FOTEC
Forschungsunternehmen
-der FH Wiener Neustadt-

SIKOSI

Konzept

FOTEC
Forschungsunternehmen
-der FH Wiener Neustadt-

Inhalt

Interviews mit Entwicklern ... 2
Sicherheitsniveau Matrix ... 2
Minimale Sicherheit ... 2
Ausreichende Sicherheit ... 3
Maximale Sicherheit ... 3
Clientfunktionalität ... 3
Stand der Technik ... 4
Ende zu Ende Kommunikation ... 4
Key Agreement Protokolle ... 7
Diffie-Hellman Key Exchange ... 7
Elliptic Curve Diffie-Hellman Key Exchange (ECDH) ... 7
Kryptografische Protokolle (Cipher-Suite) ... 8
Extended Triple Diffie Hellman Key Agreement Protocol (X3DH) ... 8
Datensicherung / Verschlüsselung ... 13
AES ... 13
SHA ... 13
Twofish ... 14
Zu verwendende Open Source Algorithmen & Bibliotheken ... 14
Systemarchitektur ... 17

FOTEC
Forschungsunternehmen
-der FH Wiener Neustadt-

Entwickler Interviews

Um die Anforderungen an SIKOSI zu evaluieren, werden von mehreren Entwicklern Fragebögen ausgefüllt. Damit sollen Stolpersteine bei der Entwicklung von sicheren Anwendungen eruiert und wünschenswerte Funktionen eruiert werden. Der Fragebogen befindet sich im Anhang dieses Dokuments.

Sicherheitsniveau Matrix

Stufe	Beschreibung
Maximale Sicherheit	• Daten sind verschlüsselt, Key dafür wird per Key Agreement Protocol generiert. Datenverbindung selbst ist ebenfalls verschlüsselt mit TLS/HTTPs • Authentifikation am Server per Zero Knowledge Protocol. • Sensible Daten in DB per Hashing Algorithmus verschlüsselt. • Sensible Felder in Datenbank nochmal per field level encryption verschlüsselt.
Ausreichende Sicherheit	• Die Daten sind verschlüsselt, um Key zu generieren wird ein Key Agreement Protocol (z.b. Diffie Hellman) verwendet. Die Datenverbindung selbst ist unverschlüsselt. • Authentifikation am Server per Zero Knowledge Protocol. • Sonstige sensible Daten die gespeichert werden müssen werden per SHA-3 in DB gespeichert, die DB selbst ist nicht verschlüsselt.
Minimale Sicherheit	• Daten sind selbst unverschlüsselt, der Übertragungsweg ist per TLS/HTTPs verschlüsselt. • Authentifikation erfolgt mit Username und Passwort, die Passwörter werden in der Datenbank per SHA-3 gehashed und persistiert. Die DB selber ist unverschlüsselt.

Sicherheitsniveau Matrix, Quelle: FOTEC GmbH

Die Sicherheitsniveau Matrix gliedert sich grundsätzlich in drei Sicherheitsstufen, die nachfolgend erklärt werden:

Minimale Sicherheit

Die minimale Sicherheitsstufe sollte grundsätzlich nur gewählt werden, wenn es sich um keine sensiblen Daten bei der Datenübertragung handelt oder es sich um eine rein interne Anwendung handelt. Die minimale Stufe bietet zwar grundsätzlichen Schutz, kann jedoch im Vergleich zu den beiden anderen Sicherheitsstufen nicht dasselbe Maß an Sicherheit gewährleisten. Während der Übertragungsweg per TLS[1] verschlüsselt wird und es einem passiven Angreifer nicht möglich ist, den Traffic im Nachhinein zu entschlüsseln, ist dieser Vorteil nichtig, wenn es dem Angreifer gelingt sich als Man-in-the-Middle in den Datenaustausch einzuklinken, und den beiden Gesprächspartnern jeweils vorgaukeln kann, er sei der jeweils andere Gesprächspartner.

Eine weitere potentielle Schwachstelle ist die Tatsache, dass die Authentifikation ein Passwort benötigt. Während diese Vorgehensweise durchaus üblich ist, ist sie von einem Sicherheitsstandpunkt betrachtet sehr bedenklich. Nicht nur, dass es wenn auch schwierig möglich ist das Passwort, sobald es über ein Netzwerk übertragen wird, abzufangen und zu entschlüsseln, ergibt sich außerdem das Problem der Speicherung. Die Speicherung der Passwörter so wie es die minimale Sicherheitsstufe vorsieht, nämlich unter der Verwendung von einem Hash Algorithmus, ist zwar nicht per se als unsicher zu werten, jedoch für die heutige Zeit nicht mehr genug, um von einem ausreichend geschützten System auszugehen. Als Zusatzmaßnahme könnte die Datenbank selbst zusätzlich verschlüsselt werden, wodurch es insgesamt schwerer würde, an die Passwort Hashes zu gelangen, jedoch das Problem insgesamt nicht gelöst wäre, dass die Passwörter irgendwo gespeichert und über das Netzwerk übertragen werden müssten.

[1] https://en.wikipedia.org/wiki/Transport_Layer_Security

Ausreichende Sicherheit

Im Gegensatz zur minimalen Sicherheitsstufe wird bei dieser Stufe nicht die Verbindung selbst, sondern die Daten verschlüsselt. Um zu verhindern, dass ein passiver Angreifer der sich in die Verbindung einklinkt die gesendeten Datenpakete entschlüsseln kann, wird hier ein sogenanntes Key-Agreement Protokoll[2] zur Verwendung gezogen.

Diese Art von Protokoll basiert darauf, dass der Schlüssel mit dem die Daten verschlüsselt werden nicht von einer Seite generiert und daraufhin der anderen Seite übermittelt wird, sondern sich beide Parteien auf eine Weise auf den Schlüssel einigen, die es nicht zu lässt, dass ein (oder mehrere) unbefugter Dritter diesen Schlüssel eruieren kann.

Key Agreement Protokolle werden in einem nachfolgenden Kapitel erklärt. Die grundsätzliche Funktionsweise beläuft sich jedoch darauf, dass sich beide Parteien auf eine gemeinsame Basis sowie einen gemeinsamen Modulo einigen, daraufhin jeweils eine geheime Zahl wählen, und Berechnungen durchführen an deren Ende beide dasselbe Ergebnis vorliegen haben, ohne jemals genug Informationen über das Netzwerk gesendet zu haben, um es einem Angreifer zu ermöglichen, auf das Passwort rück zu schließen.

Um der bereits angesprochenen Problematik der Authentifikation mit Passwort entgegen zu wirken, verwendet diese Sicherheitsstufe für die Authentifikation ein Protokoll das nach dem Zero-Knowledge-Prinzip vorgeht. In einem späteren Kapitel wird auf dieses Prinzip inklusive zweier Protokolle die danach vorgehen genauer eingegangen, die grundlegende Idee eines solchen Protokolls beläuft sich jedoch darauf, bei der Authentifikation am Server dem Server lediglich zu beweisen das Passwort zu kennen, ohne jemals in die Notwendigkeit geraten das Passwort direkt zu übermitteln, wodurch dieses weder aus einer Datenbank ausgelesen, noch während des Transports abgefangen werden kann.

Maximale Sicherheit

Die maximale Sicherheitsstufe setzt es sich wie unschwer aus dem Namen abzuleiten ist zum Ziel, das höchste Maß an Sicherheit zu bieten. Wie auch bei der vorigen Variante werden hier die Daten selbst mithilfe eines Key-Agreement Protokolls verschlüsselt. Um zusätzliche Sicherheit zu bieten, findet die Datenübertragung hier jedoch ebenfalls verschlüsselt statt. Wie auch bei der ausreichenden Sicherheitsstufe findet die Authentifikation des Benutzers nicht über ein Senden des Passworts, sondern über das Zero-Knowledge Prinzip statt. Für jene sensiblen Daten, die in der Datenbank gespeichert werden müssen, beispielsweise den User Verifier und das User Salt (siehe Zero-Knowledge Prinzip), wird ein Verschlüsselungsalgorithmus gewählt. Außerdem ist es vorgesehen, dass zusätzlich zu den Daten noch die Datenbank auf Feldebene verschlüsselt wird. Mögliche Verschlüsselungsalgorithmen werden im nachfolgenden Kapitel behandelt.

Clientfunktionalität

Der zu implementierende Client soll idealerweise alle sicherheitsrelevanten Bereiche der Anwendungsentwicklung abdecken, mindestens jedoch die folgenden:

Datensicherheit

Dies umfasst Funktionen die zur Ver- und Entschlüsselung von Daten notwendig sind. Dabei soll es unerheblich sein ob es den Datentransport oder die Datensicherung betrifft. Ein Entwickler soll durch

[2] https://en.wikipedia.org/wiki/Key-agreement_protocol

einfache Methodenaufrufe dazu in der Lage zu sein, Klartext zu verschlüsseln sowie verschlüsselte Daten zu lesen.

Authentifizierung

SIKOSI soll über Methoden verfügen, die einen Zero Knowledge Abgleich von Authentifizierungsinformationen erlauben. Konkret bedeutet dies, dass sich ein Client gegenüber einem Server oder einem anderen Client authentifizieren kann, ohne dass tatsächlich ein Passwort übermittelt wird. Der Vorteil an einer solchen Methode ist, dass ein möglicher Angriffsvektor entfällt. Die Details sind im Kapitel Zero-Knowledge Prinzip beschrieben.

Code Security

In der Softwareentwicklung wird oft zusätzliche Funktionalität in Form von Programmbibliotheken bzw. Programmpaketen (z.B. NuGet) in ein Projekt eingebunden. Es fehlt jedoch meist die Zeit, den eingebundenen Code auf seine Sicherheit zu überprüfen. Aus diesem Grund soll SIKOSI auch Funktionen zur Verfügung stellen, die einen Entwickler warnt, falls extern eingebundener Code als potentiell gefährdend erkannt wird. Dies soll durch Abgleich mittels OWASP[3] oder der OSS Index Datenbank[4] geschehen.

Stand der Technik

Verfügbare Technologien für sichere Ende zu Ende Kommunikation sowie Datenspeicherung.

Ende zu Ende Kommunikation

Zero-Knowledge Prinzip

Das Zero-Knowledge Prinzip ist eine Art der Authentifizierung, die es erlaubt einem Partner zu beweisen ein Passwort zu kennen, ohne in die Notwendigkeit zu gelangen, das Passwort selbst an den Gesprächspartner zu schicken. Auf diese Weise ist es einem Angreifer weder möglich das Passwort selbst zu sehen, noch das Passwort zu erraten, da diesem dafür nicht die nötigen Informationen zur Verfügung stehen.

Dieses Prinzip findet sich auch in der PAKE Technologie wieder. PAKE (Password Authenticated Key Exchange) bietet ähnliche Eigenschaften wie das Zero-Knowledge Prinzip. Ein PAKE ist eine spezielle Form eines Key-Agreement Protokolls, bei dem der Schlüssel basierend darauf generiert wird, ob der Benutzer ein Passwort kennt. Hierbei wird das Passwort jedoch ähnlich wie beim Zero-Knowledge Prinzip, weder an den Server geschickt, noch irgendwo gespeichert. Der grundlegende Einsatzfall eines PAKEs ist zwar der verschlüsselte Informationsaustausch durch generieren eines Schlüssels, es bietet jedoch auch die Möglichkeit lediglich zur Verifizierung eines Benutzers zu dienen. Zwei PAKE Protokolle werden nachfolgend vorgestellt.

Secure Remote Password (SRP)[5]

Secure Remote Password war eines der ersten PAKE und wurde 1998 entwickelt. Das Protokoll besteht grundsätzlich aus drei Schritten.

- Der *Registration*, beim erstmaligen Registrieren eines Benutzers.
- Der *Authentifizierung* beim Anmelden des Benutzers.

[3] https://owasp.org/
[4] https://ossindex.sonatype.org/
[5] https://blog.1password.com/developers-how-we-use-srp-and-you-can-too/

- Der *Verifizierung* nach dem Schritt der Authentifizierung.

Im Schritt der Registration generiert der Benutzer ein zufälliges *Salt*[6]. Aus dem Benutzernamen, dem Passwort und dem generierten Salt berechnet der Benutzer als nächstes mittels einer Key-Derivation-Funktion[7] einen Wert **x**. Im letzten Schritt generiert der Benutzer aus dem errechneten Wert **x**, sowie einer SRP-Gruppe[8] einen Verifier. Dieser Verifier wird jetzt zusammen mit dem Salt an den Server geschickt, welcher für den jeweiligen Benutzer den Verifier und das Salt speichert.

Im Schritt der Authentifizierung sendet der Server das Salt, sowie die SRP Gruppe für den jeweiligen Benutzer auf Anfrage an den Client. Der Client berechnet, wie auch im Schritt der Registration, mittels einer KDF den Wert x. Zusätzlich generieren Client und Server jeweils einen geheimen und einen öffentlichen Wert mithilfe der SRP Gruppe, die mit klein **a** und klein **b**, sowie groß **A** und groß **B** betitelt werden. Die errechneten öffentlichen Werte werden nun jeweils an den Partner geschickt, sodass am Ende Client und Server jeweils folgende Daten besitzen:

Client: x, a, B

Server: Verifier, b, A

Nun können beide Teilnehmer aus den jeweils ihnen bekannten Werten einen gemeinsamen Schlüssel generieren.

Der letzte Schritt der Verifizierung zielt darauf ab, zu bestätigen, dass beide Teilnehmer tatsächlich denselben Schlüssel erhalten haben. Dafür verschlüsselt der Client eine Nachricht mit dem gerade gewonnenen Schlüssel, schickt diese an den Server, welcher die Nachricht entschlüsselt und bestätigt, dass die Nachricht korrekt angekommen ist.

SRP – Vorteile:

- Speichert weder Passwort, noch einen Passwort Hash am Server.
- Es ist leicht zu implementieren und es existiert Code diesbezüglich.
- Im Toolkit OpenSSL ist funktionierender SRP-Code zu finden.
- Resistent gegen einen passiven Angreifer.

SRP – Nachteile:

- Das Design gilt als schlecht erweiterbar im Hinblick auf elliptische Kurven, welche in der Kryptographie oft genutzt werden, da es mit ihnen möglich ist, dieselbe Sicherheit mit kürzeren Schlüsseln zu erreichen.
- Das Protokoll ist so gebaut, dass es den Salt Wert gleich mit der ersten Nachricht an den Benutzer herausgibt. Das macht das Protokoll verwundbar für pre-computation attacks[9] [10]

[6] Ein Salt beschreibt einen zufälligen String.
[7] https://en.wikipedia.org/wiki/Key_derivation_function
[8] https://tools.ietf.org/html/rfc5054#page-16
[9] https://www.geeksforgeeks.org/understanding-rainbow-table-attack/
[10] Artikel hierzu: https://blog.cryptographyengineering.com/should-you-use-srp/

OPAQUE[11]

Opaque gehört wie auch SRP, zur Familie der PAKE Protokolle, behebt aber einige der Nachteile von letzterem. Beispielsweise ist es möglich, OPAQUE relativ einfach mit elliptischen Kurven zu verwenden, wodurch es insgesamt wesentlich effizienter ist. Außerdem löst OPAQUE das Problem, das durch das Senden des User Salts entsteht, indem für OPAQUE anstatt das Salt auf Anfrage an den User zu senden, mithilfe einer sogenannten „Oblivious PRF“. Das Prinzip dieser Oblivious PRF beruht darauf, dass Server und Client zusammen eine Funktion der Form **PRF(Salt, Passwort)** berechnen können, in der beide Parteien den Input der jeweils anderen Partei nicht kennen.

Oblivious PRF (Oblivious Pseudo Random Function)

Die Idee der Oblivious PRF ist es, dass der Client das Passwort und der Server das Salt besitzt, und anhand dessen ein relevantes Ergebnis gewonnen werden kann, ohne dass weder Passwort noch Salt der jeweils anderen Partei bekannt sind. Der gewünschte Output der PRF wird durch die Formel $\mathbf{H(P)^s}$, also der Hash des Passworts hoch dem Salt, angeschrieben. Für die Berechnung wird auf der Seite des Clients zu aller erst der Hash des Passworts berechnet. Die Hash Funktion muss außerdem die Eigenschaft haben, Elemente die gehashed werden in Form einer zyklischen Gruppe zu hashen.

Dieser Hash wird anschließend mit einem zufälligen skalaren Wert[12] **r** verunreinigt um **C** zu erhalten. Die Formel die sich aus diesem Schritt ergibt lautet angeschrieben $\mathbf{C = H(P)^r}$. Nachdem der Wert C an den Server geschickt wird, führt dieser die Berechnung in der Form $\mathbf{R = C^s}$ fort, und schickt das Ergebnis R zurück an den Client. Die aktuelle Formel lässt sich anschreiben in der Form $\mathbf{R = H(P)^{rs}}$. Der Client berechnet nun unter der Einbeziehung des inversen Elements vom ursprünglich gewählten zufälligen Wert r den Wert R‘, welcher das gewünschte Ergebnis der PRF Funktion darstellt.

Die einzelnen Schritte in Form von Formeln lauten wie folgt:

Schritt 1 (Client): $\mathbf{C = H(P)^r}$

Schritt 2 (Server): $\mathbf{R = H(P)^{rs}}$

Schritt 3[13] (Client): $\mathbf{O = R^{ir}}$

Nun liegt das gewünschte Ergebnis in der Form $\mathbf{O = H(P)^s}$ vor.

Überprüfung ob der Client das korrekte Passwort eingegeben hat

Um den Client Authentifizieren zu können, ist es nötig zu wissen, ob das eingegebene Passwort das korrekte war, oder nicht. Die Problematik die sich hier ergibt ist jene, dass der Server weder den Schlüssel kennt den der Client sich errechnet hat, noch weiß ob es der korrekte Schlüssel ist. Um diese Problematik zu lösen, werden beim Registrationsvorgang auf Seite des Users (per Key-Agreement Protokoll) ein Public und ein private Key generiert. Anschließend werden der private Key des Clients mit dem Public Key des Servers unter dem errechneten Schlüssel **K** verschlüsselt. Der daraus resultierende Schlüsseltext wird jetzt in der Datenbank des Servers gespeichert.

Bei jedem darauffolgenden Authentifizierungsvorgang muss der Client erneut den Schlüssel **K** unter Angabe des Passworts berechnen. Schickt nun der Server den gespeicherten verschlüsselten Text an

---

[11] OPAQUE wissenschaftl. Dokument: https://eprint.iacr.org/2018/163.pdf
OPAQUE Blog Eintrag: https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/

[12] Eine Variable die nur einen Wert speichert, im Gegensatz zu einem Array.

[13] **O** steht hierbei für das gewünschte Ergebnis in der Form der PRF.
ir steht hierbei für das inverse Element zu dem Wert **r**.

den Client, kann dieser den Text nur dann entschlüsseln, wenn er sich in Besitz des korrekten Schlüssels **K** befindet, was wiederum nur der Fall sein kann, wenn der Client das Passwort auch tatsächlich kennt und richtig angegeben hat. Kann der Text entschlüsselt werden, ist das korrekte Passwort dem Client also bekannt, kann nun mit dem darin enthaltenen öffentlichen Server Schlüssel ein Authentifizierungsverfahren gestartet werden bei dem die Kommunikationspartner jeweils die Angaben des anderen mit den gespeicherten public keys des jeweils anderen vergleichen.

Key Agreement Protokolle

Die Funktionsweise eines Key Agreement Protokolls erlaubt es einen kryptographischen Schlüssel zum Verschlüsseln von Nachrichten gemeinsam mit dem Kommunikationspartner zu generieren. Im Gegensatz zur herkömmlichen Variante, bei der ein Schlüssel von einer Partei generiert, und der anderen übermittelt wurde, ist diese Variante gegenüber passiven Angreifern geschützt.

Eine erste Variante eines Key Agreement Protokolls wurde mit dem Diffie-Hellman Key Exchange entwickelt.

Diffie-Hellman Key Exchange

Dieser beruht auf der Annahme, dass die diskrete Exponentialfunktion leicht (kurze Dauer) funktioniert, jedoch die Umkehrung der diskreten Exponentialfunktion (diskreter Logarithmus) schwer (sehr lange Dauer). Das ist bis heute noch nicht bewiesen, allerdings gibt es noch keinen Algorithmus der das Problem des diskreten Logarithmus effizient lösen kann.

Der Diffie-Hellman Schlüsselaustausch erlaubt es über ein ansonsten unsicheres Netzwerk sicher zu kommunizieren. Um das zu erreichen, einigen sich die beiden Parteien zunächst auf einen gemeinsamen Modulo, sowie eine gemeinsame Basis.

Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

Dabei wird die praktisch nicht umkehrbare Funktion der diskreten Exponentialfunktion ersetzt durch die Addition von 2 Punkten auf einer elliptischen Kurve über einem endlichen Körper.
Der große Vorteil daran ist, dass sich die Schlüssellänge bei gleichbleibender Sicherheit reduziert. Je größer die Sicherheit sein soll desto größer wird auch der Unterschied in der benötigten Schlüssellänge.

RSA key size (bits)	ECC key size (bits)	Security level (bits)	Ratio of key size
1024	160	80	6.4:1
2048	224	112	9.1:1
3072	256	128	12:1
7680	384	192	20:1
15360	512	256	30:1

Quelle: https://www.semanticscholar.org/paper/A-Reconfigurable-High-Speed-ECC-Processor-Over-NIST-Ding-Li/6f82ddeda686c9bc6262dbfee0c84502e414de84#extracted

Kryptografische Protokolle (Cipher-Suite)

Um einen sicheren Datenaustausch zu gewährleisten müssen die Daten nicht nur verschlüsselt werden, sondern es müssen auch die verschiedenen Probleme bei der Übertragung der Daten und der Kommunikation gelöst werden.

1) Schlüsselaustausch: RSA, DH, ECDH

Asymmetrische Verschlüsselung um gemeinsamen Schlüssel auszutauschen

2) Authentifizierung/Signieren: RSA, DSA, ECDSA, EdDSA

Um Man-in-the-Middle-Angriffe zu verhindern

3) Verschlüsselung: AES

Symmetrische Verschlüsselung wesentlich schneller als asymmetrische Verschlüsselung

4) Hashfunktion: SHA2, SHA3, HMAC

Zur Integritätsprüfung (ob Daten verändert wurden)

Elliptic Curve (EC) – Verfahren sind die moderneren Varianten beim Schlüsselaustausch sowie beim Signieren. Vorteile sind bedeutend kürzere Schlüssel und (mit den richtigen Kurven) auch effizienter in der Berechnung der Schlüssel. Aber bei der Auswahl der Kurve muss man vorsichtig sein, da die von der NIST empfohlenen Kurven im Verdacht stehen Backdoors der NSA eingebaut zu haben (angeblich per reinem Zufall ausgewählte Punkte auf der Kurve könnten so gewählt worden sein, dass Entschlüsselung schneller funktioniert als angegeben). Allgemeine Empfehlungen gehen daher Richtung Curve25519 (spezielle elliptische Kurven mit dem Primzahl-Modulo 2^255 – 19), Edwards-Kurven und Twisted-Edwards-Kurven, die eine schnelle effiziente Berechnung der Punkte auf der Kurve ermöglichen. Die Curve25519 wird bereits in vielen Anwendungen verwendet[14,15].

Extended Triple Diffie Hellman Key Agreement Protocol (X3DH)

https://signal.org/docs/specifications/x3dh/

IK_A Alice's identity key

EK_A Alice’s ephemeral key

IK_B Bob's identity key

SPK_B Bob's signed prekey

OPK_B Bob's one-time prekey

[14] https://ianix.com/pub/curve25519-deployment.html
[15] https://ianix.com/pub/ed25519-deployment.html

Schritt 1: Bob veröffentlicht (i.e. Upload auf Server) Prekey-Bundle (mehrere öffentliche Schlüssel für unterschiedliche Zwecke) – siehe https://signal.org/docs/specifications/x3dh/#publishing-keys

- Bob's identity key IK_B
- Bob's signed prekey SPK_B
- Bob's prekey signature $Sig(IK_B, Encode(SPK_B))$
- A set of Bob's one-time prekeys (OPK_B^1, OPK_B^2, OPK_B^3, ...)

Schritt 2: Alice holt Bob's Prekey-Bundle vom Server und verifiziert Prekey-Signature.

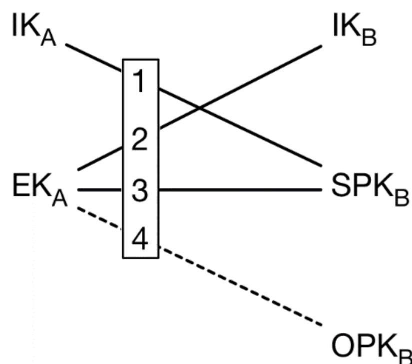
https://signal.org/docs/specifications/x3dh/#sending-the-initial-message

Wenn Ok, berechnet Alice:

```
    DH1 = DH(IK_A, SPK_B)
    DH2 = DH(EK_A, IK_B)
    DH3 = DH(EK_A, SPK_B)
    SK = KDF(DH1 || DH2 || DH3)
```

Oder (wenn Bob's Prekey-Bundle auch einen One-Time-Key (OPK_B) enthält):

```
    DH1 = DH(IK_A, SPK_B)
    DH2 = DH(EK_A, IK_B)
    DH3 = DH(EK_A, SPK_B)
    DH4 = DH(EK_A, OPK_B)
    SK = KDF(DH1 || DH2 || DH3 || DH4)
```



Quelle: https://signal.org/docs/specifications/x3dh/#sending-the-initial-message

Dann sendet Alice die erste Nachricht an Bob mit folgendem Inhalt:

- Alice's identity key IK_A
- Alice's ephemeral key EK_A
- Identifiers stating which of Bob's prekeys Alice used
- An initial ciphertext using *AD* (`AD = Encode(IK_A) || Encode(IK_B)`) as associated data and using an encryption key which is either *SK* or the output from some cryptographic PRF keyed by *SK*.

Schritt 3:

https://signal.org/docs/specifications/x3dh/#receiving-the-initial-message

Bob berechnet SK ebenfalls aus IK_A und EK_A (aus erster Nachricht von Alice) sowie seinem öffentlichen Schlüssel IK_B und den passenden privaten Schlüssel des von Alice verwendeten öffentlichen Schlüssel von Bob.

Danach werden Nachrichten mit SK oder einem von SK abgeleiteten Schlüssel verschlüsselt.

Wenn einer der Vorgänge nicht erfolgreich abgeschlossen wird, wird alles verworfen!

Double Ratchet Algorithmus

- KDF → Eine Funktion die zwei Inputs bekommt, einen zufälligen Schlüssel und einen geheimen Input, und aus diesen zwei Parametern einen Schlüssel (folglich KDF-Key genannt) generiert.

Ein wichtiges Konzept beim Double Ratchet Algorithmus ist die sogenannte KDF-Chain. Anstatt lediglich eine einzige KDF zu nutzen und mit ihr einen Schlüssel zu generieren, werden bei einer KDF-Chain mehrere KDF hintereinander gereiht. Dabei wird ein Teil des jeweils resultierenden KDF-Keys als Output Schlüssel genutzt, während der andere Teil als Input Key für die nächste KDF genutzt wird.

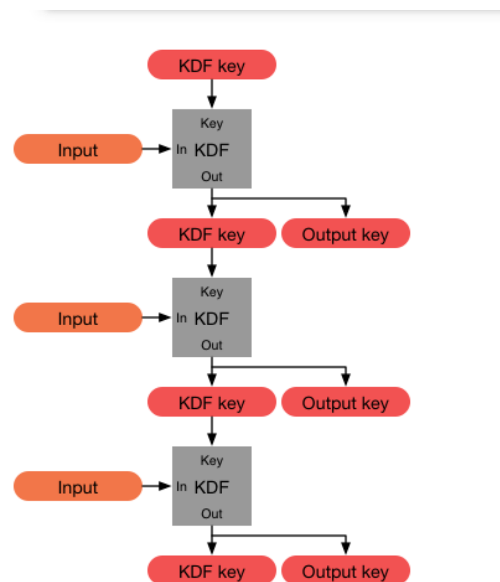


Abbildung 1 KDF-Chain. Quelle: https://signal.org/docs/specifications/doubleratchet/

Jeder der beiden Teilnehmer einer Double-Ratchet Session speichert einen KDF Key für drei KDF-Chains. Die Root-Chain, die Sending-Chain und die Receiving-Chain. Die Sending-Chain des einen Teilnehmers deckt sich mit der Receiving-Chain des anderen Teilnehmers, und umgekehrt.

Bei jedem Austausch einer Nachricht werden neue Diffie Hellmann public Keys generiert, und die jeweils resultierenden Geheimnisse werden als KDF Keys für die Receiving und Input Chain verwendet.

Symmetric – Key Ratchet

Die Input und Receiving KDF-Chains sind dafür verantwortlich, dass jede Nachricht die verschickt wird, mit einem einzigartigen Schlüssel verschlüsselt ist, der gelöscht werden kann, sobald der Vorgang des Nachricht Ent bzw. Verschlüsselns abgeschlossen ist.

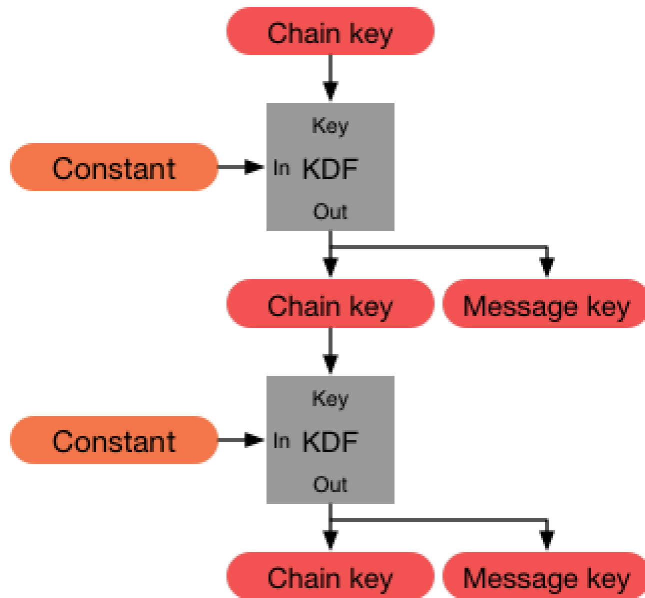


Abbildung 2 Symmetric Key Ratchet. Quelle: https://signal.org/docs/specifications/doubleratchet/

Diffie-Hellman Ratchet

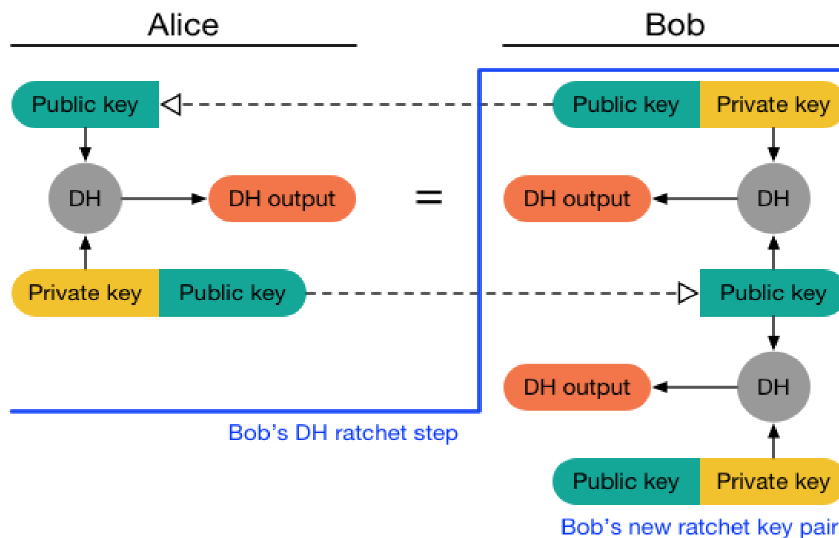


Abbildung 3 DH-Ratchet mit abwechselndem erneutem generieren des DH Outputs. Quelle: https://signal.org/docs/specifications/doubleratchet/

Diffie-Hellman Key Ratchet

Beide Teilnehmer generieren zunächst ein Diffie-Hellman Key-Pair, welches nachfolgend als Ratchet Key Pair bezeichnet wird. Jede Nachricht enthält nun im Header den aktuellen Ratchet Public Key. Sollte sich der Public Key des anderen nun ändern, findet lokal ein DH Ratchet Step statt. Dieser DH Ratchet Step resultiert in einem völlig neuen Ratchet Key Pair. Dadurch entsteht ein Hin und Her bei dem jeweils eine Partei das eigene Key Pair ändert. Dies resultiert in dem Vorteil, dass selbst wenn es

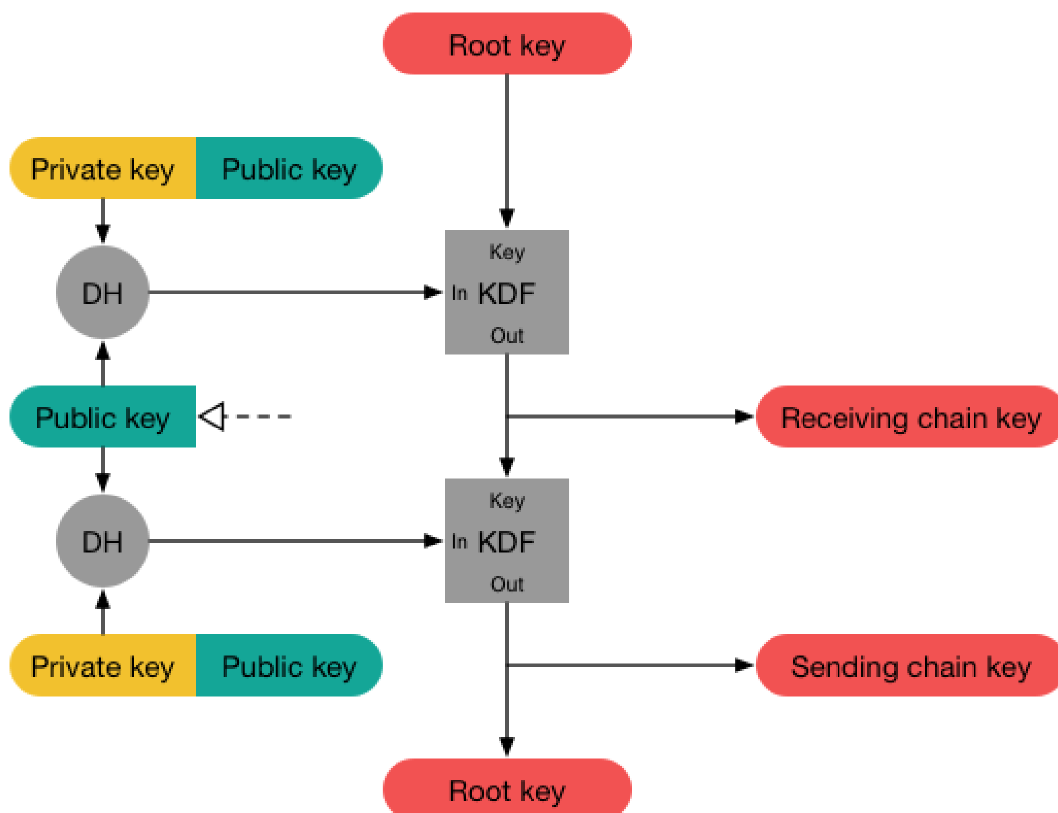
einem Angreifer gelingen würde, den private Key eines Teilnehmers zu stehlen, dieser Key sehr bald mit einem völlig unabhängigen neuen privaten Schlüssel ersetzt werden würde.

In jener Grafik ist der Ablauf des Diffie-Hellman Ratchet dargestellt. Nachdem beide Parteien ein Public/Private Key-Pair generiert haben, schickt Bob seinen Public Key an Alice. Alice generiert aus dem eigenen Private Key, sowie Bobs Public Key den DH-Output. Bei ihrer nächsten Nachricht übermittelt sie ihren eigenen Public Key an Bob. Nun berechnet Bob aus seinem eigenen Private Key, sowie Alices Public Key einen DH Output. Dieser DH Output ist derselbe den Alice im Schritt davor berechnet hat.

Anschließend verwirft Bob sein aktuelles Paar aus Public und Private Key und generiert einen völlig neuen DH Output und das Spiel wiederholt sich von vorne.

Die jeweils neu generierten DH Outputs werden als Chain Keys, also als Input KDF Keys für die Root Chain genutzt. Die Outputs der Root Chain können dann in weiterer Folge als Keys für die Receiving und Input Chain genutzt werden.

Im Double-Ratchet werden die Konzepte aus dem DH-Ratchet und dem Symmetric Key Ratchet vereint. Wenn eine Nachricht empfangen wird, wird ein Symmetric-Key Ratchet Step durchgeführt. Sollte im



Ein einziger vollständiger Double-Ratchet Step. Die Root Chain wird 2x geupdated, woraus ein Receiving Chain Key und ein Sending Chain Key gewonnen werden.

Header der Nachricht ein neuer Public Key empfangen werden, so wird vor diesem Symmetric-Key Ratchet Step ein DH-Ratchet Step durchgeführt.

Um double Ratchet zu implementieren braucht es folgende Funktionen:

- GENERATE_DH → Gibt ein Diffie Hellman Key Pair zurück
- DH(dh_pair, dh_pub) → Gibt den Output von einer Diffie Hellman Berechnung zurück zwischen private Key und public Key.
- KDF_RK(rk, dh_out) → Retourniert ein Schlüsselpaar (32 byte chain key, 32 byte message key).
- KDF_CK(ck) → Retourniert ein Schlüsselpaar (32 byte chain key, 32 byte message key)
- ENCRYPT(mk, plaintext, associated_data) → Gibt Ciphertext des plaintext über dem Message Key verschlüsselt zurück.
- DECRYPT(mk, ciphertext, associated_data) → Gibt Plaintext des Ciphertexts über dem Message Key entschlüsselt zurück.
- HEADER(dh_pair, pn, n) → Erstellt neuen Message Header der den public key des DH Key pairs, sowie die vorherige Kettenlänge und die Nachrichten Nummer enthält.
- CONCAT(ad, header) → Enkodiert einen Nachrichten Header in eine Byte Sequenz

Datensicherung / Verschlüsselung

Es folgt eine Übersicht über derzeit gängige Verschlüsselungsalgorithmen. Diese können sowohl für die Kommunikation als auch für die Datensicherung verwendet werden.

AES

Den Advanced Encryption Standard gibt es in den Ausführungen 128, 196, sowie 256. Alle drei Varianten ent- und verschlüsseln Daten in Blöcken zu je 128 Bit, verwenden dafür jedoch unterschiedliche Schlüssellängen. Die hierbei verwendeten Schlüssel sind symmetrische Schlüssel.

Die Verschlüsselungsmethoden AES-196 und AES-256 gelten als ausreichend, um sensible Informationen der Klasse „Geheim“, sowie „Streng Geheim“ zu verschlüsseln, und werden auch für diese Zwecke von US-Behörden verwendet. Direkte Angriffe auf den Algorithmus sind bislang noch nicht gelungen, im Jahr 2011 veröffentlichen Kryptologen den Biclique-Angriff, welcher erstmals theoretische Relevanz aufweisen konnte, jedoch die praktische Sicherheit von AES nicht minderte, aufgrund eines zu hohen rechnerischen Aufwands. Erfolgreiche Angriffe gegen Applikationen bei denen AES in Verwendung war, gelangen aufgrund von Lücken bei der kryptografischen Implementierung an einem Endgerät, nicht wegen Schwächen im Algorithmus.

SHA

Bei Versionen des Secure Hash Algorithmus sind die Versionen SHA-2, sowie SHA-3 relevant. Bis 2012 war der SHA-Standard SHA-2. Die dritte Version des Algorithmus wurde erstmals im Jahr 2012 veröffentlicht, als der Algorithmus den gleichnamigen Wettbewerb, veranstaltet vom National Institute of Standards and Technology, gewann.

Die Version SHA-2 schlägt den Nachfolger im Punkt Performanz. Im Gegensatz zu SHA-2 basiert SHA-3 nicht auf der Merkle-Damgard Struktur, wodurch einerseits ein gewisses Level an Sicherheit gewonnen wird, da ein eventuelles knacken des SHA-2 Algorithmus die Sicherheit des SHA-3 Algorithmus in

keinster Weise betreffen würde. Der anderen Architektur ist es jedoch auch geschuldet, dass der SHA-3 deutlich langsamer als sein Vorgänger ist.[16]

Twofish

Twofish ist ein weiterer Block-Cipher der die Daten in Blöcken zu je 128 Bit verschlüsselt. Gültige Schlüssellängen sind sämtliche Längen bis 256 Bit. Bei Twofish handelt es sich um einen Feistel-Cipher[17], was bedeutet das pro Iteration eine Hälfte des Datenblocks durch eine F Funktion geschickt wird, wonach beide Blöcke mit einem XOR vereint werden.[18]

Der Algorithmus gilt generell als performant und kann laut Entwickler flexibel eingesetzt werden.

Zu verwendende Open Source Algorithmen & Bibliotheken

Für die Implementierung von SIKOSI kommen mehrere Algorithmen und Programmbibliotheken in Frage. Einige davon decken die meisten sicherheitsrelevanten Anwendungsfälle ab, andere jedoch stellen nur Funktionalität für einen spezifischen Bereich zur Verfügung. Aus diesem Grund folgt eine kurze Beschreibung der in Frage kommenden Komponenten.

Kryptographie

Hierbei handelt es sich um Bibliotheken die für die Ver- und Ent-schlüsselung von Daten in Frage kommen.

Chaos.NaCl[19]

Diese Bibliothek wird von der Messaging App Threema[20] verwendet und stellt Methoden zur Ver-/Entschlüsselung sowie für den Schlüsselaustausch zur Verfügung.

Lizenz	Public Domain[21]
Hashing Algorithmus	SHA-512
Schlüssel-Algorithmen	Curve25519 (Montgomery), Ed25519
Verschlüsselung	XSalsa20Poly1305
Framework	>= .NET 2.0

NSec[22]

NSec ist eine moderne Kryptographie Bibliothek die auf .NET Core basiert. Die Bibliothek ist auf leichte Anwendbarkeit sowie auf Geschwindigkeit ausgelegt.

Lizenz	MIT
Hashing Algorithmen	BLAKE2b, SHA-256, SHA-512
Schlüssel-Algorithmen	X25519, Ed25519
Verschlüsselung	AES-256 GCM, ChaCha20-Poly1305
Framework	.NET Core 3.1, NET Core 2.1

[16] https://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction
[17] https://www.commonlounge.com/discussion/df78c412191849029996f37b1089f3a4
[18] https://www.schneier.com/academic/archives/1998/12/the_twofish_encrypti.html
[19] https://github.com/CodesInChaos/Chaos.NaCl
[20] https://threema.ch/press-files/cryptography_whitepaper.pdf
[21] https://github.com/CodesInChaos/Chaos.NaCl/blob/master/License.txt
[22] https://nsec.rocks/

BcryptNet[23]

Diese Bibliothek ist eine Portierung der Java Bibliothek jBCrypt und stellt damit eine Implementierung des Blowfish Passwort Hashing Algorithmus dar. Das bedeutet, dass die Bibliothek auf die Verschlüsselung von Passwörtern ausgelegt ist.

Lizenz	MIT
Hashing Algorithmen	SHA-256, SHA-384, SHA-512
Schlüssel-Algorithmen	
Verschlüsselung	Blowfish
Framework	.NET Core 3.1, >= .NET 2.0

Bouncy Castle C# API[24]

The Legion of the Bouncy Castle ist ein Zusammenschluss von Entwicklern die sich der Implementierung von Kryptographie Algorithmen verschrieben haben. Die von Ihnen zur Verfügung gestellte API kann als „All in One“ Lösung angesehen werden. Aus diesem Grund wird hier auch nicht auf den kompletten Funktionsumfang eingegangen, da dieser detailliert auf der Website des Projekts aufgeführt wird. Die Bouncy Castle Bibliothekssammlung steht unter Kritik, da tw. veraltete Implementierungen bzw. „nur“ Portierungen von Java verwendet werden.

Lizenz	MIT[25] Derivat
Framework	>= .NET 2.0, ASP.NET Core 1; Android: >= SDK V15;

SecurityDriven.NET/inferno[26]

Inferno kann ebenfalls als „All in One“ Lösung angesehen werden. Im Gegensatz zu Bouncy Castle wird jedoch auf moderne Technologien und auf professionelle Audits gesetzt. Auch bei Inferno wird an dieser Stelle nicht der volle Funktionsumfang aufgelistet, stattdessen wird auf die Projekt Website verwiesen.

Auffällig ist jedenfalls die sehr detaillierte Dokumentation, da diese nicht nur die Verwendung der Funktionalitäten abdeckt, sondern auch die Designentscheidungen der Entwickler umfasst.

Lizenz	MIT
Framework	.NET Core 2.1, .NET 4.6.2

Codesicherheit

Wie bereits im Kapitel Clientfunktionalität beschrieben, soll SIKOSI die Entwickler nicht nur im Bereich Datensicherheit, sondern auch im Bereich der Codesicherheit unterstützen.

[23] https://github.com/BcryptNet/bcrypt.net
[24] http://bouncycastle.org/csharp/index.html
[25] https://www.bouncycastle.org/licence.html
[26] https://securitydriven.net/inferno/

OWASP SafeNuGet[27]

Hierbei handelt es sich um ein Tool das NuGet Pakete und Bibliotheken die in ein Projekt eingebunden werden analysiert. Wenn bei der Analyse sicherheitsauffälliger Code gefunden wird, wird dies dem Entwickler mitgeteilt. Je nach Konfiguration kann sogar der Kompilierungsprozess gestoppt werden, d.h. ein Entwickler kann keine Anwendungen kompilieren solange die auffälligen Stellen nicht zumindest überprüft worden sind. Für die Analyse werden die Daten der OWASP TOP 10 2013 verwendet.

Ein Nachteil dieser Komponente kann das Alter sein, konkret wurde das letzte Mal im Jahr 2017 daran gearbeitet. Zusätzlich sind auch die Daten der OWASP Top 10 2013 mittlerweile veraltet.

Lizenz

Auch die Lizenzsituation ist noch unklar, diese muss erst mit den verantwortlichen Personen abgeklärt werden.

DevAudit[28]

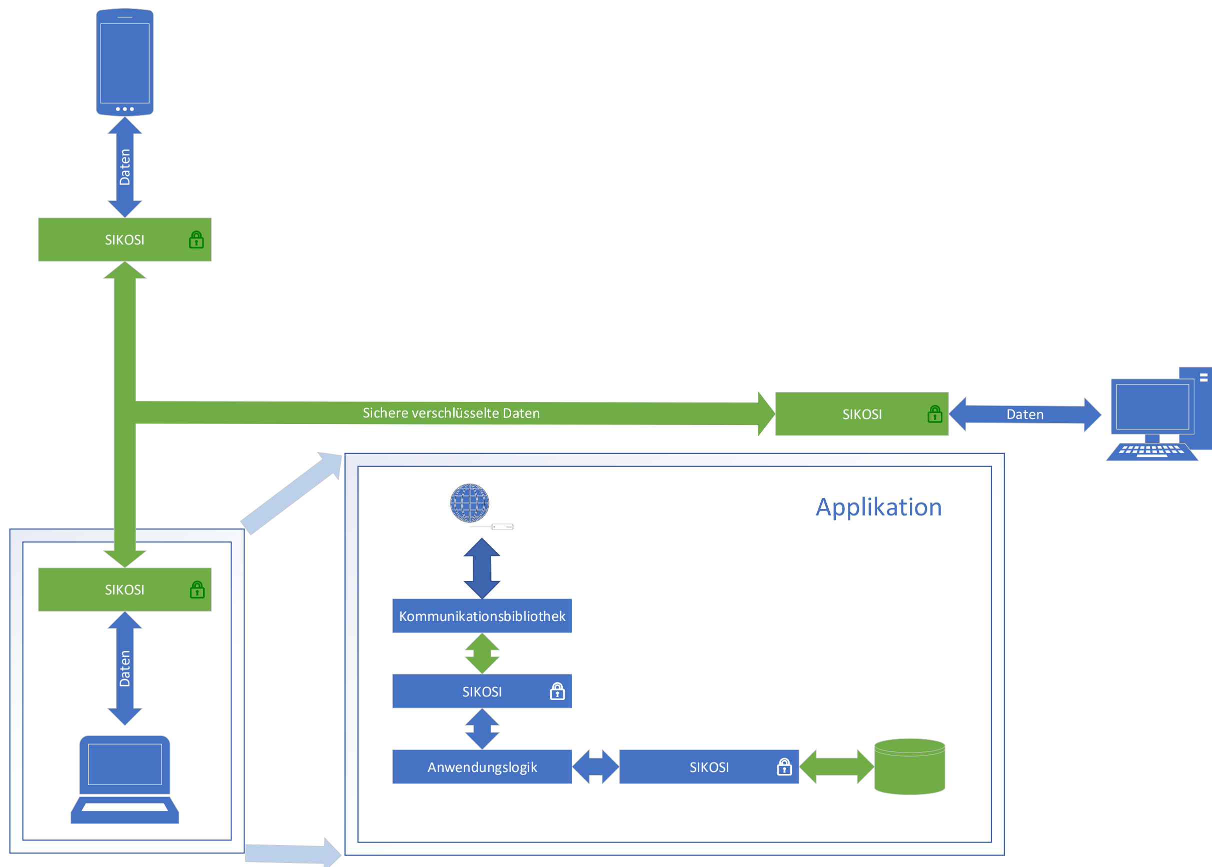
Das Ziel von DevAudit ist dasselbe wie das von OWASP SafeNuGet, d.h. einbundene Softwarekomponenten werden auf Sicherheitslücken überprüft. Im Gegensatz zu SafeNuGet verwendet es für die Analyse Daten von dem OSS Index[29], welcher laufend aktualisiert wird.

Ein weiterer Unterschied zu SafeNuGet ist auch, dass die Codebasis[30] von DevAudit aktiv weiterentwickelt wird, also auf einem aktuelleren Stand ist.

Lizenz: BSD-3

[27] https://docs.myget.org/docs/how-to/checking-nuget-package-vulnerabilities-with-owasp-safenuget
[28] https://github.com/OSSIndex/DevAudit/
[29] https://ossindex.sonatype.org/
[30] https://github.com/OSSIndex/DevAudit/

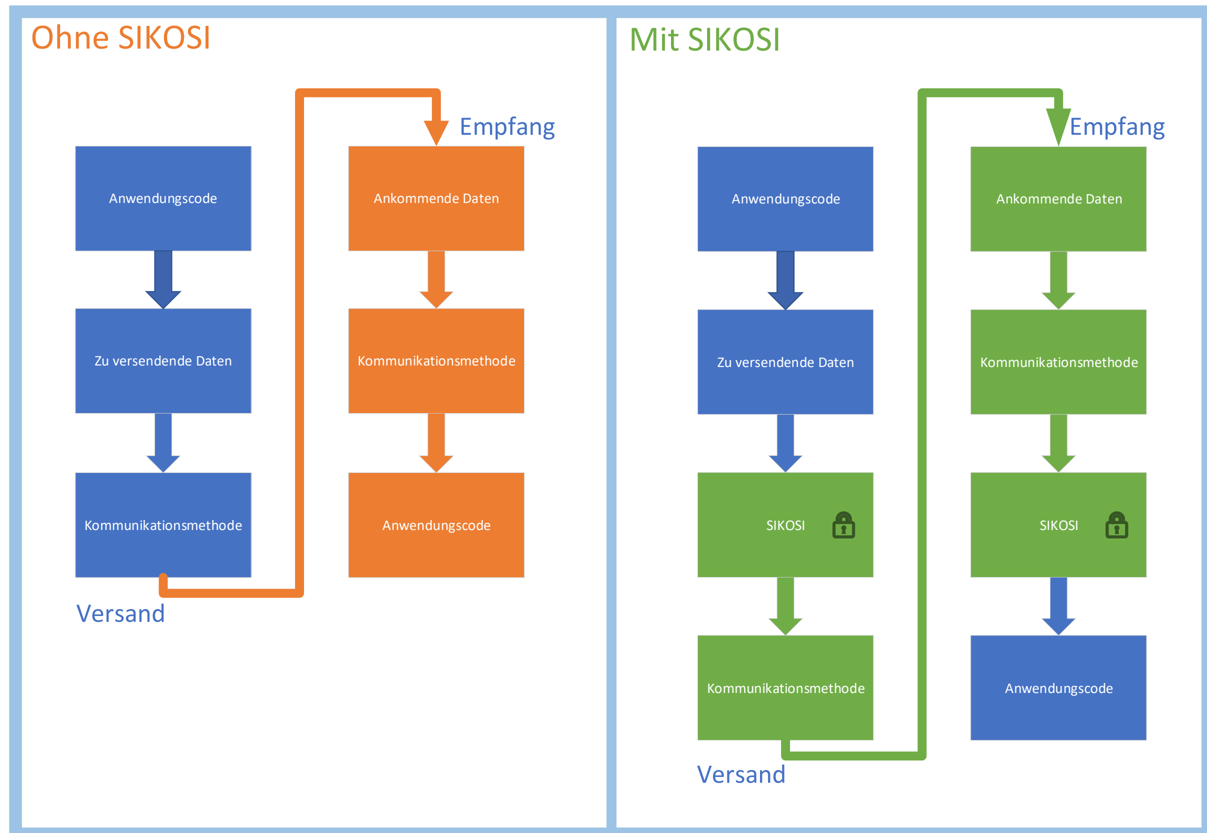
Systemarchitektur



Geplanter Architekturaufbau von SIKOSI

Wie bereits erwähnt ist das Ziel von SIKOSI die Schaffung von SDKs die von Softwareentwicklern leicht eingebunden werden können. Konkret heißt das, dass SIKOSI als NuGet oder als Bibliothek in ein bestehendes Projekt eingebunden werden kann. Sobald dies erfolgt ist, kann SIKOSI beim Datentransport und bei der Datensicherung „zwischengeschaltet“ werden. Angenommen, der Datenverkehr in einem Projekt läuft vor der Einbindung von SIKOSI so ab, dass Daten direkt und unverschlüsselt empfangen und versendet werden (siehe Abbildung „Vergleich des Ablaufs ohne und mit SIKOSI“).

Dann ist das Ziel, dass der Entwickler SIKOSI zu seinem Projekt hinzufügt und als zusätzliche Schicht zwischen Versand und Empfang einbindet. Dadurch kann er sicherstellen, dass die Daten sicher versendet und empfangen werden (Siehe Abbildung „Vergleich des Ablaufs ohne und mit SIKOSI“). Die gleiche Vorgehensweise kann für das Sichern und Laden von Daten direkt auf einem Gerät verwendet werden.



Vergleich des Ablaufs ohne und mit SIKOSI

Anhang

SIKOSI – Entwickler-Fragebogen

Herzlich Willkommen!

Im Rahmen des Projekts "SIKOSI - Sichere Kommunikation und Sicherung von Daten" führt die Firma FOTEC Forschungs- und Technologietransfer GmbH eine Umfrage unter Software-Entwicklern zum Thema IT-Security bzw. Verschlüsselung von Daten durch.

Das Ziel dieses Projekts ist die Implementierung von Security-Features für Software-Entwickler zu erleichtern. Dabei dient diese Umfrage zur Erhebung der Schwierigkeiten und Probleme die Software-Entwickler bei der sicheren Verarbeitung von Daten haben.

Eine genauere Beschreibung und Infos über den Fortschritt des Projekts SIKOSI finden Sie unter https://www.netidee.at/sikosi

Als Ergebnis des Projekts werden u.a. Open-Source Software Development Kits (SDK) entstehen, die allen Entwicklern auf Github zur Verfügung gestellt werden.

Ihre Antworten in dieser Umfrage werden selbstverständlich anonym behandelt.

Die Umfrage wird nur etwa 5 Minuten Ihrer Zeit beanspruchen.

Vielen Dank!

FOTEC
Forschungsunternehmen
-der FH Wiener Neustadt-

Q1 Wie würden Sie selbst Ihre Expertise in Bezug auf Sicherheit von Anwendungen im Allgemeinen einschätzen?

- ◯ Anfänger
- ◯ Fortgeschritten
- ◯ Experte

Q2 Wie würden Sie selbst Ihre Expertise in Bezug auf Verschlüsselungsalgorithmen einschätzen?

- ◯ Anfänger
- ◯ Fortgeschritten
- ◯ Experte

Q3 Bitte nennen Sie Ihre favorisierten Technologien:
(Mehrfachnennung möglich)

- ◯ Programmiersprache ____
- ◯ Entwicklungsumgebung ____
- ◯ Betriebssystem ____
- ◯ Verschlüsselungsbibliothek ____

Q4 An der Entwicklung welcher Art von Applikationen sind Sie vorrangig beteiligt?

- ☐ Webapplikation
- ☐ Desktopapplikation
- ☐ Mobile Applikation
- ☐ Clientapplikation
- ☐ Serverapplikation
- ☐ Andere: ____

Q5 Sind in diesen Applikationen die sichere Übertragung bzw. Speicherung von Daten ein entscheidender Faktor für den Erfolg der Applikation?

- ○ Ja
- ○ Nein

Q6 Sind Sie an der Implementierung der Sicherheits-Features involviert?

- ○ Ja
- ○ Nein

Display This Question:

If Q6 = Ja

Q7 Was sind für Sie die größten Schwierigkeiten bei der Implementierung dieser Technologien?

____

____

____

____

____

Q8 Welche Technologien haben Sie schon benutzt um eine sichere Übertragung und Speicherung von Daten zu gewährleisten?

____

____

____

____

____

Q9 Wenn Sie externe Bibliotheken verwenden, um Sicherheits-Features (wie z.B. Verschlüsselung) zu implementieren: Nach welchen Kriterien wählen Sie diese Bibliotheken aus?

____

____

____

____

____

Q10 Wie oft werden fehlerhafte Implementierungen von Sicherheits-Features mittels externer Bibliotheken bei Ihnen erkannt?

- ◯ Selten bis nie
- ◯ Schon immer wieder
- ◯ (Fast) immer
- ◯ Keine Angabe

Q11 Zu welchem Zeitpunkt machen Sie sich zum ersten Mal Gedanken über die sichere Übertragung bzw. Speicherung von Daten in einer Anwendung?

- ◯ In der Planungsphase
- ◯ Während der Implementierung der Anwendung
- ◯ Wenn Grundfunktionalität der Anwendung fertig
- ◯ Gar nicht

Display This Question:

If Q11 != In der Planungsphase

Q12 Warum wird die sichere Übertragung bzw. Speicherung von Daten nicht schon in der Planungsphase der Anwendung genau durchdacht und die zu verwendenden Technologien festgelegt?

- ☐ Zeitmangel
- ☐ Fehlendes Wissen / Unsicherheit
- ☐ Nicht so wichtig
- ☐ Andere: ____
- ☐ Keine Angabe

Display This Question:

If Q11 = In der Planungsphase

Q13 Warum wird Ihrer Meinung nach bei anderen Projekten/Entwicklern die sichere Übertragung bzw. Speicherung von Daten nicht schon in der Planungsphase der Anwendung genau durchdacht und die zu verwendenden Technologien festgelegt?

- ☐ Zeitmangel
- ☐ Fehlendes Wissen / Unsicherheit
- ☐ Nicht so wichtig
- ☐ Andere: ____
- ☐ Weiß nicht

Q14 Wenn Sie bestimmen könnten welche Funktionen eine neue Security-Bibliothek beinhalten muss, welche Funktionen wären dann für Sie die wichtigsten?
z.B. Hashfunktion SHA256, Diffie-Hellman-Schlüsselaustausch, ...
(Mehrfachnennungen erwünscht)

____

____

____

____

____

Q15 Würden Sie eine Bibliothek für sichere Übertragung und Speicherung von Daten mit sehr einfach gehaltenen Funktionsaufrufen bevorzugen, wenn Sie dafür Einschränkungen bei der Auswahl der verwendeten Verschlüsselungsalgorithmen in Kauf nehmen müssten?

◯ Ja

◯ Nein

◯ Weiß nicht

Q17 Würden Sie sich Leitfäden bzw. Checklisten für die Implementierung von Sicherheitsfunktionalitäten (wie z.B. Verschlüsselung) wünschen und diese in Ihrer täglichen Arbeit verwenden?

◯ Ja

◯ Nein

◯ Weiß nicht

Q18 Kreuzen Sie bitte die Begriffe/Fragestellungen an, die Ihnen NICHT ganz klar sind und für die Sie selbst eine Erklärung bräuchten, wenn Sie danach gefragt werden:

- ☐ Ende-zu-Ende-Verschlüsselung
- ☐ Unterschied zwischen symmetrischer und asymmetrischer Verschlüsselung
- ☐ Unterschied zwischen Blockverschlüsselung und Stromverschlüsselung
- ☐ Data Encryption Standard (DES)
- ☐ Ist DES heute noch sicher?
- ☐ Triple Data Encryption Standard (3DES)
- ☐ Ist 3DES heute noch sicher?
- ☐ Advanced Encryption Standard (AES)
- ☐ Ist AES heute noch sicher?
- ☐ Secure Sockets Layer/Transport Layer Security (SSL/TLS)
- ☐ Klassischer Diffie-Hellman-Schlüsselaustausch (DH)
- ☐ Elliptic Curve Diffie-Hellman-Schlüsselaustausch (ECDH)
- ☐ Wichtigster Vorteil des ECDH gegenüber DH
- ☐ Extended Triple Diffie Hellman Key Agreement Protocol (X3DH)
- ☐ Double Ratchet Algorithmus
- ☐ RSA-Verschlüsselung
- ☐ RSA für Schlüsselaustausch oder als Digitale Signatur?

- [ ] Wozu dient digitale Signatur?
- [ ] Digital Signature Algorithm (DSA)
- [ ] Elliptic Curve Digital Signature Algorithm (ECDSA)
- [ ] Was ist eine Hashfunktion?
- [ ] Zero-Knowledge-Prinzip
- [ ] Password Authenticated Key Exchange (PAKE)
- [ ] Message Authentication Code (MAC)
- [ ] Hash-Based Authentication Code (HMAC)
- [ ] Blockchain