Nibyou | FOR A BETTER HEALTH.

Lösungen für Voraussetzungen an die Kryptographie von Gesundheitsdatenplattformen

Luca Kiebel, CTO
luca.kiebel@nibyou.de

Stand: 29. Juni 2022

Inhaltsverzeichnis

1 Einleitung

Nibyou entwickelt eine cloud-basierte Gesundheitsdatenplattform (GDP) gleichen Namens. Diese Plattform nutzt Ende-zu-Ende-Verschlüsselung und Verschlüsselung-im-Ruhezustand um Ihren Nutzern sichere Speicherung und sicheres Teilen ihrer persönlichen Gesundheitsdaten zu ermöglichen. Ziel dieser Plattform ist die Verbreitung einer Praxissoftware für den ernährungstherapeutischen Markt, mithilfe welcher Experten[1] und Patienten aktuelle und vergangene Gesundheitsdaten sicher miteinander austauschen können.
Dieses Dokument beschreibt im weiteren, welche Schritte in GDP genutzt werden können, um Voraussetzungen an Datensicherheit zu erfüllen.

1.1 Ende-zu-Ende Verschlüsselung

Während der Registrierung eines neuen Nutzerkontos wird unter anderem ein sogenanntes Schlüsselpaar erstellt, bestehend aus einem öffentlich bekannten *public key* und einem geheimen *private key*. Diese Schlüssel werden eingesetzt um generierte Daten Ende-zu-Ende, also z.B. vom Patienten zum Experten, zu verschlüsseln. Auf den dadurch entstehenden gesicherten Bereich können Nutzer durch eine Front-End Anwendung, wie unsere Web-App, zugreifen.

In diesem gesicherten Bereich können Patienten und Experten Daten hochladen, und auch wieder darauf zugreifen. Zu keinem Zeitpunkt hat Nibyou dabei Zugriff auf unverschlüsselte Daten.

Die Daten können vom Patienten mit anderen Experten einfach geteilt werden, indem im System datensatz- oder datensortenspezifische Schlüssel ausgetauscht werden.

1.2 Verschlüsselung-im-Ruhezustand

Die weniger sicherheitskritischen Daten, wie Namen oder Versichertennummern werden auf unseren Servern ebenfalls verschlüsselt gelagert. Diese Verschlüsselung basiert auf dem sogenannten *Encryption-at-rest* (EAT) Prinzip. *At-rest* bedeutet, dass die Daten dann verschlüsselt werden, wenn sie an ihrem Ziel, also unseren Servern, angekommen sind und wieder entschlüsselt werden, sobald sie bewegt, also Beispielsweise an unsere Frontend-Anwendungen geschickt, werden. Hierzu verwenden wir die EAT Integration von MongoDB Enterprise[1], sowie Festplattenverschlüsselung auf unseren Servern, und, wie auch in der Ende-zu-Ende Verschlüsselung, AES256-GCM, mehr dazu in 2.1.2.

[1] Diätassistenten, Ökotrophologen, Ernährungsmediziner oder Ernährungswissenschaftler

2 Verschlüsselungsmethoden

Neben den Erklärungen zu dem Thema Verschlüsselungen in diesem Dokument, empfehlen sich die Wikipedia Artikel zu den Themen Verschlüsselung, dem RSA-Kryptosystem (oder allgemein zu asymmetrischen Kryptosystemen) und den Artikeln zu AES oder analog allgemein zu symmetrischen Kryptosystemen. Daneben empfiehlt sich die Standardliteratur [2]–[6].

2.1 Symmetrische Verschlüsselung

Verschlüsselung ist das grundlegende Ziel der sicheren Kryptographie. Sie macht Daten unverständlich und sorgt somit für Vertraulichkeit der Daten. Die zu verschlüsselnden Daten heißen Nachrichten (m für message), oder einfach Text. Einem Algorithmus namens Chiffre (c für cipher) wird die Nachricht übergeben und in diesem mit einem Schlüssel (k für key) verschlüsselt. Das Ergebnis dieser Funktion heißt Chiffrat (C für cyphertext). Wird derselbe Schlüssel für ver- und entschlüsselung verwendet, handelt es sich um einen symmetrischen Algorithmus oder eine symmetrische Chiffre.

2.1.1 Advanced Encryption Standard (AES)

Teil unseres Kryptographiesystems ist der Einsatz des Advanced Encryption Standards, der seit 2001 der Standard für Verschlüsselungen des US-Militärs ist und hier für Dokumente mit höchster Geheimhaltungsstufe eingesetzt werden darf[7].
AES unterstützt regelmäßig, aber abhängig von der Implementation, drei Schlüssellängen: 128, 192 und 256 Bits. Bei Nibyou wird ausschließlich AES-256 verwendet, um maximale Sicherheit zu garantieren.

2.1.2 AES Betriebsmodus

Um größere Datenmengen als die Blocklänge von AES (also 128 Bit) verschlüsseln zu können, muss der gewählte Algorithmus mit einem sogenannten Betriebsmodus verwendet werden. Größere Datenmengen müssen also in geeigneter Weise in einzelne Abschnitte geteilt werden.
Der naive Weg, einfach die Nachricht m in n Bestandteile der Länge 128 Bit aufzuteilen und einzeln (z.B. parallel) zu verschlüsseln ist zwar schnell, führt aber auch dazu, dass die Reihenfolge der Abschnitte nicht garantiert werden kann.
Betriebsmodi, wie der *Cipher Block Chaining Mode* garantieren zwar die Reihenfolge, lassen sich aber nicht parallel durchführen.
Der Galois/Counter Mode (GCM) (AES-128-GCM) funktioniert jedoch ganz anders. Wie der Name schon sagt, kombiniert GCM die Galois-Feldmultiplikation mit der Betriebsart des Zählerverfahrens für Blockchiffren. Der Zählermodus wurde entwickelt, um Blockchiffren in Stromchiffren zu verwandeln, bei denen jeder Block mit einem Pseudozufallswert aus einem SSchlüsselstrom"verschlüsselt wird. Bei diesem Konzept wird

dies durch die Verwendung aufeinanderfolgender Werte eines inkrementierenden SZählersërreicht, so dass jeder Block mit einem einmaligen Wert verschlüsselt wird, der wahrscheinlich nicht wieder vorkommt. Die Galois-Feld-Multiplikationskomponente erweitert dieses Konzept, indem jeder Block als ein eigenes endliches Feld für die Verschlüsselung auf der Grundlage des AES-Standards betrachtet wird.

AES-GCM wird parallel ausgeführt, was bedeutet, dass der Durchsatz deutlich höher ist als bei AES-CBC, da der Verschlüsselungs-Overhead verringert wird. Jeder Block mit AES-GCM kann unabhängig verschlüsselt werden. Die Arbeitsweise von AES-GCM kann sowohl bei der Verschlüsselung als auch bei der Entschlüsselung parallel durchgeführt werden.

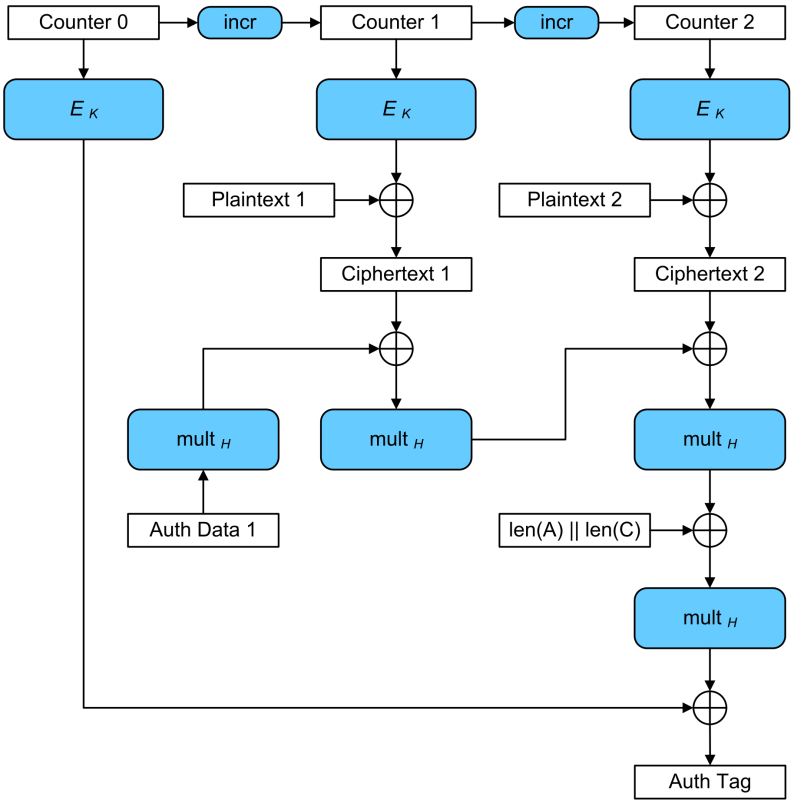

Abbildung 1: Galois/Counter Mode

2.2 Asymmetrische Verschlüsselung

Das gravierendste Problem von symmetrischer Verschlüsselung ist, dass Schlüssel über einen bereits gesicherten Kanal ausgetauscht werden, oder über Methoden, wie den Diffie-Hellman-Schlüsselaustausch zuerst generiert werden müssen. Beide Methoden sind für unseren Anwendungsfall nicht nutzbar. Wir können auf der einen Seite nicht davon ausgehen, dass Schlüssel sicher ausgetauscht werden können, und auf der anderen

Schlüssel nicht von nur zwei Seiten aus erstellen lassen, und Daten mehr als zwei Seiten zugänglich machen.

Bei der asymmetrischen Verschlüsselung gibt es unterschiedliche Schlüssel zum ver- und entschlüsseln einer Nachricht. Jeder Nutzer in einem solchen Kryptosystem besitzt ein Schlüsselpaar aus zwei mathematisch verwandten Zahlen. Daten die mit einem der beiden Schlüssel verschlüsselt werden können ausschließlich mit dem anderen wieder entschlüsselt werden. Das erlaubt zwei Nutzungsmöglichkeiten: Verschlüsselung mit dem öffentlichen *public key*, wonach das Chiffrat nur mit dem geheimen *priavte key* entschlüsselt werden kann, und Signierung mit dem *priavte key*, wonach mit dem *public key* sichergestellt werden kann, dass nur der Eigentümer des *private key* eine gegebene Nachricht hätte erstellen können.

2.2.1 Rivest–Shamir–Adleman (RSA)

Die Sicherheit der Verwendung zweier mathematisch verwandter Schlüssel beruht auf der Benutzung von sogenannten Einwegfunktionen, also Funktionen, die in eine Richtung einfach durchzuführen sind, in die andere aber mit sinnvollem Aufwand unmöglich aufzulösen sind. Bei RSA wird hierzu eine Eigenschaft der Faktorisierung von großen Zahlen genutzt: während die Multiplikation zweier Primzahlen für Computer und Menschen sehr einfach ist, ist die umgekehrte Zerlegung einer bekannten Zahl in ihre Primfaktoren sowohl für Menschen, als auch für Computer quasi nur durch Ausprobieren nach dem Brute-Force-Muster möglich. Selbst mit aktuellester Technik und riesigen Quantencomputern ist das „knacken“ eines einzelnen 2048-Bit RSA Schlüssels kaum möglich. C. Gidney und M. Ekerå theorisieren, dass mit 20 Millionen Qubits RSA-2048 in acht Stunden geknackt werden kann[8]. Die größten Quantencomputer sind eher in der Größe von 50 Qubits[9] nicht gerade zeiteffizient.

2.3 Verwendete Zwischenlösung

Wie man erkennen kann haben symmetrische und asymmetrische Verschlüsselungen einige Vor- und Nachteile. Asymmetrische Verschlüsselung ist zwar um einige Zehnerpotenzen langsamer als symmetrische, aber deutlich übersichtlicher im Schlüsselmanagement. Um die Vorteile beider Verschlüsselungsmethoden nutzen zu können, wird bei Nibyou folgende Hybridverschlüsselung verwendet:

Eine Nachricht m wird mit einem symmetrischen Schlüssel k zu dem Chiffrat C_m verschlüsselt. Dieser Schlüssel wird zum Austausch mit dem Empfänger mit dem *public key* E_b verschlüsselt und an diesen gesendet. Der Empfänger kann nun mit seinem *private key* D_b in einer eher zeitintensiven Operation k wieder entschlüsseln und mit diesem Schlüssel c_m zu m in deutlich weniger Zeit, als dies mit einem asymmetrischen Kryptographiesystem möglich wäre, entschlüsseln. Üblicherweise werden Sender und Empfänger Alice (a) und Bob (b) genannt.

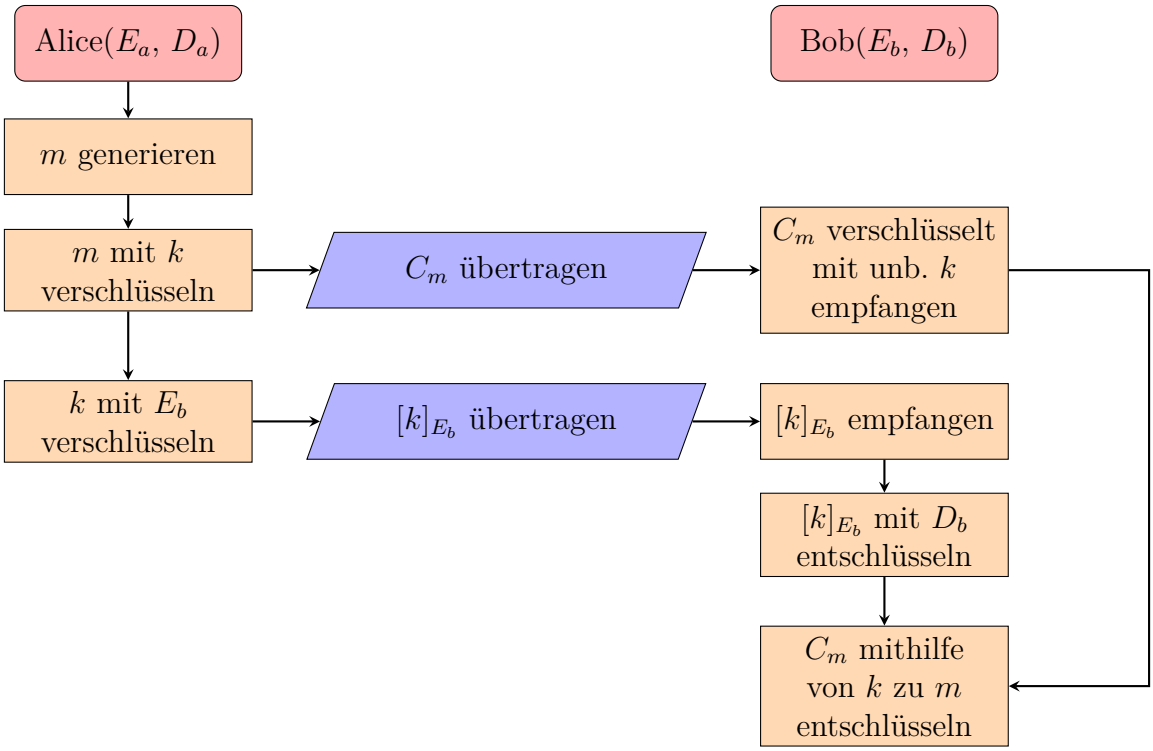

Abbildung 2: Nachrichtenübertragung mit Ende-zu-Ende Verschlüsselung

Soll hier nun ein weiterer Teilnehmer hinzugefügt werden, muss Alice lediglich C_m übertragen, und k mit dem öffentlichen Schlüssel des neuen Teilnehmers E_c zu $[k]_{E_c}$ verschlüsseln und versenden. Alice spart sich also die erneute Verschlüsselung der Nachricht m, und muss nur das kleine Datum k verschlüsseln[2].

[2]Für den Fall, dass m kleiner als 128 Bit ist, wäre das mögliche Zeitersparnis durch direkten asymmetrischen Austausch zu vernachlässigen.

In der Praxis sorgt diese Methode allerdings dafür, dass sehr viele Schlüssel übertragen werden müssen. Nibyou setzt daher auf sogenannte *common-keys* (*ck*). Diese Schlüssel sind einzigartig für eine bestimmte Gruppe, Sorte oder für einen bestimmten Satz an Daten. So muss beispielsweise zum Teilen der Gewichtsdaten eines Patienten mit mehreren Experten nur ein Schlüssel getauscht werden. In der Datenbank sind die Gewichtsdaten dann mit der UUID des *common-keys* verknüpft. Mit dieser Methode lässt es sich auch erreichen einen Datensatz mathematisch zeitlich einzugrenzen, indem ein neuer *common-key* für zukünftige Daten erstellt wird.

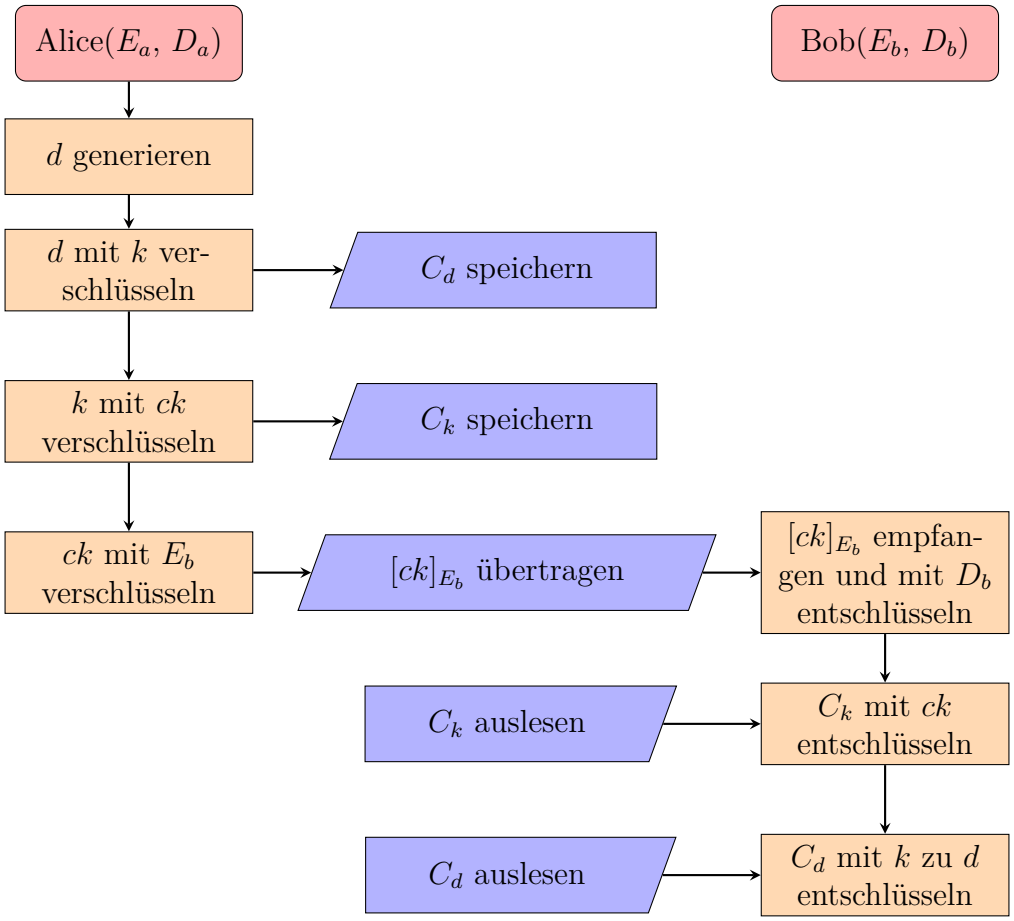

Abbildung 3: Datensicherung mit Ende-zu-Ende Verschlüsselung und *common-keys*

Die Übertragung des verschlüsselten *common-keys* muss nur einmal stattfinden, danach kann Bob selbstständig von Alice symmetrisch verschlüsselte Daten (C_d und C_k) aus der Datenbank auslesen.

3 Schlüssel

Um die in der GDP gespeicherten Daten zu verschlüsseln, werden zu verschiedenen Zeitpunkten diverse Schlüssel angelegt. Die Übertragung der Daten ist durch TLS 1.2 und TLS 1.3[10] (wo möglich[3]) Ende-zu-Ende verschlüsselt.

3.1 Asymmetrisches Schlüsselpaar des Nutzers (RSA)

Bei der Accounterstellung wird ein Schlüsselpaar für die Nutzung im RSA-Kryptosystem mit 2048 Bit Länge generiert. Dieses Schlüsselpaar besteht für die gesamte Lebensdauer des Accounts. Der öffentliche Schlüssels des Paares wird unverschlüsselt mit der UUID des Nutzers in der Datenbank gespeichert und kann so von allen anderen Nutzern zur Verschlüsselung von Nachrichten, die für den Nutzer bestimmt sind, genutzt werden.

3.1.1 Probleme mit dem Umgang mit privaten RSA-Schlüsseln

Der private Teil des Schlüsselpaares kann nicht unverschlüsselt in der Datenbank gespeichert werden. In dem typischen Einsatzgebiet von asymmetrischen Kryptosystemen verlässt der private Schlüssel nie das Gerät, auf dem er erstellt wurde. Diese Abschottung sorgt dafür, dass kein Dritter den Nutzer nachahmen kann, weder indem er Nachrichten, die an den Nutzer gerichtet sind, entschlüsselt, noch indem er sich beim Signieren als der Nutzer ausgibt.

Diese Methode zur Sicherung des privaten Schlüssels funktioniert allerdings nur kaum für Cloud-Anwendungen wie Nibyou. Sobald ein Nutzer mit mehreren Geräten auf die gleiche Datenbasis zugreifen, und diese entschlüsseln können will, muss der private Schlüssel zwischen den diversen Geräten kopiert werden. Dieser Vorgang ist mühselig und erfordert dem Nutzer ein sehr hohes Maß an Sicherheitsbewusstsein im Umgang mit den Schlüsseln ab. Besser ist es das erstellte Schlüsselpaar nie in die Nähe des Nutzers zu bringen.

3.1.2 Speicherung des privaten RSA-Schlüssels

Nibyou setzt zum Speichern des privaten Schlüssels das Nutzerpasswort ein. Auf dem Gerät des Nutzers wird zum Zeitpunkt der Accounterstellung ein RSA-Schlüsselpaar erstellt. Der öffentliche Schlüssel kann, wie oben beschrieben sofort veröffentlich werden. Zur Sicherung des privaten Schlüssels wird auf dem Gerät des Nutzers ein Schlüssel aus dem eingegebenen Passwort erstellt. Dies geschiet mithilfe der *Password-Based Key Derivation Function 2 (PBKDF2)* von den Entwicklern von RSA, und sorgt so unabahägig von der Komplexität des Nutzerpasswortes[4] für sichere symmetrische Schlüssel

[3] Unser Infrastrukturpartner Cloudflare ist bemüht, TLS 1.3 schnellstmöglich in alle modernen Browser zu integrieren und so möglichst schnell für die Verbreitung der neuen und schnellsten Sicherheitsstandards im Netz zu sorgen[11]

[4] Nutzer sind dennoch angehalten sichere Passwörter zu erstellen (8 Zeichen Mindestlänge, Groß- & Kleinbuchstaben, Zahlen und Sonderzeichen) und werden auf die Benutzung von Passwortmanagern

um damit den privaten Schlüssel verschlüsselt in der Datenbank zu speichern. Dieses Schlüsselderivat heißt kd_{pw} und das Nutzerpasswort pw.

3.1.3 Wiederherstellungsschlüssel für den privaten RSA-Schlüssel

Da Nibyou zur Verschlüsselung des privaten Schlüssels das Nutzerpasswort nutzt, besteht die Gefahr, dass der Nutzer sein Passwort vergisst und Zugang seinem Benutzerkonto verliert. Dies würde bedeuten, dass alle zuvor gesendeten Nachrichten und alle zuvor erstellten Dokumente unbrauchbar mit einem öffentlichen Schlüssel verschlüsselt wurden, zu dem es keinen privaten Schlüssel mehr gibt[5]. Um dieser Möglichkeit entgegenzuwirken wird zur Accounterstellung ebenfalls ein Wiederherstellungsschlüssel erstellt. Dieser Schlüssel ist eine BIP-39[6] Mnemonik[12] von einer zufällig generierten 128 Bit großen Zahl. Ein solcher Wiederherstellungsschlüssel könnte wie folgt aussehen:

0x7ff	0x034	0x2d1	0x607	0x08b	0x1cc
zylinder	altertum	gitarre	roggen	asteroid	eiszeit

Abbildung 4: BIP-39 Repräsentation eines Wiederherstellungsschlüssels

Hierzu stellt Nibyou eine Liste an Wörtern bereit, die den Wiederherstellungsschlüssel abbilden[7]. Zur Verschlüsselung wird auch auf diesen Schlüssel die PBKDF2 ausgeführt. Das Schlüsselderivat heißt kd_{rk} und der Wiederherstellungsschlüssel rk.

Der Wiederherstellungsschlüssel wird dem Nutzer bei der Accounterstellung einmalig angezeigt. Dem Nutzer wird die Möglichkeit gegeben eine PDF mit dem Schlüssel auszudrucken. Nach bestätigung, dass der Schlüssel gesichert wurde, geht der Accounterstellungsvorgang weiter. Wurde der Wiederherstellungsschlüssung verwendet muss er, wie das Passwort neu erstellt werden. Unabhängig von diesen Vorgängen können sowohl Passwort als auch Wiederherstellungsschlüssel beliebig oft geändert werden. Die Wiederherstellung funktioniert, indem neben dem mit dem Nutzerpasswort verschlüsselten privaten Schlüssel noch der private Schlüssel mit dem mit Wiederherstellungsschlüssel verschlüsselt, gespeichert wird. So können beide „Passwörter“ genutzt werden um den privaten Schlüssel zu entschlüsseln.

3.2 Common Key (ck)

Es wird bei der Accounterstellung zunächst ein Common Key pro Datensorte erstellt. Mit diesen können symmetrische Datenschlüssel (in vorherigen Diagrammen nur k ge-

hingewiesen.

[5] Vergleiche 2.2.1. Könnte man sinvoll vom öffentlichen den privaten Schlüssel ableiten, wäre RSA unsicher.

[6] BIP-39 ist ein *Bitcoin Improvement Proposal*, also ein Vorschlag die Architektur von Bitcoin zu verbessern. Mehr dazu hier.

[7] nibyou/bip-39-de

nannt) verschlüsselt gespeichert werden. Somit werden alle Datensätze mit einem eigenen Schlüssel verschlüsselt, und diese Schlüssel können sicher mit einem Common Key pro Datensorte gespeichert werden. In der Datenbank müssen Datensätze oder Datensorten nicht direkt an einen Nutzer gebunden sein, sondern sind an eine Common Key UUID verbunden, der Schlüssel wird geteilt und Nutzer können so auf die geteilten Daten zugreifen.

Bei Chats und Gruppenchats auf der Nibyou Plattform werden diese Schlüssel auch verwendet, um Nachrichten zu verschlüsseln. Dies erlaubt es, pro Chatraum nur einen Common Key mit anderen Teilnehmnern teilen zu müssen. Auch vergangene Nachrichten können so, ohne sie zwischendurch mühselig zu entschlüsseln, mit neuen Teilnehmern geteilt werden.

3.3 Streuwertschlüssel

Wie jedes moderne Programm kennt auch Nibyou die Passwörter ihrer Kunden nicht. Um dies sicherzustellen werden Passwörter durch sogenannte Hashingfunktionen mit Einweg-Verschlüsselung in Streuwerte umgewandelt, die nach der Umwandlung nicht mehr in das eigentliche Passwort zurückverwandelt werden können. Nibyou setzt hierzu zwei Hashing-Algorithmen ein. Zur Registrierung und später zum Login wird das Passwort des Nutzers im Klartext an den Server gesendet und hier mit dem aktuell sichersten[13] Hashingalgorithmus verschlüsselt. Die Sicherheit von Hashingalgorithmen hängt heutzutage besonders von zwei Faktoren ab:

- Wie sicher ist es, dass keine zwei ungleichen Klartexte zu gleichen Streuwerten werden?
- Wie schwer ist es Systeme zu bauen, die möglichst viele Hashes pro Zeit erstellen können?

Alle heute gängigen Hashingalgorithmen haben mit dem ersten Faktor (sog. Kollisionen) keine Probleme mehr[8].
Der Zweite Faktor ist weniger einfach zu beseitigen. Früher (und auch zum Teil noch heutzutage) wurden Hashingalgorithmen verwendet, die es relativ einfach machten eine riesige Anzahl an Hashes mit günstiger Hardware in kürzester Zeit zu erstellen[9]. Moderne Algorithmen nutzen daher sogenannte *work factors*, wie z.B.:

- Wie häufig wird der Algorithmus angewendet?

[8] Man kann Kollisionen zwar nie ausschließen (mit unendlich viel Zeit lässt sich mit einem handelsüblichen Rechner für jeden Hashingalgorithmus eine Kollision finden, einfach da die Menge der erstellbaren Streuwerte endlich, die Menge der Klartexte aber unendlich ist), aber die Wahrscheinlichkeit soweit drücken, dass bei Argon2 diese 2^{-113} beträgt. [14]

[9] Zum Vergleich, ein GPU Cluster, der die Software des HashCat Projekts ausführt, kam 2012 auf 180 Milliarden *MD5* Hashes pro Sekunde (Link). Die Wahrscheinlichkeit Kollisionen in *MD5* zu finden beträgt ca. $1/2^{64}$, man kann also mit 100 dieser Cluster in unter 2 Wochen für jeden möglichen *MD5* Hash eine Kollision finden.

- Wie viel Arbeitsspeicher wird verwendet?
- Wie viele Threads werden gleichzeitig genutzt um den Streuwert zu ermitteln?

Diese sorgen dafür, dass selbst auf modernster Hardware Implementationen von modernen Hashingalgorithmen mehrere Dutzend bis mehrere Hundert Millisekunden zur Ausführung brauchen.
Der Aktuell beste Hashingalgorithmus in der Abwehr von Angriffen mit riesiger Anzahl and GPUs ist *Argon2*[15].
Nibyou verwendet demnach *Argon2* zur sicheren Verwahrung der Nutzerpasswörter, GDP sollten sich generell an aktuelle Studien zu dem Thema Hashing wenden, da vor allem mit den Vorsprüngen im Bereich Quantum Computing bewährte Hashingalgorithmen jederzeit unsicher werden können.
Neben der Speicherung des Passworts muss dieses auch zur Verschlüsselung des privaten Schlüsselteils des RSA-Schlüsselpaares genutzt werden.

4 Datenspende

4.1 Datenprozessoren

Nibyou bietet gemeinnützigen Gesundheitsorganisationen und Universitäten die Möglichkeit, sich für eine verbesserte Datenversorgung von Forschern und Experten als sogenannte Datenprozessoren (DP) zu registrieren. Als DP hat eine Organisation die Möglichkeit anonymisierte Daten von Nutzern, die ein Teilen mit spezifischen DP freigegeben haben, zur statistischen Auswertung zu Analysieren. So kann auf Dauer beispielsweise besser untersucht werden, welche Auswirkungen bestimmte Behandlungsmethoden haben, und den Experten so mögliche Methoden in der Behandlung vorgeschlagen werden. Nibyou selbst setzt das Prinzip der Datenspende ein um genau so Experten Vorschläge für Behandlungen zu machen. Nibyou wird niemals aufgrund von statistischen Daten Diagnosen erstellen, oder direkt Vorschläge für Behandlungen an Patienten senden.

4.2 Vorgänge zur Sicherstellung der Anonymisierung

Um sicherzustellen, dass Daten, die von DP mit beispielsweise Machine Learning Algorithmen analysiert werden sollen,
1. nicht auf einzelne Patienten zurückverfolgt werden können, sowie
2. nicht von dritten Nutzern für einen Patienten erstellt (gefälscht) werden können,
müssen folgende Voraussetzungen erfüllt werden:

4.2.1 Datenanonymisierung

Die geteilten Gesundheitsdaten werden vor dem Teilen vom Server geladen, auf dem Gerät entschlüsselt, von jeglichen eindeutigen Identifikationsmerkmalen bereinigt und, wie in 4.2.2 erklärt, wieder verschlüsselt. Das bedeutet, die geteilten Daten beinhalten keine UUIDs, Email-Adresen, Namen, Adressen, usw. Somit kann sichergestellt werden, dass DPs die Nutzer nicht anhand der erhaltenen Daten deanonymisieren können.

4.2.2 Datensignierung & Fälschungssicherheit

Den anonymisierten Daten wird der öffentliche Teil eines RSA-Schlüsselsatzes (E_{ds}) beigefügt. Der Schlüsselsatz wird ähnlich dem RSA-Schlüsselsatz des Nutzers, mit dem PBKDF2-Derivat des Passwortes im Profil des Nutzers gespeichert, dadurch wird sichergestellt, dass der Nutzer nicht anhand von E_{ds} erkannt werden kann. Ein Hashwert dieses Schlüssels kann immer wieder zur Zuordnung von Datensätzen im System des DP genutzt werden. Um zu versichern, dass nur der Nutzer, der die Daten erstellt hat diese auch mit dem öffentlichen Schlüssel teilt, werden die Daten mit dem privaten Schlüssel des Satzes (D_{ds}) verschlüsselt. Diese umgekehrte Verschlüsselung sorgt dafür, dass der DP feststellen kann, ob die gesendeten Daten sich mit dem beigefügten öffentlichen Schlüssel E_{ds} wieder entschlüsseln lassen. So kann nur ein Nutzer, der sowohl E_{ds}, als auch D_{ds} produzieren kann auch die Daten teilen. Die Überprüfung dieser Fälschungssicherung obliegt dem DP.

Literatur

[1] MongoDB, Inc., *Encryption at Rest*, [Online; Stand 29. Juni 2022]. Adresse: https://docs.mongodb.com/manual/core/security-encryption-at-rest/.

[2] F. L. Bauer, *Entzifferte Geheimnisse. Methoden und Maximen der Kryptologie*, 3. Aufl. Springer Verlag, 2000, ISBN: 3-540-67931-6.

[3] L. A. Bertram und G. van Dooble, et al., *Nomenclatura – Encyclopedia of modern Cryptography and Internet Security. From AutoCrypt and Exponential Encryption to Zero-Knowledge-Proof Keys*. Books on Demand, 2019, ISBN: 978-3746-06668-4.

[4] J. A. Buchmann, *Einführung in die Kryptographie*, 4. Aufl. Springer Verlag, 2008, ISBN: 978-3-540-74451-1.

[5] B. Schneier, *Angewandte Kryptographie. Protokolle, Algorithmen und Sourcecode in C*. Pearson Studium, 2006, ISBN: 3-8273-7228-3.

[6] E. J. Topol, *Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again*. Basic Books, Inc., Mär 2019, ISBN: 978-1-5416-4463-2.

[7] Wikipedia, *Advanced Encryption Standard — Wikipedia, Die freie Enzyklopädie*, [Online; Stand 29. Juni 2022], 2021. Adresse: https://de.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=211518819.

[8] C. Gidney und M. Ekerå, „How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits,“ *Quantum*, Jg. 5, S. 433, Apr. 2021, ISSN: 2521-327X. DOI: 10.22331/q-2021-04-15-433. Adresse: http://dx.doi.org/10.22331/q-2021-04-15-433.

[9] S. E. Yunakovsky, M. Kot, N. O. Pozhar u. a., *Towards security recommendations for public-key infrastructures for production environments in the post-quantum era*, 2021. arXiv: 2105.01324 [quant-ph].

[10] Internet Engineering Task Force (IETF), E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, [Online; Stand], Aug. 2018. Adresse: https://datatracker.ietf.org/doc/html/rfc8446.

[11] Cloudflare, Inc., F. Valsorda, *An overview of TLS 1.3 and QA*, [Online; Stand 29. Juni 2022], Sep. 2016. Adresse: https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a.

[12] Wikipedia, *Mnemotechnik — Wikipedia, Die freie Enzyklopädie*, [Online; Stand 29. Juni 2022], 2021. Adresse: https://de.wikipedia.org/w/index.php?title=Mnemotechnik&oldid=207274041.

[13] J.-P. Aumasson, T. Arcieri, D. Chestnykh u. a., *Password Hashing Competition*, [Online; Stand 29. Juni 2022]. Adresse: https://www.password-hashing.net.

[14] A. Biryukov, D. Dinu und D. Khovratovich, *Argon and Argon2*, University of Luxembourg, Hrsg., [Online; Stand 29. Juni 2022]. Adresse: https://orbilu.uni.lu/bitstream/10993/19901/1/Argon.pdf.

[15] A. Biryukov, D. Dinu und D. Khovratovich, *Argon2: the memory-hard function for password hashing and other applications*, University of Luxembourg, Hrsg., [Online; Stand 29. Juni 2022]. Adresse: https://raw.githubusercontent.com/P-H-C/phc-winner-argon2/master/argon2-specs.pdf.

Abbildungsverzeichnis