Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Matematica

TELEGRAM: DESCRIZIONE SICUREZZA

Tesi di Laurea in Crittografia

Relatore:
Prof.
DAVIDE
ALIFFI

Presentata da:
FRANCESCO BUDA

III Sessione
Anno Accademico 2014/2015

Introduzione

Telegram, introdotto per la prima volta nel 2013, è un'applicazione di messaggistica istantanea per dispositivi mobili che, dalla data di rilascio ad oggi, vanta un incremento dei suoi utenti attivi fino a 100 milioni ed a 15 miliardi di messaggi spediti giornalmente. Esso offre due modalità di conversazione principali: chat *standard e chat* segrete . Per la prima viene utilizzata una cifratura *client-server*, in cui, una volta raggiunto il server, i messaggi rimangono *in chiaro*, così permettendo sincronizzazione tra più dispositivi e chat di gruppo. Per le chat segrete si utilizza invece una cifratura *end-to-end*, ovvero tra i singoli dispositivi, in cui lo scambio di messaggi è ancora svolto tramite il server, ma senza prevederne il salvataggio in chiaro; vi è perciò un aumento dal punto di vista della privacy, poichè la sicurezza viene drasticmente aumentata, ma, insieme, anche la perdita della possibilità di sincronizzazione multipla.
A differenza di altri servizi analoghi (quali whatsapp, facebook ecc) Telegram ha deciso di sviluppare un proprio Protocollo originale, noto come MTProto, costruito su basi deboli ma in modo tale che attacchi conosciuti non vadano a buon fine. La documentazione ufficiale riguardante il software-client è open source, cioè liberamente analizzabile in dettaglio, mentre quella riguardante il server rimane tutt'ora privata. A seguito di un attenta analisi svolta dalla "Electronics Frontier Foundation" nel febbraio del 2015, la modalità di messaggistica a chat segreta si assicura i massimi voti per la sua sicurezza.
Lo scopo di questa tesi è perciò dapprima quello di dare una panoramica sulle chat standard e i relativi protocolli, per poi analizzare dettagliatamente le chat segrete e le fondamenta matematiche riguardanti la crittografia in esse utilizzata.

Indice

Elenco delle figure

Capitolo 1

Nozioni Preliminari

1.1 Infinite Garble Extenstion (IGE)

IGE è una modalità di cifrario a blocchi avente la proprietà di propagazione degli errori infinita. IGE usa la seguente concatenazione di blocchi:

$$c_i = \mathrm{f}_K(x_i \oplus c_{i-1}) \oplus m_{i-1}$$

con f_K funzione di cifratura con chiave K per il cifrario a blocchi .
A differenza di modalità quali CBC o CTR, che utilizzano un solo vettore iniziale, IGE utilizza sia un blocco di testo in chiaro iniziale m_0 che un blocco di testo cifrato inziale c_0. Secondo la documentazione ufficiale, il testo in chiaro iniziale viene ricavato usando una chiave casuale $K_0 : c_0 = \mathrm{f}_{K_0}(m_0)$, ma secondo l'implementazione in OpenSSL può essere usato un blocco arbitrario come paramentro.
Come mostrato in figura 1.1 la catena di operazioni per la cifratura (con funzione C_K, chiave K) è data da:
$c_1 = m_0 \oplus \mathrm{C}_K(m_1 \oplus c_0)$
$c_2 = m_1 \oplus \mathrm{C}_K(m_2 \oplus c_1)$
....
$c_i = m_{i-1} \oplus \mathrm{C}_K(m_i \oplus c_{i-1})$

Mentre dalla figura 1.2 segue la decifrazione (con funzione D_K, chiave K) per ogni blocco:

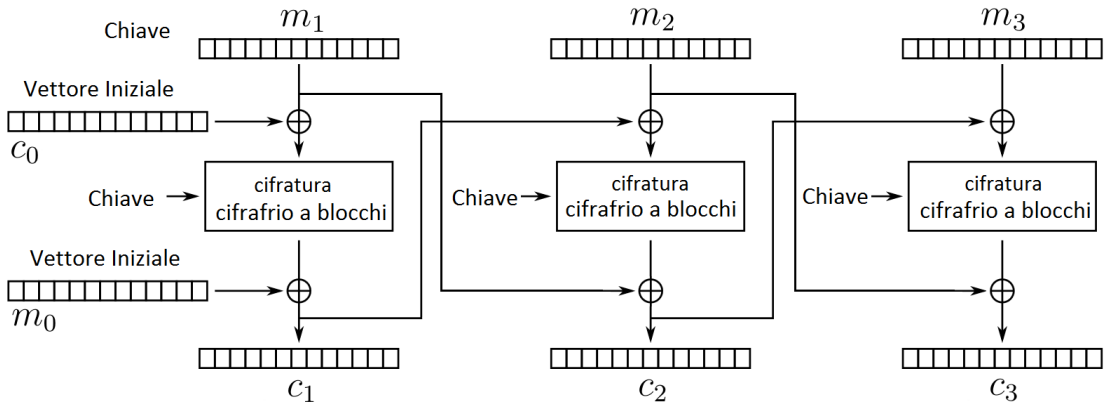

Figura 1.1: Diagramma crittografia IGE

$$m_1 = c_0 \oplus \mathrm{D}_K(c_1 \oplus m_0)$$
$$m_2 = c_1 \oplus \mathrm{D}_K(c_2 \oplus m_1)$$
....
$$m_i = c_{i-1} \oplus \mathrm{D}_K(c_i \oplus m_{i-1})$$

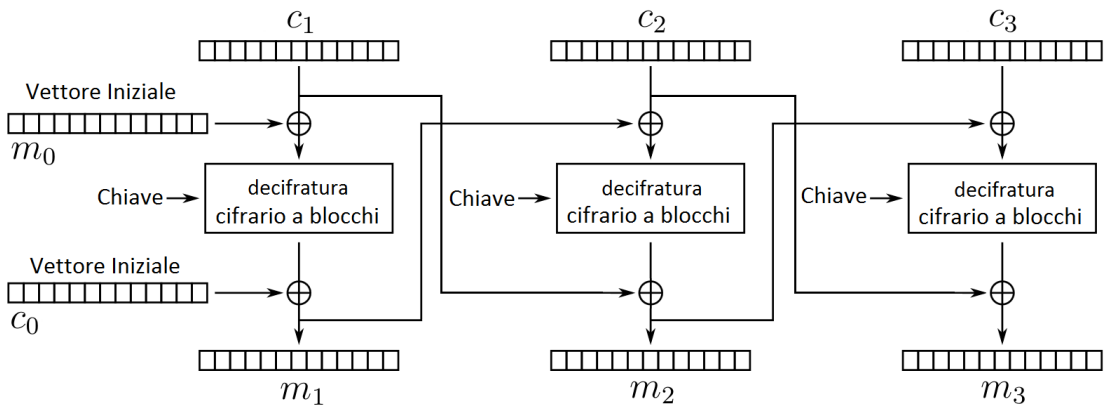

Figura 1.2: Diagramma crittografia IGE

Come accennato precendetemente, IGE ha la proprietà di propagare gli errori in avanti indefinitamente, ed infatti in figura 1.3 è mostrato come un errore in un singolo blocco renda il rispettivo blocco decriptato, così come i successivi, erronei.

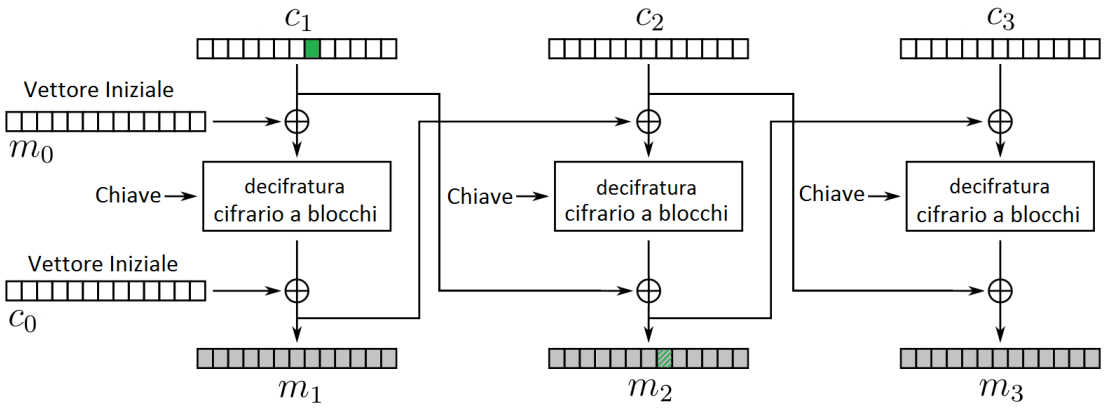

Figura 1.3: Propagazione degli errori in IGE

1.2 SHA1

SHA1 è una funzione hash crittografica, pubblicata per la prima volta nel 1995, che prendendo in input un messaggio di lunghezza variabile, inferiore a $2^{64} - 1$ bit, produce un "impronta del messaggio" di lunghezza fissa pari a 160-bit. Questa è la funzione hash più utilizzata al giorno d'oggi, anche se esistono documentazioni in cui viene provato che SHA1 sia crittograficamente penetrabile, sebbene ad alti costi.

Capitolo 2

Protocolli

In questo capitolo si andrà a vedere più nel dettaglio il protocollo MTProto usato in Telegram e il suo utilizzo per chat standard e chat segrete (differente dal primo perchè implementato con un layer aggiuntivo). Si analizzerà inizialmente l'autenticazione iniziale tra utente e sever, e come è stabilita una connessione end-to-end sicura criptata tra 2 utenti e ecc.

2.1 Registrazione Dispositivo

Agli occhi dell'utente medio la registrazione dei propri dispotivoi. viene effettuata in due semplici passi a seguito della prima installazione di Telegram. Inizialmente viene richiesto l'inserimento del proprio numero di cellulare, utilizzato per autenticazioni dell'utente per login futuri, mentre il secondo passo consiste nella richiesta di un codice a 5 cifre, trasmesso tramite SMS, volto alla verifica del numero di cellulare. A seguito di questo inserimento vi è l'inizializzazione del vero e proprio protocollo di autenticazione.

Questo protocollo si svolge come segue, per un utente $\boldsymbol{U}$ che si registra verso il server $\boldsymbol{S}$.

1. $\boldsymbol{U}$ invia una richiesta a $\boldsymbol{S}$ con una stringa casuale detta *nonce*.

2. ***S*** risponde con un altra stringa casuale *server_nonce*, un numero composto n, e un'impronta digitale di una chiave pubblica per il sistema RSA.

3. ***U*** decompone n in fattori primi p' e q in modo tale che $p < q$[1]. ***U*** inoltre ha una lista di chiavi pubbliche RSA e seleziona la chiave $server_{pK}$ che corrisponde all'impronta digitale ricevuta.

4. ***U*** sceglie una nuova stringa casuale *new_nonce* che a differenza delle precedenti non verà trasmessa in chiaro, poi crea un hash SHA-1 dei tre *nonce* ed i numeri n, p e q. L'impronta viene criptata tramite RSA con chiave $server_{pK}$ e poi inviata ad ***S***.

5. ***S*** risponde inviando i parametri g, p, g_a , necessari per un'esecuzione del protocollo Diffie-Hellman, criptati con AES-256 in modalità IGE, usando chiavi temporanee AES e AES-IV derivate da *server_nonce* e *new_nonce*.

6. ***U*** sceglie un esponente b e calcola $g_b = g^b \bmod p$, e $auth\_key = (g_a)^b \bmod p$. g_b viene poi inviato a ***S*** criptandolo con AES-256 in modalità IGE, usando le due chiavi temporanee AES e AES-IV. In questo modo U ed S completano lo scambio della chiave di DH.

Entrambi, *nonce* e *server_nonce*, sono stringhe a 128-bit e, in seguito al passo **2.**, vanno a formare la coppia (*nonce*, *server_nonce*) che sarà sempre presente nei messaggi successivi, sia in chiaro che criptata parzialmente. Questa definisce una "sessione temporanea" la cui sicurezza nei confronti di attacchi a sessioni parallele è data dalla casualità della scelta di *server_nonce*.
n é un intero composto a 64-bit, *new_nonce* è una stringa a 256-bit e l'impronta digitale della chiave pubblica RSA sono i 64-esimi bit meno significativi dell'immagine tramite SHA1 della chiave pubblica del server.

[1]La decomposizione in fattori primi di n è una "proof of work", infatti essa richiede tempo e serve ad evitare che il server sia inondato da richieste di registrazione, ad esempio per un attacco DoS

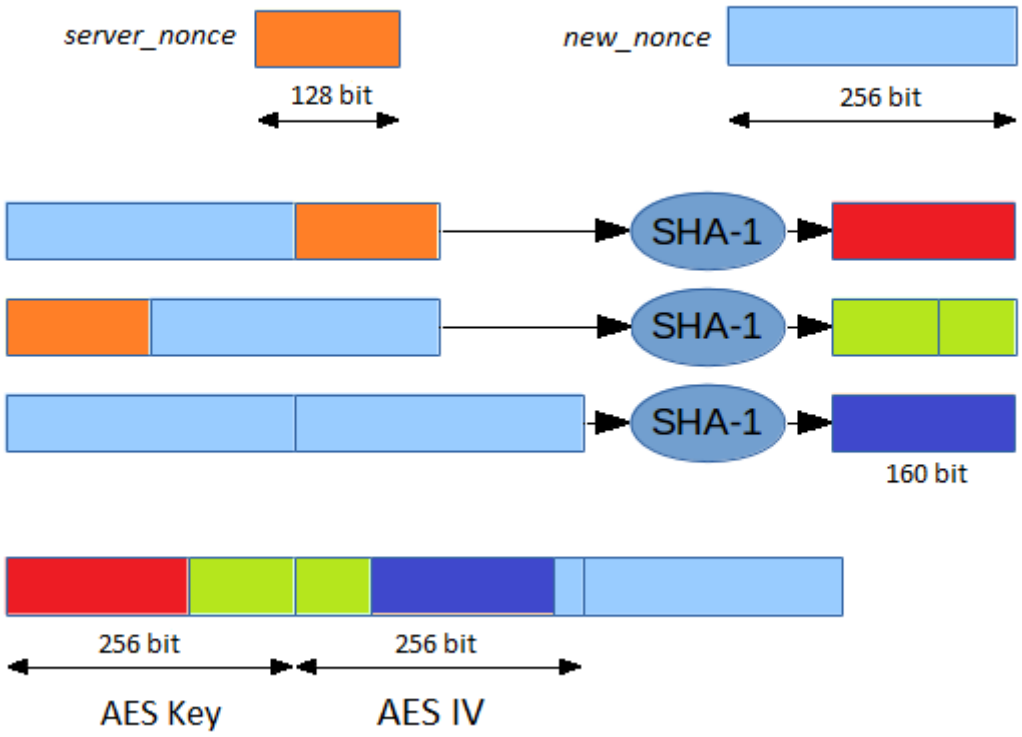

Figura 2.1: Derivazione Chiavi Temporanee

I parametri g, p, g_a non vengono inviati in chiaro, ma è richiesta la derivazione delle chiavi a 256-bit AES e AES IV (mostrata in figura 2.1), operata tramite hash multipli SHA-1, calcolate a partire da *server_nonce* e *new_nonce*, e una successiva concatenazione di stringhe [2]. Decriptando la risposta si ottengono dunque i parametri di DH,*nonce*, *server_nonce* e una stampa dell'ora del server usata per sincronizzazione[3].

Subito dopo il passo *5.* l'utente U deve verificare che p sia un primo sicuro, cioè che $q = \frac{p-1}{2}$ sia un primo e che $2^{2047} < p < 2^{2048}$; che g sia uguale a $2, 3, 4, 5, 6$ o 7

[2]Un processo analogo verrà descritto più nel dettaglio nella sezione 2.4.1

[3]g, pg_a, sebbene siano parametri di DH pubblici, vengono cifrati con RSA per garantire l'autenticità ed evitare attacchi Man-in-The-Middle

e che generi un sottogruppo ciclico di ordine primo $\frac{p-1}{2}$, e che $1 < g_a < p-1$. Se di una queste condizioni non è soddisfatta, il protoccolo viene annullato.

Si ha che, dopo il calcolo di $g_a^b mod\ p$ e $g_b^a mod\ p$ rispettivamete, l'utente e il server condividono una chiave segreta $auth\_key$ usata esclusivamente per comunicazione utente-server e chat standard e, la conseguente avvenuta registrazione e autorizzazione del dispositivo dell'utente.

2.2 Creazione di una Chiave Segreta Condivisa tra due utenti

Come accennato in precedenza, una delle caratteristiche più rilevanti di Telegram è la presenza delle chat segrete, in particolare della crittografia in esse utilizzata. Queste consistono in un collegamento tra due utenti in cui i messaggi vengono cifrati tramite una chiave segreta da essi, e solo essi, condivisa. La creazione della chiave segreta tra due utenti **A** e **B** avviene come segue:

1. **A** contatta il server ottenendo i parametri pubblici di DH (un primo p, e un intero g) e una sequenza casuale r_{server}.
   A deve verifcare che il p ricevuto sia un primo sicuro, cioè che $q = \frac{p-1}{2}$ sia un primo e che $2^{2047} < p < 2^{2048}$ e che g sia uguale a $2, 3, 4, 5, 6$ o 7 che generi un sottogruppo ciclico di ordine primo $\frac{p-1}{2}$. Inoltre, per evitare calcoli superflui, i parametri di DH vengono salvati in cache dall'utente insieme alla loro versione, e nel caso in cui una nuova chat segreta voglia essere inizializzata, se essi risultano ancora validi per il server, allora nella risposta è contenuto esclusivamente r_{server}.

2. **A** genera un valore casuale a, calcola il valore pubblico di DH $g_a = g^a \text{mod}\ p$ e lo invia a **B**.
   Per generare un valore sufficientemente casuale, nel caso il generatore di numeri casuali dell'utente sia debole, Telegram si assicura che ciò venga effettuato nel modo seguente:

$$a \ = r_{utente} \oplus r_{server}$$

ottenendo un valore a 2048-Bit.

3. **B** riceve la richiesta per la nuova chat e una volta accettato, contatta il server per ottenere i parametri di DH e per generare il proprio valore casuale b.
   La richiesta da parte di **A** viene effettuata tramite l'invio dei seguenti parametri:

   - **id** ID della chat
   - **access_hash** valore hash dipendente dall'user ID
   - **date** Data creazione della chat
   - **admin_id** ID del creatore e amministratore della chat
   - **participant_id** ID del secondo partecipante alla chat
   - **g_a** $g_a = g^a \text{mod } p$
   - **nonce** stringa di 256bytes vuota

   Se anche solo uno dei primi cinque parametri non risulta corretto, allora la richiesta si interrompe, altrimenti viene visualizzata a B, con possibilità di accettarla o negarla, insieme ad alcune informazioni base sull'utente A. Il valore b è calcolato in modo analogo ad a.

4. **B** calcola $g_b = g^b \text{mod } p$, la conseguente chiave segreta $aut\_key = g_a^b$ e la sua impronta digitale $key\_fingerprint$, inviando poi la coppia $(g_b, key\_fingerprint)$ ad **A** .
   Entrambi gli utenti sono tenuti a verificare che $1 < g_a, g_b < p - 1$ ed è anche consigliato controllare che $2^{2048-64} < g_a, g_b < p - 2^{2048-64}$. Sia g_a che g_b devono avere lunghezza pari a 2048-Bit e se ciò non è verificato ad essi vengono aggiunti degli 0 fino a lunghezza cercata ($padding$). Poichè **B** riceve g_a per primo, può calcolare $auth\_key = g_a^b$ e la relativa impronta digitale $key\_fingerprint$ che verrà poi usata come controllo di sicurezza per lo scambio della chiave. Essa consiste nie 64-Bit meno significativi del valore hash SHA-1 di $auth\_key$. Si ha inoltre la condizione che $auth\_key$ sia lunga

esattamente 256-Bit e nel caso in cui non lo sia anche ad essa vengono aggiunti degli 0.

5. **A** calcola $g_b^a = auth\_key$, verificando che la relativa impronta digitale sia uguale a $key\_fingerprint$, da cui si ottiene il successo dello scambio della chiave segreta.

Analizzando i passaggi appena descritti si evince che sebbene venga utilizzato il server di Telegram per stabilire una connessione tra 2 utenti, questo non può venire a conoscenza della chiave segreta condivisa da essi in quanto non è a conoscenza del valore a e b, rispettivamente. In più, poichè lo scambio della chiave di Diffie-Hellmann è basato sul problema del logaritmo discreto, questo assicura che nessun attacco passivo possa portare alla conoscenza di $auth\_key$.

Telegram non ha un vero e proprio protocollo di autenticazione all'interno delle chat segrete, ma per garantirla utilizza la "visualizzazione della chiave". Questa viene generata dai primi 128 Bits del valore hash SHA-1 della chiave condivisa, e viene mostrata all'utente sotto forma di una matrice 12x12, in cui ogni cella é colorata con uno tra quattro colori (come mostrato in figura 2.2).
Questa caratteristica, sebbene originale, viene considerata come una debolezza di Telegram, in quanto le due immagini (o i due testi) sono intese per essere confrontate di persona, assicurandosi che non vi siano differenze. Ciò va però a distruggere lo scopo di una chat, cioè quello di comunicare a distanza, per cui si ottiene che la maggior parte degli utenti confronta le chiavi semplicemente inviando i rispettivi screenshot nella nuova chat non ancora autenticata. Questo si è dimostrato debole ad attacchi del tipo $man - in - the - middle$, in quanto un avversario che abbia completato un protocollo DH separatamente con ciascuno degli utenti, può facilmente ingannare gli utenti inviando screenshot da esso creati; in ogni se gli utenti seguono il protocollo nella forma prevista, allora la chat da loro creata può essere considerata autenticata e sicura.

Figura 2.2: Visualizzazione Chiave Segreta su iOS.

Capitolo 3

Cifratura Messaggi

Telegram è conosciuto come uno dei pochi servizi che utilizza un proprio schema crittografico "MTProto", ideato per essere computazionalmente veloce su dispositivi mobili e con sicurezza massima. Vediamo ora come avviene la cifratura all'interno di una chat segreta. I messaggi non vengono cifrati direttamente a partire dal testo in chiaro, ma vengono prima inseriti in "pacchetti" così costruiti:

1. Creazione del pacchetto $DecryptedMessage$ contenente il testo in chiaro e parametri aggiuntivi, così costruito:

· **random_id:**	ID casuale del messaggio, generato dal mittente.
· **random_bytes:**	Insieme di bytes casuali.
· **ttl:**	Tempo di vita del messaggio deciso dal mittente.
· **message:**	Testo del messaggio.
· **media:**	Tipologia di file inserito (audio,video ecc.).

2. Per compatibilità all'indietro[1] vi è l'inserimento del pacchetto precedente all'interno di $decryptedMessageLayer$, secondo i seguenti passi:

[1]Se ne discuterà in dettaglio nella sezione 3.4.

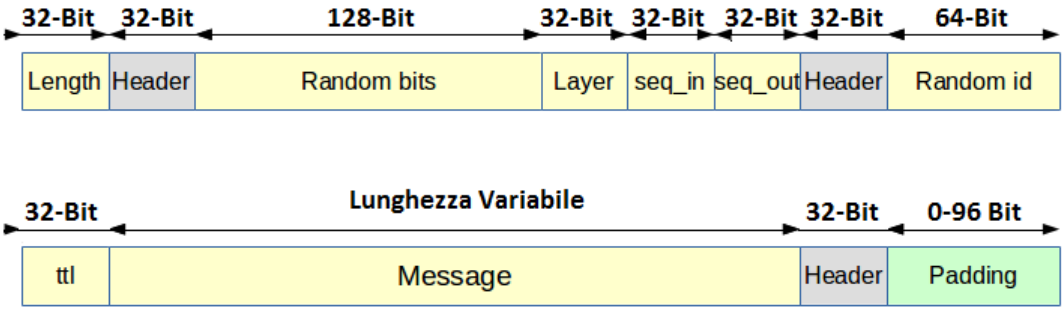

Figura 3.1: Contenuto del Payload in attesa di essere cifrato.

· **random_bytes:**	Insieme di bytes casuali.
· **layer:**	Versione del protocollo utilizzato.
· **seq_in:**	Numero messaggi ricevuti dall'admin.
· **seq_out**	Numero messaggi inviati dall'admin.
· **message:**	"pacchetto" $DecryptedMessage$.

All'inzio di esso viene poi aggiunto un "pacchetto" *Length*, contenente l'effettiva lunghezza di $decryptedMessageLayer$, ottenendo dunque l'insieme di dati detto *Payload*. In figura 3.1 viene mostrato come si presenta il *Payload*, con l'aggiunta di un padding, subito prima della cifratura, di cui:

• **Header:** All'interno del *Payload* vi sono tre intestazioni dipendenti dalla versione del protocollo utilizzato, di cui l'ultima indicatrice del tipo di file inserito.

• **Random bits:** 120-Bit di questo blocco rappresentano i bytes casuali visti in precedenza. Questi vengono inseriti per evitare il riconoscimento di messaggi cifrati brevi. Gli 8-Bit mancanti esprimono la lunghezza di *Randombits* in bytes.

• **Padding:** Viene aggiunto un padding per ottenere un insieme di dati di lunghezza divisibile per 16 bytes (128-bits, lunghezza di un blocco AES).
Il resto dei parametri è definito come visto in precedenza.

In figura 3.2 viene poi mostrato in via schematica l'utilizzo di MTProto per cifrare il *payload* in vista di essere inviato nella chat, di cui:

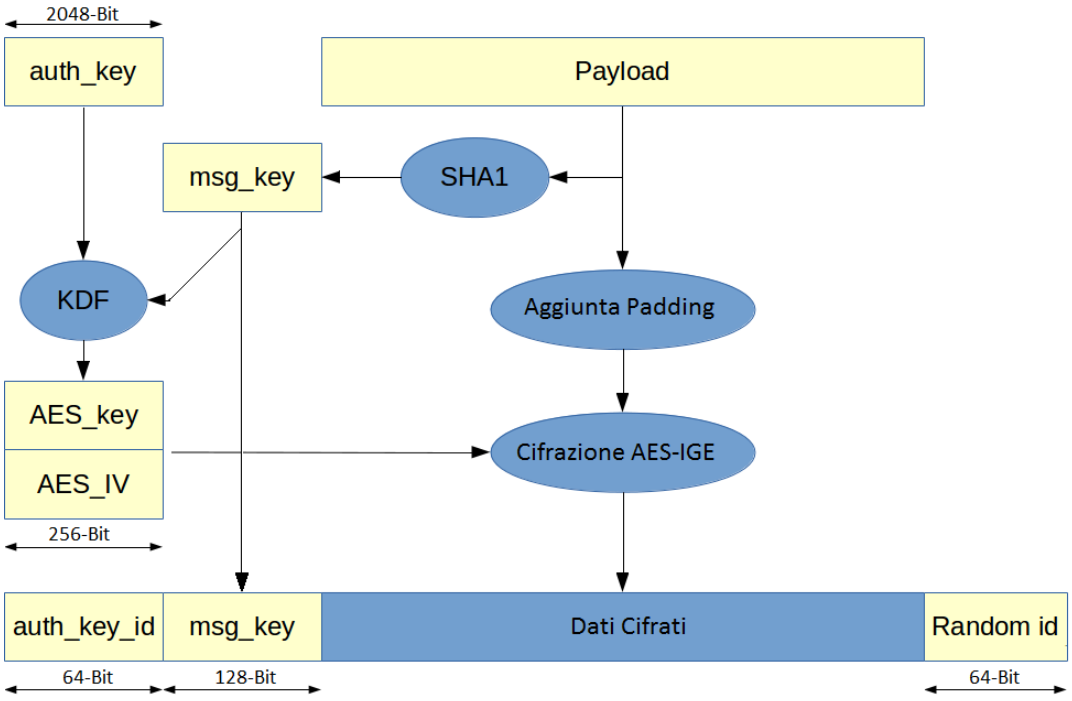

Figura 3.2: Schema Cifrazione Chat Segrete in MTProto

• **auth_key:** Chiave segreta condivisa ottenuta secondo il protocollo visto in sezione 2.2.

• **Payload:** "Pacchetto" contenente il messaggio, come visto in figura 3.1.

• **msg_key:** 128 Bits meno significativi dell'hash SHA-1 calcolata sul contenuto da cifrare (senza padding).

• **AES_key, AES_IV:** Chiave e vettore iniziale a 256-Bit necessari per AES in modalità IGE (ottenute tramite KDF, come vedremo nella sottosezione 2.4.1).

• **auth_key_id:** 64-Bits meno significativi dell'hash SHA-1 calcolata su *auth_key*.

3.0.1 KDF: Funzione di derivazione delle chiavi temporanee

Come anticipato, MTProto utilizza AES in modalità IGE con chiavi a 256 bit, il che necessita di un vettore iniziale. Si ha che per ogni messaggio avviene la creazione di un nuovo vettore iniziale e chiave AES, derivanti da *auth_key* e dal messaggio stesso. Questo per assicurare che solo un utente in possesso di *auth_key* possa decifrare i messaggi cifrati a partire da essa. La chiave e il vettore iniziale vengono calcolati tramite molteplici funzioni hash SHA-1, questo poichè gli ouput di SHA-1 sono sempre lunghi 160-Bit mentre la lunghezza cercata è di 512-Bit. Ad essi vengono dati come input 256-Bit di *auth_key* e i 128-Bit di *msg_key*. In figura 3.3 viene schematizzato l'intero processo, e si può notare come solo i 1028-Bit più significativi di *auth_key* vengano utilizzati e come la posizione di *msg_key* venga cambiata ogni volta per aggiungere una maggiore casualità al risultato. La chiave *AES_key* e il vettore iniziale *AES IV* vengono dunque "assemblati" combinando sotto-stringhe di ognuno dei quattro output derivati da SHA-1 fino ad ottenere stringhe a 256-Bit . Secondo la documentazione ufficiale di Telegram, ogni sottostringa viene calcolata tramite una funzione $substr(input, skip, length)$ in cui *skip* indica il numero di bytes da saltare e *lenght* il numero di bytes da leggere da *input*, ottenendo la sequente catena di calcoli fino a AES_KEY e $AES\ IV$.

$$
\begin{aligned}
sha1\_a &= \text{SHA1}\ (msg\_key + substr(auth\_key, 0, 32)\\
sha1\_b &= \text{SHA1}\ (substr(auth\_key, 32, 16) + msg\_key + substr(auth\_key, 48, 16))\\
sha1\_c &= \text{SHA1}\ (substr(auth\_key, 64, 32) + msg\_key)\\
sha1\_d &= \text{SHA1}\ (msg\_key + substr(auth\_key, 96, 32))\\
\text{AES\_KEY} &= substr(sha1\_a, 0, 8) + substr(sha1\_b, 8, 12) + substr(sha1\_c, 4, 12)\\
\text{AES\_IV} &= substr(sha1\_a, 8, 12) + substr(sha1\_b, 8, 12) + substr(sha\_c, 16, 12)\\
&\quad + substr(sha1\_d, 0, 8)
\end{aligned}
$$

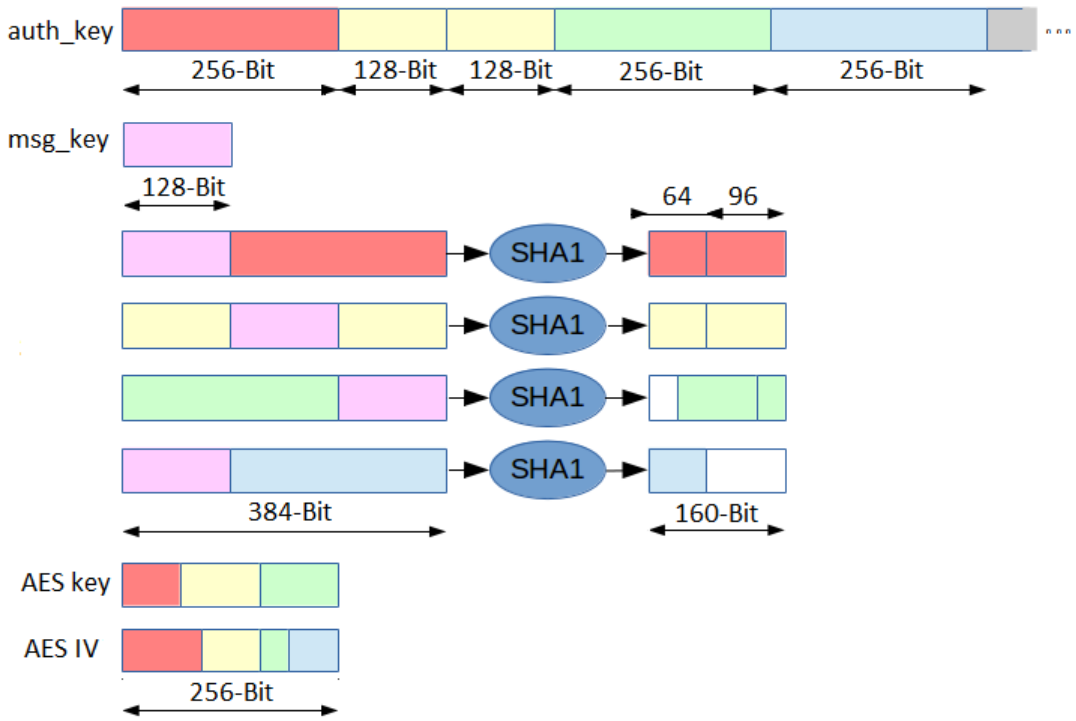

Figura 3.3: Funzione di derivazione delle chiavi temporanee usata in MTProto

3.1 Sequenza di Numeri nelle Chat Segrete

In crittografia si ha la regola generale di non sottovalutare un avversario, ma di considerare che egli abbia capacità di calcolo illimitata, poichè polinomiale. Nel caso di Telegram non è perciò sufficiente affermare che lo scambio della chiave segreta condivisa sia assolutamente sicuro, ma bisogna accertarsi anche della sicurezza verso altre tipologie di attacco, ad esempio di tipo *replay*, *mirroring* o simili.

- **Replay-Attack:** Attacco in cui una tramissione di dati viene maliziosamente ripetuta o ritardata da parte di un avversario che ha precedentemente intercettato i dati. Nel caso di una chat, è l'attacco secondo cui un messaggio viene registrato e re-inserito all'interno della conversazione senza che l'autore originale del messaggio ne sia a conoscenza.
- **Mirroring-Attack:** Attacco secondo il quale un avversario registra un messaggio inviato da un utente e glielo re-invia instantaneamente. In questo

modo il mittente del messaggio è portato a credere che il secondo utente della chat abbia semplicemente risposto con un messaggio identico.

Nel primo caso una contromisura efficace può essere quella di inserire un contatore che tiene traccia dell'ordine in cui i messaggi vengono visualizzati, ma ciò non è sufficiente per la seconda tipologia di attacco poichè il messaggio fraudolento verrebe considerato dal contatore come un messaggio autentico. É per questo che in MTProto vengono utilizzati due contatori, *seq_in* e *seg_out*. Nella loro forma standard vengono inizializzati con $(seq\_in, seq\_out) := (0, 0)$ e incrementati di 1 ogni qualvolta un messaggio (di servizio o conversazione) viene inviato/ricevuto agli occhi dell'admin della chat. Considerando due utenti **A** e **B** (con **A** admin della chat) ,i contatori vengono poi trasformati secondo la formula:

$$2 * seq\_counter + x$$

Dove x è calcolato seguendo la regola:

	seq_in	**seq_out**
Inviato da **A**	0	1
Inviato da **B**	1	0

I contatori trasformati vengono inseriti in ogni messaggio, mentre quelli standard vengono mantenuti dai singoli utenti. Si ha quindi che il bit meno significativo di ogni *seq_counter* è diverso per messaggi ricevuti o inviati, da cui si ottiene la protezione contro Mirror-Attack.

3.1.1 Controlli di sicurezza

Controllo di Seq_out

Ogni client deve verificare che ha ricevuto messaggi contenenti il *seq_out* associato, a partire da 0 fino a un valore **C**. Naturalmente il messaggio seguente è atteso con valore di $seq\_out = \mathbf{C} + 1$. Se ciò non accade viene attuata una delle seguenti opzioni:

· **seq_out** $\leq \mathbf{C}$ Il client deve allora trascurare il messaggio senza verificarne il contenuto.

· **seq_out** $> \mathbf{C} + 1$ Ciò può significare che qualche messaggio è stato perso dal server a causa di fallimenti tecnici, o dovuto al messaggio diventato obsoleto. Una soluzione rapida è quella di terminare la chat ma vengono prima effettuati ulteriori controlli.

Per da gestire in modo corretto i messaggi in arrivo dopo che un salto è stata individuata la maancanza di uno o più messaggi(**seq_out** $> \mathbf{C} + 1$), essi vengono inseriti in una lista "d'attesa" sul client locale, effettuando poi una richiesta al client remoto tramite il comando *decryptedMessageActionResend* contenente i seguenti parametri:

start_seq :	*seq_out* del primo messaggio da re-inviare.
end_seq :	*seq_out* dell'ultimo messaggio da re-inviare.

I rispettivi *seq_out* sono facilmente calcolabili sommando 2 al *seq_out* dell'ultimo messaggio prima del salto e sottaendo 2 a quello del messaggio ricevuto con sequenza errata.
Si ha che nella stragrande maggioranza dei casi i "salti" vengono gestiti come sopra descritto, senza perciò attuare modifiche alla chat. Vi sono però altri casi che vengono gestiti in maniera più drastica:

· Il verificarsi di due "salti" contemporanei è molto raro (assumendo che il server e il client funzionino normalmente) perciò vi è la conseguente terminazione della chat.
· Se un client ricevente la richiesta *decryptedMessageActionResend* è impossibilitato a soddisfarla, allora vi è la terminazione della chat.

Controllo di Seq_in

Ogni client deve verificare che *seq_in* dei messaggi ricevuti sia *valido*. Ciò è effettuato secondo i criteri:

· **seq_in** deve formare una sequenza non decrescente di interi non negativi.

· **seq_in** deve essere valido al momento della ricezione del messagio, cioè prendendo per esempio **D** come il *seq_out* dell'ultimo messaggio da noi inviato, allora il *seq_in* ricevuto non deve essere maggiore di **D** + 1. Questo non solo assicura sicurezza, ma permette alla chat di inserire i messaggi nel corretto ordine di invio-ricezione. Per esempio, nel caso in cui sia appena avvenuta la creazine di una chat segreta, se un utente dopo aver inviato 5 messaggi ricevesse un messaggio con $seq\_in = 2$ allora questo sarebbe forzatamente collocato subito dopo il secondo messaggio inviato. Questo permette la manipolazione e la correzione di messaggi *ritardati*.

Se *seq_in* non soddisfa questi criteri, il cliet termina la chat immediatamente. Ciò potrebbe accadere in caso di comportamenti maligni da parte del server/client dal lato del client remoto.

3.2 Decifratura Messaggi

La decifrazione dei messaggi usata in MTProto risulta talmente simile al processo di cifratura che nella documentazione ufficiale di Telegram essa viene presentata semplicemente come una sua inversione, con l'aggiunta di un solo controllo di sicurezza. Nello schema in figura 3.4 infatti è mostrato solamente come venga controllato che msg_key, derivante dal messaggio decifrato, sia uguale a msg_key' ricevuto insieme al messaggio cifrato. Tuttavia ciò è incompleto in quanto i controlli operati sono molteplici, come vedremo ora in dettaglio. L'algoritmo 1 prende come input $auth\_key, auth\_key\_id', msg\_key'$ e $encrypted\_data$ (questi sono proprio i parametri contenuti nel messaggio ricevuto), e utilizza funzioni quali SHA1($input$), $substr(input, skip, lenght)$,KDF($input, input'$), computeAuthKeyID($input$) che calcola l'ID dell'input dato e readLenght($input$) che calcola la lunghezza dell'input dato.

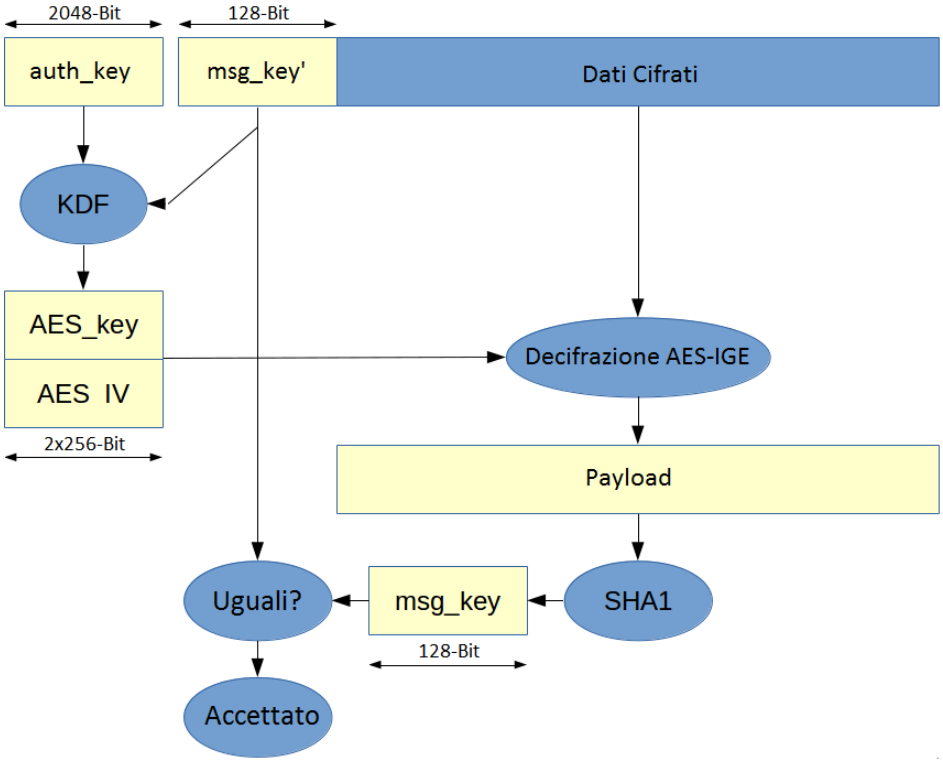

Figura 3.4: Schema Decifratura Chat Segrete in MTProto

Algoritmo 1 Algoritmo di decifrazione in MTProto

Input: $auth\_key, auth\_key\_id', msg\_key'$;

1: $auth\_key\_id \leftarrow$ computeAuthKeyId($auth\_key$);
2: **if** $auth\_key\_id \neq auth\_key\_id'$:;
3: discard;
4: AES_key,AES_$iv \leftarrow$ KDF($auth\_key, msg\_key'$);
5: $payload \leftarrow$ AES_IGE_$Decrypt$($encrypted\_data$,AES_key,AES_iv);
6: $lenght \leftarrow$ readLenght($payload$);
7: **if** $lenght$ out of bounds:;
8: discard;
9: $payload \leftarrow$ removePadding($payload$);
10: $msg\_key \leftarrow$ computeMsgKey($payload$);
11: **if** $msg\_key \neq msg\_key'$:;
12: discard;
13: $seq\_in, seq\_out \leftarrow$ computeCounters($payload$);
14: **if** $seq\_in, seq\_out \neq$ local seq_in, seq_out:;
15: discard;
16: return $payload$;

Un particolare importante che viene omesso in 3.4 è quello della rimozione del padding (passo 9, algoritmo 1) prima del calcolo tramite SHA1 di msg_key. Questo è per assicurarsi che msg_key venga calcolata a partire dal testo del messaggio e non da dati aggiuntivi casuali. Vediamo anche come venga controllata la lunghezza del *payload* (passo 7, algoritmo 1) in modo tale che non ecceda la lunghezza del testo in chiaro se non di 15 bytes (che corrisponde esattamente alla massima lunghezza di $Random\_Bits$).
Si nota dunque come ogni controllo, se non soddisfatto, porta allo scarto del messaggio, senza alcuna visualizzazione d'errore sul dispositivo

3.3 Segretezza in avanti (Perfect Forward Secrecy)

La *Segretezzainavanti* di un sistema crittografico è la proprietà che assicura che una chiave di sessione ottenuta da un set di chiavi a lungo termine non venga compromessa, anche nel caso in cui una delle chiavi a lungo termine sia compromessa. La compromissione di una chiave infatti permette la decifrazione esclusivamente del testo cifrato con la chiave stessa, e non del testo completo.
In Telegram questo equivarrebbe al caso in cui la chiave $auth\_key$ venga compromessa, portando cosi alla possibilità di accedere, da parte di un avversario, a tutti i messaggi della chat. Questo viene evitato introducendo un contatore $key\_counter$ che tiene traccia del numero di volte che un $auth\_key$ viene utilizzato per derivare la chiave temporanea AES$_key$ necessaria per la cifratura o decifrazione. Si ha che ogni qualvolta il contatore raggiunge quota 100, o la chiave è in uso da più di una settimana, i clients eseguono il protocollo di creazione di una nuova chiave condivisa. Le vecchie chiavi vengono poi distrutte e non possono più essere ricostruite in alcun modo. Lo scambio della nuova chiave avviene in modo analogo alla prima $auth\_key$ se non per il fatto che i paramentri di DH (p, g) non vengono ritrasmessi, ma riutilizzati dallo scambio precedente, e la creazione del numero casuale a non è più "assistita", cioè non vi è più l'utilizzo del numero casuale creato dal server. Una volta poi che un utente ha calcolato i propri parametri pubblici di DH, essi vengono

inviati all'utente B tramite il canale sicuro, stabilito dalla chat segreta, già in uso. In questo protocollo, sebbene sia operato un continuo cambiamento della chiave segreta, non vengono applicate differenze nè all'impronta digitale $key\_fingerprints$ nè alla visualizzazione della chiave, quindi l'autenticazione da parte di ogni utente è operata sempre in egual modo. Una volta scartata la chiave obsoleta, i messaggi precedenti rimangono comunque decifrati e leggibili da parte di entrambi gli utenti. Questa rappresenta da un lato una caratteristica importante per gli utenti, perchè con l'aggiornamento della chiave non vi è perdita di messaggi, ma da un altro una debolezza del protocollo, perchè se un avversario viene in possesso di un$auth_key$ può dunque derivarne la chiave temporanea AES_key e accedere alla lettura di 100 messaggi criptati con essa.

3.4 Compatibilità all'indietro

Dalla data di rilascio ad oggi, MTProto ha ricevuto continui aggiornamenti, portando all'incertezza di comunicazione tra dispositivi con software aggiornati a versioni (*layer*) differenti. Questo problema è stato superato grazie all'introduzione della compatibilità all'indietro dei layer più recenti.
Abbiamo visto in precedenza come ogni messaggio, prima di essere cifrato, venga inserito in un pacchetto contenente il *layer* che il rispettivo client sta utilizzando (ricordiamo che ciò vale anche per messaggi di servizio). Si ha che subito dopo la creazione dell chiave segreta *auth_key* (salvata localmente in un pacchetto cifrato *encryptedChat* con *layer* 0), l'admin della chat **A** invia il proprio *layer* a **B** tramite un messaggio di servizio, e **B** risponde in egual maniera. Entrambi i clients salvano il minore tra i due *layer* all'interno del pacchetto *encryptedChat* così da conformarsi alla stessa versione.
Vi sono caratteristiche che sono state inserite solo in data successiva a quella di rilascio di Telegram, di cui alcune di grande importanza quali:

- **Chat segrete:** Introdotte con il layer 8
- **Contatori:** Miglioratone l'utilizzo con il layer 17

- **Forward Secrecy:** Introdotta con il layer 20

Analizzando l'utilizzo dei contatori prima dell'introduzione del layer 17, si nota come essi non venivano inseriti all'interno di ogni messaggio, ma conservati esclusivamente sul server, diminuendo quindi drasticamente la sicurezza rispetto alle versioni attuali. Infatti in caso di mirror o replay-attack, i client non possedevano alcun tipo di difesa per contrastarli.
Attualmente la versione più aggiornata corrisponde al *layer* 23.

Capitolo 4

Attacchi conosciuti

La costruzione di Telegram è stata effettuata con l'obbiettivo primario di ottenere un servizio sicuro, e affidabile, contro attacchi crittografici.

4.1 Testo in Chiaro Noto (KPA)

Per definizione, un attacco a testo in chiaro noto, è un modello di attacco in cui l'avverario possiede uno o più testi in chiaro, e i relativi testi cifrat. IGE è dimostrato sicuro da attacchi di questo tipo, ma non ne vedremo il dettaglio in questa tesi.

4.2 Testo in Chiaro Scelto (CPA)

Per definizione, un attacco a testo in chiaro scelto, è un modello di attacco in cui l'avversario ha la capacità di scegliere un testo in chiaro arbitrario, ed ottenerne il relativo testo cifrato. IGE (base su cui è costruito MTProto) è dimostrato essere robusto contro attacchi CPA, ma non sicuro con attacchi BACPA (block-wise adaptive testo in chiaro scelto); questo però non vale per Telegram, e le sua implementazione AES-IGE usata in MTProto. Ciò è dovuto all'utilizzo dei contatori presenti in ogni messaggio. In più si ha che per modificare il testo in chiaro, bisognerbbe essere a conoscenza della chiave temporanea AES_key e

del vettore iniziale AES IV, entrambe dipendenti dalla chiave segreta condivisa *auth_key*, dimostrato precedentemente come non intercettabile o calcolabile.

4.3 Testo Cifrato Scelto (CCA)

Per definizione, un attacco a testo cifrato scelto è un modello di attacco in cui un avversario ottiene informazione, almeno in parte, inserendo uno o più testi cifrati nel sistema, ed ottenendo i rispettivi testi in chiaro, secondo una chiave ad esso sconosciuta. Con queste informazioni, l'avversario può cercare di risalire alla chiave segreta usata per la decifrazione.
Si è visto come in MTProto, ogni volta che un messaggio viene decifrato, si attuano controlli su *msg_key* in modo che corrisponda al risultato di SHA-1 applicato ai dati decifrati. In più, ogni testo in chiaro (decifrato) contiene sempre valori indicatori dell'effettiva lunghezza del messaggio, e le sequenze di numeri (contatori). Entrambe queste caratterische, anche considerate singolarmente, mitigano la possibilità di un attacco CCA.

4.4 Indistinguibilità Testo Cifrato Scelto (IND-CCA)

L'indistinguibilità di un testo cifrato è la proprietà secondo la quale, un avversario è impossibilitato a distinguere l'origini di una coppia di testi cifrati pur potendo cifrare/decifrare a suo piacimento.Più precisamente, vi sono tre proprietà d'indistinguibilità:

1. Testo in chiaro scelto (**IND-CPA**)
2. Testo cifrato scelto (**IND-CCA1**)
3. Testo cifrato scelto adattivo (**IND-CCA2**)

Se un crittosistema è sicuro rispetto a una di queste proprietà, allora lo è anche rispetto alle precendeti, da cui si che ha IND-CCA2 è la proprietà più "forte" tra

le tre.
La sicurezza in termini d'indistinguibilità, viene spesso presentata sotto forma di $sfida$, in cui il crittosistema viene considerato sicuro se nessun avversario può vincere la $sfida$ con probabilità maggiore di un avversario che è forzato a scegliere casualmente.
Nel caso di Telegram, si ha che IGE è dimostrato essere non sicuro verso IND-CPA, ma ciò non influisce, se non in via teorica, nella sua sicurezza. In termini pratici, per MTProto si ha che, sotto certe circostanze, un avversario potrebbe intercettare un messaggio cifrato (dopo che è stato inviato), cambiarne una parte (senza essere in grado di modificare il testo del messaggio in alcun modo), e inviarlo all'utente destinatario originale del messaggio. In questo caso la decifrazione avviene senza errori e il messaggio viene ricevuto senza che gli utenti si accorgano dell'infiltrazione dell'avversario. Si nota come l'attacco dell'avversario non sia una vera e propria minaccia, in quanto il messaggio viene ugualmente recapitato e rimane leggibile solo dagli utenti originali della chat.
Quindi si può affermare che, sebbene MTProto, e in particolare IGE, non sia sicuro sotto l'aspetto IND-CCA questo non rappresenta una "grave" debolezza del protocollo.

4.4.1 IND-BACPA

Il termine "blockwise-adaptive" allude alla capacità di un avversario di poter osservare i risultati dell'inserimento di uno o più blocchi all'interno di un testo in chiaro, prima di passare al blocco successivo, permettendogli così di adattare l'attacco in base alle informazioni man mano ottenute. Questa nozione viene descritta tramite la $sfida$ seguente.
Si considera un avversario $\boldsymbol{A}$, un crittosistema con funzioni (Gen, Enc, Dec) indicanti rispettivamente la funzione che genera la chiave $\boldsymbol{k}$, la funzione di cifratura e quella di decifrazione:

1. Viene generata la chiave $\boldsymbol{k}$ ottenuta tramite Gen, e scelto un bit casuale $\boldsymbol{b} \leftarrow \{0, 1\}$.

2. ***A*** produce una coppia di blocchi di un messaggio in chiaro m_{01}, m_{11} di lunghezza uguale.

3. Un oracolo calcola un blocco cifrato $c_1 = Enc_k(m_{b1})$ e poi esso viene "inviato" ad ***A***.

4. Dopo aver osservato c_1, ***A*** genera un'altra coppia di blocchi in chiaro m_{02}, m_{12} ricevendo c_2. Dopo un numero di passi polinomiale, ***A*** fornisce in output ***b'*** .

5. L'output viene definito 1 se ***b = b'***, ossia se l'avversario vince la sfida che consiste nell'individuare il valore di ***b***, e 0 altrimenti.

Dunque, il sistema crittografico preso in esame è sicuro rispetto a IND-BACPA se per ogni avversario polinomiale ***A***, esiste una funzione *negl* trascurabile[1] tale che

$$Pr(Output = 1) \leq 1/2 + negl$$

cioè se un avversario ha probabilità non maggiore di vincere la sfida, rispetto a un avversario forzato a indovinare.

Attacco IND-BACPA

Per quanto riguarda IGE, esso è dimostrato essere sicuro contro attacchi CPA ma insicuro contro attacchi BACPA. Analizzando infatti la regola di cifratura a blocchi IGE

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

si nota come un blocco in chiaro x_i venga sommato con il blocco cifrato precedente c_{i-1}, prima di effettuare il calcolo mediante la funzione di cifratura. Sfruttando questa caratteristica, un avversario può dunque vincere la $sfida$ di BACPA trovando una collisione tra due blocchi prima che essi siano inseriti nella funzione di cifratura. In questo modo i risultanti blocchi cifrati saranno gli stessi.

[1]Una funzione $f : \mathbb{N} \to \mathbb{N}$ è detta trascurabile se per ogni polinomio $p(u)$ esiste una costante u_0 tale che $f(u) \leq \frac{1}{p(u)} \quad \forall u > u_0$

Vediamo ora un esempio di attacco su IGE, considerando un avversario $\boldsymbol{A}$ e un oracolo di cifratura $\boldsymbol{O}$:

1. Viene generata la chiave $\mathbf{k}$. $\boldsymbol{O}$ sceglie casualmente due blocchi iniziali m_0, c_0.

2. $\boldsymbol{A}$ sceglie due blocchi uguali m_{01}, m_{11} e li invia ad $\boldsymbol{O}$.

3. $\boldsymbol{O}$ calcola
$$c_1 = m_0 \oplus Enc_k(m_{b1} \oplus c_0)$$

4. $\boldsymbol{A}$ sceglie altri due blocchi uguali m_{02}, m_{12}.

5. $\boldsymbol{O}$ calcola
$$c_2 = m_{b1} \oplus Enc_k(m_{b2} \oplus c_1)$$

6. $\boldsymbol{A}$ calcola
$$m_{03} = m_{02} \oplus c_1 \oplus c_2$$
e sceglie un $m_{13} \neq m_{03}$. (In questo passo si ha l'aspetto adattivo dell'attacco)

7. $\boldsymbol{O}$ calcola
$$c_3 = m_{b2} \oplus Enc_k(m_{b3} \oplus c_2)$$

8. $\boldsymbol{A}$ controlla se
$$m_{01} \oplus m_{02} \oplus c_3 \stackrel{?}{=} c_2$$
Se questo è vero allora $\boldsymbol{A}$ può affermare che b = 0, altrimenti b = 1, e quindi $\boldsymbol{A}$ ha vinto la sfida BACPA.

Si nota come in questo tipo di attacco l'avversario vinca la sfida BACPA con probabilità 1; infatti grazie a come è scelto m_{03} si ha che se b = 0 allora:

$$\begin{aligned} c_3 &= m_{02} \oplus Enc_k((m_{02} \oplus c_1 \oplus \cancel{c_2}) \oplus \cancel{c_2}) \\ &= m_{02} \oplus Enc_k(m_{02} \oplus c_1) \end{aligned}$$

Quindi

$$\begin{aligned} m_{01} \oplus m_{02} \oplus c_3 &= m_{01} \oplus \cancel{m_{02}} \oplus \cancel{m_{02}} \oplus Enc_k(m_{02} \oplus c_1) \\ &= m_{01} \oplus Enc_k(m_{02} \oplus c_1) \\ &= c2 \end{aligned}$$

Se il risultato non è c_2 si può dire con certezza che b = 1.

Si può dunque dire che ogni sistema che utilizzi IGE non è sicuro rispetto alla proprietà più debole d'indistinguibilità ("blockwise-adaptive") IND-BACPA e di conseguenza nemmeno alle successive (sezione 4.0.4). Questo non è però applicabile a MTProto in quanto le chiavi temporanee di cifratura, AES_key e AES IV, dipendono dal testo in chiaro attraverso msg_key, perciò la modifica anche di un solo blocco comporterebbe il cambiamento della chiave di cifratura (che in questo tipo di sfida non può essere modificata[2]).

4.4.2 IND-CCA

Si analizzerà ora come MTProto sia debole ad attacchi IND-CCA, sfruttando le vulnerabilità del padding aggiunto al messaggio.
Nei capitoli precedenti si è visto come il calcolo di msg_key (nel processo di cifratura) avvenga prima dell'aggiunta del padding, non garantendone perciò l'autenticità e integrità. Questo significa che un avversario può modificare i blocchi contenenti il padding senza essere in alcun modo rilevato.
L'attacco seguente sfrutta dunque il fatto che la lunghezza del padding non viene mai verificata, permettendo così ad un avversario A di aggiungere blocchi a suo piacimento. Si considera ***O*** come oracolo di cifratura e decifrazione.

1. ***A*** produce due messaggi M_0, M_1 di lunghezza uguale.
2. ***O*** sceglie in modo casuale $b \in \{0, 1\}$ e calcola il testo cifrato $C = Enc_k(M_b)$.
3. ***A*** aggiunge a C un blocco di 128-bit casuale c_r e chiede ad ***O*** di decifrare $C' = C \| c_r \neq C$.[3]

[2] Concretamente, la decifrazione diventerebbe impossibili, e il messaggio verrebbe rifiutato
[3] In questa sfida si può chiedere la decifrazione di qualsiasi blocco, purchè diverso da C.

4. ***O*** decifra C' e ne ricava la lunghezza del payload. Tutto ciò che viene dopo viene scartato, quindi sia il padding originale che il blocco aggiuntivo. Quindi ***O*** ritorna $M' = Dec_k(C\|c_r) = M_b$

5. ***A*** restituisce in output 1 se $M' = M_1$ o 0 se $M' = M_0$

Notiamo come, sebbene avvengano delle "modifiche" al messaggio cifrato esso viene decifrato senza alcun errore, passando tutti i controlli attuati e ingannando così MTProto. Con questo tipo di attacco l'avversario vince con probabilità 1, quindi MTProto non è sicuro contro attacchi IND-CCA. Sebbene questa sia una debolezza del protocollo, essa non incide fortemente sulla sua sicurezza in quanto, sebbene l'avversario possa in qualche modo intromettersi nella comunicazione tra i 2 utenti, un messaggio rimarrebbe leggibile esclusivamente agli utenti originali della chat (a meno che l'avversario disponga veramente di un oracolo di decifrazione)

4.5 Attacchi Replay

Come visto in sezione 3.1 questi attacchi vengono contrastati poichè ogni messaggio contiene *message_id* ed i contatori, cosicchè ogni messaggio può essere inviato una sola volta.

4.6 Man-In-The-Middle

Per definizione un attacco di tipo Man-In-The-Middle è un attacco secondo cui l'avversario ha la possibilita di leggere,modificare o inserire messaggi a piacere, tra due parti comunicanti tra di loro.
In Telegram, entrambe le modalità di conversazione, chat end-to-end e chat standard, sono dichiarate sicure rispetto ad attacchi Man-in-The-Middle. Durante lo scambio delle chiavi di DH, tra utente e server, i dati sono cifrati con RSA con la chiave pubblica del server. Nel caso di chat standard quindi la sicurezza da questi attacchi è affidata esclusivamente al server.Per le chat segrete invece gli utenti devono

verificare che la loro chat sia sicura confrontando la visualizzazione della chiave di sessione.[4].

[4]Mostrato in figura 2.2

Bibliografia

[1] A practical cryptanalysis of the Telegram messaging protocol. http://cs.au.dk/~jakjak/master-thesis.pdf.

[2] J. Katz and Y. Lindell. *Introduction to Modern Cryptography: Principles and Protocols*. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2007. ISBN 9781584885511. URL https://books.google.dk/books?id=TTtVKHdOcDoC.

[3] Telegram. Secret chats, end-to-end encryption. https://core.telegram.org/api/end-to-end.

[4] Telegram. Perfect Forward Secrecy. https://core.telegram.org/api/end-to-end/pfs, 2016.

[5] Telegram. FAQ for the Technically Inclined. https://core.telegram.

[6] Telegram. Mobile Protocol: Detailed Description. https://core.telegram.org/mtproto/description.

[7] Telegram. Sequence numbers in Secret Chats. https://core.telegram.org/api/end-to-end/seq_no.

[8] Telegram. 15 Billion Telegrams Delivered Daily. https://telegram.org/blog/15-billion.

Ringraziamenti

Il primo ringraziamento va al Professore Davide Aliffi, per avermi proposto questo argomento di tesi super-interessante e per la disponibilità mostrata nel corso di questo ultimo mese.

Un ringraziamento a tutta la mia famiglia, che nonostante la mia scelta di indirizzo di laurea, "bizzarra" e non aspettata, mi ha comunque sempre sostenuto e appoggiato.

Un ringraziamento a Pase, caro amico, coinquilino e compagno di banco, con cui ho condiviso tutte le mie giornate tra sveglie insopportabili, lezioni "improponibili" e pasti idilliaci, e che ha reso questo percorso più leggero e divertente.
Un super ringraziamento anche a Rend, Filo e Lu, combriccola che vorrei avere sempre vicino a me, e a tutti gli altri compagni di corso e coinquilini, che mi hanno regalato 3 anni di ricordi indimenticabili.

Un ultimo ringraziamento, ma non per importanza, va a Sofi, che mi ha sopportato,motivato e che ha reso questi ultimi mesi più brevi e calorosi.

Grazie a voi tutti e..... alla prossima!!