ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

TITOLO DELL'ELABORATO

CRITTOGRAFIA NELLE PRINCIPALI APPLICAZIONI DI MESSAGGISTICA ISTANTANEA

Elaborato in

Crittografia

Relatore

Luciano Margara

Presentata da

Luca Ferretti

Anno Accademico 2020/2021

Indice

Indice delle figure

1. Introduzione:

L'obiettivo della presente tesi è quello di fornire un'analisi dettagliata delle tecniche di crittografia adottate dalle principali applicazioni di messaggistica istantanea.

Al giorno d'oggi la privacy è un tema molto importante e l'evoluzione tecnologica del recente passato lo ha reso ancora più centrale.

I telefoni cellulari sono diventati i dispositivi di comunicazione più diffusi e vengono usati quotidianamente per scambiare informazioni di qualsiasi natura, dai saluti ad una persona che si conosce a dati riservati come password e coordinate bancarie.

Gli utenti che utilizzano dispositivi elettronici richiedono che le trasmissioni dei dati siano sicure e riservate. Per questo motivo diventa fondamentale l'utilizzo della crittografia da parte delle applicazioni che forniscono servizi basati su scambi di dati.

La crittografia (dal greco "Kryptós", nascosto) è lo studio di tecniche di comunicazione sicure allo scopo di consentire ai soli mittenti e destinatari di conoscere il contenuto delle informazioni che si scambiano.

I procedimenti si basano sull'applicazione di formule matematiche e tecniche informatiche dalle quali si ricava il testo cifrato. Quest'ultimo risulta incomprensibile e computazionalmente difficile da decifrare per tutti tranne gli effettivi destinatari.

Questo porta ad avere i seguenti vantaggi:

- Sicurezza
- Riservatezza
- Integrità

Verranno prese in considerazione le applicazioni di messaggistica istantanea più diffuse, ovvero WhatsApp Messenger, Telegram e Signal.

WhatsApp è nata come alternativa agli SMS ed ha conosciuto una crescita notevole che l'ha portata ad avere un bacino di utenza di oltre due miliardi di persone in 180 paesi.

Attualmente fa parte del gruppo "Facebook Inc." ed è largamente utilizzata per l'invio di messaggi di testo, foto, video, documenti, posizioni, per la condivisione di messaggi di stato e per effettuare chiamate e videochiamate.

L’applicazione è gratuita, necessita di una connessione internet per funzionare e per registrarsi occorre collegare il proprio numero di telefono.

Telegram è un servizio di messaggistica istantanea nato nel 2013 basato su cloud ed erogato dalla “Telegram LLC”, società con sede a Dubai.

Con più di 500 milioni di utenti attivi che la rendono una delle 10 app più scaricate al mondo, Telegram offre servizi di chat private, chat di gruppo (che possono contenere fino a 200.000 membri), bot, dirette streaming e chat segrete.

Attraverso questi canali si possono condividere messaggi di testo, immagini, video, posizioni e documenti.

È possibile accedere al proprio account senza l’esistenza di un limite massimo di dispositivi connessi contemporaneamente.

Anche Telegram è gratuita, necessita di una connessione internet per funzionare e richiede un numero di telefono a cui collegare l’account in quanto è richiesto l’inserimento di un codice di verifica inviato per SMS all’atto della registrazione.

Signal nasce nel novembre del 2015 dall’unione delle applicazioni TextSecure e RedPhone con la regia, fra gli altri, di Moxie Marlinspike, un crittografo ex capo della sicurezza di Twitter.

L’applicazione viene rilasciata dalla software house Open Whisper Systems e nel febbraio del 2018 passa alla neofondata Signal Foundation, organizzazione non profit con sede in California.

A gennaio del 2021 ha toccato quota 105 milioni di download totali.

Come WhatsApp e Telegram, anche Signal è gratuita e richiede l’inserimento del proprio numero di telefono per completare la procedura di autenticazione durante il primo utilizzo.

Sia il codice sorgente di Signal che quello di Telegram sono open source: entrambe le società hanno voluto optare per una maggiore trasparenza agli occhi degli utenti.

Diversa la scelta di WhatsApp di avere un codice closed source e di appoggiarsi sul protocollo “Signal” di Open Whisper Systems, quest’ultimo open source.

2. Concetti base:

Per poter entrare nel dettaglio e comprendere come viene usata la crittografia dalle applicazioni descritte in precedenza è necessario introdurre dei concetti di base.

2.1 Crittografia simmetrica:

Questa tecnica è caratterizzata dalla presenza di un'unica chiave privata usata per cifrare e per decifrare il messaggio.

I processi di cifratura e decifrazione sono noti, mentre ad essere segreta è la chiave che deve rimanere un'informazione conosciuta esclusivamente ai due utenti per mantenere la segretezza della comunicazione.

La difficoltà di indovinare o di venire a conoscenza della chiave privata coincide con la sicurezza della procedura.

Nasce così il problema di dover trovare un canale sicuro dove poter scambiare la chiave privata.

2.2 Crittografia asimmetrica:

Per superare il problema dello scambio della chiave privata nascono, a partire dal 1976, i primi cifrari a chiave pubblica.

Nella crittografia asimmetrica ogni utente dispone di una chiave pubblica K_{pub} e di una chiave privata K_{prv}.

La chiave pubblica è nota a tutti e viene usata per il processo di cifratura, la chiave privata invece viene utilizzata per il processo inverso e deve essere tenuta segreta.

Anche in questa tecnica i procedimenti di cifratura e decifrazione sono noti ma lavorano con chiavi distinte.

Ipotizziamo uno scenario con due utenti che vogliono comunicare: Alice e Bob.

Alice vuole inviare un messaggio a Bob quindi lo scrive e lo cifra con la K_{pub} di Bob.

Bob riceve il messaggio da Alice ed è in grado di decifrarlo utilizzando la propria K_{prv}.

La sicurezza della crittografia asimmetrica risiede nel fatto che il crittogramma è decifrabile solamente da chi possiede la K_{prv} del destinatario, quindi dal destinatario stesso.

Questo consente di garantire ai due utenti

- Sicurezza
- Riservatezza
- Confidenzialità (in quanto solo il destinatario del messaggio può decifrarlo)
- Autentificazione (il ricevente è in grado di identificare con certezza il mittente)

Il principio su cui si fondano i cifrari a chiave pubblica è l'esistenza di una funzione one-way con trap-door.

Questa proprietà caratterizza le funzioni che sono facili da calcolare ma difficili da invertire (da qui la dicitura one-way).

La difficoltà nell'invertire viene meno se si è a conoscenza di informazioni aggiuntive che rendono semplice il calcolo inverso.

Esempi di funzioni one-way con trap-door sono il calcolo del logaritmo discreto e la fattorizzazione di un numero intero.

3. Crittografia su WhatsApp, Telegram e Signal:

Le applicazioni prese in esame offrono differenti servizi di crittografia dei dati che tuttavia presentano elementi comuni.
Questi sono trattati nei paragrafi seguenti insieme ai protocolli e ai meccanismi necessari per la loro comprensione nel dettaglio.

3.1 Concetti base:

3.1.1 Crittografia end-to-end

La crittografia end-to-end, letteralmente "da un estremo all'altro", è un'implementazione della crittografia asimmetrica anche nota come E2EE (End to End Encryption).

I messaggi vengono crittografati in modo da consentire solo ed esclusivamente al relativo destinatario di poterli decifrare.

Questo avviene perché sono gli endpoint a detenere le chiavi crittografiche ed il prestatore di servizi è un semplice messaggero, il quale non ha gli strumenti per poter decifrare il contenuto che trasporta.

L'utilizzo della crittografia asimmetrica ricorda la logica della cassetta postale: tutti possono essere a conoscenza dell'indirizzo di qualcuno (chiave pubblica) a cui spedire qualcosa ma solo il destinatario ha la possibilità di aprirla (chiave privata).

Nella E2EE i processi di cifratura e decifrazione avvengono negli endpoint, facendo così trasportare dai server il messaggio codificato.

Questo assicura che i prestatori di servizi, non possedendo le chiavi degli utenti e non dovendo cifrare e decifrare il messaggio, non abbiano possibilità in alcun modo di venire a conoscenza del contenuto in forma leggibile.

I dati memorizzati nei server degli intermediari sono cifrati, dunque viene meno un fattore di rischio rilevante per la privacy degli utenti.

In questo modo, eventuali attacchi hacker aventi come obiettivo i server dei prestatori di servizi o eventuali pressioni di governi e autorità per ottenere informazioni non hanno speranze di poter acquisire dati degli utenti.

Dopo una prima analisi possiamo stilare una lista di vantaggi derivanti dall'utilizzo della crittografia end-to-end:

- Sicurezza (le tecniche utilizzate garantiscono che le comunicazioni ricevute siano decifrabili solamente dall'effettivo destinatario)
- Riservatezza (il contenuto della comunicazione è segreto e conosciuto solo dai due utenti agli endpoint)
- Confidenzialità
- Maggiore protezione dei dati nei confronti dei crittoanalisti (sia con comportamento attivo che passivo)
- I dati memorizzati sui server non hanno valore in quanto non leggibili
- Per lo stesso motivo i server sono meno soggetti ad attacchi hacker e quindi più affidabili

3.1.2 Curve ellittiche

La crittografia basata sulle curve ellittiche nasce dagli studiosi Koblitz e Miller nel 1985 ed è una tecnica di crittografia asimmetrica.

Denotiamo come campo K un insieme non vuoto dotato di due operazioni che soddisfano

- proprietà associativa
- proprietà commutativa
- esistenza elemento neutro
- esistenza inverso di ogni elemento (eccetto dello 0 nella moltiplicazione)

Una curva ellittica "E" è definita su un campo K come l'insieme di punti $(x,y) \in K^2$ che soddisfano l'equazione

$$y^2+axy+by=x^3+cx^2+dx+e$$

Le curve ellittiche devono il loro successo nell'ambito della crittografia alla proprietà di attribuire ai loro punti la struttura algebrica di un gruppo abeliano additivo. Vale a dire una legge di composizione interna che permette di associare ad ogni coppia di punti (x,y) sulla curva un terzo punto, anch'esso sulla curva.

Ad esempio nel caso di operazione somma, prendendo $P1,P2 \in$ curva, $P1+P2=P3 \in$ curva.

3.1.3 Protocollo Diffie-Hellman (DH)

Il protocollo Diffie-Hellman è un protocollo utilizzato per consentire a due utenti di scambiare una chiave segreta in un canale non sicuro.

Si basa su un meccanismo di cooperazione, la chiave K non è scelta dai due utenti singolarmente ma generata da loro in modo congiunto.

È presente il concetto di gruppo: struttura algebrica formata da un insieme non vuoto e da un'operazione binaria interna (il risultato dell'operazione fra due elementi dell'insieme è esso stesso appartenente all'insieme).

L'operazione deve essere associativa, deve esistere un elemento neutro e l'inverso di ogni elemento con la stessa logica dei campi delle curve ellittiche. Se l'operazione è anche commutativa, il gruppo viene definito abeliano.

Dato un gruppo moltiplicativo G, un elemento g si dice generatore se posso ottenere tutto il gruppo facendo potenze di g.

$$\{g^i : i \in \mathbb{Z}\} = G$$

Identifichiamo con $\mathbb{Z}^*_p$ il gruppo moltiplicativo degli interi modulo p e primi con p.

Illustriamo il procedimento con cui Alice e Bob generano una K condivisa:

- Alice sceglie a caso p numero primo, g generatore di $\mathbb{Z}^*_p$ e a numero intero >0
- Alice calcola $A=g^a \bmod(p)$ e spedisce a Bob (g,p,A)
- Bob sceglie a caso b numero
- Bob calcola $B= g^b \bmod(p)$ e spedisce ad Alice B
- Bob calcola chiave segreta $K=A^b \bmod(p)$
- Alice calcola chiave segreta $K=B^a \bmod(p)$

Le due K generate in modo cooperativo da Alice e Bob sono identiche, hanno così generato una chiave privata condivisa.

Sul canale non sicuro passano in chiaro p,g,A e B che sono quindi facilmente intercettabili. Ciò però non mina la riservatezza della chiave K poiché per calcolarla bisognerebbe risolvere $A=g^a \bmod(p)$ in funzione di "a" con A e p noti (oppure $B=g^b \bmod(p)$ con B,p noti).

Questo corrisponde al problema del logaritmo discreto di A o B in base g, problema computazionalmente difficile quindi improponibile per valori di p molto grandi con qualsiasi calcolatore.

Viene così raggiunto l'obiettivo: Alice e Bob condividono una chiave segreta creata attraverso un protocollo cooperativo scambiata in un canale non sicuro.

3.1.4 Curva ellittica Diffie-Hellman (ECDH)

La curva ellittica Diffie-Hellman sposta il problema del logaritmo discreto sulle curve ellittiche.

Il procedimento è simile a quello visto in precedenza:

- Alice e Bob scelgono un campo finito F_q ed una curva ellittica E definita sul campo F_q
- Alice e Bob scelgono e rendono pubblico un punto $P \in E$ di ordine n molto grande (dove n è il più piccolo intero positivo tale che $nP=0$)
- Alice sceglie un intero a ($a<n$) da mantenere segreto come sua chiave privata
- Alice calcola $aP \in E$ e lo manda in chiaro a Bob (chiave pubblica, corrisponde ad un punto $\in E$ scelto casualmente)
- Bob sceglie un intero b ($b<n$) da mantenere segreto come sua chiave privata
- Bob calcola $bP \in E$ e lo spedisce ad Alice in chiaro (chiave pubblica)

La chiave segreta K che Alice e Bob calcolano è così costituita:

$$K=abP \in E$$

Trovare a conoscendo P e aP (oppure trovare b conoscendo P e Pb) corrisponde alla risoluzione del problema del logaritmo discreto sulle curve ellittiche.

Si ritiene che questo problema sia notevolmente più difficile da trattare rispetto al medesimo su campi finiti.

Tutti gli algoritmi noti hanno complessità esponenziale, quindi proibitiva per numeri di ordine grande.

Le curve ellittiche risultano per certi versi migliori degli altri algoritmi a chiave pubblica anche perché, a parità di sicurezza, richiedono chiavi di dimensione minore.

3.1.5 Curve25519

La Curve25519 è una funzione ad alta sicurezza e ad alta velocità della curva ellittica Diffie-Hellman.

Ogni utente della Curve25519 ha una chiave privata e una chiave pubblica, entrambe di 32 byte.

Ogni coppia di utenti ha un segreto condiviso di 32 byte usato per autenticare e crittografare i messaggi fra di loro.

Curve25519 utilizza una curva ellittica, detta curva di Montgomery, della forma $y^2=x^3+486662x^2+x$ definita sul campo finito Fp con $p=2^{255}-19$.

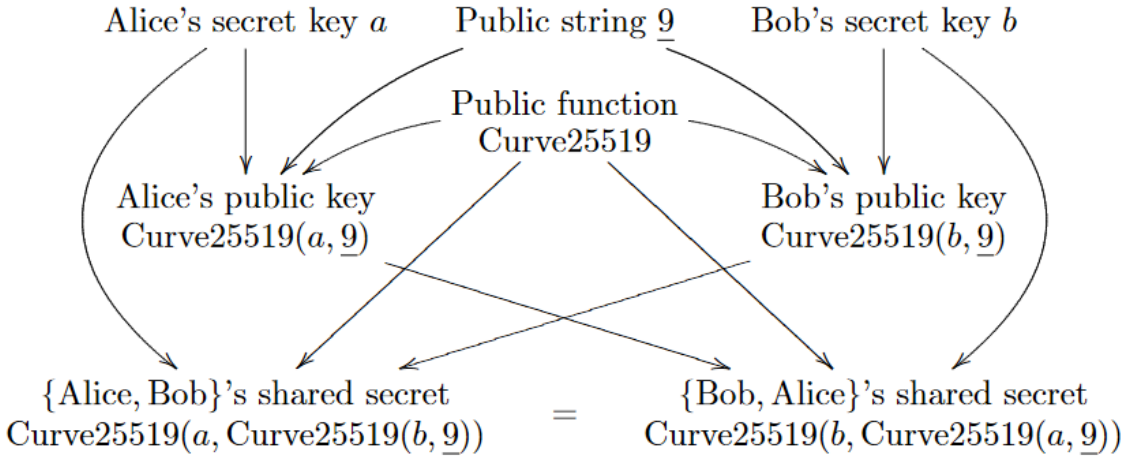

Fig. 3.1.5 Generazione segreto condiviso curve25519

Vediamo ora il processo di generazione delle chiavi e del segreto condiviso. Alice e Bob generano una propria chiave privata in modo casuale. A queste vengono applicate funzioni della Curve25519 per ottenere le rispettive chiavi pubbliche.

La chiave pubblica di Alice sarà quindi 9a mod(p) e quella di Bob 9b mod(p).

I due si scambiano le chiavi pubbliche ed ognuno effettua il prodotto scalare tra la chiave pubblica ricevuta e la propria chiave privata, ottenendo così due valori uguali di K=9ab mod(p).

Un hash del segreto condiviso può essere utilizzato come chiave segreta per autenticare i messaggi o come chiave per un sistema a crittografia simmetrica con funzione di autenticazione e cifratura.

Vantaggi della Curve25519:

- Velocità estremamente elevata
- Immune a timing attacks
- Chiavi di piccole dimensioni
- Tempo di validazione chiavi trascurabile

3.1.6 AES-256

AES (Advanced Encryption Standard) è un algoritmo di cifratura nato negli Stati Uniti a fine 2001.

Fa uso del cifrario a blocchi "Rijndael" come algoritmo di cifratura a chiave simmetrica ed è diventato uno standard per il governo degli USA.

Lavora prendendo in input una chiave che può essere lunga 128,192 o 256 bit (da qui i nomi delle versioni), ma i blocchi utilizzati hanno tutti una dimensione fissa pari a 128 bit e sono rappresentati in formato esadecimale in tabelle 4x4.

Se si vuole utilizzare una password è necessario applicare una funzione hash (ad esempio SHA-256) per ottenere una chiave a 256 bit.

Attraverso un processo di espansione della chiave iniziale, chiamato Key Expansion, vengono create nuove chiavi dette chiavi rotonde (rounded keys), le quali vengono usate nei vari round (14 per AES-256).

Ogni round ad esclusione dell'ultimo è formato dai seguenti 4 passi (layer):

1) Trasformazione SubBytes

2) Trasformazione ShiftRows

3) Trasformazione Mix Columns

4) AddRoundKey

Il quattordicesimo round si differenzia in quanto non ha la trasformazione Mix Columns.

Al principio si effettua un'operazione di AddRoundKey, uno XOR fra il messaggio e la prima round key. Il layer 1 corrisponde ad una permutazione guidata da una tabella detta "S-box" di 16x16 byte. (es. 19->x=1,y=9 -> D4)

		y														
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	F
x	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Fig. 3.1.6.1 Tabella S-box di SubBytes

Al passo 2 si esegue uno scorrimento di byte: la riga 1 non viene modificata, la 2° subisce uno shift circolare a sinistra di un byte, la terza lo subisce di 2 byte e la quarta di 3 byte.

Il layer 3 prevede operazioni sulle colonne, ciascun byte subisce una trasformazione che possiamo vedere come una moltiplicazione tra matrici.

$$\begin{bmatrix} \mathbf{02} & \mathbf{03} & \mathbf{01} & \mathbf{01} \\ \mathbf{01} & \mathbf{02} & \mathbf{03} & \mathbf{01} \\ \mathbf{01} & \mathbf{01} & \mathbf{02} & \mathbf{03} \\ \mathbf{03} & \mathbf{01} & \mathbf{01} & \mathbf{02} \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} \mathbf{s'_{0,0}} & \mathbf{s'_{0,1}} & \mathbf{s'_{0,2}} & \mathbf{s'_{0,3}} \\ \mathbf{s'_{1,0}} & \mathbf{s'_{1,1}} & \mathbf{s'_{1,2}} & \mathbf{s'_{1,3}} \\ \mathbf{s'_{2,0}} & \mathbf{s'_{2,1}} & \mathbf{s'_{2,2}} & \mathbf{s'_{2,3}} \\ \mathbf{s'_{3,0}} & \mathbf{s'_{3,1}} & \mathbf{s'_{3,2}} & \mathbf{s'_{3,3}} \end{bmatrix}$$

Fig. 3.1.6.2 Mix Columns

(La matrice a sinistra è nota come Rijndael Galois Field)

Per una generica colonna j, con $0 \leq j \leq 3$, i nuovi elementi si possono calcolare in questo modo, dove $\oplus$ è l'operazione di XOR mentre $\bullet$ è l'operazione di moltiplicazione in un campo finito.

$$s'_{0,j} = (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$

$$s'_{1,j} = s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j}$$

$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j})$$

$$s'_{3,j} = (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})$$

Fig. 3.1.6.3 Calcolo nuovi elementi Mix Columns

Al passo 4 si effettua un'operazione di XOR fra il blocco ottenuto al passaggio precedente e la chiave segreta.

L'algoritmo di AES-256 può perciò essere sintetizzato così:

Testo in chiaro -> AddRoundKey -> 13 round di: SubBytes, ShiftRows, MixColumns, AddRoundKey -> 14° round: SubBytes, ShiftRows, AddRoundKey -> Testo cifrato

AES-256 è ritenuto il più sicuro della famiglia AES grazie alla maggiore lunghezza della chiave segreta. Avendo una dimensione di 256 bit sono di fatto impraticabili attacchi "brute force". Le possibili combinazioni sono 2^{256} , un numero esponenzialmente più grande del numero di atomi osservabili nell'universo.

3.1.7 HMAC (Hash-based Message Authentication Code)

HMAC è un algoritmo per autenticazione di messaggi che lavora applicando una funzione hash al messaggio insieme ad una chiave segreta. L'hash risultate viene detto digest o firma.

Il destinatario del messaggio, il quale possiede la chiave segreta, può eseguire l'hash del messaggio con il medesimo algoritmo e controllare la firma generata verificando la corrispondenza con quella ricevuta.

In questo modo viene autenticato il messaggio e verificata la sua integrità.

Possiamo riassumere il procedimento con la seguente formula.

$$\mathrm{HMAC}(K, m) = H\left((K' \oplus \mathrm{opad}) \| H\left((K' \oplus \mathrm{ipad}) \| M'\right)\right)$$

Fig. 3.1.7.1 Formula HMAC

K' si ottiene dalla chiave segreta K in base alla sua lunghezza: se è maggiore della dimensione del blocco K'=H(K), altrimenti K'=K+padding di zeri per arrivare alla dimensione del blocco.

Avendo "b" numero di bit in un blocco, denotiamo come ipad un blocco contenente il valore 36 in esadecimale ripetuto b/8 volte e opad un blocco simile avente come valore che si ripete 5C.

Si effettua uno XOR iniziale tra K' e ipad ed al risultato ("Si") viene concatenato il messaggio suddiviso in "l" blocchi da b bit.

Si applica quindi la funzione hash a Si||M sfruttando un vettore di inizializzazione "iv" opportuno. Se l'hash generato ha lunghezza minore di "b" si completa con un'operazione di zero padding.

Si procede allo XOR fra K' e opad generando "S0" e a questo si concatena l'hash generato in precedenza. Abbiamo così ottenuto l'argomento da passare alla funzione H (inizializzata servendosi di "iv" opportuno).

Il costo computazionale di HMAC è dello stesso ordine di grandezza della funzione hash utilizzata.

Ulteriori pregi sono la possibilità di poter sostituire facilmente la funzione H e la garanzia di un discreto livello di sicurezza sfruttando le sue proprietà one-way.

Nella figura seguente è illustrato il procedimento dettagliato spiegato nella pagina precedente.

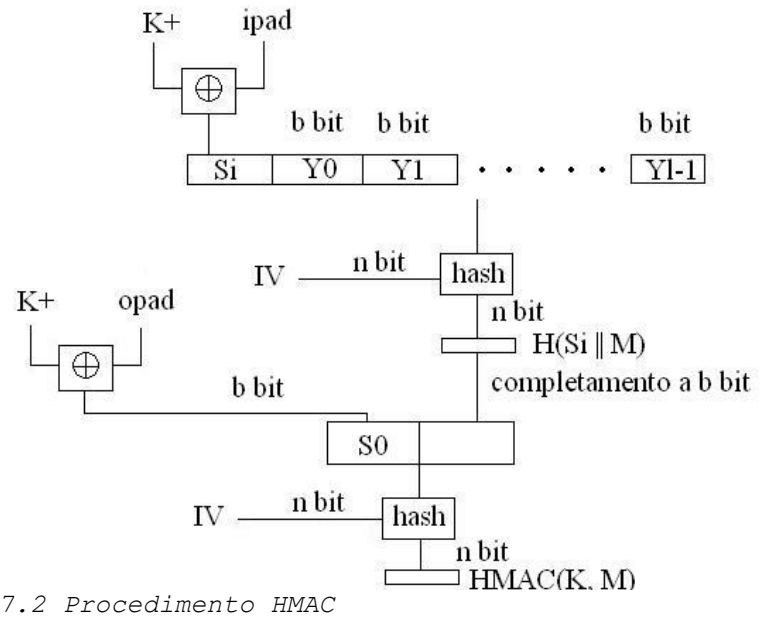

Fig. 3.1.7.2 Procedimento HMAC

3.1.8 SHA-256

La famiglia SHA (Secure Hash Algorithm) comprende diversi algoritmi che generano un digest di lunghezza fissa a partire da una stringa di lunghezza variabile: fra i più noti SHA-1, SHA-244, SHA-256, SHA-384 e SHA-512.

L'output di SHA-256 ha una dimensione di 256 bit. L'algoritmo opera con blocchi di 512 bit (16x32) ed ogni blocco richiede 64 round.

Partiamo definendo le operazioni alla base del funzionamento:

- AND,OR,XOR rispettivamente indicate come $\wedge, \vee, \oplus$
- Sottrazione bit a bit, -
- Somma modulo 2^{32}, A+B
- RotR(A,n), shift circolare a destra di n bit su A
- ShR(A,n), shift a destra di n bit su A
- A||B, concatenazione fra A e B

Illustriamo inoltre le funzioni che verranno usate:

$$
\begin{aligned}
Ch(X,Y,Z) &= (X \wedge Y) \oplus (\overline{X} \wedge Z),\\
Maj(X,Y,Z) &= (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z),\\
\Sigma_0(X) &= RotR(X,2) \oplus RotR(X,13) \oplus RotR(X,22),\\
\Sigma_1(X) &= RotR(X,6) \oplus RotR(X,11) \oplus RotR(X,25),\\
\sigma_0(X) &= RotR(X,7) \oplus RotR(X,18) \oplus ShR(X,3),\\
\sigma_1(X) &= RotR(X,17) \oplus RotR(X,19) \oplus ShR(X,10)
\end{aligned}
$$

Fig. 3.1.8.1 Funzioni utilizzate da SHA-256

Di seguito le 64 word K_i date dai primi 32 bit delle parti frazionarie delle radici cubiche dei primi 64 numeri primi.

0x428a2f98	0x71374491	0xb5c0fbcf	0xe9b5dba5	0x3956c25b	0x59f111f1	0x923f82a4	0xab1c5ed5
0xd807aa98	0x12835b01	0x243185be	0x550c7dc3	0x72be5d74	0x80deb1fe	0x9bdc06a7	0xc19bf174
0xe49b69c1	0xefbe4786	0x0fc19dc6	0x240ca1cc	0x2de92c6f	0x4a7484aa	0x5cb0a9dc	0x76f988da
0x983e5152	0xa831c66d	0xb00327c8	0xbf597fc7	0xc6e00bf3	0xd5a79147	0x06ca6351	0x14292967
0x27b70a85	0x2e1b2138	0x4d2c6dfc	0x53380d13	0x650a7354	0x766a0abb	0x81c2c92e	0x92722c85
0xa2bfe8a1	0xa81a664b	0xc24b8b70	0xc76c51a3	0xd192e819	0xd6990624	0xf40e3585	0x106aa070
0x19a4c116	0x1e376c08	0x2748774c	0x34b0bcb5	0x391c0cb3	0x4ed8aa4a	0x5b9cca4f	0x682e6ff3
0x748f82ee	0x78a5636f	0x84c87814	0x8cc70208	0x90befffa	0xa4506ceb	0xbef9a3f7	0xc67178f2

Fig. 3.1.8.2 64 word utilizzate da SHA-256

Per assicurarsi che i blocchi abbiano una dimensione pari ad un multiplo di 512 bit si eseguono 3 passaggi

- inizialmente viene aggiunto un bit=1
- successivamente vengono aggiunti k bit=0, con k il più piccolo intero positivo tale che $L+1+k \equiv 448 \bmod(512)$ e L dimensione in bit del messaggio iniziale. Si ottiene così un messaggio a cui mancano esattamente 64 bit per avere una lunghezza multipla di 512
- i 64 bit mancanti vengono riempiti inserendo L, si ottiene così una struttura come quella illustrata nell'immagine seguente

Fig. 3.1.8.3 Struttura blocchi SHA-256

Per ogni blocco $M \in \{0,1\}^{512}$, le 64 word di 32 bit ognuna sono così costruite:

- le prime 16 si ottengono suddividendo M in blocchi da 32 bit $M=W_1||W_2||\ldots||W_{16}$
- le successive 48 si ottengono mediante la formula

$W_i = W_{i-16}+\sigma_0(W_{i-15})+W_{i-7}+\sigma_1(W_{i-2})$, con $17 \leq i \leq 64$

Dove $\sigma_0(x)=RotR(x,7) \oplus RotR(x,18) \oplus ShR(x,3)$ e $\sigma_1(x)=RotR(x,17) \oplus RotR(x,19) \oplus ShR(x,10)$

Inizializziamo 8 variabili con i primi 32 bit della parte frazionaria delle radici quadrate dei primi 8 numeri primi:

$H_1^{(0)}$=0x6a09e667 $H_2^{(0)}$= 0xbb67ae85 $H_3^{(0)}$= 0x3c6ef372 $H_4^{(0)}$= 0xa54ff53a
$H_5^{(0)}$= 0x510e527f $H_6^{(0)}$= 0x9b05688c $H_7^{(0)}$= 0x1f83d9ab $H_8^{(0)}$= 0x5be0cd19

I blocchi del messaggio $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$ sono così uno alla volta processati:

for t=1 to N

- Costruzione dei 64 blocchi W_i partendo da $M^{(t)}$come descritto in precedenza
- $(a, b, c, d, e, f, g, h) = (H_1^{(t-1)}, H_2^{(t-1)}, H_3^{(t-1)}, H_4^{(t-1)}, H_5^{(t-1)}, H_6^{(t-1)}, H_7^{(t-1)}, H_8^{(t-1)})$
- for i=0 to 63
  - $T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_i + W_i$
  - $T_2 = \Sigma_0(a) + Maj(a, b, c)$
  - h=g
  - g=f
  - f=e
  - e=d+T1
  - d=c
  - c=b
  - b=a
  - $a = T_1 + T_2$

End for

- $H_1^{(t)} = H_1^{(t-1)} + a$, $H_2^{(t)} = H_2^{(t-1)} + b$, $H_3^{(t)} = H_3^{(t-1)} + c$, $H_4^{(t)} = H_4^{(t-1)} + d$, $H_5^{(t)} = H_5^{(t-1)} + e$, $H_6^{(t)} = H_6^{(t-1)} + f$, $H_7^{(t)} = H_7^{(t-1)} + g$, $H_8^{(t)} = H_8^{(t-1)} + h$

End for

L'hash finale da 256 bit del messaggio è dato dalla concatenazione degli $H^{(N)}$ dopo l'ultimo ciclo:

$H = H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)} || H_5^{(N)} || H_6^{(N)} || H_7^{(N)} || H_8^{(N)}$

4. Implementazione su WhatsApp

Dopo una dettagliata analisi ed una piena comprensione dei protocolli e dei meccanismi di crittografia descritti, possiamo vedere come vengono utilizzati da WhatsApp Messenger per rendere le comunicazioni fra gli utenti sicure e riservate.

Vengono usati tre tipi di chiavi pubbliche:

- Identity Key Pair, una coppia di chiavi Curve25519 a lungo termine generate nel momento dell'installazione dell'applicazione
- Signed Pre Key, una coppia di chiavi Curve25519 a medio termine generate all'installazione, firmate dall'Identity Key Pair e rigenerate periodicamente
- One-Time Pre Keys, una coda di coppie di chiavi Curve25519 ad uso singolo, generate all'installazione e rifornita secondo necessità

Definiamo tre tipi di chiavi di sessione:

- Root Key, un valore di 32 byte usato per creare Chain Keys
- Chain Key, un valore di 32 byte usato per creare Message Keys
- Message Key, un valore di 80 byte usato per crittografare il contenuto dei messaggi: 32 byte sono usati per una chiave AES-256, 32 per una chiave HMAC-SHA-256 e i restanti 16 per un opportuno vettore di inizializzazione "iv"

4.1 Registrazione:

All'atto della registrazione il client WhatsApp trasmette al server la propria Identity Key, la propria Signed Pre Key (con la sua firma) ed un'istanza di One-Time Pre Key.

Il server WhatsApp memorizza queste chiavi pubbliche associate all'identificativo dell'utente.

4.2 Inizializzazione sessione:

Un client WhatsApp per comunicare con un altro utente deve prima stabilire una sessione crittografata.

Una volta stabilita non ci sarà più bisogno di ricrearla fino a che la sessione iniziale non viene persa a causa di eventi come modifiche al dispositivo o reinstallazione dell'app.

Vediamo i passi eseguiti dal client, detto "initiator", per stabilire una sessione crittografata tra due utenti.

1) Richiede le chiavi pubbliche del destinatario Identity Key, Signed Pre Key e un'istanza di One-Time Pre Key
2) Il server fornisce le chiavi richieste e rimuove dalla coda delle One-Time Pre Key quella trasmessa all'initiator. Se l'ultimo lotto di queste chiavi è stato consumato e successivamente non rifornito, non viene restituita alcuna One-Time Pre Key
3) Memorizza l'Identity Key del destinatario come $I_{recipient}$, la Signed Pre Key come $S_{recipient}$ e la One-Time Pre Key come $O_{recipient}$
4) Genera una coppia di chiavi Curve25519 effimera e la memorizza come $E_{intiator}$
5) Salva la propria Identity Key come $I_{initiator}$
6) Calcola il master secret con la seguente formula $master\_secret=ECDH(I_{initiator},S_{recipient})||ECDH(E_{intiator},I_{recipient})||ECDH(E_{intiator},S_{recipient})||ECDH(E_{intiator},O_{recipient})$
7) Usa HKDF per creare una Root Key e Chain Keys a partire dal master secret calcolato

HKDF è una funzione di derivazione di chiavi basata sul messaggio hash di HMAC. Lo schema utilizzato, detto XtX (eXtract-then-eXpand), può essere descritto in due passi:

- $K = Hash(SKM)$
- $KM = PRF(K,"1")||PRF(K,"2")|| \ldots ||PRF(K,"t")$

Il primo modulo prende come parametro SKM (il nostro master_secret) e ne estrae una chiave K pseudorandom di lunghezza fissa. Il secondo consiste nell'espansione della chiave K in ulteriori chiavi attraverso la funzione pseudorandom PRF.

4.3 Ricezione sessione:

Dopo aver stabilito una sessione crittografata, l'initiator può mandare messaggi al destinatario (recipient) anche se quest'ultimo è offline. Fino a che il recipient non risponde, i messaggi dell'initiator includono nell'intestazione le informazioni necessarie al destinatario per creare una sessione corrispondente, ovvero $E_{intiator}$ e $I_{initiator}$.

Quando il destinatario riceve uno di questi messaggi con le informazioni di setup effettua le seguenti operazioni:

- Calcola il corrispondente master_secret utilizzando le sue chiavi private e le chiavi pubbliche contenute nell'intestazione
- Cancella la One-Time Pre Key usata dal mittente

4.4 Scambio di messaggi:

A sessione stabilita, i due utenti possono scambiarsi messaggi protetti attraverso una Message Key usando AES256 in modalità CBC per la cifratura e HMAC-SHA-256 per l'autenticazione.

CBC (Cipher Block Chaining) è una tecnica attraverso la quale cifrando lo stesso messaggio in chiaro ripetutamente si ottengono blocchi cifrati distinti. Questo perché l'input dell'algoritmo di cifratura è il risultato dello XOR fra il testo cifrato precedente ed il blocco in chiaro corrente.

La Message Key è effimera e cambia ad ogni messaggio, per cui una volta usata per cifrarne uno non può essere ricostruita a partire dallo stato della sessione dopo che questo è stato trasmesso. Inoltre, ogni roundtrip dei vari messaggi ci sarà un nuovo accordo sulla ECDH da utilizzare al fine di creare una nuova Chain Key.

Le Message Keys sono derivate dalla Chain Key del mittente che effettua uno "scatto in avanti" ad ogni messaggio inviato, non sarà quindi possibile derivarne il valore corrente o passato a partire dalle Message Key.

Vengono così calcolate:

- Message Key = HMAC-SHA-256(Chain Key, 0x01)
- Chain Key = HMAC-SHA-256(Chain Key, 0x02) (Aggiornamento)

Ad ogni trasmissione di un messaggio viene creata una chiave pubblica Curve25519 e spedita insieme ad esso. Una volta ricevuta una risposta, una nuova Chain Key e Root Key sono calcolate nel seguente modo:

- ephemeral_secret = ECDH(Ehpemeral$_{sender}$, Ephemeral$_{recipient}$)
- Chain Key, Root Key = HKDF(Root Key, ephemeral_secret)

Una Chain viene utilizzata per mandare messaggi solo da un utente, perciò le Message keys non sono riciclate. Per come queste ultime e le Chain Keys sono calcolate è tecnicamente possibile che i messaggi arrivino in ritardo, fuori ordine o che vengano persi.

4.5 Scambio di media e altri allegati:

La crittografia end-to-end di WhatsApp copre anche gli allegati di ogni tipo (video, audio, immagini o file) e di qualsiasi dimensione.

1) Il mittente genera due chiavi da 32 byte entrambe effimere: una chiave AES256 ed una HMAC-SHA-256
2) Successivamente cifra l'allegato con la chiave AES256 in modalità CBC servendosi di un vettore "iv" random e accoda un MAC del testo cifrato utilizzando HMAC-SHA-256
3) Il mittente carica l'allegato cifrato in un archivio BLOB
4) Il mittente trasmette normalmente il messaggio cifrato che contiene la chiave crittografica, la chiave HMAC, un hash SHA-256 del blob cifrato ed un puntatore al blob in questione nell'archivio BLOB
5) Il destinatario decifra il messaggio, recupera il blob crittografato dall'archivio, ne verifica l'hash SHA-256 per essere certo dell'integrità del contenuto, verifica il MAC e decifra il testo in chiaro

4.6 Messaggi nei gruppi:

Per descrivere la crittografia sui messaggi nei gruppi è necessaria una differenziazione preliminare. Solitamente le principali alternative utilizzate nelle chat di gruppo ricadono su due possibili architetture:

- Client-side-fan-out: il mittente cifra il messaggio con le chiavi di ogni membro del gruppo, inviando lo stesso messaggio ad ogni componente singolarmente. In questo modo, in un gruppo di 30 persone il mittente invia 30 messaggi e la chat di gruppo può essere vista come un insieme di molte chat private fra due membri. Vengono usati gli stessi protocolli ma si possono avere costi elevati.

- Server-side-fan-out: il mittente cifra il messaggio e lo invia una volta, il compito di farlo arrivare agli altri membri del gruppo spetta al server. Come è possibile adottare questa architettura e mantenere la E2EE? Il mittente cifra il messaggio con una chiave che è conosciuta da tutti i membri del gruppo. Il tutto avviene tramite un segreto condiviso e con la crittografia simmetrica.

I messaggi nei gruppi WhatsApp si basano su sessioni crittografate a coppie per garantire un'efficiente architettura server-side-fan-out per la maggior parte dei messaggi inviati nei gruppi.

Questo si ottiene utilizzando un componente del protocollo Signal chiamato "Sender Keys".

Vediamo ora i passaggi eseguiti la prima volta che un membro di un gruppo manda un messaggio:

1) Il mittente genera una Chain Key random di 32 byte
2) Genera inoltre una coppia di chiavi Curve25519 "Signature Key"
3) Il mittente combina la Chain Key e la chiave pubblica della Signature Key formando un messaggio Sender Key
4) Infine cifra individualmente la Sender Key per ogni membro del gruppo usando un protocollo di messaggistica a coppie

Per i successivi messaggi nel gruppo gli step sono i seguenti:

1) Il mittente deriva una Message Key dalla Chain Key (che aggiorna successivamente)
2) Cifra il messaggio facendo uso di AES256 in modalità CBC
3) Firma il testo cifrato con la Signature Key
4) Trasmette il messaggio cifrato al server che si occupa di inviarlo a tutti i membri (server-side-fan-out)

Ogni volta che un membro del gruppo esce, i partecipanti provvedono a cancellare la propria Sender Key e riparte il procedimento.

4.7 Chiamate e videochiamate:

Anche le chiamate e le videochiamate WhatsApp sono protette dalla crittografia end-to-end.

I passaggi per il setup delle chiamate sono i seguenti:

1) Il chiamante crea una sessione crittografata con il destinatario come descritto nel capitolo 4.2.
2) Genera un master secret SRTP (Secure Real Time Transport Protocol) random di 32 byte
3) Trasmette un messaggio cifrato al destinatario che segnala una chiamata in arrivo e contiene il master secret SRTP
4) Se il destinatario risponde ha inizio una chiamata crittografata SRTP

4.8 Stati:

Gli stati su WhatsApp sono crittografati in maniera analoga a come lo sono i messaggi di gruppo. Il primo stato inviato ad un insieme di destinatari segue i medesimi step del primo messaggio inviato in una chat di gruppo. Gli stati successivi, se destinati agli stessi utenti, seguono i passaggi dei successivi messaggi del gruppo. Quando un utente rimuove un destinatario degli stati dalle impostazioni WhatsApp o cancella il suo numero dalla rubrica, elimina la propria Sender Key e ricomincia dal principio.

5. Implementazione su Telegram

Telegram differisce da WhatsApp in quanto è un servizio di messaggistica basato su cloud con sincronizzazione istantanea.

Al fine di poter garantire affidabilità anche su connessioni meno stabili e velocità anche quando si trasmettono file di grosse dimensioni (fino a 2GB ciascuno), viene utilizzato il protocollo proprietario MTProto. Prima di analizzarlo è necessario introdurre un algoritmo di crittografia da esso utilizzato: SHA-1.

5.1 SHA-1

SHA-1 è una delle prime versioni degli algoritmi della famiglia SHA.

Prende in input un messaggio di lunghezza variabile (massimo $2^{64}-1$ bit) e restituisce un digest di 160 bit.

In primo luogo ad un messaggio vengono aggiunti dei bit di imbottitura ed un unsigned int di 64 bit in modo da ottenere un nuovo messaggio con lunghezza multipla di 512 bit.

Vengono definite cinque costanti esadecimali H: H_0=67452301, H_1=EFCDAB89, H_2=98BADCFE, H_3=10325476, H_4=C3D2E1F0.

Illustriamo due funzioni che verranno utilizzate dall'algoritmo.

$$F(B,C,D,t) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & se\ 0 \le t \le 19 \\ B \oplus C \oplus D & se\ 20 \le t \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & se\ 40 \le t \le 59 \\ B \oplus C \oplus D & se\ 60 \le t \le 79 \end{cases}$$

$$K(t) = \begin{cases} 5A827999 & se\ 0 \le t \le 19 \\ 6ED9EBA1 & se\ 20 \le t \le 39 \\ 8F1BBCDC & se\ 40 \le t \le 59 \\ CA62C1D6 & se\ 60 \le t \le 79 \end{cases}$$

Fig. 5.1 Funzioni utilizzate da SHA-1

I blocchi del messaggio costruito $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$ sono così uno alla volta processati:

for i=1 to N

- $M^{(i)}=W_0||W_1||...||W_{15}$,dove ogni W_j, detto parola, è formata da 32 bit. Le prime 16 parole corrispondono ai primi 12 blocchi di M

- for t=16 to 79
    - $W_t=(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \leftarrow 1$, dove $\leftarrow 1$ indica uno shift di una posizione a sinistra

  End for
- $A=H_0$, $B=H_1$, $C=H_2$, $D=H_3$, $E=H_4$
- for t=0 to 79
    - $T=(A \leftarrow 5)+F(B,C,D,t)+E+W_t+K(t)$
    - $E=D$, $D=C$, $C=(B \leftarrow 30)$, $B=A$, $A=T$

  End for
- $H_0= H_0+A$, $H_1= H_1+B$, $H_2= H_2+C$, $H_3= H_3+D$, $H_4= H_4+E$,

End for

L'hash finale da 160 bit del messaggio è dato dalla concatenazione degli H dopo l'ultimo ciclo:

$H = H_0||H_1||H_2||H_3||H_4$

5.2 MTProto 2.0:

La versione osservata del protocollo MTProto è la 2.0, consigliata da Telegram e largamente utilizzata dagli utenti. La versione 1.0 è deprecata ma ancora supportata per ragioni di retrocompatibilità.

Il protocollo può essere virtualmente suddiviso in tre componenti indipendenti:

- Alto livello: definisce il metodo con cui le query API e le risposte vengono convertite in messaggi binari.
- Crittografia: definisce il modo in cui i messaggi vengono crittografati prima di essere trasmessi attraverso il protocollo di trasporto.
- Trasporto: il metodo attraverso cui client e server si scambiano messaggi su un protocollo di rete esistente quale HTTP, TCP, UDP e websocket.

MTProto supporta due livelli differenti di crittografia utilizzati nelle due tipologie di chat messe a disposizione degli utenti: chat cloud e chat segrete.

5.3 Chat cloud:

Nelle chat cloud, private e di gruppo, viene utilizzata la crittografia server-client.

I messaggi e i media delle chat cloud vengono archiviati in maniera crittografata nel Cloud di Telegram per permettere l'accesso da tutti i dispositivi.

Vediamo ora come funziona la crittografia server-client di MTProto 2.0.

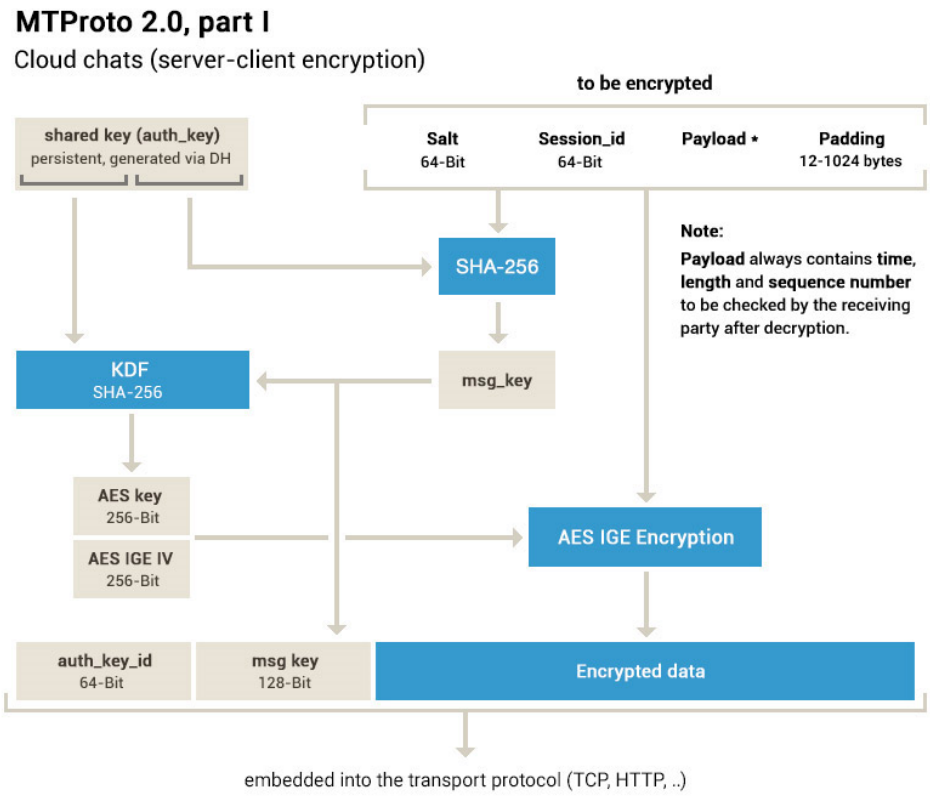

Fig. 5.3 MTProto 2.0 chat cloud

Ogni messaggio da crittografare contiene, oltre al corpo, le seguenti informazioni aggiuntive: server salt (64 bit random generati dal client per distinguere le singole sessioni), session id, message sequence number (32 bit), message length e time. Queste informazioni vanno controllate in fase di decifrazione per evitare problemi fra i componenti.

A partire dal time viene generato un msg_id di 64 bit che identifica un messaggio all'interno di una sessione; i successivi avranno un msg_id incrementato in modo monotono.

Authorization key (auth_key) è una chiave a 2048 bit creata all'atto della registrazione, utilizzando il protocollo Diffie-Hellman, condivisa fra il dispositivo del client e il server.

La message key (msg_key) si ottiene prendendo i 128 bit centrali del digest restituito dalla funzione hash SHA-256 applicata al messaggio da crittografare preceduto da un frammento di 32 byte dell'auth_key.

La msg_key e i 1024 bit più significativi dell'auth_key vengono usati per calcolare una chiave AES di 256 bit indicata come "AES key" e un vettore di inizializzazione "AES IGE IV" di uguale dimensione.

Il procedimento per ottenerli si basa sulla funzione SHA-256 ed è il seguente:

- msg_key_large = SHA-256(substr(auth_key, 88+x, 32) + plaintext + random_padding);
- msg_key = substr(msg_key_large, 8, 16);
- sha256_a = SHA-256(msg_key + substr(auth_key, x, 36));
- sha256_b = SHA-256(substr(auth_key, 40+x, 36) + msg_key);
- AES KEY = substr(sha256_a, 0, 8) + substr(sha256_b, 8, 16) + substr(sha256_a, 24, 8);
- AES IGE IV = substr(sha256_b, 0, 8) + substr(sha256_a, 8, 16) + substr(sha256_b, 24, 8);

dove x=0 per i messaggi dal client al server e x=8 per quelli con percorso opposto.

AES key e AES IGE IV vengono utilizzati per cifrare il messaggio attraverso AES-256 in modalità IGE.

IGE (Infinite Garble Extension) ha come peculiarità quella di essere una modalità di cifrario a blocchi con propagazione degli errori infinita. Può essere descritta dalla seguente formula:

$$c_i = f_K(x_i \oplus c_{i-1}) \oplus m_{i-1}$$

dove f_K è la funzione di cifratura e per la prima iterazione viene utilizzato il vettore di inizializzazione.

I 64 bit meno significativi del digest ottenuto applicando SHA-1 all'auth_key definiscono auth_key_id. Questa rappresenta la specifica chiave utilizzata per crittografare il messaggio.

Le varie auth_key_id devono essere univoche e per questo motivo in caso di collisione viene replicato il procedimento generando una nuova auth_key.

Sul protocollo di trasporto viene inviato il messaggio cifrato preceduto dall'auth_key_id e dalla msg_key come in figura 5.3.

Alla ricezione di un messaggio viene ricalcolata la msg_key con il medesimo procedimento e si controlla la corrispondenza con quella ricevuta. Inoltre si verifica che il msg_id abbia parità pari per i messaggi inviati dal client al server e dispari per quelli con percorso inverso.

5.4 Chat segrete:

Le chat segrete di Telegram sono protette dalla crittografia end-to-end. Le chiavi di decifrazione sono custodite nei due dispositivi che le creano e sono note esclusivamente ad essi, non al server che non ha quindi possibilità di accedervi e di decifrarne il contenuto.

I messaggi inviati nelle chat segrete non vengono memorizzati sul cloud e di conseguenza le conversazioni possono essere viste solo dal dispositivo di origine.

Non è permesso inoltrare i messaggi, è supportata l'autodistruzione con l'ausilio di un timer e l'eliminazione dei messaggi avviene per ambo le parti. Le chat segrete vengono perse una volta disconnessi.

Nell'immagine seguente il funzionamento della crittografia end-to-end nelle chat segrete che si può vedere come l'aggiunta di un livello di crittografia client-client a quello client-server.

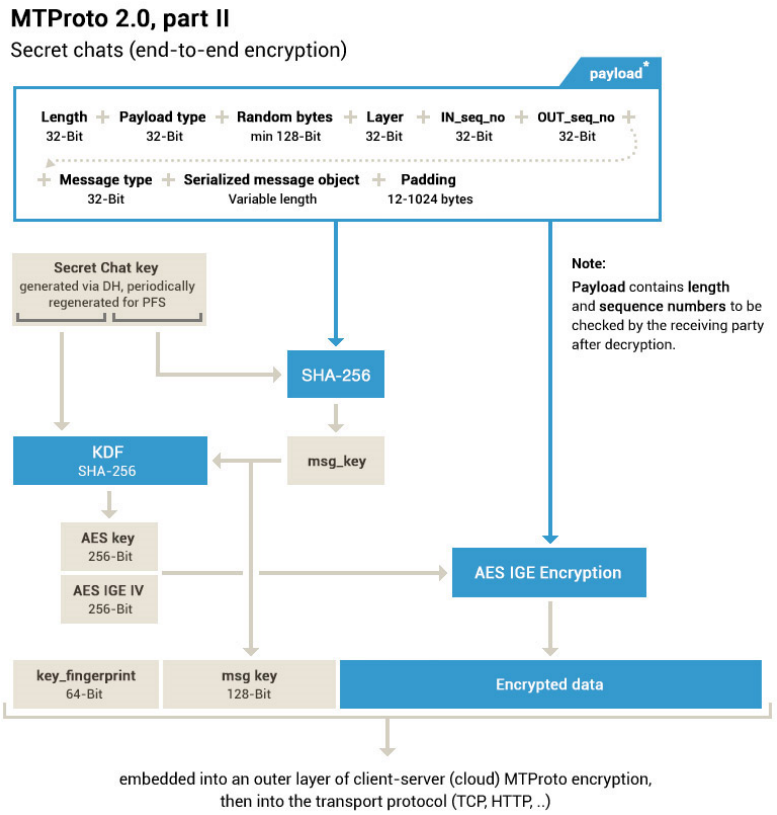

Fig. 5.4 MTProto 2.0 chat segrete

Il procedimento per la E2EE nelle chat segrete è simile a quello eseguito per le chat cloud.

Il messaggio da cifrare contiene maggiori informazioni aggiuntive su di esso e sui livelli di sicurezza utilizzati.

Quella che nella crittografia client-server era l'auth_key viene sostituita dalla Secret Chat key, una chiave generata con l'utilizzo del protocollo Diffie-Hellman e periodicamente rigenerata per il Perfect Forward Secrecy (PFS).

Il PFS riguarda i client ufficiali di Telegram e consiste nella rigenerazione della chiave una volta che è stata usata per cifrare o decifrare più di 100 messaggi oppure usata per più di una settimana, a condizione che sia stata utilizzata per almeno un messaggio.

La msg_key si ottiene, in modo analogo alle chat cloud, prendendo i 128 bit centrali di SHA-256(32 byte meno significativi Secret Chat key + messaggio da crittografare).

AES key e il vettore AES IGE IV si calcolano come visto in precedenza utilizzando la Secret Chat key al posto di auth_key.

Partendo da questi viene cifrato il messaggio, sempre con AES-256 in modalità IGE.

Infine si prendono come key_fingerprint i 64 bit meno significativi del digest restituito da SHA-1 applicato alla Secret Chat key.

Il messaggio cifrato, preceduto in ordine dalla key_fingerprint e dalla msg_key, subisce un processo di cifratura client-server prima di essere affidato al protocollo di trasporto per l'invio.

Come nelle chat cloud, alla ricezione di un messaggio c'è una fase di verifica della corrispondenza fra la msg_key ricevuta e quella che viene ricalcolata.

6. Implementazione su Signal

Signal utilizza l'omonimo protocollo crittografico sviluppato da Open Whisper Systems per offrire un servizio di crittografia end-to-end ai propri utenti.

Il protocollo Signal consente di avere ulteriori vantaggi, oltre a quelli caratteristici della E2EE, quali future secrecy e forward secrecy.

Future secrecy (o backward secrecy) è una proprietà importante in quanto garantisce che l'intercettazione di una chiave, sia a lungo termine che usata per cifrare un singolo messaggio, non comprometta le future chiavi da essa ricavate e quindi la segretezza delle successive comunicazioni.

Forward secrecy consiste nello stesso concetto applicato al passato: l'intercettazione di una chiave non intacca la segretezza dei messaggi scambiati in precedenza.

Queste due proprietà sono possibili grazie all'algoritmo Double Ratchet, approfondito nel paragrafo 6.5, usato in combinazione con Extended Triple Diffie-Hellman (X3DH).

Il protocollo Signal può essere suddiviso in 3 fasi: registrazione, configurazione della sessione e conversazione.

6.1 Registrazione:

La registrazione prevede uno scambio di informazioni fra il nuovo utente ed il server. Per registrarsi occorre inviare al server una serie di chiavi pubbliche Diffie-Hellman:

- un'identity key "ik" a lungo termine,
- una signed pre-key "spk" a medio termine firmata con la propria identity key privata,
- una lista di one-time pre-keys, chiavi effimere a singolo utilizzo denotate come "opk".

La propria identity key viene inviata solamente una volta mentre le signed pre-key devono essere rinnovate ogni settimana o mese. Nuove one-time pre-keys vengono richieste dal server quando sono quasi terminate quelle disponibili.

Ogni chiave è parte di una coppia con la rispettiva chiave privata, che rimane conosciuta solo dall'utente.

Le chiavi pubbliche corrispondenti sono inviate e memorizzate sul server che provvederà a fornire le informazioni necessarie agli utenti che vorranno contattarlo.

6.2 Configurazione della sessione

Illustriamo uno scenario comunicativo in cui Alice, una volta registratasi, voglia contattare Bob.

Alice richiede al server le chiavi pubbliche di Bob e le combina con le proprie chiavi private generando una shared secret key "ssk".

Questa viene utilizzata per cifrare il messaggio da inviare a Bob.

Alla ricezione del messaggio, Bob recupera dal server le chiavi pubbliche di Alice e calcola la stessa ssk basandosi su queste ultime e sulle sue chiavi private.

La generazione di una shared secret key condivisa avviene utilizzando una versione del protocollo X3DH come spiegato nel paragrafo seguente.

6.3 Extended Triple Diffie-Hellman (X3DH)

Il protocollo Extended Triple Diffie-Hellman è stato sviluppato da Open Whisper Systems per garantire lo scambio delle chiavi anche quando una delle parti è offline.

Vediamo nel dettaglio cosa succede quando Alice vuole mettersi in contatto con Bob.

Alice richiede al server i dati necessari per poterlo contattare e riceve in risposta un set di chiavi pubbliche di Bob, detto "pre-key bundle", contenente la sua identity key ik_B, la sua signed pre-key spk_B ed una one-time pre-key opk_B.

Una volta inviato il pre-key bundle, il server cancella la opk che gli aveva assegnato.

Alla ricezione del set di chiavi, Alice verifica la firma della spk_B e genera una ephemeral key "ek_A".

Successivamente calcola la shared secret key facendo uso della Key Derivation Function (KDF) nel seguente modo:

$$k1=DH(ik_A,spk_B),\ k2=DH(ek_A,ik_B),$$

$$k3=DH(ek_A,spk_B),\ k4=DH(ek_A,opk_B),$$

$$ssk=KDF(k1||k2||k3||k4),$$

dove DH(x,y) è una funzione ECDH che calcola una shared secret key basata sulle due chiavi passate come parametri.

Alice invia il messaggio iniziale composto dalla sua ik_A, dalla ek_A, da un identificativo ID_{opkB} che indica la opk_B usata e dalla ik_A concatenata con ik_B e insieme cifrate con ssk.

Alla ricezione del messaggio, Bob calcola allo stesso modo la ssk e decifra il contenuto. Ultimati i controlli, Bob cancella la coppia della opk_B usata e i due utenti possono continuare a scambiarsi messaggi.

6.4 Conversazione:

Una volta superato lo scambio preliminare delle chiavi inizia la fase di conversazione. Questa fase si appoggia sull'algoritmo Double Ratchet spiegato nel paragrafo seguente.

6.5 Double Ratchet:

Il termine "ratchet" può essere tradotto come cricchetto meccanico e va ad indicare un meccanismo che si muove a scatti esclusivamente in avanti. Generalmente fa riferimento ad applicazioni iterative di funzioni one-way.

L'algoritmo Double Ratchet viene utilizzato da Signal in combinazione con X3DH, dove X3DH ha la funzione di far cominciare la comunicazione mentre Double Ratchet riguarda le conversazioni in corso.

Anche Double Ratchet fa uso di una Key Derivation Function ma usa l'output di una KDF come input di una KDF successiva, creando così una catena di KDF.

L'applicazione di una catena di KDF conferisce al protocollo importanti proprietà quali forward secrecy e quella di far sembrare l'output finale una sequenza randomica.

Double Ratchet si suddivide in Diffie-Hellman ratchet e symmetric ratchet, combinati per raggiungere i vantaggi sopraelencati.

Le chiavi in output del DH ratchet sono passate in input al symmetric ratchet (fig. 6.5.1), utilizzato per cifrare i messaggi.

La figura 6.5.2 mostra quello che avviene all'interno del Diffie-Hellman ratchet.

Alice e Bob generano una coppia di ratchet key. Quando Bob vuole scrivere ad Alice le invia una copia della sua ratchet key pubblica pk^1_B. Partendo da quest'ultima e dalla sua ratchet key privata, Alice genera una nuova ssk.

A questo punto Alice invia a Bob la sua ratchet key pubblica in modo che Bob possa calcolare la stessa ssk con il medesimo procedimento e mantenere la consistenza della secret shared key.

Quando Bob vuole inviare un nuovo messaggio, genera una nuova coppia di ratchet key e la utilizza per calcolare una nuova ssk insieme alla ratchet key pubblica di Alice precedente. Successivamente invia la nuova pk^2_B ad Alice che a sua volta calcola la ssk aggiornata.

Il medesimo procedimento, dipendentemente da chi è il mittente, si ripete ad ogni messaggio scambiato all'interno della chat.

Il solo Diffie-Hellman ratchet non offre tutte le garanzie cercate a livello di sicurezza. Infatti esso è vulnerabile ad attacchi di tipo man-in-the-middle (MITM), attacchi dove un intruso può riuscire a leggere, modificare o inviare messaggi piazzandosi "in mezzo" ad una comunicazione.

Questo perché le coppie di ratchet key generate durante gli scambi di messaggi non sono legate alle identità dei due utenti.

Per porre rimedio a questo problema è necessario che la shared secret key sia legata all'output dell'algoritmo X3DH.

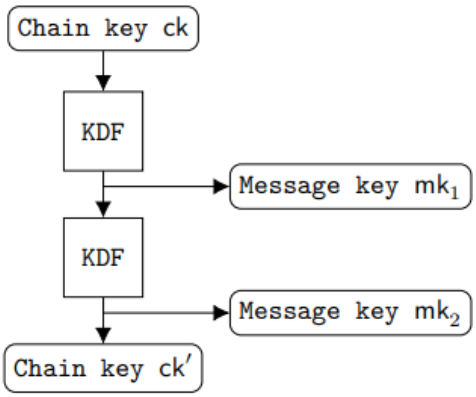

Fig. 6.5.1 Catena di KDF

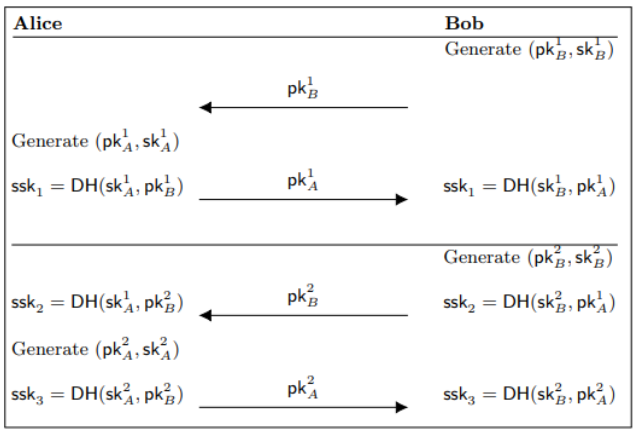

Fig. 6.5.2 Diffie-Hellman ratchet

Le chiavi in output del DH ratchet, più precisamente la receiving chain key ck_r e la sending chain key ck_s, vengono passate in input al symmetric ratchet andando a creare uno schema come quello illustrato nella figura 6.5.3. In questo modo, la catena di KDF garantisce in aggiunta un' importante proprietà ricercata: la future secrecy.

Le sending chain keys vengono usate per derivare le chiavi per cifrare il messaggio, le receiving chain keys invece permettono di ricavare le chiavi per la decifrazione dei messaggi ricevuti.

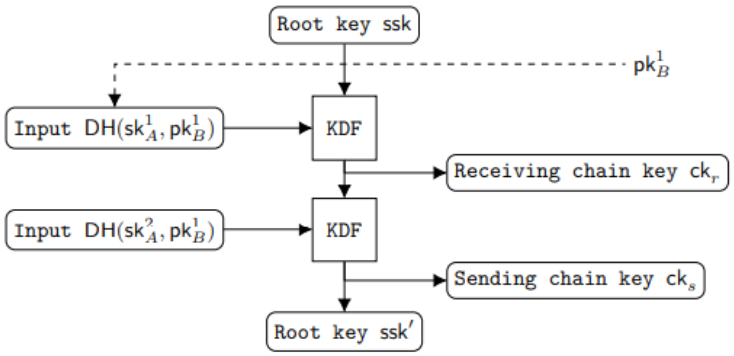

Fig. 6.5.3 Funzionamento symmetric ratchet

Analizziamo l'algoritmo Double Ratchet dal punto di vista di Alice, rappresentato in figura 6.5.4.

Inizialmente riceve la pk^1_B di Bob e la usa per ricavare una nuova ssk che diventa la receiving chain key ck^1_r. Bob invia tre messaggi ed Alice per decifrarli fa fare tre step in avanti alla receiving message key calcolando così mk^1_r, mk^2_r e mk^3_r.

Successivamente Alice vuole inviare due messaggi a Bob, quindi genera una nuova ratchet key (pk^2_A, sk^2_A) e invia a Bob pk^2_A.

Calcola una nuova ssk che corrisponde alla sending chain key ck^1_S e fa fare due step in avanti alla sending message key calcolando così mk^1_S, mk^2_S.

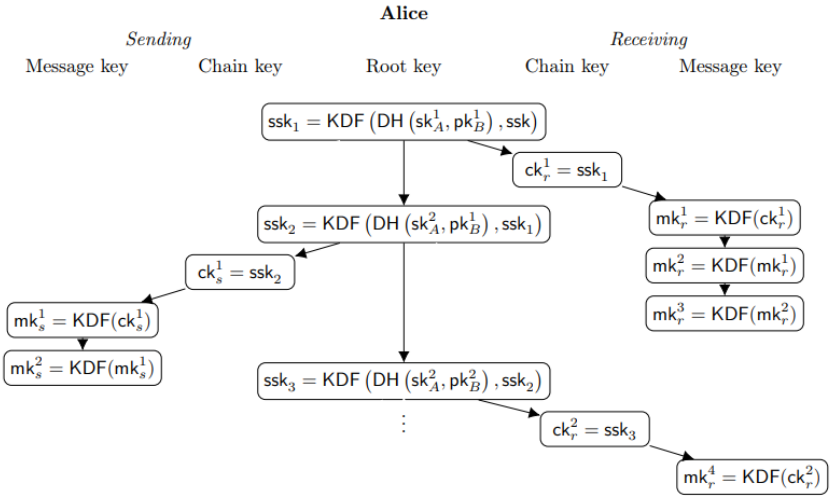

Fig. 6.5.4 Double Ratchet di Alice

Per comprendere meglio i meccanismi alla base dell'algoritmo Double Ratchet possiamo immaginare che Alice e Bob abbiano tre ratchet ciascuno dedicati alla loro conversazione: un Diffie-Hellman ratchet, un sending ratchet ed un receiving ratchet.

Tutti i ratchet, in seguito allo scambio Diffie-Hellman preventivo, iniziano dallo stesso punto.

Quando Alice vuole inviare un messaggio fa fare uno scatto in avanti al suo sending ratchet, ottenendo in output una message key A_1. Bob fa scattare il suo receiving ratchet ottenendo la stessa A_1 con la quale decifra il messaggio ricevuto.

Per inviare un nuovo messaggio, Alice fa fare un nuovo scatto al sending ratchet ottenendo così A_2. La medesima chiave la ottiene Bob una volta ricevuto il messaggio e fatto fare uno scatto al proprio receiving ratchet, potendo così decifrare il contenuto.

Quando Bob vuole inviare un messaggio fa fare uno scatto al suo sending ratchet ottenendo la message key B_1. Alice per decifrarlo usa l'equivalente chiave B_1 ottenuta dal suo receiving ratchet.

Attraverso questa tecnica, il sending ratchet di Alice è sempre sincronizzato con il receiving ratchet di Bob ed allo stesso modo il sending ratchet di Bob con il receiving ratchet di Alice.

A questi concetti va aggiunto un meccanismo che garantisca future secrecy. Infatti se venisse intercettata A_2, ad esempio, la conoscenza della message key consegnerebbe a chi l'ha carpita la possibilità di venire a conoscenza dei messaggi futuri inviati dallo stesso ratchet.

La future secrecy viene garantita sfruttando il Diffie-Hellman ratchet.

Ad ogni messaggio il mittente invia una nuova chiave pubblica DH che fa effettuare uno scatto in avanti al proprio DH ratchet e a quello del ricevente. Questo aggiornamento della chiave DH porta ad un azzeramento degli altri due ratchet.

Effettuare il rinnovamento della chiave pubblica Diffie-Hellman ad ogni messaggio porta, insieme ai meccanismi descritti in precedenza, ad avere sia forward secrecy che future secrecy.

7. Conclusioni

I concetti di sicurezza e di privacy sono costantemente sotto i riflettori in una società dove comunicare attraverso applicazioni di messaggistica istantanea è parte integrante del quotidiano.

Generalmente gli utenti cercano di usare un'applicazione che consista in un buon compromesso fra questi due principi e le funzionalità offerte.

Sicurezza e privacy vanno però distinte: la sicurezza consiste nel proteggere i dati da accessi non autorizzati, mentre la privacy corrisponde alla salvaguardia dell'identità.

Prendiamo ora in considerazione il concetto di sicurezza e confrontiamo come viene trattato dalle applicazioni approfondite.

WhatsApp offre ai propri utenti un servizio di crittografia end-to-end basato sul protocollo Signal. Quest'ultimo è open source e viene ritenuto un protocollo estremamente valido per garantire la segretezza delle comunicazioni.

Dal 15 ottobre 2021 WhatsApp ha avviato una nuova funzionalità che permette di poter superare una delle principali critiche alla sicurezza che venivano mosse nei suoi confronti.

Un punto debole dell'applicazione infatti risiedeva nei backup. Fino a prima dell'ultimo aggiornamento, i dati venivano cifrati utilizzando una chiave posseduta da WhatsApp stesso ed inviata all'utente.

Il fatto che le chiavi di decifrazione fossero memorizzate sui server dava in sostanza la possibilità, a chi vi aveva accesso, di venire a conoscenza dei dati memorizzati.

Inoltre, essendo WhatsApp di proprietà di un'azienda con sede negli USA, in base alle leggi nazionali in materia di sorveglianza sarebbe stata obbligata a fornire le chiavi dei backup alle autorità governative in caso di richiesta formale.

A questi fattori vanno aggiunti il rischio di una possibile intercettazione della chiave di decifrazione nel momento dell'invio all'utente ed il fatto che la cifratura riguardava solo i messaggi di testo, escludendo gli allegati trasmessi.

Il nuovo aggiornamento offre la possibilità di superare le vulnerabilità sopraelencate attivando dalle impostazioni di backup la modalità "Backup crittografato end-to-end".

Una volta attivata, l'utente dovrà salvare una chiave crittografica di 64 bit o creare una password associata alla chiave. WhatsApp la memorizza in un Hardware Security Module (HSM) che svolge la funzione di una cassetta di sicurezza. In questo modo l'applicazione non conosce la chiave o la password ad essa associata ma sa solamente che esiste in un HSM.

Per questo motivo, l'inserimento errato ripetuto più volte renderà i dati inaccessibili in modo permanente.

In questo modo i dati salvati sui server cloud remoti sono indecifrabili senza il possesso della chiave di crittografia.

La sicurezza su Telegram è gestita in modo differente dalle altre applicazioni presenti sul mercato.

Supporta la crittografia end-to-end ma essa non è attiva per impostazione predefinita. Il suo utilizzo è limitato alle chat segrete, mentre le chat cloud sono protette dalla crittografia server-client.

Questa scelta è bersaglio di buona parte delle critiche mosse verso l'applicazione poiché la E2EE è considerata la base per comunicazioni sicure. Non adottarla per le chat cloud significa che le loro chiavi di crittografia sono situate nei server di Telegram e questo costituisce un fattore di rischio per la segretezza.

Per scongiurarlo, Telegram fa uso di un'infrastruttura distribuita in cui i dati vengono spezzettati fra più datacenter sparsi per il globo. Questi sono soggetti a diverse giurisdizioni, in questo modo sarebbero necessari numerosi ordini da tribunali appartenenti a differenti potestà giudiziali per ottenere dei dati. Ad oggi Telegram afferma di aver divulgato 0 byte a terzi, inclusi i governi.

I backup non riguardano i messaggi scambiati all'interno di chat segrete, i quali non vengono memorizzati e non sono quindi accessibili da dispositivi diversi da quello di origine della conversazione.

Signal utilizza l'omonimo protocollo crittografico, usato anche da WhatsApp, per offrire un servizio di E2EE ai propri utenti.

L'applicazione non conserva backup delle chat ma offre la possibilità di crearne uno locale e cifrato attraverso una passphrase decisa dall'utente.

La scelta di Signal di non effettuare backup va ad eliminare alla radice i potenziali rischi collegati.

Il concetto di privacy citato in precedenza riguarda principalmente i metadati, ovvero dati accessori. Un'applicazione garantisce un buon livello di privacy se raccoglie un numero basso di metadati e li anonimizza cifrandoli.

La gestione della privacy da parte di WhatsApp è deficitaria in quanto è l'applicazione che fra le tre raccoglie più metadati per distacco, non li crittografa e li memorizza in chiaro.

WhatsApp raccoglie informazioni sul dispositivo (identificativo, numero di telefono, dati sulle prestazioni), dati sulla pubblicità, contatti, posizione e cronologia delle posizioni, indirizzo IP, cronologia degli acquisti, informazioni sui pagamenti e data e ora di messaggi e chiamate.

Telegram invece raccoglie il nome utente, il numero di telefono, i contatti e l'identificativo dell'utente. Questi dati non vengono utilizzati per profilazione pubblicitaria ma, stando a quanto dichiarato, per il solo fine di offrire il servizio.

Similmente a WhatsApp, Telegram conserva i metadati in forma non cifrata.

Fra le applicazioni in esame, quella che fornisce un livello di privacy migliore è Signal. Gli unici dati raccolti dall'applicazione sono il numero di telefono, necessario per il funzionamento, la data di creazione dell'account e l'orario dell'ultima connessione al server.

I metadati legati alla conversazione quali data e ora di invio, mittente e destinatario sono cifrati prima di essere inviati e non vengono memorizzati.

Fra le applicazioni approfondite non ce n'è una che prevalga nettamente sulle altre in tutti gli aspetti.

I fattori da valutare sono numerosi: la sicurezza, la privacy, i rischi, le funzionalità offerte, l'esperienza dell'utente, la qualità dei servizi e la diffusione dell'applicazione.

WhatsApp basa la sicurezza delle conversazioni sul protocollo Signal, ritenuto estremamente valido, offre numerose funzionalità con un buon livello di qualità ed è molto diffusa in tutto il mondo. Tuttavia il suo codice sorgente non è open source, i backup non sono crittografati E2E di default, raccoglie numerosi dati sugli utenti e nei paesi extra-UE, dopo il più recente aggiornamento della privacy policy, questi dati potrebbero essere utilizzati da Facebook al fine di migliorare l'esperienza dell'utente.

Telegram ha un codice open source, offre la E2EE nelle chat segrete, raccoglie pochi dati sugli utenti e ha numerose funzionalità di buona qualità, alcune singolari come canali e bot. D'altro lato i backup e le chat cloud non sono crittografati end-to-end.

Signal offre un livello di sicurezza maggiore in quanto le chat sono tutte protette dalla E2EE, è l'applicazione che raccoglie meno dati sugli utenti, è open source e non memorizza i messaggi scambiati.
Di contro l'assenza di backup potrebbe essere vista come la mancanza di una comodità da parte di alcuni utenti, offre meno funzionalità rispetto alla concorrenza ed ha un bacino di utenza inferiore.

Considerando tutti gli elementi elencati ed il proprio caso d'uso, ogni utente sceglie l'applicazione, o le applicazioni, da utilizzare per i suoi scopi.

Indipendentemente da dove ricada la decisione, è importante tenere a mente che qualsiasi livello di sicurezza fornito dalle applicazioni deve essere necessariamente accompagnato, per non venire vanificato, da un comportamento responsabile da parte dell'utente. Occorre sempre tenere alta la soglia dell'attenzione sui propri dispositivi e sull'utilizzo che ne viene fatto al fine di evitare spiacevoli inconvenienti.

8. Bibliografia

E2EE su WhatsApp: https://scontent.whatsapp.net/v/t39.8562-34/122249142_469857720642275_2152527586907531259_n.pdf/WA_Security_WhitePaper.pdf?ccb=1-3&_nc_sid=2fbf2a&_nc_ohc=8yzI5v6NjWsAX-L9KfB&_nc_ht=scontent.whatsapp.net&oh=d491490cc8fcabf6edcbc625f78ed5aa&oe=60D8E499

Curve25519: https://cr.yp.to/ecdh/curve25519-20060209.pdf

Immagini e dettagli AES:
https://amslaurea.unibo.it/2641/1/tasini_andrea_tesi.pdf

https://amslaurea.unibo.it/7274/1/Chieco_Davide_tesi.pdf

HMAC: https://nazarenolatella.myblog.it/2009/10/05/hmac/

SHA-256:
https://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiygMqFgqvxAhVzhv0HHdqKBroQFjATegQIJxAE&url=https%3A%2F%2Fwww.researchgate.net%2Ffile.PostFileLoader.html%3Fid%3D534b393ad3df3e04508b45ad%26assetKey%3DAS%253A273514844622849%2540144222242926 0&usg=AOvVaw1TvK13udFN1J1ZwmfGiRWs

Telegram: https://core.telegram.org/techfaq

MTProto: https://core.telegram.org/mtproto/description

Signal: https://signal.org/docs/

Protocollo Signal:
https://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwi8p7_T9pr0AhVM-qQKHRDyA68QFnoECAMQAQ&url=https%3A%2F%2Fwww.ru.nl%2Fpublish%2Fpages%2F769526%2Fz00b_2019_thesis_dion_van_dam_2019_eerder.pdf&usg=AOvVaw0fOCKcPDtGKe9ti5R250F5

https://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiq6feM95r0AhUxMewKHU15DbMQFnoECAUQAQ&url=https%3A%2F%2Feprint.iacr.org%2F2016%2F1013.pdf&usg=AOvVaw0-YpW7gQWtJPM8vF51A287