INTRODUCTION TO INTERNET AND SECURITY

FRÜHJAHRSSEMESTER 2021

PROJEKTARBEIT

CRYPTOCHAT

July 18, 2021

Max Jappert
Colin Saladin
Noah Müller

Inhaltsverzeichnis

1 Einleitung

Unser Projekt erweitert mit dem Subjective Chat eine schon bestehende Funktionalität, die von der Gruppe #3 im FS 2020 implementiert wurde. Unsere Erweiterung bereichert ihre Implementation um ein asymmetrisches Verschlüsselungsverfahren. Die als Chat Events gespeicherten Nachrichten aus dem Subjective Chat sind in der letztjährigen Implementation unverschlüsselt, also innerhalb des Chat Feeds (als SQlite-Datenbank) im Klartext lesbar. Mit unserer Implementation werden diese mit einem gemeinsamen Private Key verschlüsselt, der mit einem Diffie-Hellman-Merkle Key-Exchange mit jedem Nachrichtenzyklus neu verhandelt wird. Die Nachrichten werden mit *Cipher Block Chaining* (CBC) verschlüsselt. Unsere Implementation ist forward- und backward-secure, da mit jeder gesendeten Nachricht ein neuer Public Key und somit mit jeder erhaltenen Nachricht ein neuer Private Key generiert wird. Somit reicht ein Private Key nur für die Entschlüsselung eines Nachrichtenzyklus. Die Schlüssel für die vorigen und zukünftigen Nachrichten sind daraus praktisch nicht berechenbar.

2 Hintergründe

Was bereits implementiert wurde und wie unser Projekt an die bestehenden Implementationen anknüpft soll im Folgenden diskutiert werden.

2.1 Subjective Chat

Die Funktionalität des Subjective Chats wurde bereits im Rahmen der Vorlesung *Introduction to Internet and Security* im FS 2020 von der Gruppe #3 implementiert. Sie stellt die Basis unseres Projektes dar und ermöglicht das Versenden von Nachrichten zwischen zwei oder mehr Personen. Es handelt sich um eine funktionsfähige Chat-Applikation, die mit einer grafischen Benutzeroberfläche ausgestattet ist. Die Applikation knüpft an ein dezentrales Netzwerk an, da über einen USB-Stick serverlos kommuniziert werden kann. Die Nachrichten werden als Chat Events im Chat Feed gespeichert, darin sind die Nachrichten ohne unsere Implementation im Klartext gespeichert.

2.2 Double Ratchet Algorithmus

Mit der Idee eines Cryptochats ging die Wahl des Verschlüsselungsalgorithmus einher. Der heutige Industriestandard ist der Double Ratchet Algorithmus, der bei WhatsApp, Telegram, Facebook Messenger und Signal verwendet wird.

Eine Implementation wurde bereits im Rahmen des Redezentralisierungsseminars im HS 2020 von der Gruppe 3 implementiert, jedoch ohne graphische Benutzeroberfläche. Unsere ursprüngliche Idee was das Migrieren dieser Implementation in den Subjective Chat, so dass dessen Nachrichten mit dem Double-Ratchet Algorithmus verschlüsselt würden.

Der Algorithmus garantiert neben sehr sicherer Verschlüsselung auch Forward- und Backward-Secrecy. Wird also ein privater Schlüssel geknackt, kann höchstens eine Handvoll Nachrichten entschlüsselt werden. Wie der Double Ratchet Algorithmus funktioniert lässt sich mit zwei physikalischen Ratschen veranschaulichen. Unglücklicherweise konnten wir auch nach langem Ausprobieren den bereits implementierten Double-Ratchet Algorithmus nicht zum Laufen bringen. Wir waren lange mit der Gruppe in Kontakt, die diesen implementierte,

doch auch sie konnten uns nur bedingt helfen. Zusätzlich zu der Tatsache, dass die Implementation auch ohne unsere Einwirkung schlichtweg nicht funktionierte, kam die schwache Modularisierung des Codes, welche auch im Fall einer funktionierenden Implementation ein einfaches Migrieren mehr oder weniger verunmöglichte. Deshalb entschieden wir uns nach ca. 15 Stunden gemeinsamer Arbeit, eine primitivere Verschlüsselung zu implementieren und einigten uns auf *Cipher Block Chaining* (CBC).

2.3 Cipher Block Chaining

Da dieses Konzept bereits ausführlich in der Vorlesung behandelt wurde, wird sich dieser Report keiner sehr detaillierten Beschreibung bemühen. CBC ist ein Block Cipher, der Regelmässigkeiten im Ciphertext verhindert, in dem vor der Verschlüsselung des Blocks B dieser mit dem zuvor verschlüsselten Block $k(A)$ XORed wird, so dass schlussendlich $k(B) := k(k(A) \bigoplus B)$ gilt. Somit tauchen Regelmässigkeiten im Klartext nicht im Ciphertext auf. Für die Verschlüsselung des ersten Blocks wird ein Initialization Vector (IV) verwendet. Dieser wird im Klartext übermittelt und für jede Nachricht zufällig gewählt.

3 Implementierung, Konfiguration und Setup

Damit Chatnachrichten ausgetauscht werden können, müssen zuerst alle Clients einen Masterfeed erstellen und eine Verbindung aufbauen. Ein Tutorial hierfür im Videoformat ist im Ordner *20-fs-iac-lec/groups/Demo_D* zu finden, oder im Textformat in unserem *README*. Unsere Implementation baut auf den folgenden Projekten auf:

- LogStore (Verwaltung der Feeds in Form von SQlite-Datenbanken)
- LogMerge (Synchronisierung der Feeds)
- Sneakernet (Ermöglicht die Kommunikation via USB-Sticks)
- FeedControl (Ermöglicht das Folgen eines gewissen Feeds, in unserem Fall ist dies wichtig für das Folgen des Chat Feeds, dessen Verschlüsselung wir implementierten)
- SubjectiveChat (Siehe oben)

Die Verwendung des Subjective Chats mit unserer Implementation ist identisch mit der Verwendung ohne. Unser Beitrag spielt sich sozusagen hinter den Kulissen ab, nämlich darin, wie die Nachrichten in die Feeds gespeichert und aus den Feeds ausgelesen werden. Anstatt dass, wie im "normalen" Subjective Chat, die geschriebenen Nachrichten im vom Protokoll vorgegebenen Format direkt abgespeichert werden, durchlaufen sie beim Absenden bzw. Abspeichern ("gesendet" wird ja manuell) den folgenden Prozess: Sie werden mit CBC verschlüsselt und zusammen mit dem IV an die Stelle der Nachricht gespeichert. Zusätzlich wird der aktuelle Public Key angehängt. Beim Lesen der Nachrichten geht unsere Implementation folgendermassen vor: IV, Ciphertext und Public Key werden aus dem Chat Event herausgelesen. Mit dem Private Key und dem IV wird der Ciphertext entschlüsselt und in eine .json-Datei im Klartext lokal gespeichert. Danach wird ein neuer Public Key generiert und anhand dessen und dem erhaltenen Public Key ein neuer Private Key generiert, der für das Entschlüsseln der nächsten Nachrichten verwendet wird. Die Grösse des generierten Private Keys beträgt 32 Byte. Um einen zusätzlichen Austausch der USB-Sticks zu verhindern entschieden wir uns dazu, den ersten Key

Exchange zusammen mit den ersten versandten Nachrichten zu implementieren. Dies bedeutet jedoch, dass alle Nachrichten die versandt wurden, bevor eine Antwort erhalten wurde, nur pseudo-verschlüsselt sind. Sie sind zwar verschlüsselt, jedoch mit einem symmetrischen Schlüssel, der dem Event wie sonst der Public Key angehängt wird. Somit sieht die Nachricht im Feed auf den ersten Blick verschlüsselt aus, doch kann nicht nur der andere Client, sondern jede Instanz die Nachricht entschlüsseln, da der Chat Event sowohl den IV, wie auch den Schlüssel enthält.

Format

Das Folgende ist ein Beispiel für einen Event-Datenbank Eintrag[1] mit unserer Implementation. Die einzelnen Teile des Eintrags sind farblich markiert und darunter benannt.


```
colin#split:#9aZb/frCHcmsSJd1MLtJbA==:n5sfi/Ihz/
z6WBbAQlXT1Q==#split:#msg#split:#pubkey#split:#999483407485526184576412605019
680342733103802772068984866968213076570358185774009788536113414068064732806
200004481533766324051017534582524195565467435802309561307702810211548181142 4
681593692846269303429750106327220328183958684933149148996611024371143395366 3
446675571481677567699504780974169321330445581107710184428366355931772360609 1
730324826313569282811953320961203246039882759780622381784982149282369351216 0
346733781491662740648983154082227981913002550333256042948762091250163252984 5
716608635511111793301288659276665538658278377268061773796460463284217336943 9
070653620825273253544860202976910554396151062792597150 9
```

Benutzername
Initiation Vector (IV)
Verschlüsselte Nachricht
Eventtyp
Public Key

4 Resultate

Während unserem Prozess zur Erstellung der Applikation sind wir auf viele Probleme gestossen, die wir teilweise durch eine Anpassung unserer Projektziele eliminieren mussten. Unsere Applikation ist dazu in der Lage, Text- und Bildnachrichten verschlüsselt an eine Partei zu übertragen. Diese werden nun nicht mehr als Klartext in der Datenbank abgespeichert sondern sind mit einem 32-Byte Schlüssel verschlüsselt. Da eine vollständige Implementierung des Double-Ratchet Algorithmus nicht möglich war, haben wir uns dazu entschieden, verschiedene Meilensteine zu erreichen, um uns stetig diesem Ziel anzunähern. Aufgrund dieser Erkenntnis haben wir mit einem zusätzlichen Zwischenschritt eine der beiden Ratschen implementiert - die Diffie-Hellman-Merkle Ratsche. Dies hat zur Folge, dass nach jedem Nachrichtenzyklus ein neues geteiltes Geheimnis kreiert wird. Um Leser:innen einen besseren Überblick zu verschaffen, soll der im Folgenden definierte Ablauf dienlich sein:

1. User1 generiert einen Public Key und sendet diesen angehängt an eine Nachricht an User2. Die erste

[1] Die Event-Datenbank ist die Datei *eventDatabase.sqlite*, darin werden u. A. die Chat Events gespeichert.

Nachricht ist nicht sicher, da der benötigte Schlüssel für das Entschlüsseln der Nachricht mit der ersten Nachricht mitgeschickt wird.

2. User2 generiert einen Public Key und erstellt mit beiden Public Keys ein geteiltes Geheimnis.
3. User2 schickt eine mit dem geteilten Geheimnis verschlüsselte Nachricht an User1, inklusive seinem Public Key.
4. User1 kreiert nun ebenfalls das geteilte Geheimnis und kann somit die erhaltene Nachricht entschlüsseln. User1 erstellt einen neuen Public Key und schickt diesen zusammen mit einer Nachricht (verschlüsselt, mit IV) an User2.
5. User2 entschlüsselt mit dem alten geteilten Geheimnis die Nachricht, generiert einen neuen Public Key und erstellt mit den neuen Public Keys ein neues geteiltes Geheimnis.

5 Diskussion

5.1 Von der Idee zum Endprodukt

Der ursprünglichen Idee, eine perfekte Verschmelzung vom Subjective Chat mit dem Double Ratchet Algorithmus zu erzielen, mussten wir leider abtrünnig werden, da einige der Schwierigkeiten nahezu unmöglich zu überwinden waren. Da die Implementation des Double-Ratchet Algorithmus in seiner BACnet-Form nicht funktioniert, sind wir nach einem Brainstorming auf die Idee gekommen, uns mit dem Konzept von Meilensteinen möglichst nahe zum Konzept von Double Ratchet hinzubewegen. Somit war es uns möglich, grössere Probleme langsam anzugehen und die Komplexität schrittweise zu erhöhen.

Übersicht über die Meilensteine

Zu beginn haben wir die Chat-Nachrichten mittels CBC und einem zufällig-generierten Schlüssel verschlüsselt, welchen wir der Nachricht gleich darauf angefügt haben. Nachdem dieser erste Meilenstein implementiert wurde, setzten wir uns das Ziel, mittels einem Diffie-Hellman-Merkle Key Exchange einen gemeinsamen Private Key auszutauschen und die Nachrichten mit diesem zu verschlüsseln. Der nächste und gleichzeitig auch letzte Meilenstein, den wir realisieren konnten, war die Erweiterung der asymmetrischen Kryptographie durch eine Ratsche des Double-Ratchet Algorithmus, mit welcher nach jedem Zyklus von Nachrichten ein neuer privater Schlüssel ausgetauscht wird. Somit bietet die Implementation Forward- und Backward Secrecy. Unsere Implementation ist forward- und backward-secure. Der vielleicht nicht ganz so triviale Unterschied zum Double Ratchet Algorithmus lässt sich damit erklären, dass bei unserer Implementation jeweils nur jeder Zyklus so verschlüsselt wird und nicht jede einzelne Nachricht.

5.2 Gruppenchat-Funktionalität

Die bereits implementierte Gruppenchat-Funktion verlangt offensichtlich eine andere Implementierung des Verschlüsselungsverfahrens, da das Protokoll mit mehr als zwei Teilnehmern grundlegend Verschieden ist. Um unserem Ziel einen weiteren Schritt näher zu kommen haben wir im Endeffekt einen Versuch unternommen, die Sicherheit des Gruppenchats ebenfalls zu gewährleisten. Unglücklicherweise stellte sich dieses Unterfangen

als viel schwieriger heraus, als wir erwartet hatten. Die von uns verwendete Implementation zur Erzeugung von gemeinsamen Privatschlüsseln war unbrauchbar für mehr als zwei Chat-Teilnehmer, da die ausgetauschten Schlüssel, nicht alleinig durch die, an die Nachricht angehängten, öffentlichen Schlüssel erzeugt werden können. Dies bewirkte, dass wir eine Alternative dazu finden mussten, wobei wir schlussendlich gescheitert sind. Unsere Bemühungen sind im Repository auskommentiert und bereit von einer weiteren Gruppe, in einem späteren Semester, den Feinschliff verpasst zu kriegen.

5.3 Ausblick

Auf den von uns erarbeiteten Ergebnissen ist es durchaus möglich, mit diesen Konzepten weiterzuarbeiten und Neuerungen hinzuzufügen. Es wäre zum Beispiel denkbar, den tatsächlichen Double Ratchet Algorithmus zu implementieren und der Anwendung eine funktionierende und optimalerweise sogar effiziente Gruppenchatfunktion hinzuzufügen. Wenn man sich mit dem Ausblick an einen etwas objektiveren Standpunkt begibt, begegnet man der Auffälligkeit, dass das BACnet noch vermaschter miteinander interagieren könnte. Wir denken diesbezüglich, dass die kommenden Jahrgänge das Potential dieser internen Vermaschung immer mehr ausschöpfen können.

5.4 Reflexion

Obwohl wir auf viele Probleme gestossen sind, sind wir mit unserer Eigenleistung im Rahmen dieser Arbeit zufrieden. Das Projekt war anspruchsvoll und erlaubte es uns, den in der Vorlesung behandelten Stoff praktisch umzusetzen und somit zu vertiefen. Insbesondere aufgrund unserer stetigen Erhöhung der Komplexität und dem hinzufügen von mehr Funktionalitäten sind wir unserem Ursprungsziel nahe gekommen. Hätten wir darauf beharrt, direkt einen Double Ratchet Algorithmus zum laufen zu bringen, wären wir aller Wahrscheinlichkeit nach viel weniger weit gekommen. Somit konnten wir viel Neues lernen und können diesen Report mit gutem Gewissen an dieser Stelle abschliessen.

6 Referenzen

- https://signal.org/docs/specifications/doubleratchet/ (14.07.2021)
- https://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode (14.07.2021)
- https://en.wikipedia.org/wiki/Double_Ratchet_Algorithm (14.07.2021)
- https://pypi.org/project/pyDH/ (14.07.2021)
- https://github.com/cn-uofbasel/BACnet/ (14.07.2021)
- https://www.sqlite.org/index.html (14.07.2021)