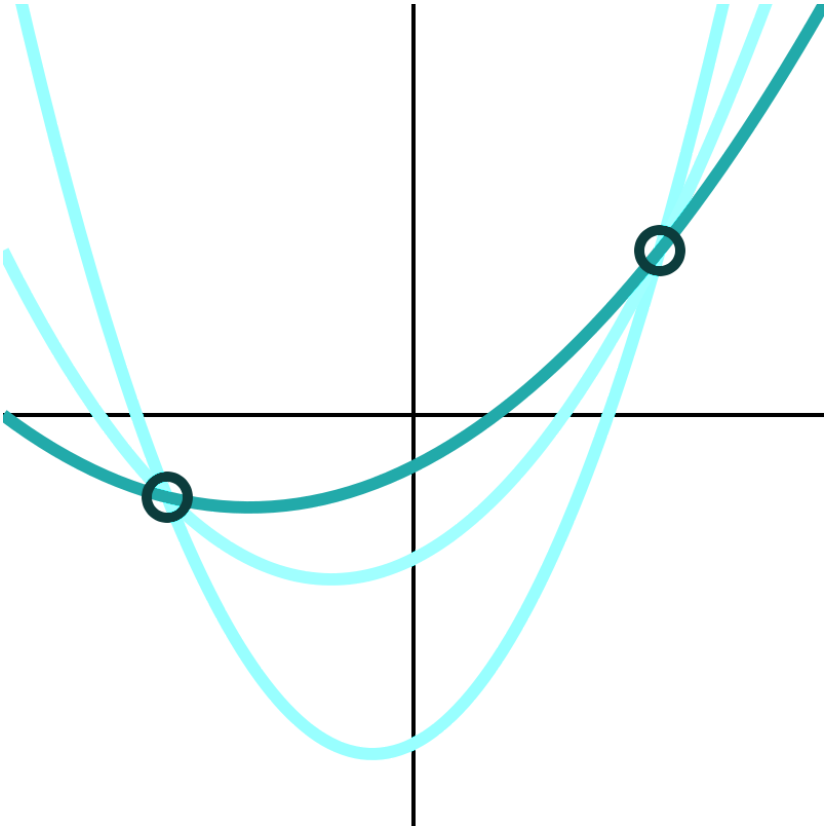
Introduction to Internet and Security

Frühjahrssemester 2021

Gruppe 6 - Secret Sharing



18. Juli, 2021

Colin Fingerlin
Matias Carballo
Olivier Mattmann

Abstract

Im Rahmen der Vorlesung “Introduction to Internet and Security” konnten wir als Gruppe ein Projekt im BACnet umsetzen. Wir haben uns dafür entschieden eine App zu schreiben, die es Usern ermöglicht Secrets von bis zu 4 Kb unter Kontakten aufzuteilen und bei Bedarf diese anzufragen und den Secret wiederherzustellen. Wir haben den Usern diese Funktionalität durch eine benutzerfreundliche UI zur Verfügung gestellt.

Neben dieser Haupt-Funktionalität gibt es die Möglichkeit die Daten der Applikation zu verschlüsseln. Durch ein modulares Design haben wir uns die Möglichkeit offengelassen, das Projekt zu erweitern oder es für andere Applikationen ausserhalb des BACnets anzupassen.

Einführung und Hintergrund

Das BACnet-Projekt verläuft parallel zur Vorlesung “Introduction to Internet and Security”. Das Ziel war elektronische Kommunikation zu ermöglichen, ohne auf das Internet zurückzugreifen. Das Netzwerk sollte ohne Client-Server-Architektur auskommen, sondern dezentral aufgebaut sein.

Im Rahmen dieses Projektes wollte unsere Gruppe den Usern es ermöglichen innerhalb des BACnet Secret Sharing nutzen zu können.

In Kryptographie ist Secret Sharing ein Mechanismus, bei dem Shards einer wichtigen privaten Information in ein Netzwerk oder Gruppe geteilt werden. Die einzelnen Shards sind an sich nutzlos, man kann aus einem Shard nicht auf die Gesamtinformation schliessen. Wenn man jedoch eine vorgegebene Anzahl Shards erhält, kann man das Secret wiederherstellen. Das Splitten und Wiederherstellen verläuft nach dem Shamir Secret Sharing Algorithmus[1], welcher durch polynomiale Interpolation funktioniert.

Ein Secret wird zu einem Polynom eines beliebig gewählten Ranges verwandelt. Dieser Rang entspricht der Anzahl der Shards, die benötigt werden das Secret wiederherzustellen. Die Shards, welche verschickt werden repräsentieren Punkte auf diesem Polynom. Falls man Fragmente im Besitz von mindestens der Anzahl des Ranges ist, kann das Polynom durch Interpolation wiederhergestellt werden und somit auch das Secret. Unsere Aufgabe war es das Verteilen und Zurückholen solcher Shards über die bereits vorhandene Infrastruktur zu ermöglichen.

[1] http://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf

Implementation

Unsere Secret Sharing Applikation stellt folgende Funktionalitäten zur Verfügung:

- Shards an Kontakte senden
- Shards anfordern
- Secrets wiederherstellen

Beim ersten Ausführen der Applikation muss vom User ein Passwort gewählt werden. Dieses Passwort wird neben den Schlüsselpaaren der Feeds auch zur Verschlüsselung verwendet. Da es nicht gewollt ist, dass alle die versendeten Events einsehen können verschlüsseln wir einen AES Key mit dem Public-Key des Empfängers. Dieser AES Key kann dann verwendet werden um den eigentlichen Inhalt des Events zu entschlüsseln. Insgesamt verwenden wir drei Typen von Events. Im folgenden Abschnitt werden diese näher erläutert;

SHARE Typ

Ein Event des Typs SHARE wird verwendet um einen Shard an jemanden zu senden. Der Inhalt besteht aus dem Shard und dem Namen, welcher dem Secret zugeteilt worden ist. Der Name und der Shard sind jeweils mit dem Passwort des Eigentümers vom Secret verschlüsselt. Der Empfänger speichert das Shard-Namen-Paar in einem Buffer.

REQUEST Typ

Ein Event des Typs REQUEST wird verwendet um einen Shard bei einem Kontakt anzufragen. Das Event beinhaltet den verschlüsselten Namen des Secrets bzw. des angefragten Shards. Beim Empfang eines solchen Events wird der Shard mittels einem REPLY Event an den Anfrager gesendet, sofern ein Shard mit solchem Namen sich im Buffer befindet. Der Anfrager muss nicht zwingenderweise der Eigentümer des Secrets sein. Dennoch wird der Shard vom Buffer versendet, denn er ist immer noch durch das Passwort des Eigentümers verschlüsselt. Versucht ein Client auf diese Weise Shards vom Urheber anzufragen, lehnt die Applikation die Request ab.

REPLY Typ

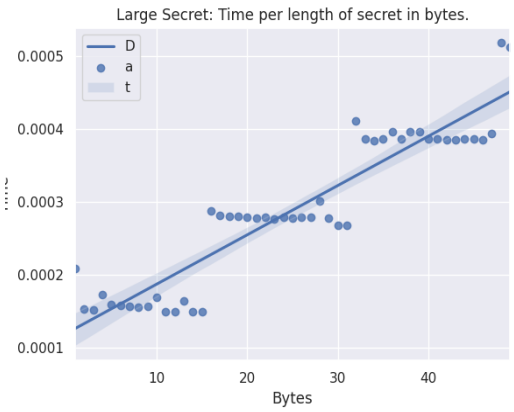
Ein Event des Typs REPLY wird verwendet um eine angefragten Shard zurückzusenden. Die Applikation versendet dieses Event automatisch beim Empfangen einer gültigen Anfrage. Das Event beinhaltet den verschlüsselten Namen des Secrets, sowie der verschlüsselte Shard. Wenn man ein Event des Typs REPLY erhält wird der Shard entschlüsselt und in seinem eigenen Buffer abgelegt, bis das Secret wiederhergestellt wurde.

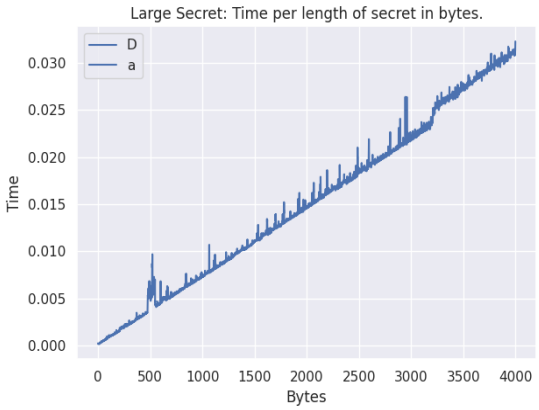
Unsere App nutzt selber ein Login mit einem Passwort, um die Sicherheit der Secrets zu gewährleisten. Der Status der App werden lokal in einem JSON-Format abgelegt, wir haben aber den Nutzern die Möglichkeit gegeben diese Status-files mit der App selber zu entcrypten und zu decrypten.

Resultate und Diskussion

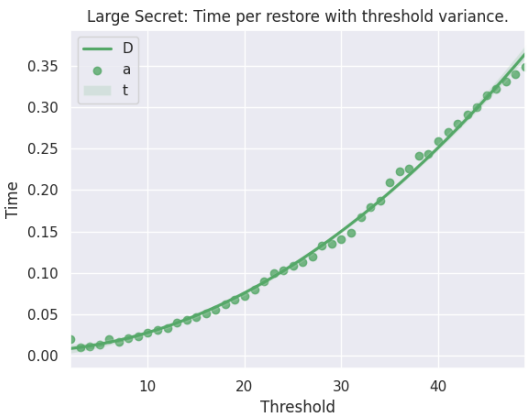
Das Ergebnis unseres Projekts ist eine benutzerfreundliche App, die es ermöglicht Secret Sharing im BACnet zu betreiben mit Secrets bis zu 4.08Kb groß, neben andere Funktionalitäten wie zum Beispiel das Encrypten von lokalen Files (**fencrypt module**). Die App ist so modular wie möglich aufgebaut, so dass man sie in anderen Umgebungen mit leichten Anpassungen auch nutzen könnte.

Eines der grossen Probleme den wir hatten war, dass das Shamir Secret Sharing Algorithmus nicht für längere Keys in einer Bibliothek zu finden ist, sondern meist für 16 Bytes lange Secrets. Wir haben für unsere App die mögliche Länge erweitert auf 256 * 16 Bytes, etwa 4.08Kb. Weiter bemerken wir, dass es möglich ist mit weiteren Anpassungen beliebig große Dateien auf diese Weise zu verschlüsseln. Die Komplexität der erweiterten Variante bleibt in angemessenen Rahmen für eine Benutzer Applikation.

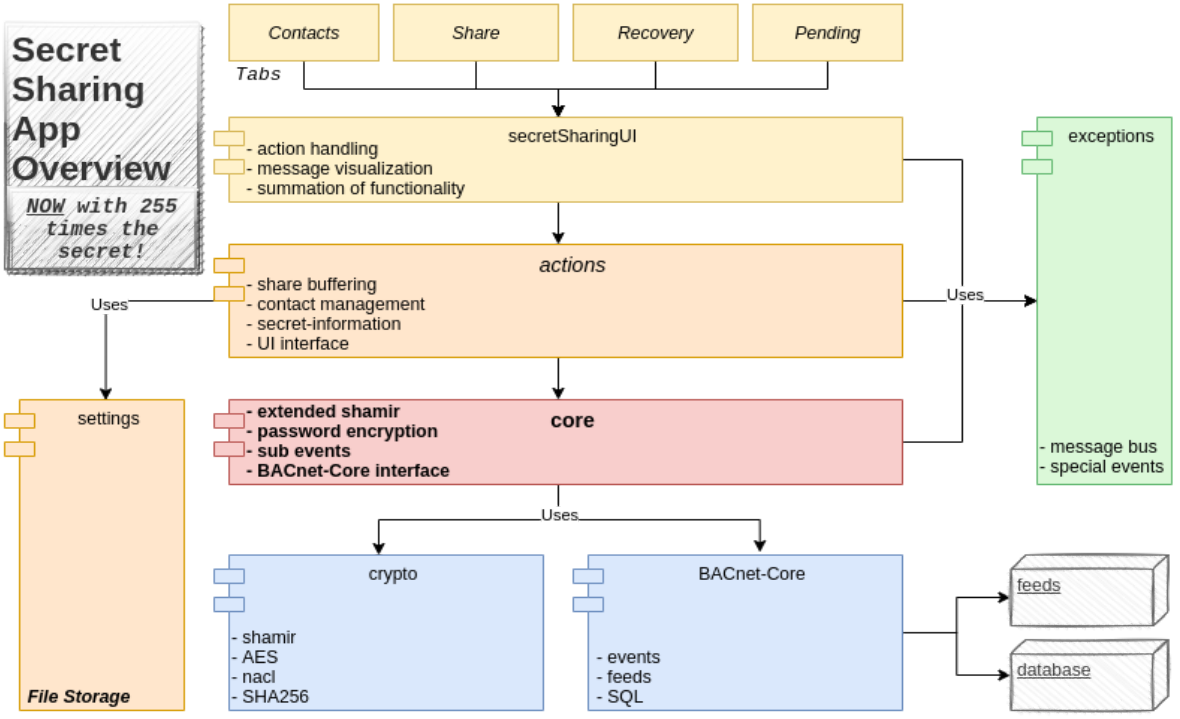




In diesen Plots sehen wir das erstellen der Pakete bei variabler länge des Secrets ist **O(n)** komplex, behaltet also die Komplexität der ursprünglichen Shamir Methode. Bei der Wiederherstellung haben wir mit variablem Grad des Polynoms (Threshold) eine **leicht quadratische Konvergenz**, während Shamir grundsätzlich O(t*log²t) aufweist.



Die Modularität lässt sich anhand eines UML Diagramms darstellen. Der Benutzer erhält über Custom Tabs zugang zu Funktionalitäten des **secretSharingUI** Dieses hat eine wohl-definierte Schnittstelle mit dem ***actions module***, welches die Abkapselung zum Back-End der Applikation darstellt. Vom actions module gibt es Zugriff auf das **settings module**, welches Datenspeicherung zu Applikations-Zwecken tätigt und zum **core module** welches die Kern-Funktionalitäten der Applikation kapselt. Das **exceptions module** ist ein sekundärer Nachrichtenkanal zwischen den Schichten der Applikation und übernimmt spezielle Vorkommnisse und Fehler in der Applikation. *All dies hat zur Folge, dass das core module von Drittparteien ohne Wechselwirkungen importiert werden kann.*



Zum Zeitpunkt der Fertigstellung der Applikation ist jedoch der BACnet-Core nicht zugänglich und wird deshalb statt mit stubs mit älteren Modulen überbrückt um das Testen zu ermöglichen (**database_connector module**). Jedoch ist es ein leichtes die Applikation für ein neues Modul zu konfigurieren da lediglich wenige Methoden im core module Zugang fordern. Das core module wurde weiter mit testing versehen um die Funktionalität zu sichern, diese Tests befinden sich in *06-SecretSharing/tests/testing.py*.

Anhang

Benutzte libraries:

PyQt5 - https://pypi.org/project/PyQt5/ (GUI)

PyCryptoDome - https://pypi.org/project/pycryptodome/ (Shamir Algorithm, AES)

PyNaCl - https://pypi.org/project/PyNaCl/ (Key-Pairs)

bcrypt - https://pypi.org/project/bcrypt/ (Passwords)