Universität Hamburg
Fachbereich Informatik

Hinweise für das Erscheinungsbild von Seminar-, Studien- und Bachelor-, Master- und Diplomarbeiten

am Arbeitsbereich Sicherheit in Verteilten Systemen (SVS)

Prof. Dr. Hannes Federrath

28. November 2014

(Muster für das Deckblatt: siehe letzte Seite dieser Hinweise)

Zusammenfassung

Für den eiligen Leser sollen auf etwa einer halben, maximal einer Seite die wichtigsten Inhalte, Erkenntnisse, Neuerungen bzw. Ergebnisse der Arbeit beschrieben werden.

Durch eine solche Zusammenfassung (im engl. auch Abstract genannt) am Anfang der Arbeit wird die Arbeit deutlich aufgewertet. Hier sollte vermittelt werden, warum der Leser die Arbeit lesen sollte.

Inhaltsverzeichnis

1 Einleitung

2 Funktionsweise

3 DTE

Die DTE, Abkürzung für *distribution-transforming encoder*, dient zum Abbilden einer Nachricht M aus dem Message Space $\mathcal{M}$ auf einen Seed S aus dem Seed Space $\mathcal{S}$. Gleichermaßen soll sie die Möglichkeit bieten, von einem Seed auf die ursprüngliche Nachricht abzubilden. Eine DTE ist also ein Tupel von Algorithmen

$$DTE = (encode, decode)$$

wobei $encode$ einen meist randomisierten Algorithmus der Form $\mathcal{M} \rightarrow \mathcal{S}$ und $decode$ einen deterministischen Algorithmus der Form $\mathcal{S} \rightarrow \mathcal{M}$ beschreibt.

Ein DTE-Schema $(encode, decode)$ wird als *korrekt* bezeichnet, wenn für jede Nachricht $M \in \mathcal{M}$, die mit $encode$ in den Seed Space $\mathcal{S}$ und mit $decode$ anschließend wieder in den Message Space $\mathcal{M}$ abgebildet wird, das Resultat wieder die ursprüngliche Nachricht M ist. Formal kann dies geschrieben werden als

$$P(decode(encode(M)) = M) = 1 \qquad f.a. M \in \mathcal{M}$$

wobei P ein Maß für die Wahrscheinlichkeit für das in den Klammern stehende Ereignis ist.

Bei der Konstruktion einer DTE ist nicht nur dies zu beachten. Wichtig ist ebenfalls, die Verteilung der Wahrscheinlichkeiten der Nachrichten im Message Space zu kennen. Entsprechend dieser Wahrscheinlichkeiten wird einer Nachricht eine Anzahl von Seeds zur Kodierung zugewiesen.

Bei der Verschlüsselung einer Nachricht $M \in \mathcal{M}$ wird der Algorithmus $encode$ verwendet. Dabei wird eine Nachricht aus dem Message Space durch einen Seed aus dem Seed Space kodiert. Wie eingangs bereits erwähnt handelt es sich bei $encode$ um einen randomisierten Algorithmus. Das liegt an der Tatsache, dass

- $\mathcal{S}, \mathcal{M}$
- (DTE = (encode, decode))
- Zufall bei der encode Funktion -> Encode keine Funktion im mathematischen Sinne
- Eindeutigkeit bei der decode Funktion

4 Verschlüsselungsschema

Nachdem eine Nachricht aus dem Message Space - wie im vorherigen Kapitel beschrieben - durch eine DTE auf den Seed Space abgebildet wurde, folgt die Verschlüsselung des Ergebnisses. Hierfür schlägt [JR14b] zwei verschiedene Vorgehensweisen vor. Die unter Verwendung dieser Vorgehensweisen entstehenden Honey Encryption-Schemata werden im Folgenden dargestellt.

4.1 Hashbasierte Verschlüsselung

Im ersten Schritt ...

$\text{HEnc}_{\text{Hash}}(M, K)$
$S \xleftarrow{<r>} \text{DTE}(M)$
$K_D = \text{PBKDF}(K)$
$R \xleftarrow{<r>} 0,1^n$
$H = \text{HF}(K_D, R)$
$C = H \oplus S$
Return (C, R)

Abbildung 1: Hashbasierte Verschlüsselung

$\text{HDec}_{\text{Hash}}((C, R), K)$
$K_D = \text{PBKDF}(K)$
$H = \text{HF}(K_D, R)$
$S = H \oplus C$
$M = \text{DTE}^{-1}(S)$
Return M

Abbildung 2: Hashbasierte Entschlüsselung

4.2 Auf Blockchiffren basierte Verschlüsselung

$$
\begin{aligned}
&\text{HEnc}_{\text{Block}}(M, K)\\
&\quad S \overset{<r>}{=} \text{DTE}(M)\\
&\quad R \overset{<r>}{=} 0,1^k\\
&\quad K' = \text{HF}(K, R)\\
&\\
&\quad P = \epsilon\\
&\quad \text{For } i = 1 \text{ to } \left\lceil \frac{|S|}{n} \right\rceil\\
&\qquad P = P \parallel \text{Enc}(K', i)\\
&\\
&\quad C = P[1..|S|] \oplus S\\
&\quad \text{Return } (C, R)
\end{aligned}
$$

Abbildung 3: Verschlüsselung mit Blockchiffre (CTR-Modus)

$$
\begin{aligned}
&\text{HDec}_{\text{Block}}((C, R), K)\\
&\quad K' = \text{HF}(K, R)\\
&\\
&\quad P = \epsilon\\
&\quad \text{For } i = 1 \text{ to } \left\lceil \frac{|S|}{n} \right\rceil\\
&\qquad P = P \parallel \text{Enc}(K', i)\\
&\\
&\quad S = P[1..|S|] \oplus M\\
&\quad M = \text{DTE}^{-1}(S)\\
&\quad \text{Return } M
\end{aligned}
$$

Abbildung 4: Entschlüsselung mit Blockchiffre (CTR-Modus)

5 Möglichkeiten und Probleme

Durch die enorm hohe Sicherheit der Verschlüsselung ergeben sich vor allem Anwendungsfälle, bei denen hochsensible Daten verschlüsselt werden sollen. Juels und Ristenpart ([JR14b, JR14a]) geben dafür einige Beispiele, wie das Verschlüsseln von RSA-Schlüsseln oder Kreditkartennum-

mern. Ebenfalls könnten Passwort-Safes/-Manager mit Honey Encryption vor Zugriffen von außen geschützt werden. Generell ist diese Verschlüsselungsmethode auf strukturierte Daten anwendbar, von denen die Generierung und der Aufbau bekannt sind. Dies ist schließlich, wie in Abschnitt 3 beschrieben, notwendig zur Konstruktion einer sicheren und invertierbaren DTE. Dementsprechend ist Honey Encryption nicht oder nur eingeschränkt für Freitext, wie Notizen oder E-Mails, geeignet. Ein Grund dafür ist die Tatsache, dass die Menge aller Nachrichten bekannt sein muss. Diese ist bei Freitext quasi unbegrenzt.

Ebenfalls sollte beachtet werden, dass für jeden Anwendungsbereich, in dem Honey Encrytion genutzt werden soll, eine eigene DTE konstruiert werden muss. Ein universaler Ansatz existiert dazu nicht. Der Nutzen der Verschlüsselung mit Honey Encryption sollte also verhältnismäßig groß im Vergleich zum Aufwand der Erstellung stehen. Dieser Aufwand ist in jedem Fall enorm. Es muss nicht nur die Menge aller Nachrichten bekannt sein, sondern auch die Verteilung der Wahrscheinlichkeiten dieser. Bei Passwort-Managern beispielsweise ist die Menge der vom Nutzer verschlüsselten Passwörter meist abhängig vom Nutzer selbst. Falls der Nutzer nämlich keine zufällig generierten Passwörter verwendet, sind Passwörter, die Teile des Namens, des Geburtsdatums, des Namens der Lieblingsband, des Haustieres oder anderer persönlicher Daten beinhalten, sehr wahrscheinlich. Die Verteilung der Wahrscheinlichkeiten der Nachrichten, in diesem Fall der Passwörter des Nutzers, sind also von Nutzer zu Nutzer unterschiedlich. Diese muss aber zur Konstruktion einer guten DTE bekannt sein. Was passiert aber, wenn die DTE nicht optimal gewählt ist und es für einen Angreifer erkennbar ist, dass er die richtige Lösung, zum Beispiel die Passwörter des Nutzers im Safe, gefunden hat? Dann fällt die Methode auf eine Brute-Force-Attacke zurück. Der Angreifer muss also jede Passwort-Kombination für den Safe ausprobieren, bis er das richtige Passwort gefunden hat. Die Sicherheit von Honey Encryption ist dann ähnlich der Sicherheit von herkömmlichen Verschlüsselungsmethoden, wie zum Beispiel AES. Zwar gibt es bei Honey Encryption mehrere Passwörter, die aufgrund der Hash-Funktion von Passwort auf Seed-Space die gleiche Nachricht erzeugen, was die Wahrscheinlichkeit erhöht, dass ein solches Passwort bei einem Brute-Force-Angriff gefunden wird. Allerdings wird im Gegensatz zu beispielsweise AES nie direkt angegeben, ob das eingegebene Passwort korrekt ist. Der Angreifer hat also einen enormen Mehraufwand, wenn er überprüfen will, ob er ein korrektes Passwort gefunden hat. Schließlich muss er bei jedem Versuch erneut überprüfen, ob die resultierende Nachricht sinnvoll ist oder nicht. Je schlechter die DTE gewählt ist, desto einfacher fällt es dem Angreifer. Dennoch ist es schwer, die Nachrichten auf Plausibilität zu prüfen.

Neben der Erstellung der DTE ist auch die Speicherung der DTE ein Problem. Da sie eine Funktion ist, die für jeden Anwendungsfall neu erstellt werden muss, muss sie auch für jeden Anwendungsfall gespeichert werden. Kann eine Nachricht nicht so einfach aus einem Seed berechnet werden, so, muss eine Datenstruktur gespeichert werden, die von einer Nachricht auf einen Seed und umgekehrt abbilden kann. Diese tabellen-ähnliche Struktur ist für einen ausreichend großen Message Space sehr umfangreich. Ist die Anzahl der Nachrichten im Message Space gleich n und die Speicherung einer Nachricht m Bits teuer, dann brauchen wir *mindestens*

$$(\lceil \log_2(n) \rceil + m) \cdot n$$

Bits zur Speicherung der Datenstruktur.

Beispiel: Es existieren $n = 2^{16} = 65536$ mögliche Nachrichten, die gleichverteilt auf den Seed Space abgebildet werden sollen. Dabei soll jeder Nachricht nur ein Seed zugewiesen werden. Für jede Nachricht ist zudem ein String mit maximal 10 ASCII-Zeichen, also ein Speicherbedarf von $m = 10$ Bytes, also 80 Bits. Die Datenstruktur zum Speichern aller Nachrichten — mit ihrem Index als Seed — wird dann mindestens

$$\begin{aligned}(\lceil \log_2(65536) \rceil + 80) \cdot 65536 &= (16 + 80) \cdot 65536 \\ &= 96 \cdot 65536 \\ &= 6291456\end{aligned}$$

Bits benötigen. Das sind 786432 Bytes, also ungefähr 800 Kilobytes. Um eine Nachricht aus dem Message Space zu verschlüsseln, wird also Speicherplatz für die DTE (inklusive ihrer Datenstruktur), den resultierenden Ciphertext, sowie die Hashfunktion und eine eventuell vorhandene Verschlüsselungsfunktion (nach Abschnitt 4) benötigt. Ein String von maximal 10 ASCII-Zeichen aus der Menge der Nachrichten zu verschlüsseln, wird zu einem fast ein Megabyte großen Paket. Gerade das Verschicken von geheimen Nachrichten wird dadurch extrem erschwert, da dieses Paket bei neuen Anwendungsfällen erneut erstellt und übermittelt werden muss. Dieses vergleichsweise große Datenverhältnis zwischen *Verfahren zum Ver- und Entschlüsseln* und *Ciphertext* relativiert sich aber bei steigender Anzahl von übermittelten Ciphertexten. Ist die Länge der Nachrichten relativ lang, werden aber auf kurze Seeds abgebildet, sind die Ciphertexte kürzer als bei anderen Verschlüsselungsverfahren. Mit steigender Anzahl an zu speichernden oder zu übertragenden Ciphertexten teilt sich die Größe der Ver- und Entschlüsselungsmethoden auf die einzelnen Cyphertexte auf und wir erhalten möglicherweise zum Schluss immer noch kleinere zu speichernde Datenmengen als bei anderen Verschlüsselungsverfahren mit gleicher Ciphertextanzahl.

Ein anderer Bereich für Probleme von Honey Encryption ist die Erstellung der Hashfunktion, die vom Key Space auf den Seed Space abbildet. Da die Größe des Seed Spaces für jede Menge von Nachrichten unterschiedlich ist, muss auch die Hashfunktion an die Größe des Seed Spaces angepasst werden. Wie bei jeder Hashfunktion muss ebenfalls sicher gestellt werden, dass Werte aus dem Definitionsbereich durch die Funktion nur in den erlaubten Wertebereich abgebildet werden. Außerhalb des Definitionsbereichs ist das Resultat der Hashfunktion egal. Wird zum Beispiel ein vierstelliger Pin zum Ver- und Entschlüsseln von einer Nachricht verlangt, so muss unsere Hashfunktion lediglich die gültige Menge aller Zeichenkette — “0000” bis “9999” — berücksichtigen. Alle Werte außerhalb dieses Bereichs, beispielsweise Zeichenketten mit mehr oder weniger Zeichen, müssen nicht beachtet werden. Dies stellt zwar eine kleine Erleichterung für den Ersteller der Hashfunktion dar, allerdings ist die Generierung einer guten Hashfunktion, die die Anforderungen aus Abschnitt 4 erfüllen, wie schon das Erstellen einer guten DTE, extrem aufwändig.

Einer der größten Vorteile von Honey Encryption ist gleichzeitig auch einer ihrer größten Nachteile. Die Tatsache, dass unter Eingabe jedes möglichen Schlüssels ein plausibler Klartext angezeigt wird, könnte dem Nutzererlebnis schaden. Gibt nämlich ein Nutzer das Passwort falsch ein, bekommt er normalerweise den Hinweis, dass die Eingabe nicht korrekt ist. Bei Honey Encryption wird dem Nutzer diese Hilfestellung nicht gegeben. Der Nutzer weiß gar nicht, ob er

das Passwort richtig eingegeben hat, bzw. ob er überhaupt im Besitz des richtigen Passwortes ist (falls ihm das oben erwähnte Paket von einer anderen Person geschickt wurde — der Schlüssel ist dabei beispielsweise mündlich übertragen worden). Diese Problematik lässt sich nicht einfach lösen. Juels und Ristenpart stellen insgesamt drei Ansätze vor ([Jue14, JR14b]), die sogenannte *Typo-Safety* zu gewährleisten. Einerseits könnte zum Ciphertext eine Prüfsumme des Passwortes gespeichert werden. So würden nach einer Fehleingabe des Passwortes die Prüfsummen nicht mehr übereinstimmen und der Nutzer könnte entsprechend gewarnt werden. Logischerweise schränkt dies jedoch den Key Space ein, da die Anzahl der möglichen Schlüssel dezimiert wird. Diese Möglichkeit bietet also weniger Schutz für eine bessere Benutzbarkeit. Ein weiterer Ansatz ist die Überprüfung der entschlüsselten Daten. Dies klappt aber nur bei Daten wie Kreditkarten, bei denen der Anbieter überprüfen kann, ob die Nummer eine gültige und zum Kunden gehörende ist. Damit der Anbieter sicherstellen kann, dass ein Kunde und nicht ein Angreifer auf den Dienst zugreifen, wäre es nach Juels und Ristenpart möglich, beispielsweise die ersten beiden Ziffern der Kreditkartennummer als Klartext im Honey Encryption Verfahren zu speichern. Nutzt ein Angreifer dann irgendeine Kreditkartennummer, weiß der Anbieter, dass es sich hierbei um keinen Tippfehler des Passwortes, sondern um einen Angriff von einem unberechtigten Dritten handelt. Läge ein Schreibfehler vor, stimmten wenigstens die ersten beiden Ziffern überein und der Anbieter kann den Kunden nach einer erneuten Eingabe des Passwortes fragen. Hierbei wird der Message Space verkleinert, was auch zu weniger Sicherheit führt. Wie auch schon der Ansatz davor sollte diese Methode dementsprechend mit Vorsicht angewendet werden. Der dritte und letzte Vorschlag seitens der beiden Autoren ist es, eine Farbe mit den anderen Informationen zusammen zu verschlüsseln. Diese Farbe wird ebenfalls wieder hergestellt, wenn das richtige Passwort eingegeben wurde, und der Ciphertext wird zu einer anderen Farbe entschlüsselt, wenn es eine Fehleingabe gab. Es wird die Stärke von Honey Encryption angewendet, um dem Nutzer einen Hinweis darauf zu geben, dass er den richtigen Schlüssel eingegeben hat. Der Nutzer muss sich allerdings die ursprüngliche Farbe merken, bzw. sie wieder erkennen. Dies ist jedoch leichter, als sich den viel komplizierteren Klartext der Nachricht zu merken, da hierbei das visuelle Gedächtnis des Nutzers angesprochen wird. Diese Methode bringt allerdings nicht nur Vorteile. So wird dem Nutzer bei jeder Verschlüsselung, die er vornimmt, eine neue Farbe präsentiert. Bei der Unmenge an Logins, die wir heutzutage tätigen, könnte das für Verwirrung sorgen.

6 Fazit

Zusammenfassend lässt sich sagen, dass Honey Encryption mit ihrem neuen Ansatz, Nachrichten zu verschlüsseln, ein interessantes neues Forschungsfeld eröffnet. Sie ermöglicht es, auch kurze Schlüssel wie nutzergenerierte Passwörter zu verwenden und stellt mit ihrer Sicherheit eine enorm starke Verschlüsselung dar.

Allerdings überwiegen im jetzigen Forschungszustand die Nachteile und Probleme (aufgezeigt in Abschnitt 5). Die Generierung der einzelnen Teile des Honey Encryption Schemas ist noch zu aufwändig, um das Verfahren großflächig anzuwenden. Die Benutzung muss durch Gewährleistung der *Typo-Safety* gegeben sein und die Anwendungsgebiete der Verschlüsselung sollten weiter erforscht werden.

Die zukünftigen Forschungsbestreben sollten darauf abzielen, das Erstellen einer sicheren DTE zu vereinfachen. Ein Anfang wäre es, eine öffentlich zugängliche Sammlung von Honey Encryption Schemata für bestimmte Anwendungsfälle zur Verfügung zu stellen. Damit könnten Entwickler, die strukturierte Daten verschlüsseln wollen, auf diese zurückgreifen und müssten lediglich die Einbindung in ihr bestehendes System berücksichtigen. Eine immer größer werdende Sammlung führt somit zu weniger Aufwand für Entwickler und damit zu einer größeren und vor allem schnelleren Verbreitung des Verfahrens. Mit Hilfe modernerer Technik, statistischen Verfahren und weiteren Hilfsmitteln könnte es vielleicht sogar möglich sein, DTEs und Hashfunktionen automatisiert generieren zu lassen. Die Probleme, die dabei auftreten und die Bedingungen, die an eine gute DTE gestellt werden, stehen der Entwicklung momentan noch im Weg. Allerdings gibt es Fortschritte in der Analyse und Generierung von natürlicher Sprache, die bei der Entwicklung der genannten Systeme behilflich sein können ([Jue14]).

Ein weiteres Forschungsziel sollte es sein, die Möglichkeiten und Anwendungsbereiche von Honey Encryption zu erweitern. So könnte eine spannende Forschungsfrage sein, in wie weit es nicht doch möglich wäre, Klartexte zu verschlüsseln. Eine Idee könnte sein, die korrekte Nachricht als Grundlage für einen sehr viel größeren Message Space zu nutzen. Dabei würde eine Veränderung von Wörtern ohne Beeinträchtigung des Sinnzusammenhanges wichtig sein. Ein Beispiel wäre die Abwandlung des Satzes “Der geheime Treffpunkt ist Hamburg” in Nachrichten wie “Der geheime Treffpunkt ist Berlin”. Der Sinngehalt bliebe der gleiche, ein potentieller Angreifer könnte dann nicht entscheiden, in welcher Stadt nun der *geheime Treffpunkt* liegt. Die automatische Generierung solcher sinnverwandten Sätze wäre hier allerdings die erste Anlaufstelle, da der Message Space entsprechend groß gewählt werden muss. Dies ist so bei heutigem Kenntnisstand noch nicht möglich, allerdings könnte es in Zukunft solch ein Verfahren geben. Die Analyse der Sicherheit eines solchen Ansatzes wäre dann in einer weiteren wissenschaftlichen Arbeit zu klären.

Honey Encryption ist ein interessanter Ansatz, der noch viel Platz für Fortschritt und Verbesserung bietet. Die gebotene Sicherheit, die damit theoretisch möglich ist, sollte in der heutigen Zeit, mit immer wieder vorkommenden Passwort-Leaks und stärkeren Rechnern, Anreiz sein, weitere Forschung in dieser Richtung zu betreiben.

Literatur

[JR14a] Ari Juels and Thomas Ristenpart. Honey encryption - encryption beyond the brute-force barrier. *IEEE Security & Privacy*, 12/4:59–62, April 2014.

[JR14b] Ari Juels and Thomas Ristenpart. Honey encryption - security beyond the brute-force bound. In *EUROCRYPT 2014*, Kopenhagen, Dänemark, Mai 2014.

[Jue14] Ari Juels. The password that never was. http://crcs.seas.harvard.edu/event/ari-juels-the-password-that-never-was, März 2014.