Introduction to Internet and Security

GRUPPE 08: PEER-TO-PEER GEOCACHING APP

12. Juli 2021

Emmanuel J. Arackal Kapilas Gnanarajah Lars Waldvogel

Computer Science
Universität Basel

1 Einleitung

1.1 Einführung

Geocacher nennen die Uneingeweihten "Muggel", wie in der Harry Potter Reihe. Denn im Geocachen weiss man etwas, das die Muggel nicht wissen: fast an jedem öffentlichen Ort der Welt gibt es versteckte Gegenstände, sogenannte "Caches". Diese sind versteckt, zum Beispiel unter einer Parkbank, hinter einer Werbetafel oder auch in einem kleinen Loch im Boden, dass ein normaler Mensch nur allzu gerne übersieht.

Ein Cache kann verschiedene Formen haben: von einer Zündholzschachtel über eine Filmdose bis hin zur Tupperwarebox oder sogar einer richtigen Truhe! In diesen Caches ist ein Logbuch, in dem man sich einträgt, um zu beweisen, dass man den Cache tatsächlich auch gefunden hat. In manchen dieser Caches gibt es auch eine kleine "Tauschbörse", nach dem Motto: "Wenn du was hier lässt, kannst du auch gerne etwas mitnehmen".

Doch wie findet man solche Caches? Geocaching.com hat ein Verzeichnis von abertausenden von Caches, versteckt über die ganze Welt und doch kaum einem Muggel bekannt sind. Mit einem eigenen Account kann man selbst einen Cache verstecken und auf der Webseite hochladen.

Auf einer Karte werden einem dann die ungefähren Positionen der Caches in der Nähe angezeigt. Klickt man auf einen, so erhält man eine Beschreibung. Die Caches können mit genauen Koordinaten versteckt worden sein, mit einem kleinen Rätsel oder sogar mit grossen Rätseln mit mehreren Zwischenlösungen (Multicaches), die einen schlussendlich zur finalen Position des Caches führen.

1.2 Unsere Idee

Um eine künftige Verwirrung zu vermeiden: Wenn wir von nun an von "Caches" sprechen, dann meinen wir die Information darüber mit den Hinweisen und so weiter, und wenn wir von "physischen Caches" sprechen, dann meinen wir die kleinen Versteckten Dinge in der physischen Welt.

Das Prinzip des Geocachings ist sicher nichts neues, doch haben wir im Rahmen der Vorlesung entdeckt, dass es sehr stark zentralisiert ist. Unser Ansatz ist nun, eine dezentrale Implementation zu finden, bei der sich die Informationen der Caches via Peer-to-Peer-Verbindung (P2P) verbreiten. Um es den Leuten so einfach wie möglich zu machen, sich auszutauschen, möchten wir das Geocaching als Android App implementieren.

Die App soll die Caches über eine Bluetooth-Verbindung von einem Gerät zum Anderen übertragen. Auch sollte man in der App seine eigenen Caches erstellen können.

Wenn man einen Cache findet, sollte man diesen in der App lösen können, also sich in die "Hall of Fame" (HoF) eintragen.

Und um den Austausch möglichst zuverlässig zu gestalten, werden die Caches und die HoF-Einträge mithilfe eines Protokolls übertragen, das Secure Scuttlebug sehr ähnlich ist: Hier hat jede Person (Publisher) einen Append-Only Feed, in dem sie neue eigene Caches einträgt sowie bekannt gibt, dass sie einen Cache gelöst hat. Jeder User hat nun die Möglichkeit, die Feeds gewisser Publishers zu abonnieren (subscribe), um deren Feeds zu erhalten.

2 Implementation und Resultat

2.1 Die App

2.1.1 Implementation eines Caches

In der App können mehrere Caches verwaltet werden. Ein Cache besteht aus den folgenden Elementen:

- Titel
- Beschreibung (mit Hinweisen)
- Name des Erstellers
- Public Key zum Entschlüsseln der Hall of Fame
- Private Key um sich in die Hall of Fame einzutragen
- Verschlüsselte Hall of Fame

Des Weiteren gibt es drei Arten von Caches: eigener, ungelöster und gelöster Cache.

Ein eigener Cache ist ein Cache, den wir selbst erstellen. Dabei können wir den Titel und die Beschreibung des Cache selbst eingeben. Der Name des Erstellers wird dabei immer der eigene Name sein. Per RSA wird der private und der öffentliche Schlüssel berechnet und gespeichert. Da es sich um einen eigenen Cache handelt, wird der Name des Erstellers im Hall of Fame eingetragen. Im Hall of Fame steht ansonsten kein anderer Name (Siehe Kapitel 2.1.3). Aus diesem Grund enthält der eigene Cache alle der oben genannten Informationen. In der Detailansicht kann man sich den Private Key ansehen, um diesen im physischen Cache aufzuschreiben und zu verstecken.

Ein erhaltener, ungelöster Cache ist ein Cache, den wir von Freunden erhalten und noch nicht gelöst haben. Folglich besitzen wir den Private Key nicht. Dieser ist im physischen Cache versteckt. Findet man diesen, hat man in der Detailansicht die Möglichkeit, den Cache zu lösen. Dann wird daraus ein gelöster Cache, bei dem nun alle Informationen enthalten sind.

2.1.2 Append-Only-Feed

In unserer App wird das Prinzip von Append-Only-Feed verwendet. Der Append-Only-Feed dient zur Sicherstellung der Authentizität und Unverfälschtheit der Caches und deren Publishers. Dieser orientiert sich am Modell des Secure Scuttlebug Protocol.

Der Spieler hat einen eigenen Feed und Feeds von Freunden, welchen der Spieler subscribed hat. Der Spieler hat einen Benutzernamen und ein Public/Private Keypair. Wenn ein Nutzer das erste Mal die App öffnet, wird ihm ein Keypair generiert und er erhält einen Standard-Nutzernamen. Falls der Spieler seinen eigenen Namen ändert, wird der Keypair neu generiert. Als Konsequenz bekommt der Spieler einen neuen Feed, sodass die Daten vom vorherigen Feed verloren gehen. Dies liegt daran, dass der Feed in der App als File repräsentiert wird. Der Name des Files besteht aus dem Namen des Spielers, einem Hashtag und den letzten 4 Ziffern des öffentlichen Schlüssels.

Ein Feed besteht aus einer Liste von Einträgen (Entries). Jeder dieser Einträge hat folgende Struktur:

- Zeitstempel (Timestamp)
- ID (Position im Feed)
- die Signatur des vorherigen Entries, signiert
- Inhalt (Content)
- Typ
- Die Signatur des obigen

Bei der Signatur wird der Entry gehasht und der Hash mit dem privaten Schlüssel des Spielers verschlüsselt. Wenn jemand nun also den Entry selbst hasht, sollte dieser das gleiche sein wie die entschlüsselte Signatur. So können wir sowohl die Authentizität als auch die Integrität prüfen.

Um eine Änderung vorheriger Entries zu verhindern, verweist jeder Entry auf den Vorherigen. Falls jemand versucht, einen Entry zu manipulieren, können wir dies erkennen, da die Signatur nicht mehr stimmen wird. In späteren Posts wird diese Manipulation weiterhin ersichtlich sein.

Es gibt drei Typen von Entries: CacheEntry, HoFEntry, LogEntry. Alle drei Typen unterscheiden sich vor allem im Inhalt (Content).

Der CacheEntry beschreibt einen Cache. Somit beinhaltet der CacheEntry als Inhalt den gesamten Cache. Aus diesem Grund kann dieser somit nur dann generiert werden, falls ein eigener Cache erstellt wird.

Der HoFEntry wird erstellt, falls ein ungelöster Cache gelöst wird. Sein Inhalt ist der verschlüsselte Name des Spielers. Wichtig ist, dass die Signatur identisch sein muss wie die Signatur des ursprünglichen CacheEntry. Wieso dies relevant ist, wird im Kapitel 2.1.3 erklärt.

Der LogEntry wird erstellt, falls wir neue Daten von Freunden erhalten. Dieser beinhaltet als Inhalt eine Liste von Tupeln bestehend aus dem Namen des Publishers, die ID und die Signatur der (non-Log) Entries, die wir neu erhalten haben. Mithilfe dieser LogEntries lässt sich eine logische Zeitabfolge aller Entries bestimmen. Dies machen wir, um das Fälschen der Timestamps zu verhindern. Dazu noch mehr im Kapitel 2.3.

Falls einer dieser Entries erstellt werden sollte, wird dieser im eigenen Feed am Ende angehängt.

2.1.3 Generierung der Cache List

Die Cache List ist die Liste von Caches, die der Spieler im Hauptmenü sieht. Nun stellt sich die Frage wie sie generiert wird. Die Idee ist die folgende: Das Programm geht durch das eigene Feed durch und schaut nach CacheEntries. Falls ein CacheEntry gefunden wird, wird dessen Inhalt, ein Cache, herausgefiltert. Dieser Cache wird zu einem eigenen Cache umgewandelt. Daraufhin iteriert das Programm durch alle anderen Feeds, um zu schauen, ob es HoFEntries gibt mit der identischen Signatur wie der entsprechende CacheEntry. Das bedeutet, ob es Freunde gibt, die den Cache gelöst haben. Falls ja, wird der Inhalt des HoFEntry, der verschlüsselte Name, genommen. Dieser Name wird in das Hall of Fame des Caches hinzugefügt. Am Ende der Bearbeitung wird der Cache

zum Cache List hinzugefügt.

Falls nun der eigene Feed so bearbeitet worden ist, iteriert das Programm durch alle anderen Feeds und sieht nach CacheEntries um. Falls ein CacheEntry gefunden wird, wird dessen Inhalt herausgefiltert. Dieser Inhalt wird in einen ungelösten Cache umgewandelt. Danach sucht das Programm bei allen anderen Feeds nach entsprechenden HoFEntries. Am Ende schaut das Programm, ob der eigene Feed einen entsprechenden HoFEntry besitzt. Falls ja, wird der Inhalt zum Hall of Fame hinzugefügt und der Cache in einen gelösten Cache umgewandelt. Am Ende der Bearbeitung wird der Cache zum Cache List hinzugefügt.

2.2 RSA

2.2.1 Implementation

In diesem Projekt wurden zwei verschiedene Versionen von RSA implementiert: die ≪normale≫ RSA-Implementation [1] und die RSA-CRT-Implementation [2].

Beim Erstellen eines Cache wird bei der normalen RSA-Implementation folgendermassen vorgegangen:

Zunächst werden die Keys berechnet. Dabei wird bei der Berechnung der Keys strikt vorgegeben, dass die Länge von n in Binärformat explizit kleiner sein soll als 48 Bits. Dies dient dazu, dass die Spieler/Innen wenige Ziffern beim Eingeben des privaten Keys eintippen müssen.

Bei der Verschlüsselung eines Namens wird der Name in Blöcken zergliedert. Die Blockgrösse ist dabei die Länge von n in Binärformat durch 6. Die Zahl 6 kommt von der Länge eines Buchstabens in Base64 Encoding. Falls die Länge von n in Binärformat durch 6 ohne Rest teilbar sein sollte, dann wird die Blockgrösse zusätzlich um eins reduziert. Damit soll sichergestellt werden, dass kein Block dargestellt als eine Zahl grösser sein kann als n. Im nächsten Schritt werden die Buchstaben der einzelnen Blöcke in Base64 Encoding Format umgewandelt. Falls es dazu kommen sollte, dass ein Block gepaddet werden muss, dann wird dieser Block an den leeren Stellen mit ‘111111‘ gefüllt.

Die Verschlüsselung der Blöcke wurde mit Cipher Block Chaining kombiniert, um gleiche Ciphertexte bei Blöcken mit demselben Inhalt zu vermeiden. Am Ende wird der Ciphertext zurückgegeben, wobei angemerkt werden muss, dass der erste Block im Ciphertext der Initialisierungsvektor ist:

$$c(i) = (c(i-1) XOR m(i-1))^d mod(n)$$

mit $c(i)$ = Cipherblock an der Stelle i, $m(i)$ = Textblock an der Stelle i und $c(1)$ = Initialisierungsvektor.

Beim Entschlüsseln eines Ciphertextes kann anhand von n im öffentlichen Key die angewandte Blockgröße festgestellt werden (siehe oben). Daraufhin kann der vorgegebene Ciphertext per Entschlüsselungsverfahren kombiniert mit der Rücktransformation der Cipher Block Chaining dekodiert werden:

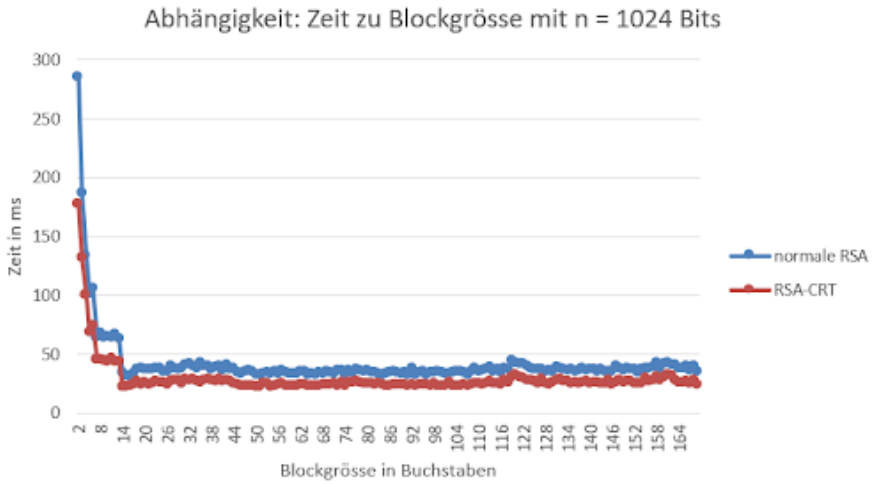
$$m(i) = c(i) XOR (c(i+1)^e mod(n))$$

Falls ein Padding erkannt werden sollte, wird dieser ignoriert.

RSA-CRT weist keine grossen Unterschiede zu der normalen RSA-Implementation. Der einzige, wichtige Unterschied liegt im Entschlüsselungsverfahren. Hier wird bei der Entschlüsselung eines Blockes die Chinese Remainder Theorem [3] angewendet. Aus diesem Grund muss dem Entschlüsselungsverfahren neben dem privaten Key auch die Primfaktorzerlegung von n bekannt sein. Folglich besteht der private Key nicht nur aus d und n, sondern auch aus p und q. Das Entschlüsselungsverfahren bleibt jedoch weiterhin gleich.

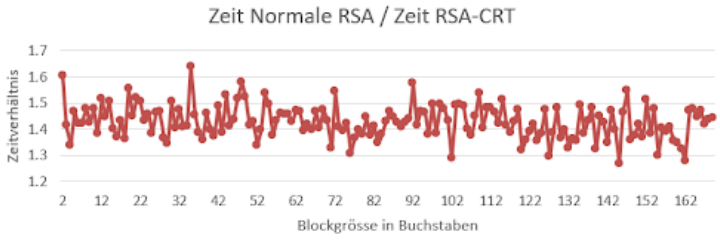
2.2.2 Resultate

Deshalb stellt sich die Frage, was die Vorteile sind von RSA-CRT im Vergleich zu der normalen RSA:



In dieser Abbildung wird die Zeit im Verhältnis zu unterschiedlichen Blockgrössen dargestellt, die gemessen wurde, um eine gegebene Nachricht in Blöcken unterteilt zu verschlüsseln und zu entschlüsseln. Dabei wurde die Messung 5 Mal durchgeführt und der Durchschnitt als Zeit im Diagramm verwendet. Es wurde ein Schlüssel der Grösse 1024 Bits benutzt.

Es kann beobachtet werden, dass sowohl bei der normalen RSA als auch bei RSA-CRT die Zeit anfangs stark abnimmt und ab der Blockgrösse 14 mehrheitlich konstant bleibt. Der Text, welches hier als Nachricht verwendet wurde, war «Lars Waldvogel». Interessant ist, dass diese Nachricht 14 Stellen hat. Des Weiteren kann beobachtet werden, dass RSA-CRT bei allen Blockgrössen weniger Zeit benötigt als die normale RSA Implementation. Das Verhältnis dieser beiden kann im nächsten Diagramm genauer untersucht werden:



In diesem Diagramm wurde das Verhältnis der gemessenen Zeit von der normalen RSA zu der gemessenen Zeit von RSA-CRT aufgetragen. Dabei kann erkannt werden, dass RSA-CRT 1.28 bis 1.64 Mal schneller war in unserer Messung als die normale RSA.

2.2.3 Diskussion

In der ersten Abbildung wurde beobachtet, dass die Zeit zuerst stark gesunken ist und ab der Blockgrösse 14 mehrheitlich konstant blieb. Die Zahl 14 ist auf die Länge der verschlüsselten Nachricht ≪Lars Waldvogel≫ zurückzuführen. Ab der Blockgrösse 14 musste die Nachricht nicht mehr unterteilt werden. Stattdessen hat die Nachricht immer in einem einzigen Block gepasst mit Padding. Aus diesem Grund musste die Exponentialrechnung, welches sowohl bei der Verschlüsselung als auch bei der Entschlüsselung am meisten performancelastig ist, nur einmal durchgeführt werden. Bei sehr kleinen Blockgrössen würde die Nachricht nicht innerhalb eines Blocks hineinpassen. Als Konsequenz müsste man mehrere Blöcke verschlüsseln. Dies würde dazu führen, dass mehr Exponentialrechnungen berechnet werden müssen. Als Konsequenz würde man mehr Zeit benötigen.

Durch die Chinese Remainder Theorem, welches die Exponentialrechnung vereinfacht, ist RSA-CRT schneller als die normale RSA. Es ist wichtig, anzumerken, dass RSA-CRT zusätzlich verschnellert werden kann, indem man bei der Entschlüsselung auch die Chinese Remainder Theorem verwendet. Dies wird bei uns nicht gemacht, da ein Spieler, welcher das Cache nicht gelöst hat, keinen Zugriff auf die Primfaktorzerlegung von n hat.

Es gibt eine Möglichkeit, das allgemeine Entschlüsselungsverfahren durch die Wahl von e zu verschnellern. In der Realität wird e zuerst gesetzt. Erst danach werden die weiteren Elemente des öffentlichen und des privaten Schlüssels berechnet. Dabei wird heutzutage entweder $e = 3$ oder $e = 65537$ gewählt, da beide Primzahlen sind und sehr effizient bei der modularen Exponentialrechnung angewendet werden können. Bei der Right-to-Left Multiplikation [4] zum Beispiel kann die Exponentialrechnung schnell durchgeführt werden, da beide Zahlen nur 2 gesetzte Bits haben. Des Weiteren kann man m^3 oder m^{65537} auch einfach berechnen, indem man den zu verschlüsselnden Wert m einmal respektive sechzehn Mal quadriert und anschliessend mit m noch einmal multipliziert. Dieser Trick macht die Entschlüsselung deutlich schneller.

In unserer App wird derzeit die normale RSA-Implementation verwendet. Dies liegt daran, dass die Spieler/Innen den privaten Schlüssel selbst eingeben müssen. Bei RSA-CRT muss man neben dem privaten Schlüssel auch die Primfaktorzerlegung von n mit eingeben. In unserer Sicht wäre das aber sehr aufwendig. Für die Zukunft wäre die Idee, den privaten Schlüssel in ein QR Code umzuwandeln, sehr interessant, da dies die genannte Problematik mit dem Eintippen vermeiden würde. Falls dies realisiert werden kann, wäre man nicht limitiert in der Länge von n in Binärformat. Im Gegenteil, man hätte die Möglichkeit, mit einem n der Grösse 2048 Bits zu arbeiten. Folglich hätte man mehr Sicherheit gegenüber Angriffen. Zurzeit kann der private Schlüssel bei uns maximal 48 Bits lang sein. Einen solchen Schlüssel zu knacken wäre sogar mit einem naiven Verfahren in Minuten machbar.

Abschliessend zu diesem Kapitel muss erwähnt werden, dass die Wahl zwischen RSA und RSA-CRT bei unserer App bei grossen Blockgrössen eher weniger relevant ist. Zwar hätte man Geschwindigkeitsnachteile bei RSA in Millisekundenbereich, aber dieser Nachteil ist in unserer Anwendung nicht sehr schwerwiegend.

2.3 Datentransfer

Der Datentransfer findet über Bluetooth statt.

Der Ablauf für den Benutzer ist folgendermassen: Zwei Personen, welche die App installiert und sich gegenseitig abonniert haben, treffen sich und koppeln ihre Geräte. Einer entscheidet sich sein Feed zu senden und klickt dafür zunächst auf den Connect-Button und danach auf den Sender-Button. Dadurch hat er bei sich einen Server gestartet, der auf eingehende Verbindungen wartet. Der andere will den Feed empfangen und klickt dafür auch zunächst auf den Connect-Button. Nun sieht er eine Liste von Geräten mit denen er gekoppelt ist unter anderem auch das Gerät seines Partners. Sobald er auf dieses Gerät klickt, verbindet er sich als Client an den Server und erhält seinen Feed. Nun kann man die Rollen vertauschen und das gleiche nochmal durchführen damit am Schluss beide auf dem neuesten Stand sind und die gleichen Daten haben.

Sobald der Server gestartet wird holt er bei sich den sein Append-Only-Feed und verschlüsselt es unter Verwendung eines selbst entworfenen Protokolls zu einem String. Wir haben empirisch herausgefunden, das es eine Limite an Bytes gibt, die man auf einmal senden bzw. empfangen kann. Daher unterteilt der Server den String in gleich grosse Blöcke und sendet zuerst die Anzahl Blöcke die er an den Client sendet wird. Der Client merkt sich diese Zahl und empfängt dementsprechend viele Blöcke von Server. Hat der Client alle Blöcke empfangen, so verwandelt er die Bytes wieder zu einem String und entschlüsselt ihn wieder unter Verwendung eines selbst entworfenen Protokolls zu einem Append-Only-Feed.

Auf dem erhaltenen Feed wird nun angeschaut, was neu ist und dementsprechend die Cache List und Hall-of-Fame geupdatet. Dafür verwandelt der Empfänger den bestehenden Feed und den erhaltenen Feed in eine Liste von Einträgen und vergleicht deren Längen. Ist der erhaltene Feed länger, so wird der File mit dem bestehenden Feed überschrieben, andernfalls wird es ignoriert. Falls es zu einem Update kommt werden die neuen Entries in einer Liste abgespeichert. Ist am Schluss diese Liste nicht leer, so wird ein LogEntry für diese neuen Entries erstellt und zum eigenen Feed angehängt.

Sobald der Sender alles versendet und der Empfänger alles empfangen hat, schliessen beide ihre Sockets und der Datentransfer ist abgeschlossen.

2.4 Resultat

Das, was am Ende aus all diesen Konzepten entstanden ist, ist unsere App. Der Zip-File der App kann in der GitHub Repository `https://github.com/LarsWaldvogel/P2PGeoCachingApp` oder in der GitHub Repository von BACnet heruntergeladen werden. In diesem Ordner befindet sich auch ein Video, welches die Benutzung der App zeigt. Im Anhang, Kapitel 4.1, ist auch ein Handbuch für die Benutzung der App aufgeführt.

3 Fazit

3.1 Lessons Learned und Schlussfolgerung

Wir haben das Ziel unseres Projektes erreicht. Wir haben eine funktionierende Android Geocaching App programmiert mit welchem man Feeds übers Bluetooth Peer-to-Peer austauschen kann. Im Rahmen dieses Projekts haben wir gelernt mit Kotlin eine App zu programmieren, wie die RSA Verschlüsselung funktioniert und optimiert werden kann und wie man per Bluetooth Daten von einem Gerät ans Andere, also Peer-to-Peer, schicken kann, um somit Informationen dezentral zu verbreiten.

Wir haben gemerkt, dass die vorhandene Bacnet Infrastruktur nicht ganz kompatibel war mit unserer App und so haben wir uns entschieden eine eigene Implementation nach der Idee des Secure Scuttle Butts vorzunehmen. Es war eine Herausforderung für uns ein eigenes Protokoll zu entwerfen, um die Feeds zu senden, empfangen und zu synchronisieren, doch wir haben uns der Herausforderung gestellt und sind zufrieden mit dem Resultat. Bei der eigenen Implementation des Protokolls haben wir viel über Secure Scuttle Butt gelernt.

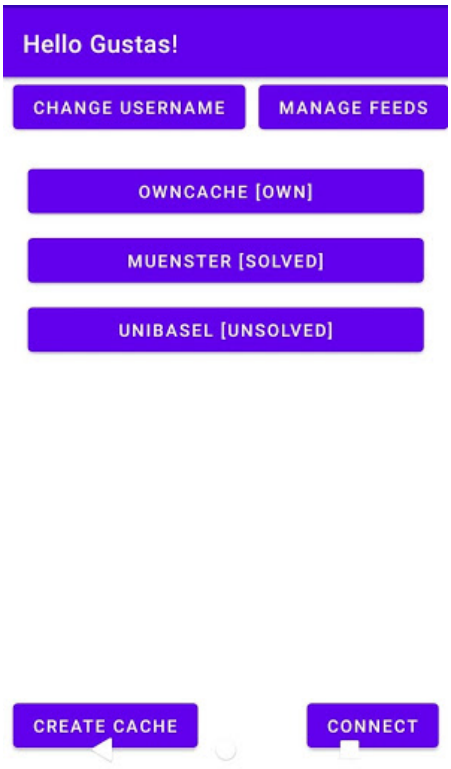
Zusätzlich haben wir gelernt, dass Kommunikation und Arbeitsteilung wichtige Punkte in einer Gruppenarbeit sind. Durch gute Planung, regelmässigen Meetings und einer guten Arbeitsteilung schon seit dem Beginn, konnten viele Missverständnisse früh geklärt werden, was im späteren Verlauf des Projekts von Vorteil war.

Für die Zukunft könnte die App auf verschiedenen Arten ergänzt und erweitert werden. Zum Beispiel mit dem Einbau von QR-Codes, um mit grossen Schlüsseln arbeiten zu können und das Lösen von Caches benutzerfreundlicher zu gestalten. Man könnte das Spiel auch erweitern, indem die Geocacher/Innen nicht nur einen einzelnen Key, sondern zum Beispiel von 5 möglichen Keys mindestens 3 finden müssen. um den Cache zu lösen - Wo wir dann beim Thema Keysplitting wären.

Im Grossen und Ganzen sind wir persönlich zufrieden und hoffen, dass dies der Anfang eines grossen Spielspasses sein wird. Möge die Jagd beginnen!

4 Anhang

4.1 Handbuch zum App “P2P Geocaching”



Startmenü

Dies ist der Startmenü, welches beim Starten der App gezeigt wird.

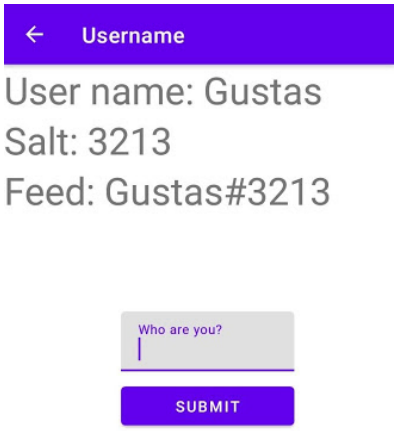
Oben links: Benutzername des Users, mit einem Button, der auf ein Fenster wechselt, in dem man den Namen ändern kann.

Oben rechts: ein Button, der einen zu einem Fenster bringt, in dem man seine Feeds einstellen kann.

Unten links: ein Button, der einen zu einem Fenster bringt, in dem man einen neuen Cache erstellen kann.

Unten recht: ein Button, mit dem man zu einem Fenster kommt, in dem man seine Daten mit anderen Nutzern austauschen kann.

Mitte: Hier sieht man die Auflistung der Caches die man hat, inklusive ihrem Typ (eigener Cache, gelöster Cache, ungelöster Cache). Klickt man auf einen Cache, kommt man zu seiner Detailansicht.



Namen ändern und Salt betrachten

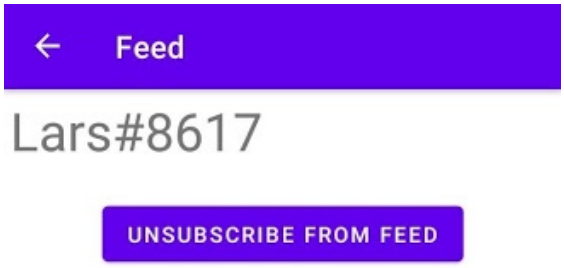
Um den Namen zu ändern, soll im Startmenü auf den Button “Change Username” gedrückt werden. Daraufhin wird dieser Fenster sichtbar. In diesem Fenster kann der jetzige Benutzername und der Salt betrachtet werden. In der Mitte befindet sich ein Textfeld. Hier kann der neue, erwünschte Name eingegeben werden. Falls daraufhin der Button “Submit” gedrückt wird, wird der Name überprüft. Falls der Name nicht nur aus Buchstaben oder Zahlen besteht, wird dieser nicht akzeptiert. Falls der Name als valid empfunden wird, bekommt der Benutzer/die Benutzerin einen neuen Salt. Die Konsequenz ist, dass die Informationen zu den eigenen Caches und Hall of Fame Einträge verloren gehen, da ein neuer Feed generiert wird.

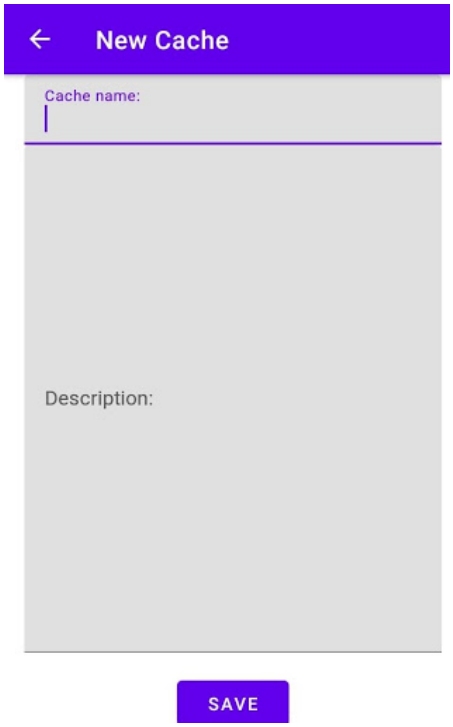
Feeds anschauen, abonnieren, deabonnieren

Um die subscribten Feeds zu betrachten, soll im Startmenü auf den Button “Manage Feeds” gedrückt werden. Daraufhin erscheint dieser Fenster.

In der Mitte werden alle abonnierten Feeds angezeigt. Falls nun auf einen Feed angeklickt wird, erscheint die Detailansicht des Feeds. Hier besteht die Möglichkeit, den Feed zu deabonnieren, indem auf den Button “Unsubscribe from Feed” gedrückt wird (siehe unten links).

Unten ist ein Button namens “Add Feed” aufgeführt. Falls auf diesen Button geklickt wird, erscheint ein Fenster, welches die Möglichkeit bietet, einen neuen Feed zu subscriben. Im Feld “Publisher Name” soll der Name des Inhabers des Feeds eingegeben werden. Im letzten Feld soll der Salt eingegeben werden, also die letzten vier Ziffern des öffentlichen Schlüssels des Inhabers. Falls alles korrekt ist und auf den Button “Save” gedrückt wird, folgt der Benutzer/die Benutzerin diesem Feed (siehe unten rechts).



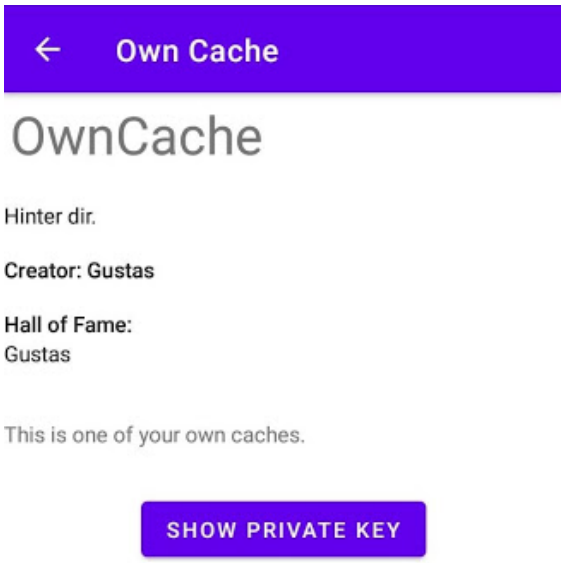


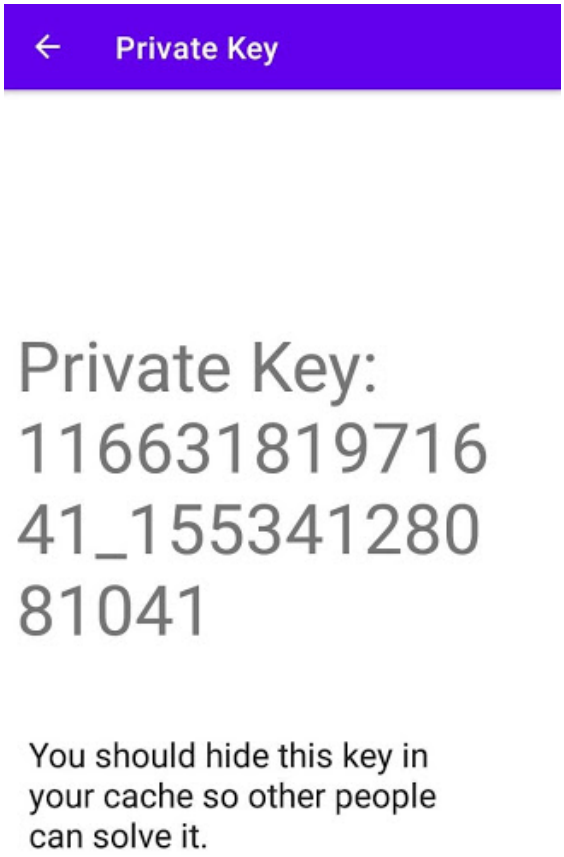
Eigenen Cache erstellen

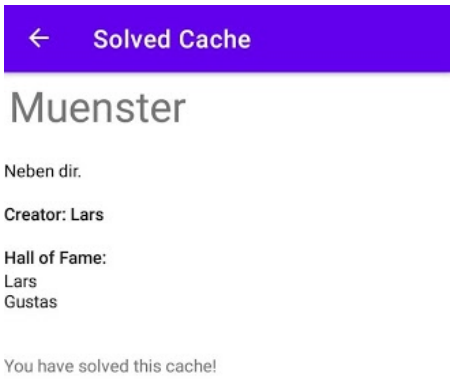
Um einen eigenen Cache zu erstellen, soll im Startmenü auf den Button “Create Cache” geklickt werden. Daraufhin erscheint dieser Fenster. Unter “Cache Name” kann der Titel, im Description Feld die Beschreibung, wo sich der physische Cache sich befindet, eingegeben werden. Falls daraufhin der Button “Save” gedrückt wird, wird der Cache erstellt.

Um den privaten Schlüssel zu sehen, muss man zurück zum Startmenü gehen und auf den entsprechenden Cache klicken. Dann erscheint ein Fenster (siehe unten links), welches nochmals die relevanten Informationen zu dem eigenen Cache zeigt.

Falls auf den Button “Show Private Key” gedrückt wird, erscheint ein Fenster (siehe unten rechts) mit dem privaten Key. Dieser soll im physischen Cache deponiert werden.

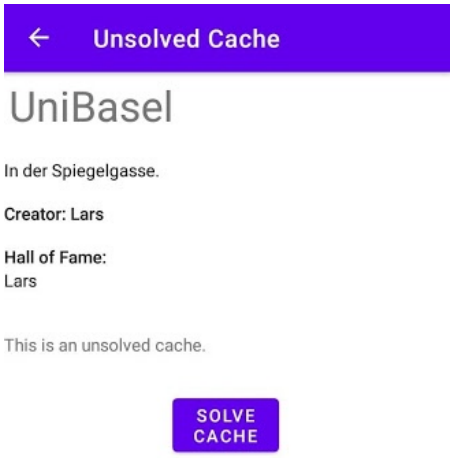






Detailansicht von gelöstem Cache

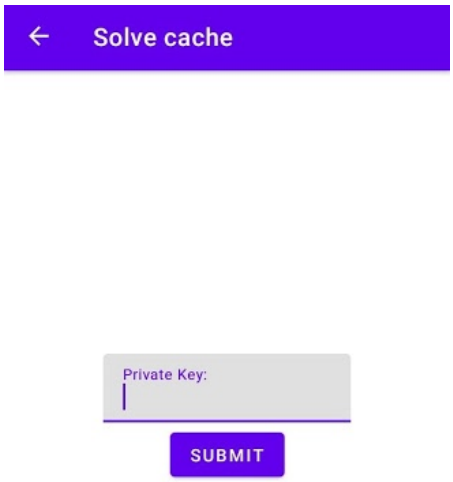
Im Startmenü kann man auf einen Cache klicken, welches als [Solved] markiert ist. Daraufhin erscheint ein Fenster wie in der Abbildung, in welcher Informationen wie der Titel, die Beschreibung, der Name des Erstellers sowie das Hall of Fame ersichtlich sind.



Detailansicht und Lösen von ungelöstem Cache

Im Startmenü kann man auf einen Cache klicken, welches als [Unsolved] markiert ist. Daraufhin erscheint ein Fenster wie in der Abbildung, in welcher Informationen wie der Titel, die Beschreibung, der Name des Erstellers sowie das Hall of Fame ersichtlich sind.

Des Weiteren hat der Benutzer/die Benutzerin die Möglichkeit, den Cache zu lösen. Hier muss der Button “Solve Cache” gedrückt werden. Daraufhin erscheint ein Fenster (siehe unten links).



In diesem Fenster kann man im Feld Private Key den gefundenen privaten Schlüssel eingeben. Das Programm überprüft anhand des bekannten, öffentlichen Schlüssels, ob der eingegebene private Schlüssel korrekt ist. Falls ja, wird der Cache als gelöst markiert. Falls nicht, geschieht nichts.

Datentransfer

Um Feedinhalte miteinander auszutauschen, soll im Startmenü auf den Button “Connect” gedrückt werden. Daraufhin erscheint ein solcher Fenster. Oben werden Geräte angezeigt, mit welchen ein Bluetooth-Verbindung eingegangen werden kann. Um den eigenen und die abonnierten Feedinhalte zu senden, soll auf den Button “Sender” gedrückt werden. Um Inhalte zu erhalten, soll direkt auf den Namen des Geräts, von welchem man empfangen möchte, gedrückt werden.

Falls Sie auf den Sender Button klicken und ihr Kollege/Kollegin auf den Namen ihres Geräts, sendet Ihr Gerät Inhalte zum Gerät ihres Kollegen/Kollegin.

Mit dem Button close kann der Zustand des Senders verlassen werden.

Es ist wichtig, anzumerken, dass die empfangenen Feeds beim Datenupdate nur berücksichtigt werden, falls die Feeds auch abonniert sind.

Literatur

[1] RSA-Kryptosystem
https://de.wikipedia.org/wiki/RSA-Kryptosystem

[2] RSA with CRT: A new cost-effective solution to thwart fault attacks
https://iacr.org/archive/ches2008/51540128/51540128.pdf

[3] Chinese remainder theorem
https://en.wikipedia.org/wiki/Chinese_remainder_theorem

[4] Modular exponentiation
https://en.wikipedia.org/wiki/Modular_exponentiation

[5] Github Repository von Peer-to-Peer Geocaching App
https://github.com/LarsWaldvogel/P2PGeoCachingApp