersuchung der Sicherheitseigenschaften von Balloon



[2]

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

[2] Hatzivasilis, George (2017): Password-Hashing Status. In Cryptography 1 (2), p. 10. DOI: 10.3390/cryptography1020010.

## Unterspezifiziert und nicht mehr unterstützt

- Das Referenzimplementierung wird nicht mehr unterstützt / weiterentwickelt [1]
- Es gab keine weiteren Veröffentlichungen (bzw. Updates) der Autoren bzgl. Balloon [2][3][4]

*Research prototype code.* Available on GitHub. *Warning: this code is NOT safe for production use! Use it only for performance tests.*

[5]



[1]

[1]

[1] <https://github.com/henrycg/balloon> [Abgerufen: 19.01.2023]  
[2] <https://dblp.uni-trier.de/pid/b/DanBoneh.html> [Abgerufen: 21.01.2023]  
[3] <https://dblp.uni-trier.de/pid/87/8037.html> [Abgerufen: 21.01.2023]  
[4] <https://dblp.uni-trier.de/pid/s/SESchechter.html> [Abgerufen: 21.01.2023]  
[5] <https://crypto.stanford.edu/balloon/> [Abgerufen: 21.01.2023]



# Zusammenfassung und Ausblick

- Was sollte man mitnehmen?
- Zukünftige Forschung

## Was sollte man mitnehmen?

- Es gibt viele Hashfunktionen für verschiedene Anwendungsfälle
- Kryptographische Hashfunktionen haben besondere Anforderungen
  - Kollisionsresistenz, Preimage-Resistenz, 2nd-Preimage Resistenz
  - Memory-Hardness, (Cache-Hardness), Password (Data) Independent Memory Access Pattern
- Memory Hard Passwort Hashing Funktionen erschweren den Aufwand von Wörterbuchangriffen
- Balloon Hashing ist eine von mehreren Alternativen, um Passwörter zu schützen
  - Diese Funktionen sind besser als iterierte Hashing-Funktionen

## Zukünftige Forschung

- Gibt es im pROM quasi perfekte Memory Hard Funktionen, die zusätzlich ein Password-Independent Memory Access Pattern haben?
  - Wäre so eine Funktion praktisch umsetzbar bzgl. Leistungsfähigkeit
- Ist es überhaupt möglich Hardware zu bauen, die den Angriff auf Balloon und Argon2, effizient umsetzen kann im pROM Kontext?
  - Wie wäre der Unterschied zu einem Angriff mit sequentieller Hardware?

# Literaturverzeichnis

- Alwen, Joel; Blocki, Jeremiah (2017): Towards Practical Attacks on Argon2i and Balloon Hashing. In : 2017 IEEE European Symposium on Security and Privacy (EuroS&P): IEEE.
- Alwen, Joel; Gazi, Peter; Kamath, Chethan; Klein, Karen; Osang, Georg; Pietrzak, Krzysztof et al. (2018): On the Memory-Hardness of Data-Independent Password-Hashing Functions. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier Lopez, Taesoo Kim (Eds.): Proceedings of the 2018 on Asia Conference on Computer and Communications Security. ASIA CCS '18: ACM Asia Conference on Computer and Communications Security. Incheon Republic of Korea, 04 06 2018 04 06 2018. New York, NY, USA: ACM, pp. 51–65.
- Alwen, Joël; Blocki, Jeremiah (2016): Efficiently Computing Data-Independent Memory-Hard Functions. In Matthew Robshaw, Jonathan Katz (Eds.): Advances in Cryptology – CRYPTO 2016, vol. 9815. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 241–271.
- Alwen, Joël; Serbinenko, Vladimir (2015): High Parallel Complexity Graphs and Memory-Hard Functions. In Rocco Servedio, Ronitt Rubinfeld (Eds.): Proceedings of the forty-seventh annual ACM symposium on Theory of Computing. STOC '15: Symposium on Theory of Computing. Portland Oregon USA, 14 06 2015 17 06 2015. New York, NY, USA: ACM, pp. 595–603.
- Aumasson, Jean-Philippe (2018): Serious Cryptography. A Practical Introduction to Modern Encryption. San Francisco, CA: No Starch Press Incorporated. Available online at <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5331332>.
- Bellare, Mihir; Rogaway, Phillip (1993): Random oracles are practical. In Dorothy Denning, Ray Pyle, Ravi Ganesan, Ravi Sandhu, Victoria Ashby (Eds.): Proceedings of the 1st ACM conference on Computer and communications security - CCS '93. the 1st ACM conference. Fairfax, Virginia, United States, 03.11.1993 - 05.11.1993. New York, New York, USA: ACM Press, pp. 62–73.
- Bellare, Mihir; Rogaway, Phillip (1996): The Exact Security of Digital Signatures-How to Sign with RSA and Rabin. In Ueli Maurer (Ed.): Advances in cryptology - EUROCRYPT '96. International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12 - 16, 1996 ; proceedings, vol. 1070. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 1070), pp. 399–416.
- Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.
- Bonneau, Joseph; Mironov, Ilya (2006): Cache-Collision Timing Attacks Against AES. In Louis Goubin, Mitsuru Matsui (Eds.): Cryptographic hardware and embedded systems - CHES 2006. 8th international workshop, Yokohama, Japan, October 10-13, 2006 ; proceedings, vol. 4249. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 4249), pp. 201–215.

- Chi, Lianhua; Zhu, Xingquan (2018): Hashing Techniques. In ACM Comput. Surv. 50 (1), pp. 1–36. DOI: 10.1145/3047307.
- Colin Percival (2009): STRONGER KEY DERIVATION VIA SEQUENTIAL MEMORY-HARD FUNCTIONS. Available online at <https://www.semanticscholar.org/paper/STRONGER-KEY-DERIVATION-VIA-SEQUENTIAL-MEMORY-HARD-Percival/7c74956d21f0466c9771bb583e2fdf854c2aedbf>.
- Dang, Quynh H. (2015): Secure Hash Standard.
- Das, Anupam; Bonneau, Joseph; Caesar, Matthew; Borisov, Nikita; Wang, XiaoFeng (2014): The Tangled Web of Password Reuse. In : Proceedings 2014 Network and Distributed System Security Symposium. Reston, VA: Internet Society.
- Dwork, Cynthia; Naor, Moni; Wee, Hoeteck (2005): Pebbling and Proofs of Work. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Eds.): Advances in Cryptology – CRYPTO 2005, vol. 3621. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 37–54.
- Dworkin, Morris J. (2015): SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.
- Einspruch, Norman G.; Hilbert, Jeffrey L. (2012): Application specific integrated circuit (ASIC) technology: Academic Press (VLSI electronics, v. 23).
- Electronic Frontier Foundation (1998): Cracking DES. Secrets of encryption research, wiretap politics & chip design. 1. ed. Sebastopol, Calif.: O'Reilly.
- Europäischen Union: Datenschutz-Grundverordnung, DSGVO, revised 2021. In : Amtsblatt der Europäischen Union (ABl. L 119, 04.05.2016). Available online at <https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32016R0679>, checked on 1/21/2023.
- Hatzivasilis, George (2017): Password-Hashing Status. In Cryptography 1 (2), p. 10. DOI: 10.3390/cryptography1020010.
- Hennessy, John L.; Patterson, David A. (2003): Computer architecture. A quantitative approach. 3rd ed. San Francisco, CA: Morgan Kaufmann Publishers (Morgan Kaufmann series in computer architecture and design).
- 2004-03: ISO/IEC 10118-3:2004. Available online at <https://www.iso.org/standard/39876.html>, checked on 1/14/2023.

- Jean-Philippe Aumasson; Samuel Neves; Zooko Wilcox-O'Hearn; Christian Winnerlein (2013): BLAKE2: simpler, smaller, fast as MD5. In Michael Jacobson, Michael Locasto, Payman Mohassel, Reihaneh Safavi-Naini, Michael J. Jacobson (Eds.): Applied cryptography and network security. 11th international conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013 ; proceedings. Proceedings of the 11th international conference on Applied Cryptography and Network Security. ACNS. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 7954), pp. 119–135. Available online at [https://www.researchgate.net/publication/262288030\\_BLAKE2\\_simpler\\_smaller\\_fast\\_as\\_MD5](https://www.researchgate.net/publication/262288030_BLAKE2_simpler_smaller_fast_as_MD5).
- Kaliski, B. (2000): PKCS #5: Password-Based Cryptography Specification Version 2.0.
- Mazurkiewicz, Antoni (1995): Introduction to Trace Theory. In Volker Diekert (Ed.): The book of traces. Singapore: World Scientific, pp. 3–41.
- Menezes, Alfred J. (1996): Handbook of Applied Cryptography. Hoboken: CRC Press (Discrete Mathematics and Its Applications). Available online at <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=1605633>.
- N Provos, D. Mazieres (1999): Bcrypt algorithm: USENIX. Available online at [https://www.usenix.org/events/usenix99/provos/provos\\_html/node5.html](https://www.usenix.org/events/usenix99/provos/provos_html/node5.html).
- Niels Kornerup (2022): Memory hard functions and persistent memory hardness. Available online at <https://www.cs.utexas.edu/~dwu4/courses/sp22/static/projects/Kornerup.pdf>.
- Paul, Wolfgang J.; Tarjan, Robert Endre; Celoni, James R. (1976): Space bounds for a game on graphs. In Math. Systems Theory 10 (1), pp. 239–251. DOI: 10.1007/BF01683275.
- Ristenpart, Thomas; Tromer, Eran; Shacham, Hovav; Savage, Stefan (2009): Hey, you, get off of my cloud. In Ehab Al-Shaer, Somesh Jha, Angelos D. Keromytis (Eds.): Proceedings of the 16th ACM conference on Computer and communications security. CCS '09: 16th ACM Conference on Computer and Communications Security 2009. Chicago Illinois USA, 09 11 2009 13 11 2009. New York, NY, USA: ACM, pp. 199–212.
- Rogaway, Phillip; Shrimpton, Thomas (2004): Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor et al. (Eds.): Fast Software Encryption, vol. 3017. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 371–388.
- Roy, Arnab; Datta, Anupam; Mitchell, John C. (2008): Formal Proofs of Cryptographic Security of Diffie-Hellman-Based Protocols. In Gilles Barthe, Cédric Fournet (Eds.): Trustworthy Global Computing, vol. 4912. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 312–329.
- Smart, Nigel P. (2003): Cryptography. An introduction. London: McGraw-Hill (McGraw-Hill education).

- Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).
- Tromer, Eran; Osvik, Dag Arne; Shamir, Adi (2010): Efficient Cache Attacks on AES, and Countermeasures. In J Cryptol 23 (1), pp. 37–71. DOI: 10.1007/s00145-009-9049-y.
- van Tilborg, Henk C. A.; Jajodia, Sushil (Eds.) (2011): Encyclopedia of Cryptography and Security. 2. ed. Boston, MA: Springer US (SpringerLink Bücher).
- Wang, Xiaoyun; Yu, Hongbo (2005): How to Break MD5 and Other Hash Functions. In Ronald Cramer (Ed.): Advances in cryptology - EUROCRYPT 2005. 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005 ; proceedings, vol. 3494. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 3494), pp. 19–35.

# Fragen

**Kontakt:**

Henry Weckermann

[henry.weckermann@studium.fernuni-hagen.de](mailto:henry.weckermann@studium.fernuni-hagen.de)

**Präsentation, Paper und weitere Unterlagen:**

[https://github.com/arantarion/Balloon\\_Seminararbeit](https://github.com/arantarion/Balloon_Seminararbeit)

# Anhang

## Hashing Abgrenzung

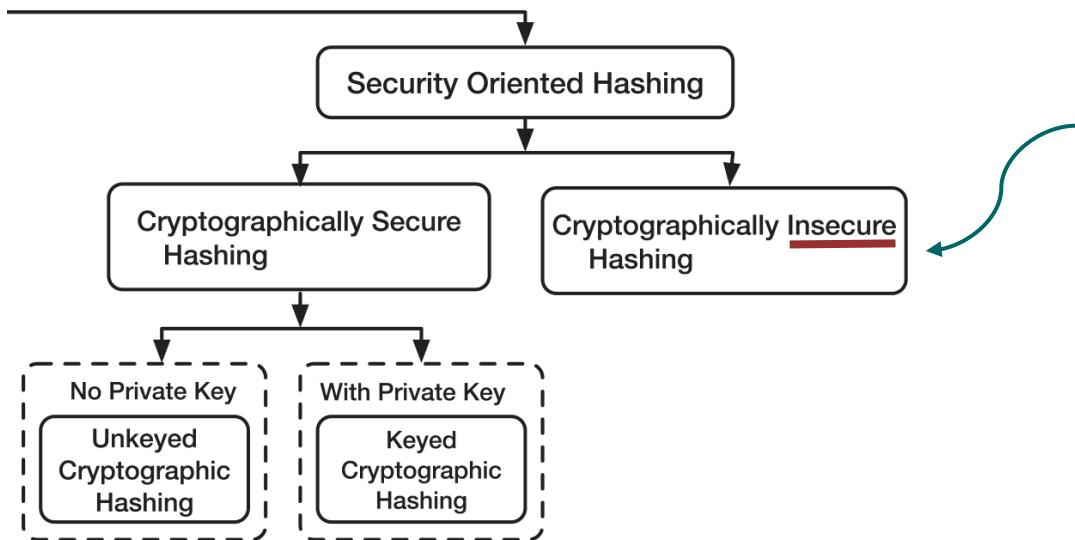
<b>Random Hashing</b>	Random Projection Hashing; Universal Hashing Locality-Sensitive Hashing (LSH)
<b>Locality Sensitive Hashing</b>	MinHash; Weighted MinHash (WMH)
	Shift-Invariant Kernel-Based Hashing (SIKH)
	Nested Subtree Hashing (NSH)
	Discriminative Clique Hashing (DICH)
	BoostMap
<b>Structured Projection</b>	Quadtree; Hilbert Curve; Z curve

<b>Cryptographic Hashing</b>	<b>Keyed Cryptographic Hashing</b>	VMAC; UMAC; PMAC; OMAC; HMAC Poly1305-AES; MD6; BLAKE2 MD2/4/5/6
	<b>Unkeyed Cryptographic Hashing</b>	SHA-1/3/224/256/384/512 HAVAL; GOST; FSB; JH; ECOH RIPEMD/-128/-160/-320

<b>Noncryptographic Hashing</b>	FNV Hash; xxHash; SuperFastHash
	MurmurHash2; lookup3; Pearson Hashing; BuzHash
	Approximate Matching
	DEK; BKDR; APartow; DJBX33A

<b>Unsupervised Hashing</b>	<b>Spectral Hashing</b>	Anchor Graph Hashing (AGH)
		Product Quantization (PQ)
		Angular Quantization-Based Binary Coding (AQBC)
		Spherical Hashing
		Isotropic Hashing
		Manhattan Hashing
		Predictable Dual-View Hashing (PDH)
		Inductive Manifold Hashing (IMH)
		Locally Linear Hashing (LLH)
		Topology Preserving Hashing (TPH)
<b>Semisupervised Hashing</b>	<b>Linear Semisupervised Hashing</b>	Kernelized LSH (KLSH)
		Multiple Feature Kernel Hashing (MFKH)
		Semisupervised Hashing (SSH)
		Semisupervised Discriminant Hashing (SSDH)
		Semisupervised Topology-Preserving Hashing (STPH)
	<b>Nonlinear Semisupervised Hashing</b>	Label-regularized Max-margin Partition (LAMP)
		Bootstrap-NSPLH
<b>Supervised Hashing</b>	<b>Linear Supervised Hashing</b>	Supervised Hashing with Binary Reduction
		Boosting Similarity-Sensitive Coding (BoostSSC)
		Binary Reconstructive Embedding (BRE)
		Minimal Loss Hashing (MLH)
		Latent Factor Hashing (LFH)
		Linear Discriminant Analysis-Based Hashing (LDAHash)
	<b>Nonlinear Supervised Hashing</b>	Kernel-Based Supervised Hashing (KSH)
		Two-Step Hashing (TSH)
		FastHash

## Taxonomie<sup>[1]</sup>



### Einsatz<sup>[2]</sup>

- Datenstrukturen (hash table, hashmap)
- Fehlererkennung
- Viele weitere Beispiele

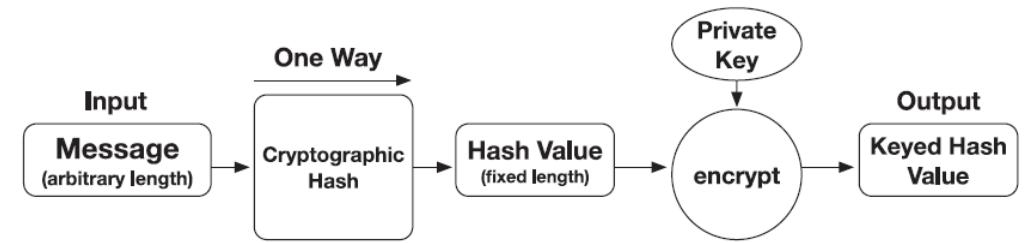
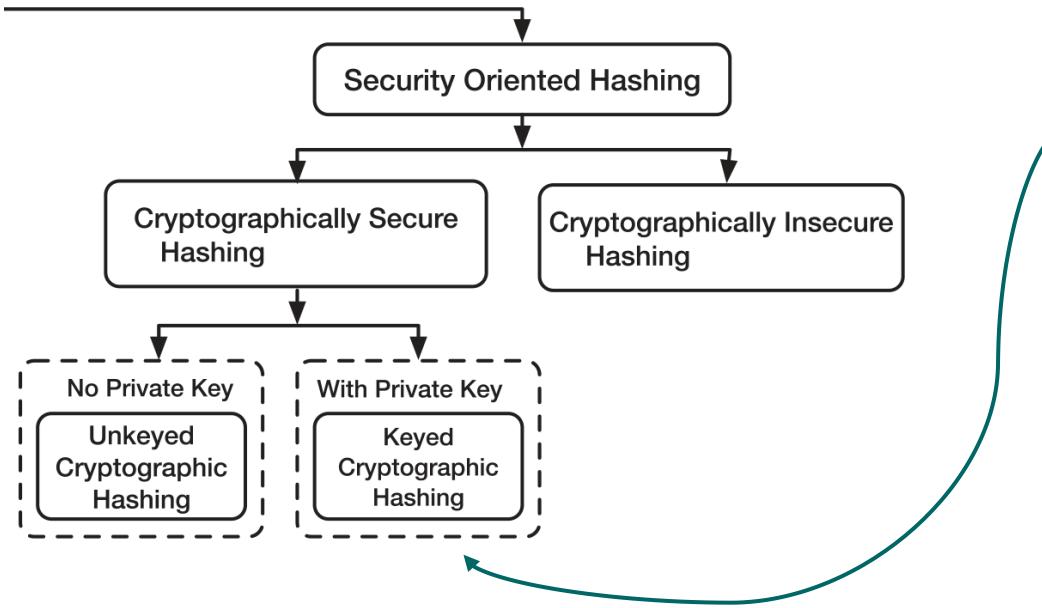
### Beispiele<sup>[1]</sup>

FNV (Fowler-Noll-Vo) Hash,  
xxHash,  
SuperFastHash,  
MurmurHash2,  
lookup3,  
Pearson Hashing,  
DJBX33A  
...

[1] Chi, Lianhua; Zhu, Xingquan (2018): Hashing Techniques. In ACM Comput. Surv. 50 (1), pp. 1–36. DOI: 10.1145/3047307.

[2] Aumasson, Jean-Philippe (2018): Serious Cryptography. A Practical Introduction to Modern Encryption. San Francisco, CA: No Starch Press Incorporated. ISBN: 1-59327-826-8

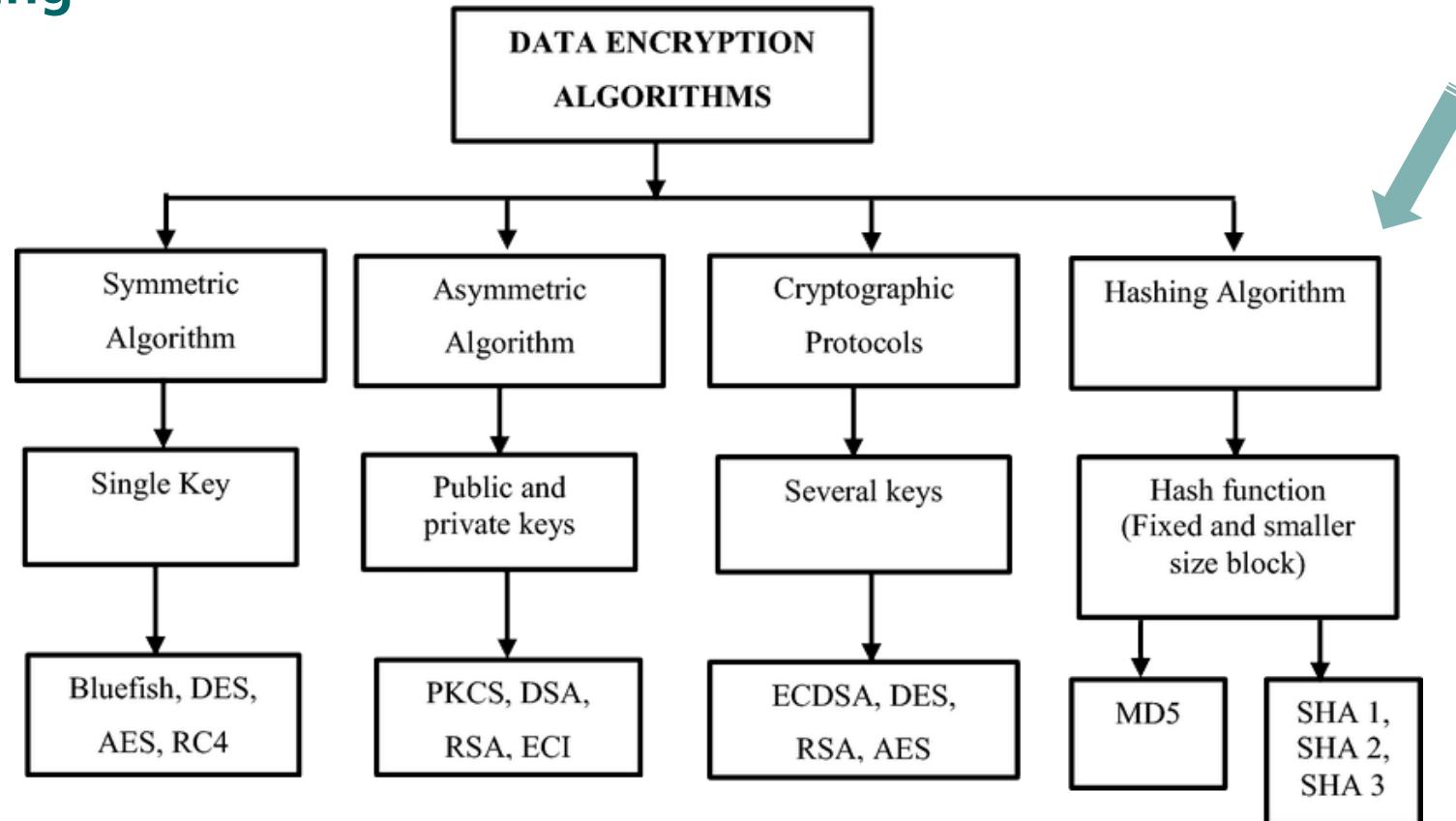
## Taxonomie<sup>[1]</sup>



**Beispiele<sup>[1]</sup>** VMAC, UMAC, HMAC, PMAC,  
Poly1305-AES,  
MD6,  
BLAKE2

[1] Chi, Lianhua; Zhu, Xingquan (2018): Hashing Techniques. In ACM Comput. Surv. 50 (1), pp. 1–36. DOI: 10.1145/3047307.

## Abgrenzung



[1] Alfa, Abraham & Alhassan, John & Olaniyi, Olayemi & Olalere, Morufu. (2021). Blockchain technology in IoT systems: current trends, methodology, problems, applications, and future directions. Journal of Reliable Intelligent Environments. 7. 10.1007/s40860-020-00116-z.

## Abgrenzung

Password-Hashing Functions	Bcrypt
Password-Based Key Derivation Functions	Scrypt Argon2 PBKDF2 bscrypt Balloon Hashing
Balanced Password Authenticated Key Exchanges	CPace
Augmented Password Authenticated Key Exchanges	SRP AuCPace BS-SPEKE
Doubly-Augmented Password Authenticated Key Exchanges	OPAQUE Double BS-SPEKE

### Mehr Infos zu PAKEs:

- [1] Boyko, Victor; MacKenzie, Philip; Patel, Sarvar (2000): Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Bart Preneel (Eds.): Advances in Cryptology — EUROCRYPT 2000, vol. 1807. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 156–171.
- [2] Katz, Jonathan; Ostrovsky, Rafail; Yung, Moti (2001): Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Birgit Pfitzmann (Eds.): Advances in Cryptology — EUROCRYPT 2001, vol. 2045. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 475–494.
- [3] MacKenzie, Philip (2001): More Efficient Password-Authenticated Key Exchange. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, David Naccache (Eds.): Topics in Cryptology — CT-RSA 2001, vol. 2020. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 361–377.

## Warum Hashing?

Pressemitteilung vom 22. November 2018 [1]: „LfDI [Landesbeauftragter für den Datenschutz und die Informationsfreiheit] Baden-Württemberg verhängt sein erstes Bußgeld in Deutschland nach der DS-GVO -Kooperation mit Aufsicht macht es glimpflich-“

Durch die Speicherung der Passwörter im Klartext verstieß das Unternehmen wesentlich gegen seine Pflicht zur Gewährleistung der Datensicherheit bei der Verarbeitung personenbezogener Daten gem. Art. 32 Abs. 1 lit a DS-GVO.

→ Bußgeld: 20.000€ (ca. 2% des letzten Jahresumsatzes)

[1] <https://www.baden-wuerttemberg.datenschutz.de/wp-content/uploads/2018/11/LfDI-Baden-W%C3%BCrttemberg-verh%C3%A4ngt-sein-erstes-Bu%C3%9Fgeld-in-Deutschland-nach-der-DS-GVO.pdf> [Abgerufen: 08.01.2023]  
Mehr Informationen: Bußgeldbescheid nach DS-GVO: [https://gdprhub.eu/images/5/f/LfDI\\_BW - O\\_1018-115.pdf](https://gdprhub.eu/images/5/f/LfDI_BW - O_1018-115.pdf) [Abgerufen: 08.01.2023]

## Bekannte (Standard-) Hashfunktionen

Name	Sicherheitsniveau	Status
MD5	$2^{64}$	Gebrochen <sup>[1]</sup>
SHA-1	$2^{80}$	Gebrochen <sup>[1]</sup>
SHA-0	$2^{80}$	Gebrochen <sup>[1]</sup>
SHA256, SHA512 <sup>[2]</sup>	$2^{128}$ bzw. $2^{256}$	
SHA-3 <sup>[3]</sup>	<i>bis zu</i> $2^{512}$	
BLAKE2s, BLAKE2b <sup>[4]</sup>	$2^{128}$ bzw. $2^{256}$	
RIPEMD-160 <sup>[5]</sup>	$2^{80}$	
Whirlpool <sup>[5]</sup>	$2^{256}$	

[1] Wang, Xiaoyun; Yu, Hongbo (2005): How to Break MD5 and Other Hash Functions. In Ronald Cramer (Ed.): Advances in cryptology - EUROCRYPT 2005. 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005 ; proceedings, vol. 3494. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 3494), pp. 19-35.

[2] Dang, Quynh H. (2015): Secure Hash Standard. Available: <https://doi.org/10.6028/NIST.FIPS.180-4>

[3] Dworkin, Morris J. (2015): SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Available: <https://doi.org/10.6028/NIST.FIPS.202>

[4] Jean-Philippe Aumasson; Samuel Neves; Zooko Wilcox-O'Hearn; Christian Winzerlein (2013): BLAKE2: simpler, smaller, faster than MD5. In Michael Jacobson, Michael Locasto, Payman Mohassel, Reihaneh Safavi-Naini, Michael J. Jacobson (Eds.): Applied cryptography and network security. 11th international conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013 ; proceedings. Proceedings of the 11th international conference on Applied Cryptography and Network Security. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 7954), pp. 119-135.

[5] 2004-03: ISO/IEC 10118-3:2004. Available online at <https://www.iso.org/standard/39876.html>

# Was ist Kryptographisch-sicheres-Hashing?

## Am Rande

Für jede Hashfunktion gibt es

- Einen Preimage Angriff in  $O(2^{n-1})$  [Bruteforce]
- Einen Second Preimage Angriff in  $O(2^{n-1})$  [Bruteforce]
- Einen Kollisionsangriff in  $O(2^{n/2})$  [Geburtstagsparadoxon]

der jeweils mit Wahrscheinlichkeit 0,5 erfolgreich ist [1][2]



Die Anzahl der Versuche  $q$ , um mit der Wahrscheinlichkeit 0,5, eine Kollision in  $Y = \{0,1\}^n$  zu finden kann durch  
$$q \approx 1,17 * \sqrt{n}$$
 beschrieben werden

Für ein beliebiges  $\epsilon$ :

$$q \approx \sqrt{2n \ln \frac{1}{1-\epsilon}}$$

[1]

[1] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

[2] Smart, Nigel P. (2003): Cryptography. An introduction. London: McGraw-Hill (McGraw-Hill education).

# Was ist Kryptographisch-sicheres-Hashing?

## Am Rande

Für jede Hashfunktion gibt es

- Einen Preimage Angriff in  $O(2^{n-1})$  [Bruteforce]
- Einen Second Preimage Angriff in  $O(2^{n-1})$  [Bruteforce]
- Einen Kollisionsangriff in  $O(2^{n/2})$  [Geburtstagsparadoxon]

der jeweils mit Wahrscheinlichkeit 0,5 erfolgreich ist [1][2]

Bei einem 40bit Hash: Mit Wahrscheinlichkeit  $\epsilon = 0,5$  würde eine Kollision in etwas mehr als  $2^{20} \approx 1.000.000$  zufälligen Hashes gefunden werden.

→ Führt dazu, dass  $n$  ungefähr doppelt so groß sein muss wie das angestrebte Sicherheitsniveau



Die Anzahl der Versuche  $q$ , um mit der Wahrscheinlichkeit 0,5, eine Kollision in  $Y = \{0,1\}^n$  zu finden kann durch  $q \approx 1,17 * \sqrt{n}$  beschrieben werden.

Für ein beliebiges  $\epsilon$ :

$$q \approx \sqrt{2n \ln \frac{1}{1-\epsilon}} \quad [1]$$

[1] Stinson, Douglas R.; Paterson, Maura B. (2019): Cryptography. Theory and practice. Fourth edition. Boca Raton, London, New York: CRC Press (Textbooks in mathematics).

[2] Smart, Nigel P. (2003): Cryptography. An introduction. London: McGraw-Hill (McGraw-Hill education).

# Anforderungen an moderne Passwort-Hashing-Algorithmen

## Random Oracle Model<sup>[1][2]</sup>

- Nützlich für formale Beweise
  - Der Aufwand eines Angriffes kann in der Anzahl der Aufrufe des Orakels ausgedrückt werden
- Keine (Hash-)Funktion kann die Eigenschaften des Zufallsorakels haben<sup>[2]</sup> → man nimmt an, dass sie zufällig genug sind und das es keine strukturellen Sicherheitslücken gibt
- Ein Beweis im Random Oracle Model ist besser als kein formeller Beweis und einfacher als im sog. „Standard Model“ (Turing-Maschinen Model)
- Viele bekannte Algorithmen sind im ROM formell bewiesen, z.B. RSA-FDH<sup>[3]</sup>, Diffie–Hellman key exchange<sup>[4]</sup>

[1] van Tilborg, Henk C. A.; Jajodia, Sushil (Eds.) (2011): Encyclopedia of Cryptography and Security. 2. ed. Boston, MA: Springer US (SpringerLink Bücher).

[2] Bellare, Mihir; Rogaway, Phillip (1993): Random oracles are practical. In Dorothy Denning, Ray Pyle, Ravi Ganesan, Ravi Sandhu, Victoria Ashby (Eds.): Proceedings of the 1st ACM conference on Computer and communications security - CCS '93. The 1st ACM conference. Fairfax, Virginia, United States, 03.11.1993 - 05.11.1993. New York, New York, USA: ACM Press, pp. 62–73.

[3] Bellare, Mihir; Rogaway, Phillip (1996): The Exact Security of Digital Signatures-How to Sign with RSA and Rabin. In Ueli Maurer (Ed.): Advances in cryptology - EUROCRYPT '96. International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12 - 16, 1996 ; proceedings, vol. 1070. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 1070), pp. 399–416.

[4] Roy, Arnab; Datta, Anupam; Mitchell, John C. (2008): Formal Proofs of Cryptographic Security of Diffie-Hellman-Based Protocols. In Gilles Barthe, Cédric Fournet (Eds.): Trustworthy Global Computing, vol. 4912. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 312–329.

## Gegenüberstellung etablierter Passwort-Hashing-Algorithmen

Hashfunktion	MHF	Data-Independent Memory Access Pattern
<b>Bcrypt</b>		X
<b>Scrypt</b>	X	
<b>Argon2d</b>	X	
<b>Argon2i</b>	X	X
<b>PBKDF2</b>		X
<b>Catena</b>	X	X
<b>Balloon Hashing</b>	X	X

## Implementierungen

C (Referenzimplementierung)

<https://github.com/henrycg/balloon>

Java

<https://github.com/codahale/balloonhash>

PHP

<https://github.com/stlixonline/php-balloon-hashing>

Python

<https://github.com/nachonavarro/balloon-hashing>

Rust

<https://github.com/stichtingorganism/balloon>

## Security Proof

### Beispiel:

Wenn  $\delta = 7$  und  $S \leq \frac{N}{8}$  wird so viel Zeit  $T$  benötigt, dass

$$S \cdot T \geq \frac{(2^R - 1)}{8 \cdot N^2}$$

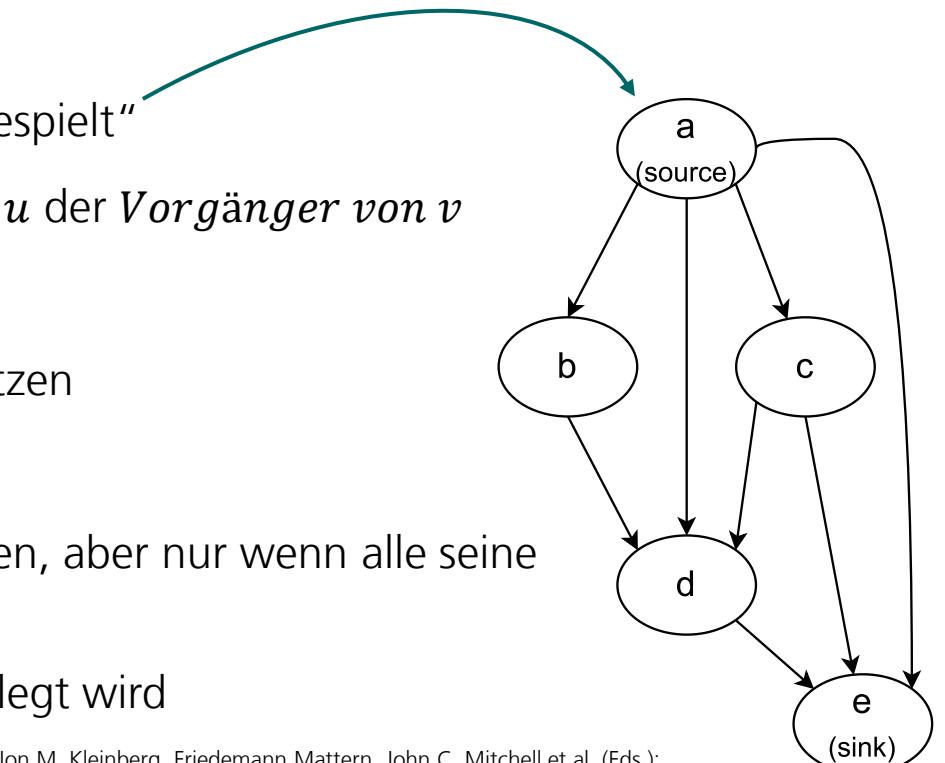
erfüllt ist.

- Wenn man nur 1/8 des Platzes nutzen will führt dies zu einer Verlangsamung die exponentiell zu der Anzahl an Runden wächst
- Bei 20 Runden bedeutet das 60000-Mal mehr Zeit

## Security Proof

### Das Graph Pebbling Spiel<sub>[1][2][3]</sub>

- Wird auf gerichteten Graphen ohne Zyklen  $G = (V, E)$  „gespielt“
- Für eine Kante  $(u, v) \in E$  ist  $v$  der **Nachfolger** von  $u$  und  $u$  der **Vorgänger** von  $v$
- folgende **Spielzüge** sind erlaubt:
  - Einen Spielstein (Pebble) auf einen Source-Knoten setzen
  - Einen Spielstein von irgendeinem Knoten entfernen
  - Einen Spielstein auf einen (nicht-source-)Knoten setzen, aber nur wenn alle seine Vorgängerknoten schon mit einem Stein belegt sind
- Das Spiel endet, wenn auf einen Sink-Knoten ein Stein gelegt wird



[1] Dwork, Cynthia; Naor, Moni; Wee, Hoeteck (2005): Pebbling and Proofs of Work. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Eds.): Advances in Cryptology – CRYPTO 2005, vol. 3621. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 37–54.

[2] Paul, Wolfgang J.; Tarjan, Robert Endre; Celoni, James R. (1976): Space bounds for a game on graphs. In Math. Systems Theory 10 (1), pp. 239–251. DOI: 10.1007/BF01683275.

[3] Sethi, Ravi (1973): Complete register allocation problems. In Alfred V. Aho, Allan B. Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, H. Raymond Strong (Eds.): Proceedings of the fifth annual ACM symposium on Theory of computing - STOC '73. the fifth annual ACM symposium. Austin, Texas, United States, 30.04.1973 - 02.05.1973. New York, New York, USA: ACM Press, pp. 182–195.

## Security Proof

### Das Graph Pebbling Spiel<sup>[1][2][3]</sup>

- Kanten sind Daten-abhängigkeiten
- Knoten sind Zwischenwerte, die für Berechnungen gebraucht werden
- Source-Knoten sind Eingabewerte / Sink-Knoten sind Ausgabewerte
- Die Steine auf dem Graphen repräsentieren Werte, die an einem Punkt in der Berechnung im Speicher sind

[1] Dwork, Cynthia; Naor, Moni; Wee, Hoeteck (2005): Pebbling and Proofs of Work. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Eds.): Advances in Cryptology – CRYPTO 2005, vol. 3621. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 37–54.

[2] Paul, Wolfgang J.; Tarjan, Robert Endre; Celoni, James R. (1976): Space bounds for a game on graphs. In Math. Systems Theory 10 (1), pp. 239–251. DOI: 10.1007/BF01683275.

[3] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Security Proof

### Das Graph Pebbling Spiel<sup>[1][2][3]</sup>

- Eine Pebbling Strategie gibt für eine Berechnung „space-time trade-offs“ an
  - Time = Anzahl der Schritte
  - Space = Anzahl der Spielsteine

[1] Dwork, Cynthia; Naor, Moni; Wee, Hoeteck (2005): Pebbling and Proofs of Work. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Eds.): Advances in Cryptology – CRYPTO 2005, vol. 3621. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 37–54.

[2] Paul, Wolfgang J.; Tarjan, Robert Endre; Celoni, James R. (1976): Space bounds for a game on graphs. In Math. Systems Theory 10 (1), pp. 239–251. DOI: 10.1007/BF01683275.

[3] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

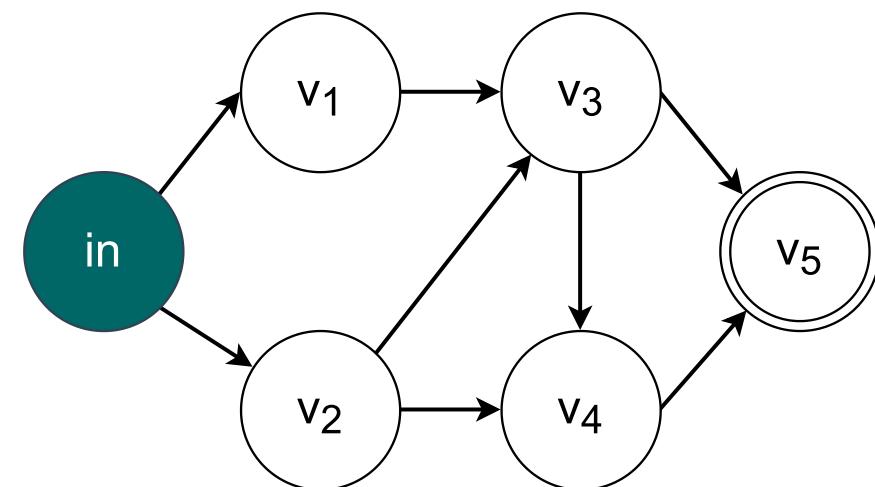
## Security Proof

### Data-Dependency Graph (DDG)<sup>[1]</sup>

```

 $v_1 = \text{hash}(\text{input}, "0")$ 
 $v_2 = \text{hash}(\text{input}, "1")$ 
 $v_3 = \text{hash}(v_1, v_2)$ 
 $v_4 = \text{hash}(v_2, v_3)$ 
 $v_5 = \text{hash}(v_3, v_4)$ 
return  $v_5$ 
    
```

Ein Knoten für jeden Aufruf  
des Zufallsorakels

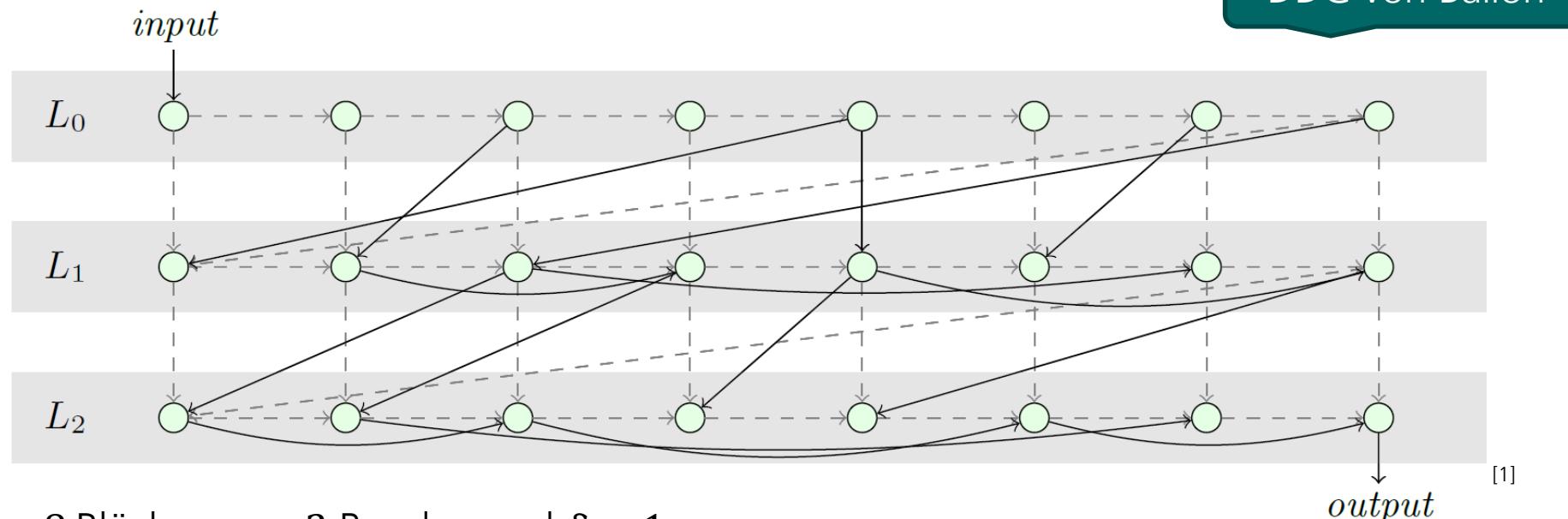


[2]

[1] Mazurkiewicz, Antoni (1995): Introduction to Trace Theory. In Volker Diekert (Ed.): The book of traces. Singapore: World Scientific, pp. 3–41.

[2] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Security Proof



Für  $n = 8$  Blöcke,  $r = 2$  Runden und  $\delta = 1$

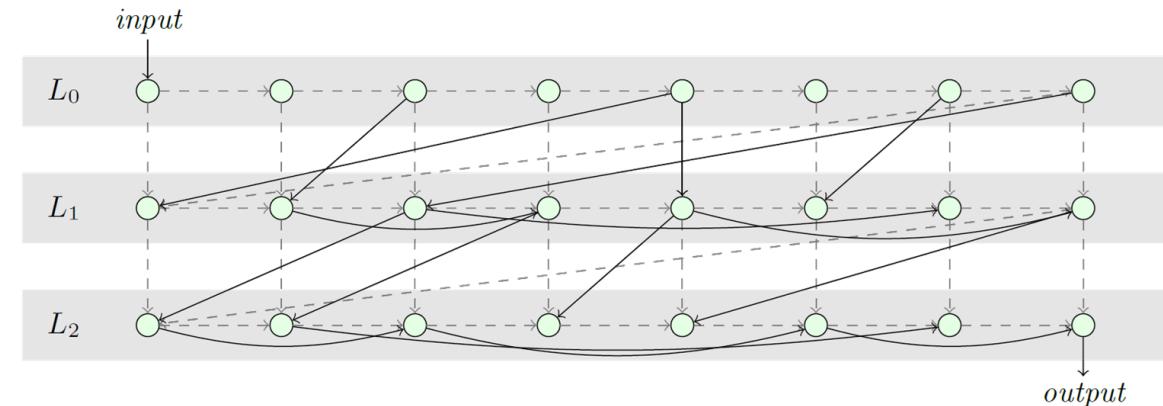
Die gestrichelten Kanten sind fest vorgegeben, die durchgängigen Linien sind die pseudozufällig ausgewählten Blöcke

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Security Proof

### Beweis – Schritt 1:

- Es kann gezeigt werden, dass der DDG von Balloon bestimmte Eigenschaften erfüllt<sup>[1]</sup>
  - **Well-Spreadedness** (gut Verteilung): Für jede Menge von  $k$ -hintereinanderfolgenden Knoten auf einem Level sind mindestens  $\frac{1}{4}$  der Knoten des vorherigen Levels auf nicht mit Spielsteinen belegten Pfaden zu diesen  $k$  Knoten
  - **Expansion** (Ausdehnung): Jede Menge von  $k$  Knoten auf einem Level hat nicht mit Steinen belegte Pfade zu mindestens  $2 \cdot k$  Knoten auf dem vorherigen Level ( $k$  hängt von  $\delta$  ab)



[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Security Proof

### Beweis – Schritt 2:[1]

- Mit den Eigenschaften aus Schritt 1 kann gezeigt werden, dass jeder Angreifer  $A$   $S$  „Platz“ und  $T$  Zeit verwenden muss, sodass die Ungleichung

$$S \cdot T \geq \frac{n^2}{32}$$

(mit hoher Wahrscheinlichkeit) erfüllt ist

- **Idee:**
  - Man nimmt eine Menge mit  $k$  vielen Werten, die viele Abhängigkeiten haben
  - Angreifer  $A$ , der weniger „Platz“ verwendet, kann diese dann nicht alle im Speicher haben, daher muss  $A$  viele Werte erneut berechnen → kostet viel Zeit

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Security Proof

### Beweis – Schritt 3:

- Die Technik von Dwork, Naor und Wee<sup>[1]</sup> wird angewendet, um einen gerichteten, azyklischen Graphen in eine Funktion  $f_G$  zu überführen
  - Wenn ein Graph  $G$  für einen graphen-respektierenden pebbling Angreifer, der Zeit und Platz sparen möchte, nicht effizient ist,
  - ist die Funktion  $f_G$  für jede Art Angreifer, der Platz und Zeit sparen möchte, nicht effizient berechenbar (im Random-Oracle Model)
- Angewandt auf Balloon kommen Boneh et al.<sup>[2]</sup> zu einer obere Schranke der Wahrscheinlichkeit, dass ein Angreifer die Funktion mit weniger Zeit und „Platz“ berechnen kann

[1] Dwork, Cynthia; Naor, Moni; Wee, Hoeteck (2005): Pebbling and Proofs of Work. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Eds.): Advances in Cryptology – CRYPTO 2005, vol. 3621. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 37–54.

[2] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Optionen für den praktischen Einsatz

### Vorschlag der Autoren von Balloon Hashing<sup>[1]</sup>

- 1. Scrypt** → Hindert parallelisierte Angriffe / Daten-abhängiges Zugriffsmuster
- 2. Balloon Hashing** → Schwächer gegen parallelisierte Angriffe / Daten-unabhängiges Muster
- 3. Kombination von 1 & 2** → Erst Ballon als Daten-unabhängigen Algorithmus und dann scrypt zur Hinderung von parallelisierten Angriffe  
→ Dieses Konstrukt schützt nicht gegen einen Angreifer, der Daten-Muster aufzeichnen kann und einen parallelisierten Angriff durchführt

[1] Boneh, Dan; Corrigan-Gibbs, Henry; Schechter, Stuart (2016): Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In Jung Hee Cheon, Tsuyoshi Takagi (Eds.): Advances in Cryptology – ASIACRYPT 2016, vol. 10031. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 220–248.

## Optionen für den praktischen Einsatz

### Weitere Vorschläge (Meine Meinung / Werte für 2023)

**Password Hashing:** Nutzerauthentifizierung bei einem Online-Dienst

- **Bcrypt** Kostenfaktor ca. so, dass die Berechnung auf der genutzten Hardware 100ms dauert
- **Scrypt** mit  $N = 32768$ ,  $r = 8$ ,  $p = 1$  und einem mind. 128bit Salt (besser 256)
- **Argon2id** mit 64 MiB Speicher Kosten, Zeitkosten von 3 und Parallelismus 1

## Optionen für den praktischen Einsatz

### Weitere Vorschläge (Meine Meinung / Werte für 2023)

**Password-based Key Derivation:** Schlüssel für Verschlüsselung aus einem Passwort ableiten

- **Scrypt** mit  $N = 1048576$ ,  $r = 8$ ,  $p = 1$  und einem mind. 128bit Salt (besser 256)
- **Argon2id** mit 1024 MiB Speicherkosten, Zeitkosten von 4 und Parallelismus 1