University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Homomorphe Verschlüsselung – Theorie und Praxis

Marvin Hahn
Julian van Dyken

Exposé für die Lehrveranstaltung
Spezielle Verfahren der IT-Sicherheit

im Studiengang
Informatik

Hochschule Emden/Leer
Prof. Dr. Patrick Fehlke
M.Eng. Frederik Gosewehr

5. Oktober 2025

Inhaltsverzeichnis

1 Einleitung

Normalerweise muss man verschlüsselte Daten entschlüsseln um damit Operationen durchzuführen z.B. Aufaddieren der Stimmen bei einer Wahl oder Daten für Maschinelles lernen. Der Nutzer muss daher einen Teil seiner Privatsphäre einbüßen, damit diese Verarbeitet werden können. Das birgt Risiken, weil die Informationen wer für wen gestimmt hat sehr sensibel sind und man auch nicht unbedingt Google seine sensiblen Daten für Maschinelles lernen zur Verfügung stellen möchte. Mit Homomorpher Verschlüsselung gibt es die Möglichkeit solche Operationen direkt auf den verschlüsselten Daten durchzuführen. Dadurch bleiben sensible Informationen geschützt. Auf die Beispiele von eben bezogen bedeutet das, dass es möglich ist ein Modell zu haben, was die Daten Homomorph auswertet, dass bedeutet, man gibt seine Informationen verschlüsselt an einen Anbieter, dieser kann diese dann auswerten und Berechnungen ausführen. Das Ergebnis dieser Berechnungen ist dann immer noch verschlüsselt. Dann bekommt man das Ergebnis zurück und kann es einfach entschlüsseln. Es ist also so, als ob die Berechnungen direkt auf dem Entschlüsselten Daten durchgeführt wurden.

2 Das Ziel der Ausarbeitung

2.1 Ziel

Grundlegende Ausarbeitung der Theorie (grundlegende Idee) hinter Homomorpher Verschlüsselung und deren Praktische Anwendung. Dabei gehen wir insbesondere auf die verschiedenen Arten der Homomorphen Verschlüsselung ein (partial, somewhat, full). Dabei werden wir beantworten, was genau diese Arten bedeutet und Beispiele für praktische Anwendungen geben. Dazu wird es eine Bewertung, ob und welche dieser Arten tatsächlich Praktisch einsetzbar sind geben, auf Basis der aktuellen Forschung.

3 Centralized vs Distributed

Versionskontrollsysteme werden hauptsächlich in zentrale (von engl. *centralized*) und verteilte (von engl. *distributed*) Systeme unterschieden. Daneben gibt es noch lokale Versionskontrolle[1], die sich nur auf einen Rechner und meist nur ein Dokument beschränkt.

[1] Siehe Scott und S. Ben 2010, S. 1.

3.1 Centralized

Zentralte Versionskontrollsysteme sind mit dem Client-Server abgebildet, wobei der Server meist nur Schnappschüsse, sprich die Versionen von Dateien speichert[2] und die Beteiligten sich nur mit dem Server synchronisieren.

3.2 Distributed

Der Begriff *verteilt* bedeutet in diesem Kontext, dass alle Beteiligten durch lokale Repositories die komplette Historie aller Dateien haben und sich untereinander synchronisieren (Peer-to-Peer).
Man spricht nicht mehr von Versionen, sondern von Revisions, also Änderungen an einer Datei (zb. neue Zeile 10: "Hallo Welt").[3]

3.3 Vor-/Nachteile

Hier werden die jeweiligen Vor- und Nachteile der beiden Systeme betrachtet und als Übersicht in Trade-offs verpackt.
Beispiel zentrale Systeme: Abhängigkeit eines Servers dafür mehr Kontrolle des Workflows (zb. durch Locking[4])
Beispiel verteilte Systeme: Es läuft offline und schnell dafür keine „neuste Version“[5]

4 Praxis

Aufgrund der in der Einführung (1) beschriebenen Probleme bei Softwareentwicklung finden Versionskontrollsysteme in der Praxis ihren Einsatz

4.1 Anwendungsfälle

Dabei kann man die Systeme nicht nur für ihren ursprünglichen Zweck[6], das Versionieren von Quellcode in Softwareprojekten nutzen, sondern auch für allmöglichen Dateien, bei denen man eine Historie haben will und die man ggf. als Team erstellt.

[2] Vgl. Kivanc u. a. 2014, S. 671.
[3] Siehe Azad o. J. für anschauliche Beispiele.
[4] Erklärung des Mechanisumses C. Michael, C.-S. Ben und Brian 2008, S. 3-4.
[5] Vgl. Azad o. J.
[6] Vgl. Blischak, Davenport und Wilson 2016.

4.2 Vorhandene Software

Beide Arten sind nicht nur theoretisch erklärbar, sondern sind bereits technisch umgesetzt und finden in Unternehmen als Standardsoftware ihren Einsatz.

4.2.1 Centralized

Durch das Open-Source-Projekt Concurrent Versions System wurde diese Art erstmalig populär und im Jahr 2000 mit Subversion[7] neu implementiert[8].

4.2.2 Distributed

An dieser Stelle sind vor allem *Git*[9] sowie das Konkurenzprodukt *Mercurial*[10] zu nennen. Beide fallen unter die Kateogrie freie Software[11] doch unterscheiden sich in Komplexität und Erlernbarkeit[12]

5 Fazit

Hier wird die Arbeit zusammengefasst und wieder auf die Stärken der beiden Arten eingegangen um ihre jeweilige Daseinsberechtigung zu untermauern.

[7] siehe C. Michael, C.-S. Ben und Brian 2008, für eine umfangreiche Einführung.
[8] siehe C. Michael, C.-S. Ben und Brian 2008, S. xi Anhang B für Unterschiede der Implementierungen.
[9] Siehe Scott und S. Ben 2010, für eine umfangreiche Einführung.
[10] Siehe Bryan 2009, für eine umfangreiche Einführung.
[11] Definition Grassmuck 2004, S.281-284.
[12] Vgl. Kirmanen 2017.

Literatur

Azad, Kalid (o. J.). *Intro to Distributed Version Control*. URL: https://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/ (besucht am 31. 05. 2019).

Blischak, John D., Emily R. Davenport und Greg Wilson (Jan. 2016). „A Quick Introduction to Version Control with Git and GitHub“. In: *PLOS Computational Biology* 12.1, S. 1–18. DOI: 10.1371/journal.pcbi.1004668. URL: https://doi.org/10.1371/journal.pcbi.1004668.

Bryan, O'Sullivan (2009). *Mercurial: The Definitive Guide*. O'Reilly Media.

C. Michael, Pilato, Collins-Sussman Ben und W. Brian (2008). *Version Control with Subversion: Next Generation Open Source Version Control*. 2. Aufl. O'Reilly Media.

Grassmuck, Volker (2004). *Freie Software. Zwischen Privat- und Gemeineigentum*. deu. Bundeszentrale für politische Bildung. ISBN: 3893315691. DOI: http://dx.doi.org/10.25969/mediarep/3575. URL: https://mediarep.org/handle/doc/4317.

Kirmanen, Antti (2017). *Git oder Mercurial: Was sollte man nutzen?* URL: https://entwickler.de/online/development/git-mercurial-versionskontrollsystem-vergleich-579792235.html (besucht am 29. 05. 2019).

Kivanc, Muslu u. a. (2014). „Transition from centralized to decentralized version control systems: a case study on reasons, barriers, and outcomes“. In: *ICSE*.

Scott, Chacon und Straub Ben (2010). *Pro Git*. 2. Aufl. Apress.