Relazione progetto Cybersecurity - Password Manager

Armando La Placa - 0687185

Prof. Gallo

1 Introduzione

Un password manager è uno strumento utilizzato per la memorizzazione sicura di password, affiancato dall'utilizzo di tecniche crittografiche per la persistenza sicura in locale o remoto. Come progetto di esame, ho sviluppato un' applicazione Android per la memorizzazione di password, usando *Flutter* come framework di sviluppo, *AES-CBC* come algoritmo di cifratura e *PDKF2* per l'hashing della chiave di cifratura.

2 Cenni di crittografia

Quando si parla di *sistema crittografico* o *schema di cifratura*, intendiamo una coppia (E, D) in cui:

- La funzione di *cifratura* $E : \mathcal{K} \times \mathcal{X} \to \mathcal{C}$ prende in input una chiave $k \in \mathcal{K}$ e un messaggio $m \in \mathcal{X}$, quindi restituisce in output un testo cifrato $c \in \mathcal{C}$.
- La funzione di *decifratura* $D : \mathcal{K} \times \mathcal{C} \to \mathcal{X}$ prende in input una chiave $k \in \mathcal{K}$ e un messaggio cifrato $c \in \mathcal{C}$, quindi restituisce in output il messaggio decifrato $x \in \mathcal{X}$.

Un *block cipher* o *cifrario a blocchi* è un cifratore deterministico definito su uno spazio finito $\mathcal{X}$ di messaggi in chiaro, e su di uno spazio $\mathcal{K}$ di chiavi. Per poter essere cifrato, il messaggio viene suddiviso in blocchi e ognuno di questi viene passato come input allo schema di cifratura.

Una *pseudo-random permutation PRP* è una funzione invertibile $E : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ che mappa ogni elemento $x \in \mathcal{X}$ in un elemento di $\mathcal{X}$. Un *Cipher Block Chaining CBC* è un PRP in cui la funzione di cifratura è definita come

$$E_{CBC}(k, m) : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$$

.

Advanced Encryption Standard AES è uno standard per la cifratura a blocchi. Esso si basa sul principio di *confusione* e *diffusione*, secondo i quali è necessario eliminare le correlazioni tra i bit del messaggio in chiaro e quelli del messaggio cifrato allo scopo di non far trasparire informazioni. AES effettua 10[1] cicli di cifratura sui blocchi in input, ognuno dei quali è diviso in 3 step.

Una *Key Derivation Function KDF* è una funzione di derivazioni chiavi (o "*salatura*") a partire da una chiave sorgente. In particolare, *Password-Based KDF PBKDF* è una funzione di salatura che prende in input un "*sale*", generato casualmente, ed effettua l'hash $H^{(c)}(key||salt)$ c volte.

Un ultimo aspetto importante della crittografia riguarda l'*integrità dei messaggi*, ovvero l'abilità di individuare messaggi modificati da utenti malevoli. Il *Message Authentication Code MAC* è una coppia di algoritmi (S, V) utilizzati per la *firma* del messaggio e la *verifica* della sua integrità.

2.1 Sicurezza

Dalla definizione di sicurezza di Shannon, l'attaccante non deve essere in grado di ottenere nessuna informazione riguardo al testo in chiaro, partendo dal solo testo cifrato.

Definiamo un *Chosen Plaintext Attack CPA* come un attacco in cui si cerca di estrapolare informazioni a partire da messaggi in chiaro. In particolare, l'attaccante manda due plaintext allo sfidante, il quale li cifrerà per poi mandarli indietro come risposta. Da questi ciphertext, l'attaccante sarà in grado di sapere se due file cifrati sono uguali o meno.

Un attacco *Chosen Ciphertext Attack CCA* permette all'attaccante di ingannare lo sfidante facendogli

[1] Nel caso in cui si utilizzi AES-128 con chiave a 128 bit.

decifrare un ciphertext non valido. Tutto ciò può essere un vantaggio per l'attaccante nel momento in cui lo sfidante deve verificare l'integrità dei messaggi. Se il messaggio non viene accettato, lo sfidante manda un carattere *bottom* $\perp$ come risposta. Se l'attaccante riesce ad ingannare lo sfidante, questo carattere non verrà inviato.

Nella derivazione di una chiave di cifratura da una chiave sorgente, ci si può affidare ad una *Key Derivation Function KDF*. Nonostante ciò, se la chiave sorgente non viene scelta in modo uniforme dallo spazio delle chiavi $\mathcal{K}$, allora la KDF non risulterà essere casuale. La procedura di *salatura* permette di uniformare la distribuzione di probabilità tra le chiavi dello spazio $\mathcal{K}$, utilizzando un *sale* non segreto.

Dato un insieme di coppie $\{(m_1, t_1), ..., (m_q, t_q)\}$ riconosciuti dal MAC, la tecnica *Chosen Message Attack CMA* si basa sull'*existential forgery*, ovvero la creazione di una nuova coppia $(m, t) \neq \{(m_1, t_1), ..., (m_q, t_q)\}$ riconosciuto dal sistema MAC.

3 Soluzione proposta

Il software proposto include l'uso delle tecniche descritte in precedenza, quali AES-CBC e il Key Derivation Function. In figura 1 è possibile visualizzare la schermata di login (sinistra) e la homepage contenente le password (destra). In aggiunta alle funzionalità di base per la memorizzazione cifrata dei dati, l'app prevede l'esportazione e l'importazione delle password memorizzate in locale, con incluso il sale utilizzato per il KDF della chiave di cifratura.

3.1 Pagine

Login Page La pagina di Login consiste di una *textbox* per l'immissione della chiave di cifratura, un'*icona informativa* contenente le istruzioni basilari di utilizzo, il sale generato (o caricato da file) per la KDF della chiave di cifratura e un bottone di accesso.

Home Page La homepage è minimale e prevede una lista di Card definite come: *icona*, *genere*, *valore-password* e *bottone di memorizzazione "Salva"*. In aggiunta alle card delle password, vi è un'icona azzurra per l'aggiunta di nuove card alla lista. Queste, se non salvate appositamente tramite il bottone "Salva", verranno eliminate al riavvio o alla chiusura dell'applicazione.

3.2 Classi di supporto

Storage La classe Storage gestisce l'accesso ai file locali e alla lettura/memorizzazione di nuovi dati. Per poter essere utilizzato, è necessario instanziare un suo oggetto passando a parametro il nome del file che si vuole creare o utilizzare. Tutte le operazioni su file vengono gestite tramite questa classe.

Picker Il Picker permette di interfacciare l'utente con l'esportazione e importazione dei file.

PasswordJSON La classe PasswordJSON definisce il modello con cui vengono memorizzate le password in locale. I suoi attributi sono: *password*, *tipo* e *id*. La password rappresenta il valore che si vuole cifrare e memorizzare; il tipo indica la tipologia di password che si sta memorizzando, ovvero:

- Password Generica;
- Account Microsoft;
- Facebook;
- Amazon;
- E-Mail generica;
- GMail;

l'id permette di gestire la persistenza in locale delle password, ovvero la loro rimozione o modifica. Questa classe viene utilizzata per la conversione in/da JSON e successiva memorizzazione su file.

CardItem Il widget CardItem viene utilizzato per la visualizzazione delle password cifrate. Esso contiene un'icona rappresentativa, un titolo rappresentativo, la password da visualizzare (o ""**********" se la chiave di cifratura è errata) e un bottone "Salva" per la persistenza in locale.

3.3 Funzionamento

Avvio

1. Caricamento della pagina *LoginPage.dart*.

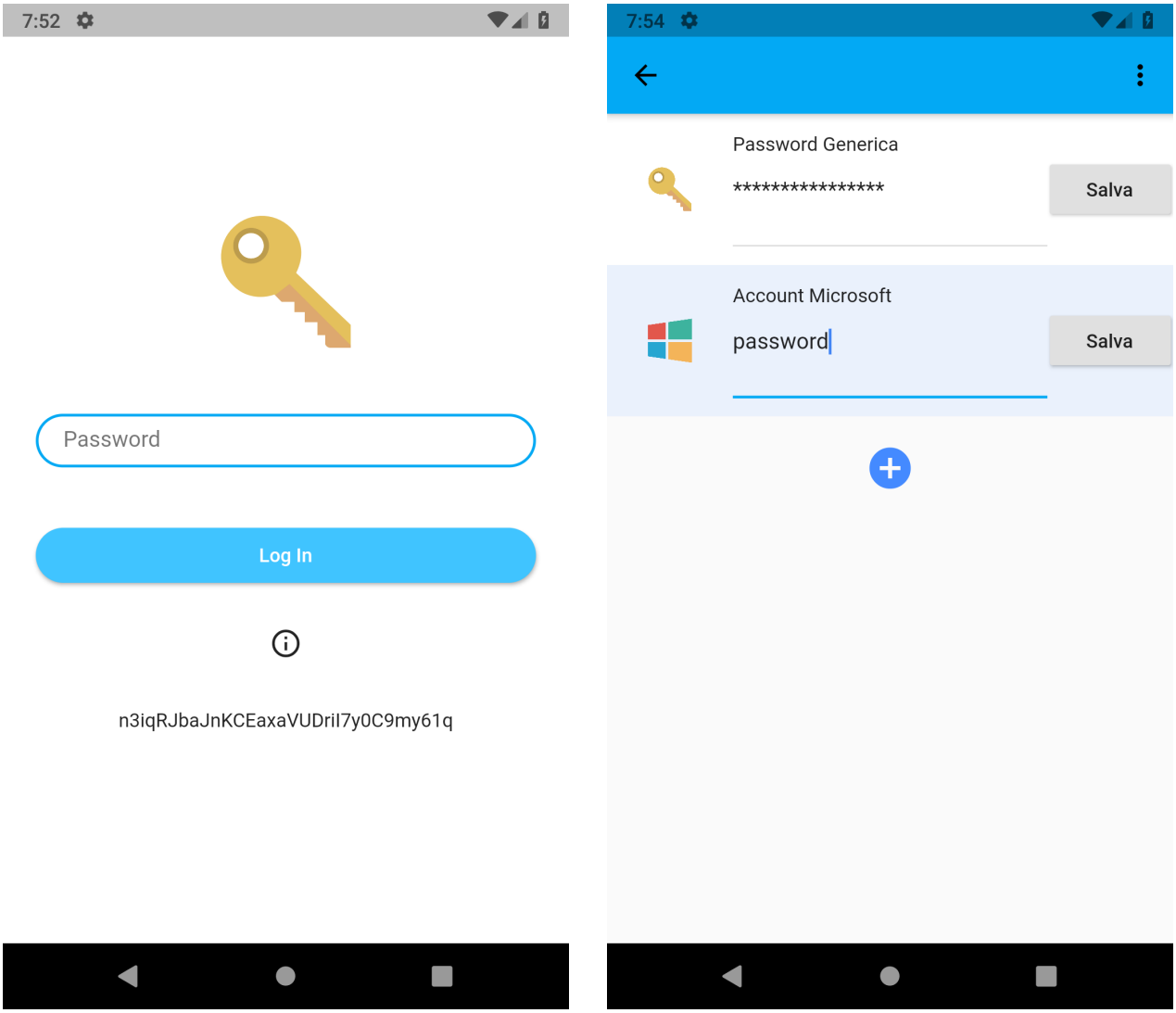

Figure 1: Pagina di Login e Homepage

2. Ricerca del sale in *salt.txt*. Se presente, lo carica in memoria, altrimenti lo genera e lo memorizza sia sulla variabile locale che su file.

3. Se la chiave immessa non ha lunghezza 0, accedi alla pagina *HomePage.dart* passando a parametro il sale e la chiave di cifratura, altrimenti restituisci messaggio di errore.

Caricamento HomePage

1. Effettua *salatura* della chiave di cifratura utilizzando un'istanza di PBKDF2 e il sale preso in input dalla pagina di Login.

2. Leggi il file *password.txt* contenente il file JSON delle password memorizzate in locale, quindi decodifica il risultato in lista.

3. Per ognuna delle password memorizzate in JSON, crea un oggetto *PasswordJSON* effettuando il parse JSON, quindi crea un *CardItem* a partire da esso.

Caricamento password memorizzate

1. Memorizza in una variabile locale *lastId* il più grande Id memorizzato su file.

2. Effettua la decifrazione della password contenuta in PasswordJSON. Se va a buon fine, visualizza la password in chiaro, altrimenti visualizza "**********". Si noti che è possibile cifrare password con chiavi diverse in contemporanea, quindi alcune CardItem potrebbero visualizzare gli asterischi mentre altri no.

Inserimento di una nuova password

1. Crea un PasswordJSON con campi null.

2. Crea un CardItem passando a parametro l'oggetto PasswordJSON creato.

3. Incrementa la variabile *lastId*.

4. Cifra la password contenuta in PasswordJSON utilizzando la chiave salata.

5. Se il bottone "Salva" viene premuto, memorizza PasswordJSON su file locale e in memoria centrale.

Importa/Esporta

1. Per l'esportazione, genera un file rinominato come *GIORNOMESEANNO-password-manager-backup.txt*. Concatena le password memorizzate in JSON con il sale, separandoli con "&&&". Apri il menù per l'esportazione su altre applicazioni.

2. Per l'importazione, separa le password JSON dal sale sfruttando il separatore "&&&", quindi memorizzali nei rispettivi file.

3.4 Difetti e punti di attacco

Poiché CBC con IV casuale non è sicuro rispetto CCA, un malfattore potrebbe modificare i testi cifrati e non causare errori di decifrazione. Inoltre, dopo essere stata calcolata, la chiave di cifratura rimane memorizzata in memoria centrale [1], ciò potrebbe comportare un rischio se un malintenzionato riesce ad entrare in possesso del dispositivo. Quest'ultimo caso potrebbe essere facilmente risolto cancellando la memoria centrale dopo un certo periodo di utilizzo.

Da uno studio effettuato sulla sicurezza dei password manager, il procedimento utilizzato per la cifratura dei messaggi non è sicuro da attacchi sull'integrità dei messaggi [2].

4 Altre soluzioni

KeePassDroid [3] Basato sull'uso di un database protetto da una password. Questo è diviso in due parti: un header e un body cifrato. L'header contiene campi utilizzati per le operazioni di cifratura e decifratura, mentre il body contiene i record XML cifrati in AES-256 CBC. In aggiunta alla cifratura AES, ad ogni record XML viene applicato una operazione di XOR con una stringa generata casualmente utilizzando Salsa20. Dopo ogni salvataggio del DB, viene generata una chiave k a 256 bit da utilizzare per codificare le password.

iOS [4] Quando un utente protegge una nota a partire dalla frase chiave dell'utente viene generata una chiave a 16 byte tramite PBKDF2 e SHA256. Il contenuto della nota è codificato tramite AES-GCM.

References

[1] Dimitris Apostolopoulos, Giannis Marinakis, Christoforos Ntantogian, and Christos Xenakis *Discovering Authentication Credentials in Volatile Memory of Android Mobile Devices.*

[2] Paolo Gasti and Kasper B. Rasmussen *On the Security of Password Manager Database Formats.*

[3] *http://www.keepassdroid.com/.*

[4] *https://www.apple.com/it/business/site/docs/iOS_Security_ Guide.pdf.*