POLITECNICO DI BARI
Dipartimento di Elettrotecnica ed Elettronica

Corso di laurea specialistica in Ingegneria Informatica

Sicurezza dei Sistemi Informatici

Relazione

"Realizzazione di un software per lo scambio sicuro di SMS tra dispositivi mobili (PDA e Smartphones)"

Docente:
Prof. Giuseppe Mastronardi

Studente:
Andrea Martelli

Introduzione

L'acronimo SMS è noto a tutti da diversi anni, ma non tutti ne conoscono il significato esteso, ovvero "Short Message Service", che indica appunto un servizio grazie al quale qualunque utente di telefonia mobile può inviare brevi messaggi di testo agli altri utenti, a costi relativamente bassi. Nonostante l'iniziale scetticismo sull'utilità del servizio, questo ha acquisito ben presto, e soprattutto grazie alle fasce più giovani di utenza, una popolarità pressoché illimitata. Sotto varie forme, è utilizzato per gli scopi più disparati, come il semplice scambio di messaggi tra normali utenti, l'invio di informazioni su traffico e condizioni meteo, il tracciamento di merci in movimento, e così via.. Proprio la semplicità, la versatilità e il basso costo del servizio lo rendono utilizzabile in un gran numero di applicazioni.

Nonostante la sua diffusione, tuttavia, ben pochi sforzi sono stati fatti per modificare il protocollo iniziale e migliorarne gli aspetti legati alla sicurezza, e questo è dovuto forse proprio al carattere inaspettato del successo ottenuto. Gli utenti finali, infatti, si scambiano ogni genere di informazioni via SMS senza essere a conoscenza del fatto che queste sono trasmesse secondo modalità non sicure e che per giunta il loro contenuto potrebbe essere facilmente letto o addirittura modificato in vari punti del canale di trasmissione.

Non potendo modificare il protocollo SMS o il canale trasmissivo, gli unici miglioramenti della sicurezza possono essere prodotti nei punti terminali della rete GSM, ovvero nei dispositivi cellulari degli utenti. Lo scopo del lavoro oggetto della presente relazione, dunque, è stato proprio quello di sviluppare un'applicazione che, mediante tecniche di crittografia e firma digitale, rendesse molto più sicuro il contenuto e l'integrità degli SMS, tutelando la privacy degli utenti. Una volta analizzate le tecnologie e gli strumenti software a disposizione, la scelta dell'ambiente di sviluppo è ricaduta sui dispositivi di tipo PDA e smartphone con sistema operativo Windows Mobile™, che possono offrire quantità di memoria e potenza di calcolo adeguate agli scopi del progetto.

1 Short Message Service (SMS)

Il termine SMS [1] (acronimo dell'inglese: Short Message Service, servizio messaggi brevi) è comunemente usato per indicare un breve messaggio di testo inviato da un telefono cellulare ad un altro, con un costo esiguo. Il termine corretto sarebbe SM (Short Message), ma ormai è invalso l'uso di indicare il singolo messaggio col nome del servizio.

Il servizio è stato originariamente sviluppato sulla rete GSM, tuttavia è ora disponibile anche su altre reti, come la UMTS ed alcune reti fisse (in Italia per ora solo Telecom). È possibile inviare SMS ad un telefono cellulare anche da un computer, tramite Internet, e dal telefono di rete fissa.

Tra i principali vantaggi percepiti dell'SMS, alla base della straordinaria diffusione di questo servizio come sistema di comunicazione, c'è il basso costo rispetto ad una lunga telefonata (vantaggio in realtà spesso inesistente, perché una conversazione via SMS è costituita da più sms a distanza di tempo, l'uno in risposta all'altro) e la possibilità di rendere la comunicazione *asincrona*, cioè leggere il messaggio in un momento successivo alla ricezione.

Il primo SMS della storia è stato inviato il 3 dicembre 1992 da un computer ad un cellulare sulla rete GSM Vodafone inglese e il testo del messaggio era "Merry Christmas". Il primo SMS da cellulare a cellulare invece venne inviato all'inizio del 1993 da uno stagista della Nokia.

Dettagli tecnici

Lo standard prevede due diverse tipologie di messaggi: quelli Point-to-Point (SMS/PP), impiegati nella comunicazione da un terminale ad un altro, e quelli tipo Cell Broadcast (SMS/CB), originati in una cella e distribuiti a tutti i terminali sotto la sua copertura.

Il messaggio ha una dimensione fissa di 140 byte. Questo si traduce in pratica nella possibilità di usare 160 caratteri di testo usando una codifica a 7 bit, oppure 140 caratteri usando una codifica a 8 bit. In lingue che usano altri caratteri rispetto all'alfabeto latino, per esempio in russo, cinese, giapponese, il messaggio è limitato a soli 70 caratteri (ognuno di 2 byte, usando il sistema Unicode).

Dal punto di vista trasmissivo, le unità di dati del messaggio SMS (vengono impiegate 6 diverse PDU, Protocol Data Unit) vengono inserite nei canali di controllo del GSM, in modo che sia possibile ricevere o inviare un messaggio durante una conversazione. Per gli scopi dell’analisi, prendiamo in considerazione esclusivamente i messaggi di tipo Point-to-Point.

Point-to-Point Messages

Sono i messaggi che un utente può inviare ad un altro utente della rete mobile. Ogni messaggio viene inviato ad un Centro Servizi (SMSC, Short Messages Services Centre) che a sua volta si preoccupa di inviarlo al terminale opportuno, se nella stessa rete GSM, oppure al Centro Servizi dell'operatore della rete del destinatario. Pertanto il percorso del singolo messaggio viene, in realtà, diviso in due: dal terminale al Centro Servizi (SMS-MO, Mobile Originated), e dal Centro Servizi al destinatario (SMS-MT, Mobile Terminated). Scopo del SMSC è, ovviamente, quello da fare lo store-and-forward dei messaggi, anche in previsione di un'eventuale irraggiungibilità momentanea del destinatario.

Dato che la connessione tra terminale e Centro Servizi è di tipo *connectionless*, non si ha garanzia né sull'invio né sulla ricezione dei messaggi SMS. Tuttavia è possibile fare richiesta di una "ricevuta di ritorno", lo Status Report del messaggio. In questo modo, una volta che il messaggio è stato correttamente inoltrato al destinatario, viene inviata una segnalazione di "messaggio consegnato" al mittente. Questo servizio in genere raddoppia il prezzo di ciascun SMS.

Dal punto di vista dei contenuti inviabili, a volte i telefoni cellulari permettono l'invio di messaggi concatenati di dimensioni superiori ai classici 160 caratteri, in realtà formati da più SMS spediti indipendentemente e ricomposti alla ricezione. Non ci addentreremo nei dettagli degli standard SMS e GSM perché esulano dagli scopi di questo lavoro.

Negli schemi seguenti è illustrato il meccanismo di trasmissione e ricezione degli SMS.

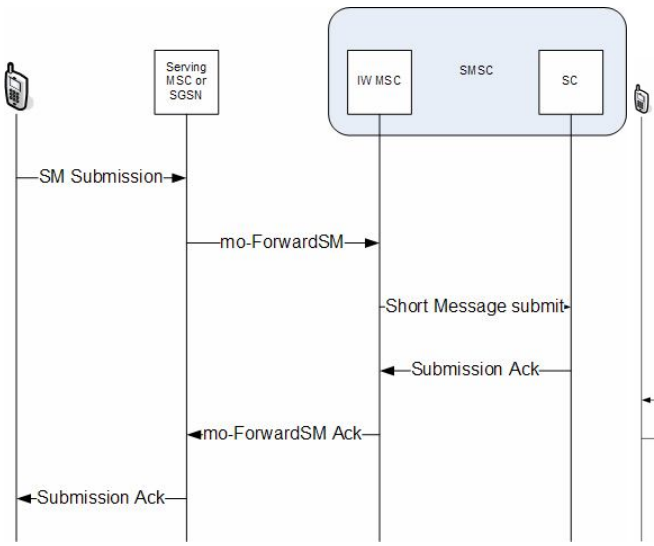


Fig. 1: Invio SMS da terminale

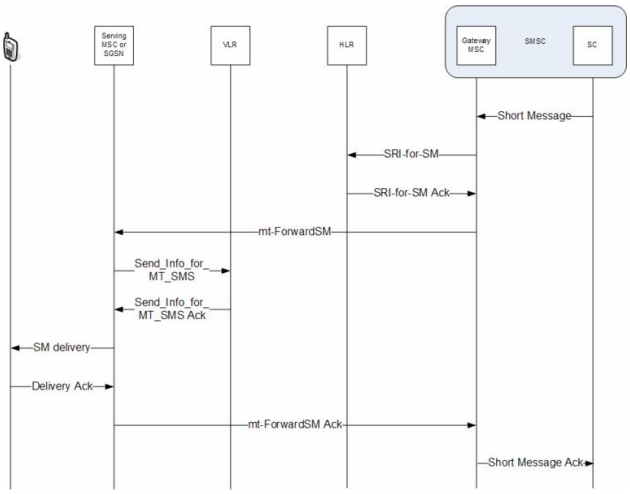


Fig. 2: Ricezione SMS dal centro servizi

La sicurezza negli SMS: i possibili attacchi

Anche se la quasi totalità degli utenti non è a conoscenza di queste problematiche, è bene ricordare che esistono dei rischi concreti nelle comunicazioni via SMS. Questi rischi, come sempre avviene nelle organizzazioni, sono in primo luogo legati agli esseri umani: non è certamente azzardato ipotizzare che all'interno delle compagnie telefoniche vi sia personale tecnico o organizzativo in grado, volendo, di accedere ai dati degli utenti, ai sistemi di logging e di leggere il contenuto degli SMS o ascoltare le telefonate. A tal proposito i recenti casi di cronaca, come quello dei dossier illegali Telecom, non dovrebbero rendere gli utenti molto fiduciosi riguardo la riservatezza delle loro comunicazioni. Oltre a questo aspetto, c'è da considerare la possibilità da parte delle autorità giudiziarie e delle forze di polizia di mettere sotto controllo i telefoni, e il fatto che una persona intercettata (o chi con questa ha normali rapporti di conoscenza) abbia il diritto, specialmente nel caso in cui non sia coinvolta in atti criminali, di tutelare la propria privacy. Vediamo ora più in dettaglio alcuni tipi di attacco al servizio SMS.

SMS Spoofing

Genericamente, lo *spoofing* consiste nel camuffare la propria reale identità facendo credere alla vittima di "essere qualcun altro". Nel caso degli SMS questa è una tecnica relativamente nuova, e consiste sostanzialmente nel modificare il *Sender ID* sostituendolo con un numero o con una sequenza alfanumerica che comparirà come mittente del messaggio. Ciò può essere fatto sia a fini leciti (es.: sostituire il numero dell'azienda con il suo nome), che illeciti.

Lo spoofing SMS è divenuto possibile specialmente da quando molti providers hanno integrato le loro reti di telecomunicazione con Internet. Sfortunatamente infatti, le web forms progettate per inviare SMS potrebbero avere vulnerabilità che un hacker potrebbe sfruttare per "bucare" il tunneling protocol che collega la rete telefonica ad Internet. Esistono inoltre dei software commerciali o open-source che consentono l'invio di messaggi "spoofati". Recentemente la Asian School of Cyber Laws ha condotto degli esperimenti a livello nazionale ed internazionale sullo spoofing ottenendo risultati soddisfacenti.

Vediamo ora come lo spoofing è possibile analizzando i campi del protocollo UCP (Universal Computer Protocol), uno dei due protocolli usati per la comunicazione tra terminali ed SMS Gateways.

Pacchetto UCP:

<STX>HEADER/DATA/CRC/<EXT>

<STX> e <EXT> indicano, rispettivamente, l'inizio e la fine del messaggio. In particolare la struttura del campo DATA (per il servizio di invio di SMS singolo) è la seguente:

AdC [string of numeric char]	destinatario
***OAdC* [string of numeric char]**	**numero sorgente del messaggio**
OAC [string of char]	codice di autenticazione del mittente
MT [1 numeric char]	tipo di SMS inviato
AMsg [string of char]	il messaggio vero e proprio

Risulta evidente che lo spoofing consiste nell'inviare un pacchetto UCP con campo OAdC modificato ad arte.

SMS Snooping

Il termine *snooping* significa letteralmente "ficcare il naso, curiosare". Nell'ambito degli SMS il tipo più semplice di snooping è quello che si potrebbe subire lasciando incustodito il proprio cellulare, e lasciando così che qualcuno legga i messaggi memorizzati. Un rischio simile si corre smarrendo la SIM card, che se non protetta da PIN può essere inserita in qualsiasi altro cellulare e se ne possono leggere i messaggi.

SMS Interception and decryption

L'attacco a più alto tasso tecnologico consiste nell'intercettazione del traffico sulla rete GSM in una certa area, mediante un'adeguata strumentazione. Anche se strumenti di questo tipo sono teoricamente in possesso solo delle forze di polizia e delle agenzie governative (come il *GSM Intercept System a5.1* dell'americano Security Intelligence Technology Group), è probabile che siano stati prodotti indipendentemente da qualche altra organizzazione per fini di ricerca.

Il sistema di intercettazione *GSM Intercept System a5.1*,

ad esempio, è in grado di intercettare tutto il traffico (voce ed SMS) in una data area e di memorizzarlo, senza interferire con la rete GSM e alterare le trasmissioni. Inoltre è in grado di decifrare il traffico GSM cifrato con l'algoritmo A5.1, la cui crittanalisi [2] è stata teorizzata ed effettuata da Alex Biryukov , Adi Shamir e David Wagner nel 1999. Nonostante questo studio abbia rivelato che un SMS cifrato con A5.1 è decifrabile in meno di un secondo (quindi sostanzialmente in *real time*) con un normale PC, quest'algoritmo viene ancora utilizzato dalla maggior parte dei gestori di rete.

È chiaro quindi che lo scambio di SMS non è un'operazione sicura al 100%, e che un sistema di cifratura delle comunicazioni porterebbe con se un grande beneficio in termini di privacy. Tuttavia le tecnologie crittografiche hanno un risvolto "negativo" quando vengono utilizzate per fini illeciti, in quanto potrebbero rendere vane le intercettazioni da parte delle forze dell'ordine. Questo spiega il perché delle politiche adottate dal governo degli Stati Uniti in merito all'esportazione di tecnologie di questo tipo. Un esempio è la JCE (Java Cryptographic Extension), che al di fuori del territorio USA è diffusa in una versione priva di alcuni algoritmi e incapace di generare chiavi superiori ad una certa lunghezza. L'"oscurantismo" americano in rapporto a queste tecnologie ha dato negli anni adito a polemiche secondo le quali in realtà molti algoritmi brevettati dall'NSA conterrebbero delle *backdoor* sfruttabili dai servizi segreti per decifrare ogni comunicazione. Tuttavia queste teorie, che hanno più a che fare con la dietrologia, non hanno per il momento avuto riscontri reali.

Si ritiene infine che nonostante ogni tecnologia abbia delle applicazioni potenzialmente negative, questa non sia una motivazione sufficiente per inibirne l'uso da parte della comunità.

2 Crittografia e firma digitale

In questo capitolo analizzeremo in generale le tecniche di crittografia e di firma digitale, con particolare attenzione agli algoritmi e ai metodi utilizzati per la realizzazione del modello di sicurezza PGP-like della nostra applicazione. Questo è basato infatti su uno schema di crittografia asimmetrica che consente lo scambio di SMS crittografati, firmati digitalmente o, nel caso più generale, di messaggi sia firmati che cifrati.

La parola crittografia deriva dall'unione di due parole greche: *kryptós* che significa nascosto, e *gráphein* che significa scrivere. La crittografia tratta delle "scritture nascoste", ovvero dei metodi per rendere un messaggio "offuscato" in modo da non essere comprensibile a persone non autorizzate a leggerlo. Un tale messaggio si chiama comunemente crittogramma.

Lo studio della crittografia e della crittanalisi, ovvero degli studi mirati alla rivelazione dei codici, si chiama comunemente *crittologia*.

La necessità di nascondere messaggi strategici da occhi nemici è antica quanto l'uomo: ci sono tracce di cifrari antichi quanto gli ebrei con il loro *codice di atbash*; gli Spartani avevano un loro particolare sistema di comunicazione dei messaggi segreti, la *scitala*; a Caio Giulio Cesare si attribuisce l'uso del cosiddetto *cifrario di Cesare*, un sistema crittografico oggi ritenuto elementare, ma emblema della nascita di un concetto totalmente nuovo e ottimo per comprendere le idee basilari della crittografia e i primi attacchi della sua "avversaria": la crittanalisi.

La storia della crittografia moderna inizia con la stesura del *“De Cifris”* di Leon Battista Alberti, che per primo insegnò a cifrare per mezzo di un disco cifrante con un alfabeto segreto mischiato da spostare ad libitum ogni due o tre parole. Anche il tedesco Tritemio prevedeva una forma di sostituzione polialfabetica, facendo scorrere l’alfabeto ordinato di un posto ad ogni lettera del chiaro. Il francese Vigenère utilizzò come base il sistema ideato da Tritemio, migliorandolo. Il suo sistema è stato considerato indecifrabile per tre secoli, finché nel 1863 il colonnello prussiano, Friedrich Kasiski pubblicò un metodo per "forzarlo", chiamato Esame Kasiski.

Qualsiasi sia il sistema crittografico utilizzato, la legge fondamentale sul corretto uso di tali tecniche fu scritta da Kerckhoffs ***("Legge di Kerckhoffs")*** nel suo libro del 1883 *"La Cryptographie Militaire"* e di seguito riportata: ***"La sicurezza di un crittosistema non deve dipendere dal tener celato il crittoalgoritmo. La sicurezza dipenderà solo dal tener celata la chiave."***

Nel 1918 Gilbert Vernam perfezionò il metodo di Vigenère proponendo l'idea di usare chiavi lunghe almeno quanto il messaggio. Successivamente, nel 1949, Claude Shannon, padre della

Teoria dell'informazione, dimostrò che questo è l'unico metodo crittografico totalmente sicuro possibile. Con il possesso di un sistema crittografico perfetto, la battaglia teorica tra crittografia e crittanalisi si è risolta con una vittoria della prima sulla seconda. Ipotizzando di voler far uso di questa insuperabile protezione, restano però aperti molti problemi di ordine pratico: bisogna infatti soddisfare gli stringenti requisiti del cifrario di Vernam: chiave lunga quanto il messaggio e mai più riutilizzabile. Tuttavia, si hanno notizie di utilizzi di questo cifrario in ambiente militare, o per la protezione delle comunicazioni del “telefono rosso” tra Washington e Mosca durante la Guerra fredda. Anche il cifrario trovato nel 1967 sul corpo di Che Guevara è una incarnazione del cifrario di Vernam.

L'attuale ricerca crittografica, avendo risolto il problema teorico della garanzia della sicurezza, si dedica al superamento dei forti limiti d'uso anzidetti. Si cercano metodi più "comodi" ma ciononostante "estremamente sicuri" (non lo saranno mai perfettamente, ma si può renderli sempre più sicuri) che, possibilmente, utilizzino chiavi corte e riutilizzabili senza compromettere la loro utilità. Al momento non esiste alcuna tecnica crittografica che si possa definire sicura in senso assoluto, tranne forse il Cifrario di Vernam: tutte le altre tecniche rendono sicuro il dato solo per un certo arco temporale e non possono garantire la durata della segretezza.

Crittografia simmetrica

La crittografia simmetrica è stata la prima tecnica crittografica ad essere inventata, e per lungo tempo è stata anche l’unica. Uno schema di crittografia simmetrica utilizza un’unica chiave per la cifratura e per la decifratura, ed è quindi utilizzata da entrambi gli interlocutori. La forza di questo tipo di crittografia, quindi, è basata non solo sulla grandezza dello spazio delle chiavi e sulla qualità dell’algoritmo, ma anche sulla segretezza della chiave. La caratteristica principale dei cifrari simmetrici è l’alta velocità anche con chiavi molto lunghe, cosa che la rende in alcuni contesti preferibile, o complementare, alla crittografia asimmetrica.

Il problema principale nell’uso di questo schema crittografico risiede nel fatto che i due interlocutori debbano in qualche modo accordarsi sulla chiave da usare. Nonostante questo sia fattibile, bisogna considerare che il canale trasmissivo usato per trasmettere la chiave potrebbe (come avviene nella stragrande maggioranza dei casi) non essere sicuro, esponendo così i due interlocutori ad un rischio inaccettabile.

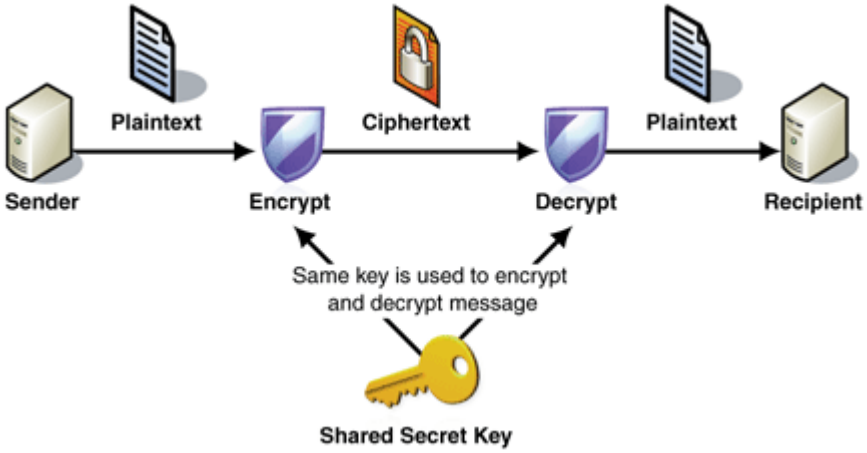


Fig. 3: schema di crittografia simmetrica

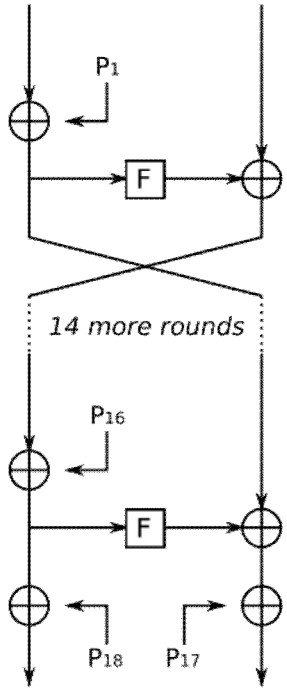
Gli algoritmi più diffusi sono il DES (Data Encryption Standard), il Triple-DES (o DES-EDE, Encryption-Decryption-Encryption), l'AES (Advanced Encryption Standard), Blowfish, Skipjack ed altri ancora. Per le sue caratteristiche, come vedremo in seguito, nell'ambito di questo lavoro l'algoritmo di cifratura simmetrica utilizzato è stato il Blowfish.

Blowfish

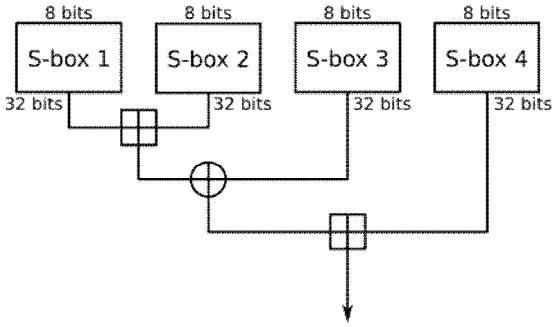
Blowfish è un algoritmo a chiave simmetrica a blocchi, ideato nel 1993 da Bruce Schneier e implementato in molti software di crittografia. Sebbene a tutt'oggi non sia reperibile una crittanalisi di Blowfish, questo algoritmo sta suscitando nuovamente interesse se implementato con una maggior dimensione dei blocchi, come nel caso di AES o Twofish.

Schneier progettò Blowfish per essere un algoritmo di utilizzo generale, utile a rimpiazzare l'allora decadente Data Encryption Standard (DES), e libero da problemi caratteristici di altri algoritmi. All'epoca molti altri sistemi di cifratura erano proprietari, coperti da brevetto o da segreti governativi. Schneier dichiarò: "Blowfish è libero da brevetti, e rimarrà tale in tutte le nazioni. L'algoritmo è di pubblico dominio, e può essere usato liberamente da chiunque".

Due delle caratteristiche di rilievo di Blowfish sono *S-box* dipendenti dalla chiave, e una lista di chiavi estremamente complessa. Nei blocchi di cifratura le *S-box* vengono utilizzate per oscurare relazioni tra il testo in chiaro e il testo cifrato seguendo il principio della *confusione* enunciato da Shannon. In generale le S-box ricevono *m* bit di ingresso e li trasformano in *n* bit di uscita. Una S-box *n x m* è implementata sotto forma di matrice e progettata appositamente per rendere l'algoritmo il più possibile resistente alla crittanalisi.



Blowfish ha una dimensione blocco a 64 bit e una lunghezza di chiave variabile fra i 32 e i 448 bit. Ha una struttura a *rete di Feistel* a 16 cicli. Ogni linea del diagramma rappresenta 32 bit. L'algoritmo gestisce due tipi di liste di sottochiavi: una di 18 elementi, definita P-array, e quattro S-Box da 256 elementi ciascuna. Ciascuna S-Box ha in ingresso 8 bit di informazioni, e produce in uscita 32 bit. A ogni ripetizione del ciclo si usa un elemento diverso del P-array, e dopo l'ultimo ciclo a ogni metà del blocco di dati viene moltiplicata in XOR con uno dei due elementi inutilizzati del P-array.



La funzione F, rappresentata nella figura a destra, divide i 32 bit in ingresso in 4 bytes, utilizzati a loro volta come ingressi delle S-Box. I risultati sono sommati in modulo 2^{32} e messi in XOR, per ottenere il risultato finale di 32 bit cifrati.

Dato che Blowfish è una rete di Feistel, può essere invertito semplicemente mettendo in XOR gli elementi 17 e 18 del P-Array, e quindi utilizzando gli elementi dell'array stesso in ordine inverso. La lista delle chiavi di Blowfish viene generata caricando i valori iniziali del P-Array e delle S-Box con rappresentazioni esadecimali delle cifre del *pi greco*, che apparentemente non hanno periodicità o schemi ripetitivi. La chiave segreta viene quindi messa in XOR con ciascun elemento del P-Array (ripetendo la chiave se necessario). Un blocco di 64 bit di valore 0 viene criptato con l'algoritmo stesso e il risultato cifrato sostituisce gli elementi P-1 e P-2. Il risultato cifrato viene quindi codificato nuovamente con le nuove sottochiavi e va a sostituire P-3 e P-4. Questo procedimento continua fino a rimpiazzare tutti gli elementi del P-Array e delle S-Box. In tutto, l'algoritmo di cifratura Blowfish viene eseguito 521 volte per generare tutte le sottochiavi, ed elaborando circa 4 Kb di dati.

Crittanalisi del Blowfish

Non esistono, o perlomeno non sono conosciute, crittanalisi di Blowfish, almeno fino al 2004, dato che una dimensione di blocco di 64 bit è oggi considerata troppo piccola, e codificando più di 2^{32} blocchi (32 GB) si potrebbero rivelare alcuni frammenti del testo originale, a causa del paradosso del compleanno. Nonostante questo, Blowfish fino ad oggi sembra abbastanza sicuro. È

importante, a tal proposito, notare come 32 GB sia una quantità di dati incredibilmente grande se considerata nell'ambito del servizio ***SMS***, in cui ogni messaggio ha una dimensione massima di 140 bytes. Per procurarsi tale mole di dati sarebbe quindi necessario ad un attaccante intercettare (32 GB)/(140 byte) = 245.426.703 messaggi cifrati con la stessa chiave (ben più dei messaggi che una persona potrebbe mandare nell'arco di una vita!!), supponendo che abbiano tutti lunghezza massima. Anche inviando 10 SMS cifrati al giorno, ad esempio, per raggiungere questa cifra occorrerebbe ripetere l'operazione per circa 67000 anni! Mentre la ridotta dimensione del blocco non crea problemi per applicazioni tradizionali come l'e-mail, potrebbe rendere inutilizzabile Blowfish per la cifratura di testi di grandi dimensioni, come ad esempio in caso di archivi.

Nel 1996 Serge Vaudenay escogitò un attacco che permetteva di forzare la cifratura di un testo conoscendo 2^{8r+1} testi in chiaro con la stessa chiave, dove r identifica il numero di cicli (rounds). Inoltre trovò una classe di chiavi deboli che possono essere identificate e rotte, con lo stesso attacco, conoscendo solo 2^{4r+1} testi in chiaro. Questo attacco non può essere usato contro il Blowfish implementato con tutti i 16 cicli. Vaudenay utilizzò una variante basata su meno cicli di cifratura. Vincent Rijmen, nella sua tesi di dottorato, introdusse un attacco differenziale di secondo grado in grado di forzare non più di 4 cicli. Non rimane quindi nessuna via conosciuta per forzare un testo codificato con 16 cicli, tranne ovviamente la forza bruta.

Perché Blowfish?

Dopo aver analizzato le principali caratteristiche di diversi algoritmi di cifratura, la scelta è ricaduta sul Blowfish per due motivi principali: innanzitutto l'efficienza computazionale, che lo rende uno degli algoritmi a chiave segreta più veloci e quindi particolarmente indicato per dispositivi a ridotte capacità di calcolo; in secondo luogo si è ritenuto che, vista la ridotta lunghezza di un SMS, al fine di non sprecare spazio prezioso, fosse necessario utilizzare un algoritmo con una dimensione dei blocchi il più possibile ridotta. Essendo in questo caso tale dimensione pari ad 8 bytes, ne consegue che un SMS cifrato possa essere costituito al massimo da 17 blocchi (8*17 = 136 bytes), con uno "spreco" di appena 4 bytes. In realtà, come vedremo in seguito, questi byte saranno vitali per il funzionamento della nostra applicazione in quanto utilizzati come header del messaggio.

Crittografia Asimmetrica

La crittografia asimmetrica, conosciuta anche come crittografia a coppia di chiavi, crittografia a chiave pubblica/privata o anche solo crittografia a chiave pubblica è un tipo di crittografia dove, come si evince dal nome, ad ogni attore coinvolto è associata una coppia di chiavi:

- la *chiave privata*, personale e segreta, viene utilizzata per decodificare un documento criptato;
- la *chiave pubblica*, che deve essere distribuita, serve a crittare un documento destinato alla persona che possiede la relativa chiave privata.

L'idea base della crittografia con coppia di chiavi diviene più chiara se si usa un'analogia postale, in cui il mittente è Alice ed il destinatario Bob e i lucchetti fanno le veci delle chiavi pubbliche e le chiavi recitano la parte delle chiavi private:

1. Alice chiede a Bob di spedirle il suo lucchetto, già aperto. La chiave dello stesso verrà però gelosamente conservata da Bob.
2. Alice riceve il lucchetto e, con esso, chiude il pacco e lo spedisce a Bob.
3. Bob riceve il pacco e può aprirlo con la chiave di cui è l'unico proprietario.

Se adesso Bob volesse mandare un altro pacco ad Alice, dovrebbe farlo chiudendolo con il lucchetto di Alice, che lei dovrebbe mandare a Bob e che solo lei potrebbe aprire. Si può notare come per secretare i pacchi ci sia bisogno del lucchetto del destinatario mentre per ricevere viene usata esclusivamente la propria chiave segreta, rendendo l'intero processo di cifratura/decifratura asimmetrico. Chiunque intercettasse il lucchetto o il messaggio chiuso non potrebbe leggerne il contenuto. Al contrario nella crittografia simmetrica, occorre il pericoloso passaggio dello scambio delle chiavi, che può essere intercettato.

Per utilizzare questo tipo di crittografia è necessario creare una coppia di chiavi, una chiave pubblica (da diffondere) ed una chiave privata (da tenere segreta). La proprietà fondamentale della coppia di chiavi pubblica/privata è che ***un messaggio cifrato usando la chiave pubblica può essere decriptato usando soltanto la chiave privata corrispondente***.

La coppia di chiavi pubblica/privata viene generata attraverso un algoritmo (ad esempio RSA o DSA) a partire da dei numeri casuali. Gli algoritmi asimmetrici sono studiati in modo tale che la conoscenza della chiave pubblica e dell'algoritmo stesso non siano sufficienti per risalire alla

chiave privata. Tale meccanismo è reso possibile grazie all'uso di funzioni unidirezionali. In realtà, in molti casi, l'impossibilità di risalire alla chiave privata non è dimostrata matematicamente, ma risulta dallo stato attuale delle conoscenze in matematica e della potenza di calcolo disponibile. Per esempio è sufficiente un piccolo computer e qualche millesimo di secondo per moltiplicare due numeri primi da 150 cifre, ma occorre il lavoro di decine di migliaia di computer per un anno per trovare i fattori primi di quel numero. Un altro possibile utilizzo dell'algoritmo di crittografia asimmetrica riguarda l'idea di firma digitale, quindi di autenticazione: un utente cifra un messaggio (più propriamente, cifra il *digest* del messaggio, come vedremo in seguito) con la propria chiave; gli altri, una volta ricevuto tale messaggio, riescono a decifrarlo solo con il certificato pubblico relativo a quella particolare chiave privata (del quale devono essere preventivamente a conoscenza, o il certificato viene spedito insieme al messaggio), per cui possono risalire con certezza alla sua identità.

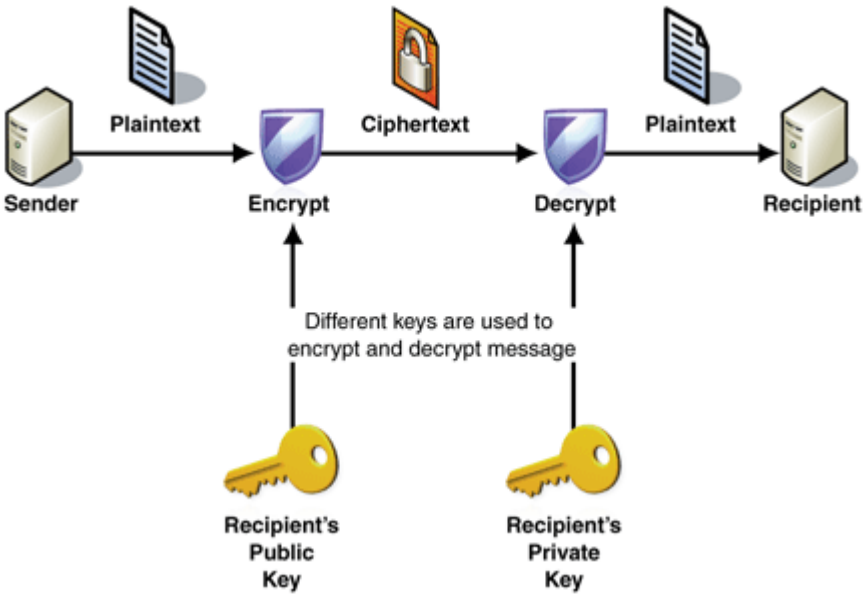


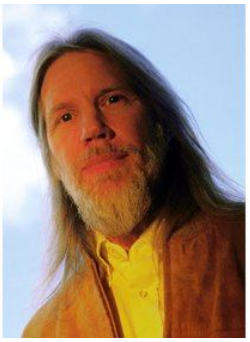
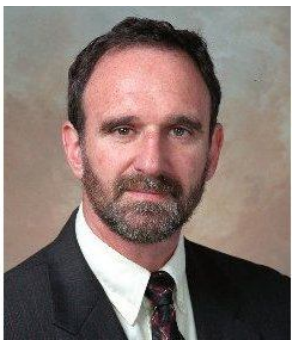
Fig. 4: Schema di cifratura a chiave pubblica

La nozione di “chiave pubblica” fu concepita e introdotta per la prima volta nel 1976 da Whitfield Diffie e Martin Hellman, in un noto articolo in cui spiegavano come, sulla base di particolari nozioni matematiche, un sistema come quello appena descritto potesse essere realizzato. Due anni dopo, nel 1978, Ronald Rivest, Adi Shamir e Len Adleman inventarono RSA, il più famoso sistema a chiave pubblica.

Occorre osservare che, nella pratica, l’algoritmo Diffide-Hellman o l’RSA vengono utilizzati solo per la generazione e lo scambio delle chiavi (operazione detta *Key Agreement*), e non per la cifratura vera e propria. Quest’operazione infatti è molto più lenta rispetto al caso della cifratura simmetrica, e nel caso dell’RSA deve scontrarsi anche con l’impossibilità di cifrare

direttamente testi di lunghezza superiore a valori molto bassi (dell'ordine dei 100 byte). Le chiavi generate, quindi, vengono usate in congiunzione ad un algoritmo di cifratura simmetrico, o al massimo per cifrare direttamente una "chiave segreta di sessione" da trasmettere e utilizzare poi ancora con un cifrario simmetrico.

Scambio di chiavi Diffie-Hellman

Lo Scambio di chiavi Diffie-Hellman (*Diffie-Hellman key exchange*) è un protocollo crittografico che consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione insicuro (pubblico) senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza. La chiave ottenuta mediante questo protocollo può essere successivamente impiegata per cifrare le comunicazioni successive tramite uno schema di crittografia simmetrica.

Nell'implementazione originale del protocollo si considera inizialmente un numero *g*, generatore del gruppo moltiplicativo degli interi modulo *p*, dove *p* è un numero primo. Uno dei due interlocutori, ad esempio Alice, sceglie un numero casuale a e calcola il valore $A = g^a \bmod p$ (dove mod indica l'operazione modulo, ovvero il resto della divisione intera) e lo invia attraverso il canale pubblico a Bob (l'altro interlocutore), assieme ai valori *g* e *p*. Bob da parte sua sceglie un numero casuale *b*, calcola $B = g^b \bmod p$ e lo invia ad Alice. A questo punto Alice calcola $K_A = B^a \bmod p$, mentre Bob calcola $K_B = A^b \bmod p$, e questi due valori si può dimostrare che sono uguali. A questo punto i due interlocutori sono entrambi in possesso della chiave segreta e possono cominciare ad usarla per cifrare le comunicazioni successive. L'implementazione più semplice del protocollo, quella originale, usa il gruppo moltiplicativo degli interi modulo *p*, dove *p* è un numero primo e *g* è generatore *mod p*.

Alice				Bob		
Sec		Calc		Calc		Sec
	p, g				p, g	
a						b
		g^a mod p			...	
	...			g^b mod p		
	$(g^b \bmod p)^a \bmod p$		→ ← =		$(g^a \bmod p)^b \bmod p$	

1. Alice e Bob si accordano per usare un numero primo $p=23$ e la base $g=5$
2. Alice sceglie in numero segreto $a=6$ e manda a Bob ($g^a \bmod p$) [$5^6 \bmod 23 = 8$]
3. Bob sceglie il intero segreto b=15 e manda ad Alice ($g^b \bmod p$) [$5^{15} \bmod 23 = 19$]

4. Alice calcola $(g^b \ mod \ p)^a \ mod \ p$ [$19^6 \bmod 23 = \mathbf{2}$]
5. Bob calcola $(g^a \ mod \ p)^b \ mod \ p$ [$8^{15} \bmod 23 = \mathbf{2}$]

Alice e Bob trovano lo stesso risultato perché g^{ab} e g^{ba} sono uguali. Si noti come solo *a*, *b* e $g^{ab} = g^{ba}$ sono segreti. Tutti gli altri numeri sono mandati in chiaro, ossia pubblici. Chiaramente i valori di *a*, *b* e *p* devono esser molto maggiori di quelli usati nell'esempio siccome è facile provare tutti i possibili valori di $g^{ab} \ mod$ 23 (vi sono in ogni caso al massimo 22 valori anche se *a* e *b* sono grandi). Se *p* è un primo di almeno 300 cifre e *a* e *b* sono almeno di 100 cifre, il miglior algoritmo conosciuto oggigiorno non può trovare *a* dati *g*, *p* e $g^a \ mod \ p$ usando tutta la potenza computazionale della terra. Questo è il problema del *logaritmo discreto*. Si noti come *g* non debba esser grande, infatti nella pratica è spesso 2 o 5.

Il logaritmo discreto

Il contesto in cui è forse più facile comprendere i logaritmi discreti è quello del gruppo moltiplicativo degli interi modulo p $\mathbb{Z}_p^*$(con p numero primo). Questo gruppo può essere visto come l'insieme degli interi $\{1, \ldots, p-1\}$con l'operazione di moltiplicazione modulo p; cioè se vogliamo la potenza (matematica) k-esima di un elemento del gruppo possiamo calcolare la sua potenza k-esima come numero intero e poi prendere il resto della divisione per p. Questo processo è indicato come potenza discreta. Ad esempio, consideriamo $\mathbb{Z}_{17}^*$; per ottenere 34 in questo gruppo, calcoliamo prima 34 = 81 e dividiamo 81 per 17, ottenendo 4 con il resto di 13; perciò nel gruppo $\mathbb{Z}_{17}^*$ è 34 = 13. Il logaritmo discreto è l'operazione inversa: dato $3^k \equiv 13 \mod 17$, quale k rende l'espressione vera? A rigore ci sarebbero infinite soluzioni intere, per la natura modulare del problema; generalmente si cerca la più piccola soluzione non negativa, che è k = 4.

Il calcolo dei logaritmi discreti è un problema di complessità NP (non polinomiale, per cui non sono noti algoritmi efficienti), mentre il problema inverso dell'esponenziazione discreta non lo è. Questa asimmetria è analoga a quella fra la fattorizzazione di interi e la moltiplicazione di interi. Entrambe queste asimmetrie sono state utilizzate per la costruzione di sistemi crittografici. Le più recenti applicazioni della crittografia usano i logaritmi discreti in sottogruppi ciclici di curve ellittiche su campi finiti.

Perché Diffie-Hellman?

Nell'ambito di questo progetto, il sistema a chiave pubblica proposto da Diffie ed Hellman è stato preferito all'RSA per diversi motivi. Innanzitutto, dal punto di vista computazionale l'esponenziazione discreta è più semplice rispetto alla determinazione e ai calcoli sui numeri primi richiesti dall'RSA, e questa è una caratteristica particolarmente apprezzabile in ambienti con scarse risorse di calcolo disponibili. In secondo luogo la cifratura con RSA genererebbe messaggi troppo lunghi per due motivi: il primo è che questo è un cifrario a blocchi di 128 bit, e un singolo SMS potrebbe contenere al più 8 blocchi, per un totale di 128 byte impegnati e 12 "sprecati" (8 se escludiamo i 4 di header che vedremo in seguito); il secondo è che, non essendo l'RSA idoneo a cifrare direttamente il testo, richiede lo scambio di una ulteriore chiave, definita "*chiave di sessione*". Questa è utilizzata per cifrare il messaggio (con DES o AES, ad esempio) e viene cifrata con RSA e allegata al messaggio. Il ricevente quindi, decifra questa chiave con la sua chiave privata RSA, e la utilizza per decifrare il messaggio, sfruttando così in maniera combinata la cifratura simmetrica e asimmetrica.

Nel campo di interesse che è stato analizzato, ovvero quello della codifica degli SMS, la necessità di includere nei messaggi una chiave di sessione che funga da chiave segreta condivisa tra mittente e destinatario è stato il motivo principale per l'esclusione dell'RSA.

Il metodo di *Diffie-Hellman* ha il *vantaggio*, invece, di consentire ai due utenti la condivisione di una chiave segreta (permanente) senza che questi se la scambino effettivamente insieme ad ogni messaggio. Una volta scambiati i cosiddetti *"key exchange"* ($g^a \bmod p$ e $g^b \bmod p$, che ricoprono il ruolo di chiave pubblica a tutti gli effetti), infatti, gli utenti sono in grado di calcolare lo stesso ***agreement***, che può essere usato per la cifratura simmetrica con Blowfish (o qualsiasi altro cifrario simmetrico).

Firma Digitale

La firma digitale è un sistema di autenticazione di documenti digitali analogo alla firma autografa su carta. È basata sull'utilizzo di un sistema crittografico a chiave pubblica (o crittografia asimmetrica). Lo scenario tipico di utilizzo di un sistema siffatto è il seguente: un mittente vuole spedire un messaggio ad un destinatario in modalità sicura. Per farlo utilizza la chiave pubblica del destinatario per la codifica del messaggio da spedire, quindi spedisce il messaggio codificato al destinatario; il destinatario riceve il messaggio codificato e adopera la sua chiave privata per ottenere il messaggio "in chiaro".

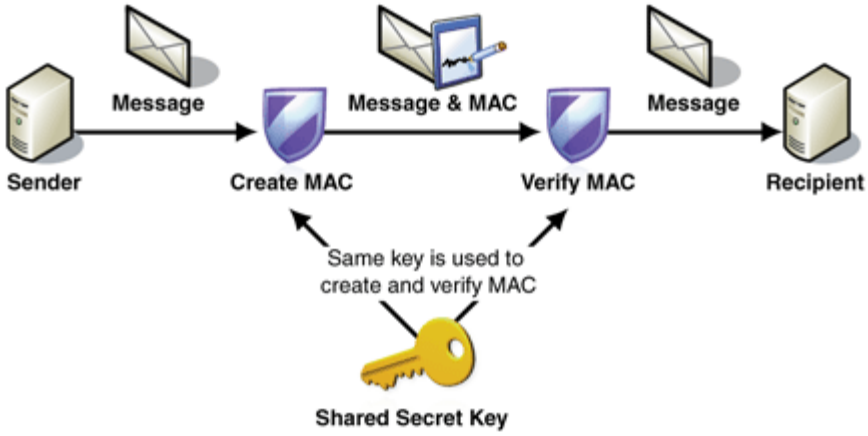


Fig. 5: schema di sistema per la firma digitale

Grazie ad un'ulteriore proprietà delle due chiavi, inversa rispetto a quella descritta, un sistema di questo tipo è adatto anche per ottenere dei documenti firmati, infatti: *la chiave pubblica di un utente è la sola in grado di poter decodificare correttamente i documenti codificati con la chiave privata di quell'utente*. Se un utente vuole creare una firma per un documento, procede nel modo seguente: con l'ausilio di una funzione *hash* ricava l'impronta digitale del documento, il *message digest*, un file di dimensione fissa che riassume le informazioni contenute nel documento, dopodiché utilizza la propria chiave privata per codificare quest'impronta digitale: il risultato di questa codifica è la creazione di una firma. La funzione *hash* è fatta in modo da rendere minima la probabilità che da testi diversi si possa ottenere il medesimo valore dell'impronta, inoltre è *one-way*, a senso unico, questo significa che dall'impronta è pressoché impossibile ottenere nuovamente il testo originario. La firma prodotta dipende dall'impronta digitale del documento e, quindi, dal documento stesso, oltre che dalla chiave privata dell'utente. A questo punto la firma viene allegata al documento.

Chiunque può verificare l'autenticità di un documento: per farlo, decodifica la firma del documento con la chiave pubblica del mittente, ottenendo l'impronta digitale del documento, e poi confronta questa con quella che si ottiene applicando la funzione *hash*, pubblica, al documento; se le due ***impronte*** sono ***uguali***, l'autenticità del documento è garantita.

Quanto appena descritto è valido nel caso generale. Dal momento che stiamo usando il sistema Diffie-Hellman, dobbiamo evidenziare una piccola differenza. In questo caso, infatti, il *message digest* non è cifrato esplicitamente con la chiave privata del mittente, ma bensì con l'*agreement* generato a partire dalla chiave pubblica dell'utente A e dalla chiave privata dell'utente B (o viceversa), che è lo stesso usato per la cifratura, come visto in precedenza. Questo è sufficiente a garantire l'autenticità del messaggio, in quanto esiste un diverso agreement tra ogni coppia di utenti.

Algoritmi di Hash

Hash ("to hash" sminuzzare, pasticciare) è un termine della lingua inglese che designa originariamente una polpettina fatta di avanzi di carne e verdure; per estensione indica un composto eterogeneo cui viene data una forma incerta: *"To make a hash of something"* vuol dire infatti creare confusione, o fare una cosa piuttosto male. Nel linguaggio scientifico, l' *hash* è una funzione univoca operante in un solo senso (ossia, che non può essere invertita), atta alla trasformazione di un testo di lunghezza arbitraria in una stringa di lunghezza fissa, relativamente limitata. Tale stringa rappresenta una sorta di "impronta digitale" del testo in chiaro, e viene detta valore di *hash*, *checksum* crittografico o *message digest*.

In informatica, la funzione di trasformazione che genera l' hash opera sui bit di un file qualsiasi, restituendo una stringa di bit di lunghezza predefinita. Spesso il nome della funzione di hash include il numero di bit che questa genera: ad esempio, SHA-256 genera una stringa di 256 bit. L'algoritmo di hash elabora qualunque mole di dati e rispetta i seguenti *requisiti*:

1) Restituisce una stringa di numeri e lettere a partire da un qualsiasi flusso di bit di qualsiasi dimensione (può essere un file ma anche una stringa). L'output è detto Digest.
2) La stringa di output è univoca per ogni documento e ne è un identificatore. Perciò, l'algoritmo è utilizzabile per la firma digitale.
3) Non è invertibile, ossia non è possibile ricostruire il documento originale a partire dalla stringa che viene restituita in output.

La firma digitale è quindi definita come il *digest* di un documento, crittografato con chiave privata. Un ulteriore uso delle funzioni di hash si ha nella derivazione di chiavi da password o passphrase. Non esiste una corrispondenza biunivoca tra l'hash e il testo. Dato che i testi possibili, con dimensione finita maggiore dell'hash, sono più degli hash possibili, per il principio dei cassetti ad almeno un hash corrisponderanno più testi possibili. Quando due testi producono lo stesso hash, si parla di collisione, e la qualità di una funzione di hash è misurata direttamente in base alla difficoltà nell'individuare due testi che generino una collisione. Per sconsigliare l'utilizzo di algoritmi di hashing in passato considerati sicuri è stato infatti sufficiente che un singolo gruppo di ricercatori riuscisse a generare una collisione.

Un hash *"crittograficamente sicuro"* non dovrebbe permettere di risalire, in un tempo congruo con la dimensione dell'hash, ad un testo che possa generarlo (che per i più resistenti è, attualmente, superiore alla durata dell'universo, applicando tutta la capacità computazionale immaginabile).

MD5

L'MD5 (acronimo di *Message Digest algorithm 5*) è un algoritmo per la crittografia dei dati a senso unico realizzato da Ronald Rivest nel 1991 e standardizzato in seguito. Questo tipo di codifica prende in input una stringa di lunghezza arbitraria e ne produce in output un'altra a 128 bit, in maniera estremamente veloce. Fu progettato come sostituto dell'MD4 non appena degli studi analitici dimostrarono la debolezza di quest'ultimo, e subito iniziarono i tentativi di attacco al nuovo algoritmo.

Nel marzo 2004 iniziò il progetto distribuito *MD5CRK* con lo scopo di dimostrare che l'MD5 era un algoritmo insicuro, trovando una collisione, usando un attacco birthday. *MD5CRK* finì in fretta, dopo che il 17 agosto 2004 fu trovata una collisione annunciata da Xiaoyun Wang, Dengguo Feng, Xuejia Lai e Hongbo Yu. Il loro attacco su un cluster IBM p690 ci impiegò solo un'ora. Dopo pochi giorni Vlastimil Klima descrisse un algoritmo migliorato, capace di costruire collisioni MD5 in poche ore su un singolo computer. Il 18 marzo 2006, Klima pubblicò un algoritmo che riusciva a trovare una collisione in un minuto su un singolo computer, usando un metodo che chiamò "calls tunneling".

Nonostante la debolezza dell'algoritmo sia stata provata, esso continua ad essere usato in moltissimi settori dell'informatica, principalmente grazie alla sua alta velocità e ridotte dimensioni. Per i motivi appena citati è stato scelto per la generazione del *digest* degli SMS.

Password Based Encryption (PBE)

Questa tecnica è molto utile per creare metodi di accesso a sistemi con un buon grado di sicurezza. È fondata principalmente sull'utilizzo di funzioni *hash*, e può essere vista sotto due diverse angolazioni.

La prima, e più semplice, applicazione consiste nella memorizzazione sicura di password, ed è massicciamente usata nei DBMS e nei sistemi di accesso dei siti web (per esempio quelli basati sul linguaggio PHP). Anziché memorizzare la password dell'utente in chiaro, se ne memorizza l'hash. Quando l'utente richiede l'accesso, viene calcolato l'hash della password immessa e confrontato con quello memorizzato, e qualora questi risultino identici viene consentito l'accesso.

La seconda, e più pertinente, applicazione consiste nella generazione di chiavi crittografiche, costituite quindi da un array di byte più o meno casuali, a partire da una password testuale e facilmente memorizzabile per un essere umano. In sostanza, alla password testuale viene

concatenato un cosiddetto *"salt"*, ovvero un prefissato array di byte scelti in modo casuale, e di questa nuova stringa viene calcolato l'hash. In questo modo si ottiene una sequenza di lunghezza prefissata di bit utilizzabile come chiave di cifratura. È da notare che, se il *salt* viene mantenuto segreto, la robustezza del sistema di password ne risente molto positivamente, in quanto migliora la resistenza agli attacchi a dizionario. In ogni caso questo sistema trae la sua forza anche dal fatto che il calcolo dell'hash richiede una certa quantità di tempo, e considerando che in molti casi la chiave è ottenuta al termine di molti cicli di hash, si comprende come un attacco possa diventare computazionalmente molto dispendioso. In aggiunta a ciò, se il *salt* è sufficientemente lungo, vengono ridotte o eliminate del tutto le collisioni hash tra passwords.

Il sistema di accesso all'applicazione oggetto di questa relazione sfrutta un sistema analogo a quello appena descritto, e che vedremo più in dettaglio in seguito.

3 Descrizione del software

Dopo aver analizzato le tecniche e i fondamenti teorici che hanno consentito la realizzazione del progetto, passiamo a descrivere la reale implementazione e le modalità di funzionamento del software che d'ora in poi chiameremo ***CryptSMS***.
Si riassumono di seguito le caratteristiche principali del software:

- Accesso protetto da PIN;
- Rubrica telefonica e keyring;
- Archiviazione cifrata dei messaggi ricevuti ed inviati mediante PBE;
- Invio di SMS in chiaro, cifrati, firmati, firmati e cifrati;
- Cambio, esportazione ed importazione sicura di chiavi e PIN;

Strumenti utilizzati

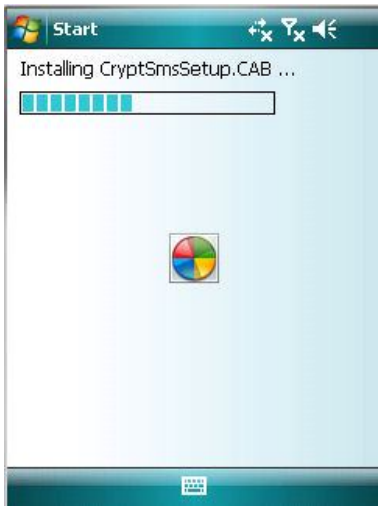
Il lavoro è iniziato considerando le possibilità implementative a livello di linguaggio di programmazione, ambiente di sviluppo e piattaforma hardware. Inizialmente, ponendo l'accento sulla portabilità del software, era stato prescelto il linguaggio Java (il cui motto, non a caso, è *"write once, run everywhere"*), ed in particolare la specifica J2ME (Java 2 Micro Editino), idonea ad essere eseguita dalle Virtual Machines in dotazione di quasi tutti i telefoni cellulari più recenti, in congiunzione con l'IDE open source Eclipse. Per la realizzazione del crittosistema sono state scelte le librerie *BouncyCastle*, anch'esse open source, ampiamente utilizzate, beneficiarie di un vasto bacino di utenza tra gli sviluppatori e offerenti un'ampia gamma di funzioni. Tuttavia, a causa delle differenze implementative nei vari dispositivi e le eccessive limitazioni, la piattaforma Java Micro è stata abbandonata, ed è stato effettuato il porting del codice già scritto nel linguaggio C# .NET 2.0.

L'ambiente di sviluppo utilizzato è stato quindi Microsoft Visual Studio 2008 in congiunzione col Windows Mobile 6 SDK. È stata inoltre utilizzata la versione C# delle API *BouncyCastle* (per la cifratura Blowfish) e un'altra libreria open source prodotta dal gruppo *Mentalis.Org* per l'algoritmo *Diffide-Hellman*. Infine, i test sono stati svolti su due PDA-Phone, un HTC P3300 e un Eten M700.

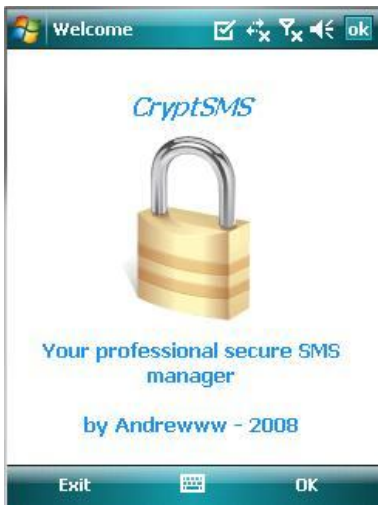
Si vuole sottolineare (ricordando il principio di Kerckhoff) come, specie nell'ambito della sicurezza, il ricorso a soluzioni ***open source*** abbia dei vantaggi sia economici che tecnologici, in quanto questi prodotti sono costantemente sottoposti al vaglio della comunità e a continui aggiornamenti.

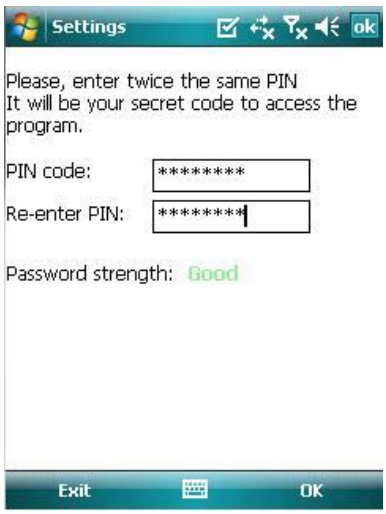
Installazione

I dettagli realizzativi verranno analizzati seguendo il filo logico dei casi d'uso e delle componenti principali del programma. Al termine dello sviluppo è stato prodotto un file di estensione *.cab* per l'installazione del software in Windows Mobile 6. Al termine di quest'operazione l'icona del programma apparirà nella cartella Programmi standard di Windows e il programma stesso potrà essere rimosso dal pannello di controllo come qualsiasi altra applicazione.



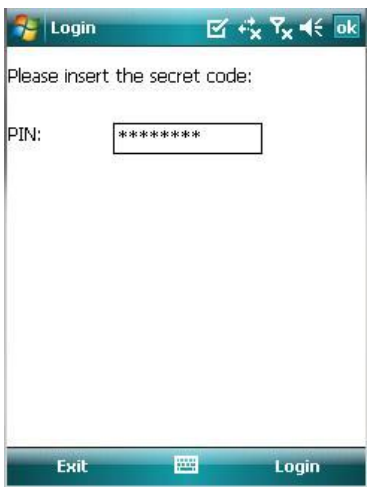
Primo accesso e Login





Al ***primo accesso*** viene chiesto all'utente di inserire due volte un codice alfanumerico che fungerà da password per l'accesso al programma. La qualità della password è testata da una funzione e può appartenere a 4 categorie: *Bad* (se contiene meno di 6 caratteri), *Normal* (se contiene almeno 6 caratteri), *Good* (se contiene due tipi di carattere tra lettere, numeri e simboli) e *Super* (se contiene tre tipi di carattere tra quelli indicati o se è lunga almeno 10 caratteri). Dalla password inserita, attraverso un meccanismo di *salt* e *hash*, viene ottenuta una chiave di 128 bit che sarà utilizzata per cifrare con Blowfish tutti i dati sensibili del programma, come le cartelle dei messaggi, le chiavi generate con Diffie-Hellman e il PIN stesso. Alla scelta del PIN, infatti, l'algoritmo di ***generazione delle chiavi*** Diffie-Hellman genera i valori numerici *p*, *g* ed *a* precedentemente visti. Questi vengono quindi criptati con Blowfish e salvati sotto forma di stringhe codificate in *Base64* in un file di tipo CSV (*Comma Separated*

Values), precisamente in *"settings.csv"*. Base64 è un sistema di numerazione posizionale che suddivide l'input in gruppi di 6 bit e trasforma ognuno nel corrispondente carattere ASCII (*American Standard Code for Information Interchange*). La comodità di questo sistema sta nel poter trattare dati binari come stringhe alfanumeriche.



Al ***login***, il meccanismo per verificare la correttezza della password è leggermente più complesso: dal PIN immesso viene generata la chiave con lo stesso metodo descritto prima, quindi viene decifrato il file contenente il PIN originale cifrato, e i due PIN vengono confrontati. Se sono identici, viene consentito l'accesso. È evidente che se il PIN inserito è errato, anche la chiave di decifratura lo è, e il pin caricato dal file non viene correttamente decifrato. In questo modo la chiave di cifratura per i dati di sistema dipende dal codice scelto dall'utente, e non esiste una chiave fissa per la cifratura del PIN stesso. Lo schema seguente sintetizza tutto il meccanismo:

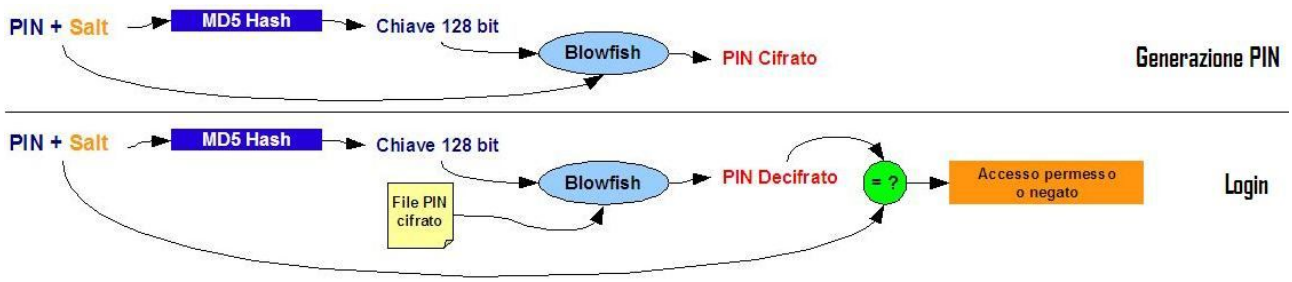
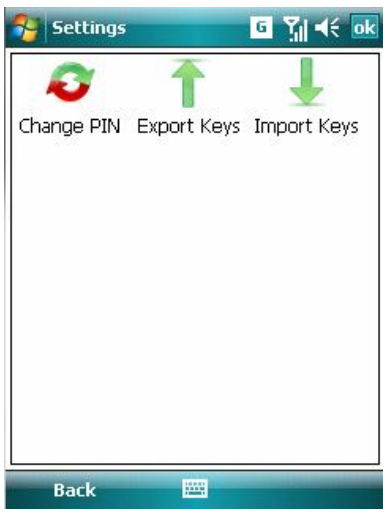


Fig. 6: schema del sistema di accesso

Gestione delle impostazioni



In qualsiasi momento durante l'utilizzo del programma, è possibile cambiare il PIN (ed insieme ad esso le chiavi crittografiche), esportare o importare il file contenente PIN e chiavi, per averne una copia di sicurezza o utilizzarle su un altro

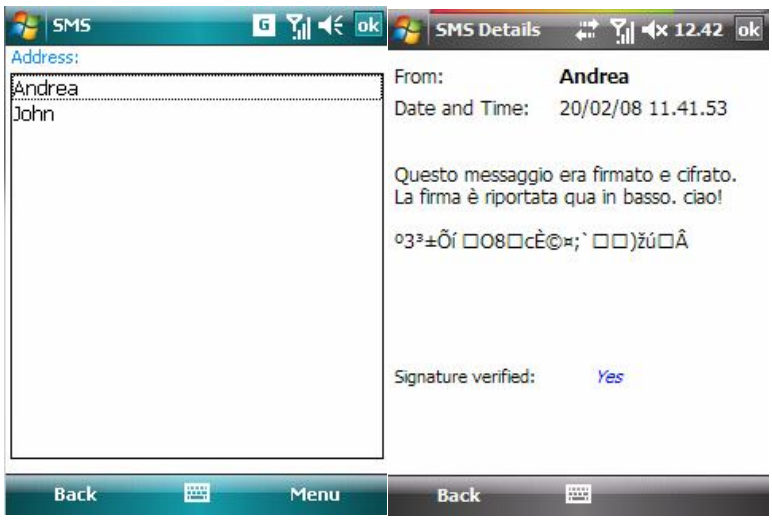
dispositivo. Le chiavi e il PIN vengono salvate in un file di tipo CSV con estensione *.csk (CryptSMS Keys)* sempre criptate, così che l'operazione di esportazione non costituisce un pericolo per la sicurezza. Infatti anche venendo in possesso di questo file, bisogna conoscere il PIN originale per essere autenticati.

Se si decide di cambiare il codice, il programma chiede innanzitutto che di inserire quello vecchio, e poi di digitare due volte il nuovo PIN. Confermando l'operazione vengono generati dei nuovi parametri che andranno a costituire delle nuove chiavi Diffie-Hellman. Inoltre, cambiando anche la chiave di cifratura "di sistema", non è più possibile leggere i vecchi SMS salvati nelle cartelle Sent e Unreaded, che quindi vengono svuotate.

```
ggpiRksmAro=,6RVZfXpqmzI=,tkoWvNJ/lZ95XFc1qgb6kPawKLUENnUz4xDYFZMwjUKYV/vtxMjCOZ81GJcLulB
U/dsTyOpQVcNjYSQPQJeBjlswbWWvZNXtgz7PguKilJHNTSPV46duSuPn7ATfEz61iqjsSkO6m/CVYmpPcXp6Au8h
SBij5abtL11/AdTobkEt7TeSQIKK2w==,S1IzPTwwPm8oTL6COO9pUgaYYYbeQJ4h
```

Fig. 7: esempio di file settings.csv

Cartelle dei messaggi

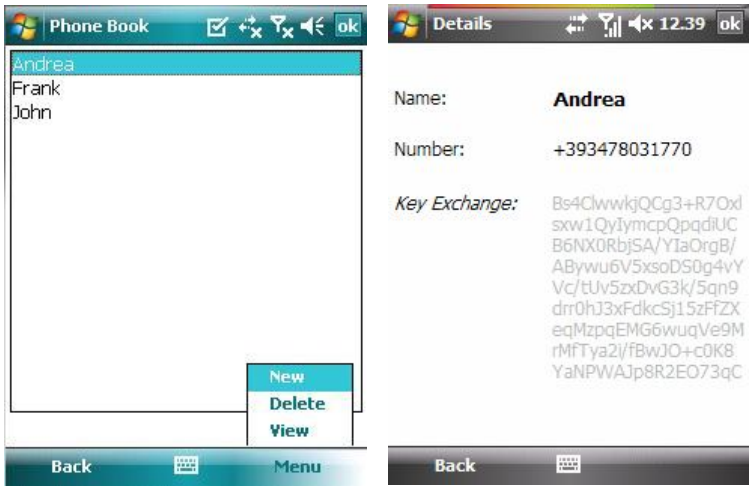


Il programma memorizza sia i messaggi ricevuti che quelli inviati, visualizzandoli in ordine cronologico. In particolare i messaggi sono suddivisi in Sent e Received, ed in ambo i casi sono conservati all'interno di file CSV(*"unreaded.csv"*, *"sent.csv"* e *"received.csv"*). Per garantire che nessuno eccetto l'utente autorizzato a conoscenza del PIN possa leggere gli SMS, il contenuto di questi viene ***cifrato*** con Blowfish e codificato in stringhe Base64 prima di essere scritto su file. Ogni messaggio all'interno del file *.csv*, è memorizzato in una riga costituita da quattro campi: mittente o destinatario, data ed ora di invio o ricezione, corpo del messaggio ed infine un flag indicante se per messaggio ricevuto è stato possibile verificare la firma digitale. Se questa è presente, inoltre, viene visualizzata in coda al messaggio.

Andrea,20/02/08_1.41.53,z9rVgDSNfEx21/p50LLqfo2mz9/IGfQGD0kLlWUeGAPWj8g65RuZ6NsKaCnEQfQyJTcNggxRLu1MIawGneJqcuwwvoDt/WcIi0PicF2HUTAZm7TSjow66S0EpjaTp1bVvU0O6Zg5KAy/dqexdB+rxQ==,True

Fig. 8: esempio di SMS inviato. Il body è criptato e codificato in Base64

Contact List



La raccolta delle informazioni relative ai contatti, come nominativo, numero ed eventualmente chiave pubblica (*key exchange*), è deputata ad un'apposita componente definita non a caso Phone Book. I dati sono memorizzati secondo le stesse modalità viste in precedenza per i messaggi e le impostazioni: ogni contatto costituisce una riga nel file *"contacts.cvs"*, e i campi stavolta sono nome, numero e chiave pubblica, qualora presente. Nelle operazioni di consultazione e manipolazione della rubrica, il contenuto di questo file viene mappato per comodità in una *HashTable* avente come chiave il nome e come valore una struttura contenente numero e chiave. La gestione dei contatti è controllata in modo tale che sia i nomi che i numeri siano univoci, cosa che rende possibili le operazioni di riconoscimento automatico *nome->numero* e *numero->nome*. Inoltre la rubrica può essere aperta in due differenti modalità che potrebbero essere definite "consultazione" e "scelta del numero", definite come segue: nella prima modalità è possibile visualizzare i dettagli dei contatti, aggiungerne di nuovi (è scontato che la chiave pubblica degli utenti può essere inserita solo dopo averla ricevuta via SMS seguendo la procedura guidata) ed eliminare quelli esistenti; la seconda modalità, invece, è usata in fase di scrittura degli SMS e consente unicamente di selezionare il contatto del destinatario.

Con questi ultimi tre paragrafi può considerarsi conclusa la descrizione dei meccanismi legati alla *persistenza* dei dati dell'applicazione. Passiamo ora ad analizzare la parte relativa al vero e proprio scambio di SMS, che è poi la parte centrale e più complessa dell'applicazione.

+393478031770,Andrea,Bs4ClwwkjQCg3+R7OxlsxW1QyIymcpQpqdiUCB6NX0RbjSA/YIaOrgB/ABywu6V5xsoDS0g4vYVc/tUv5zxDvG3k/5qn9drr0hJ3xFdkcSj15zFfZXeqMzpqEMG6wuqVe9MrMfTya2i/fBwJO+c0K8YaNPWAJp8R2EO73qCG8+c=

Fig. 9: esempio di contatto salvato in contacts.csv. La chiave pubblica è codificata in Base64

Formato degli SMS

Considerazioni preliminari

L'analisi dei requisiti del progetto ha portato a compiere un'attenta riflessione su come dovesse essere trattato il contenuto degli SMS manipolati dal programma. Innanzitutto, ricordiamo che un singolo SMS (e il programma gestisce solo messaggi singoli) può avere una lunghezza massima di 160 caratteri, utilizzando la codifica a 7 bit, e di 140 caratteri utilizzando una codifica ad 8 bit. Dal momento che è necessario trasmettere dati cifrati, e che questi sono costituiti da array di byte che possono assumere qualsiasi valore nel range 0-255, è stato obbligatorio usare una codifica ad 8 bit, in modo da non incorrere in troncamenti o altri tipi di errore in fase di codifica e trasmissione. La codifica prescelta, quindi, per trasformare array di byte in stringhe inviabili, è stata la *"windows-1252"*, adatta a codificare tutti i caratteri solitamente usati negli SMS e molto in uso nello stesso sistema operativo Windows (anche se ultimamente è stato superato da Unicode).

Un altro requisito fissato in fase di progetto è la possibilità di riconoscere i messaggi scambiati da *CryptSMS* e differenziarli dai normali messaggi in arrivo sul dispositivo. Questo è motivato anche dalla possibilità offerta da Windows Mobile di avviare automaticamente l'applicazione in funzione della ricezione di messaggi conformi a certi vincoli. In questo caso il vincolo si è concretizzato in un flag anteposto al testo del messaggio (vedremo meglio quest'aspetto in seguito).

Oltre a quanto appena detto, è necessario dotare il sistema di un metodo per determinare le caratteristiche di un messaggio ricevuto (o inviato), e cioè per capire se il messaggio è in chiaro, se è cifrato, firmato o se è una chiave pubblica.

Struttura dell'Header

La questione appena evidenziata della "riconoscibilità" del tipo di messaggio è stata risolta ideando un semplice protocollo, ovvero dotando gli SMS generati dal programma di un opportuno *header* contenente tutte le informazioni necessarie. Questo è costituito da 4 byte e ha la seguente struttura:

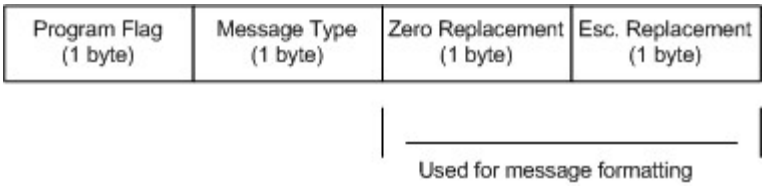


Fig. 10: struttura dell'header dei messaggi

- *Program Flag:* valore fisso 181 (µ). Identifica i messaggi inviati da CryptSMS e garantisce che quelli così contrassegnati vengano intercettati solo dal programma. È stato scelto questo valore perché è molto difficile che un normale SMS inizi col carattere µ, che tra l'altro non è disponibile su tutti i dispositivi.
- *Message Type:* indica il tipo di messaggio. 33 (*!*) = *plain*, 35 (*#*) = *encrypted*, 37 (*%*) = *signed*, 39 (*'*) = *signed & encrypted*, 41 (*)*) = *public key*;
- *Zero Replacement:* indica il valore usato per sostituire gli eventuali byte=0 nel corpo dei messaggi cifrati, firmati o nelle chiavi pubbliche.
- *Escare Replacement:* indica il valore usato per sostituire gli eventuali byte=27 (*escape character*) nel corpo dei messaggi. L'utilità degli ultimi due campi verrà esposta più chiaramente in seguito.

I campi *Encryption Flag* e *Signature Flag*, inoltre, possono assumere valore uguale e pari a 37 (#) per indicare che il messaggio costituisce una *chiave pubblica*, e vengono in tal caso come *Public Key Flag*.

In soli 4 byte quindi sono presenti tutte le informazioni necessarie alla corretta interpretazione degli SMS. È bene sottolineare che questo comporta una perdita di caratteri utili da sfruttare per la scrittura dei messaggi, ma che questa perdita è necessaria. Inoltre, facendo alcune considerazioni sulle dimensioni dei messaggi cifrati e della firma digitale, si osserva che l'impiego di questi 4 byte è ottimale e non costituisce necessariamente uno spreco. Per comprendere meglio questo aspetto analizzeremo ora la struttura dei vari tipi di messaggio.

Messaggi cifrati

Per inviare un messaggio cifrato occorre determinare la chiave di cifratura segreta e condivisa tra mittente e destinatario, secondo lo schema Diffie-Hellman. *L'agreement* và poi usato come chiave per la cifratura simmetrica con Blowfish. Tuttavia, l'algoritmo utilizzato genera agreement di 1024 bit, decisamente troppi per un algoritmo di cifratura simmetrica. È stato deciso, quindi, di utilizzare come chiave non questi 1024 bit, ma il loro hash *MD5* di 128 bit.

Bisogna considerare ora l'output dell'algoritmo Blowfish: questo genera dati di dimensione multipla di 8 byte, e quindi 140 byte potranno contenere al massimo 17 blocchi, per un totale di 136 byte impegnati. Come si vede, il tutto è studiato in modo che avanzino esattamente 4 byte, che possono essere proficuamente sfruttati per l'header! Affinché il processo produca messaggi criptati di lunghezza non superiore a 136 bytes, il messaggio originale dev'essere lungo al più ***135 caratteri*** (con una perdita effettiva di appena 5 caratteri). Questo perché *input di esattamente N blocchi producono in output N+1 blocchi*.

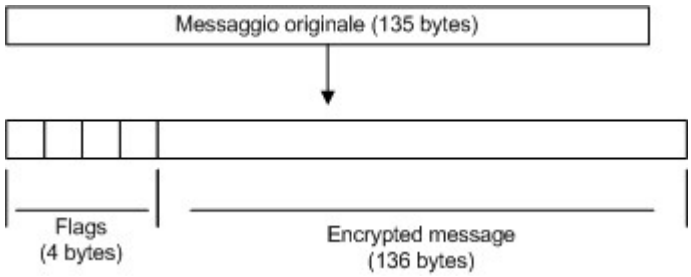


Fig. 11 struttura di un SMS cifrato

Messaggi firmati

La firma dei messaggi è calcolata secondo lo schema classico. Essa è generalmente il *digest* del messaggio cifrato con la chiave privata del mittente. L'adattamento di questo schema al metodo Diffie-Hellman e all'implementazione scelta, si traduce nel calcolo del *digest MD5* del messaggio e la sua cifratura con Blowfish, usando come chiave di cifratura *l'agreement* dei due interlocutori, che è unico e segreto. La firma risultante è lunga 24 bytes, che andranno sottratti alla massima lunghezza effettiva del testo da inviare. Considerando dunque 4 byte di header e 24 di firma, il messaggio può essere lungo al più ***112 caratteri***.

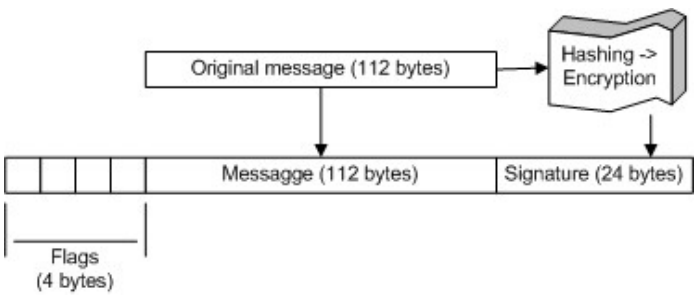


Fig. 12: struttura di un SMS firmato

Messaggi firmati e cifrati

Questo tipo di messaggi riunisce le due tecniche appena viste e garantisce il più alto livello di sicurezza in quanto, oltre a proteggere il contenuto, accerta anche la sua integrità e l'identità del mittente. La combinazione degli algoritmi di firma e cifratura può essere sintetizzata come segue: viene calcolato il digest del messaggio e viene cifrato con l'agreement, dopodichè la firma così ottenuta viene concatenata al messaggio originale. A questo punto l'insieme messaggio + firma viene cifrato con Blowfish usando come chiave sempre l'agreement dei due utenti.

In termini di lunghezza massima effettiva, ai 135 caratteri dei normali messaggi cifrati bisogna sottrarre i 24 della firma, per un valore effettivo di ***111 caratteri***.

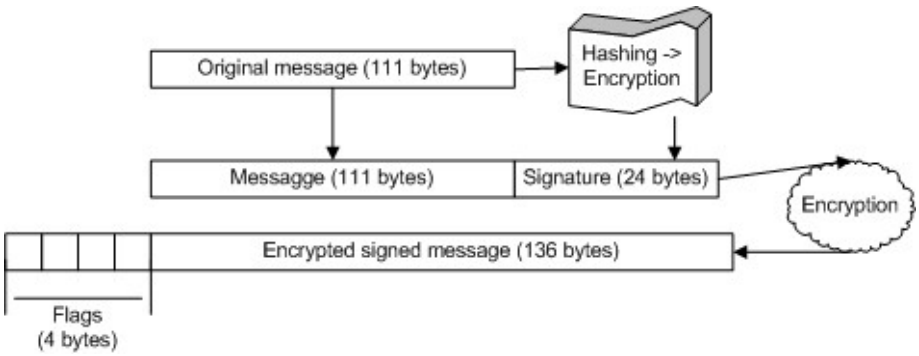


Fig. 13: struttura di un messaggio firmato e cifrato

Invio della chiave pubblica

Per inviare la chiave pubblica occorre caricare dal file *settings.csv* i parametri (*g*, *p* ed *a*) delle chiavi (dopo averli opportunamente decifrati con la chiave di sistema) e generare il *Key Exchange*, ovvero $g^a \bmod p$. Questo altro non è che un numero di 1024 bit, che può essere codificato in una stringa di 128 caratteri e inviato via SMS.

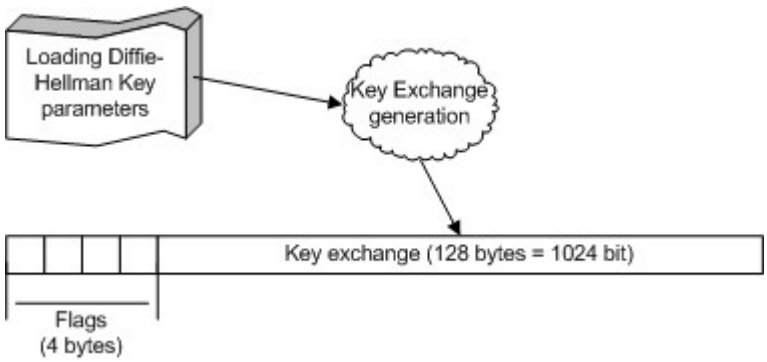


Fig. 14: Invio della chiave pubblica

Problemi riscontrati in trasmissione

In questo paragrafo sarà esplicitata l'utilità del terzo e quarto flag, i cosiddetti *"Zero Replacement"* ed *"Escape Replacement"*. Durante alcuni test i messaggi inviati risultavano troppo brevi, come se fossero stati troncati, o originavano errori di decodifica. Questo era imputabile alla presenza, all'interno della sequenza, di byte di valore pari a 0 o a 27, che venivano interpretati, in fase di codifica *windows-1252* e in fase di trasmissione, come *Null Terminator* o come *Escape Charachter*. Quest'ultimo, in particolare, veniva tradotto in fase di trasmissione nel valore 32 (*Space*). Il problema è stato risolto facendo la seguente osservazione: dal momento che la lunghezza effettiva di un messaggio non supera mai i 136 bytes, anche se questi avessero tutti valore diverso ci sarebbero comunque alcuni valori del range 0-255 non utilizzati. L'idea quindi è quella di sostituire ai byte pari a 0 e a 27, il due più piccoli valori inutilizzati (ovviamente escludendo 0 e 27), e scriverli nel due flag sopra citati, che costituiscono il terzo e quarto byte dell'header. All'atto della ricezione, per riottenere il messaggio originario sarà sufficiente sostituire tutte le occorrenze dei caratteri "sostitutivi", letti dall'header, con i valori 0 e 27.

Operando questi controlli si ha la garanzia che i messaggi non vengano troncati in fase di codifica o durante il loro transito nella rete GSM.

Invio degli SMS

Il menu "New SMS" offre la possibilità di scegliere il tipo di messaggio da inviare. In funzione di questo la maschera per l'inserimento del testo presenterà una differente limitazione. Il caso in figura è quello di un messaggio di tipo *plaintext*. Dal momento che il software è orientato all'invio di messaggi tra soggetti che si sono preventivamente scambiati le chiavi pubbliche, il destinatario del messaggio può essere unicamente scelto tra i nominativi presenti in rubrica. Qualora il tipo di messaggio richieda la conoscenza della

chiave pubblica del destinatario, ma non se ne disponga, un popup informativo avviserà l'utente del problema.

Ricezione degli SMS

La ricezione di un SMS è segnalata da una notifica temporanea a video, e dalla comparsa di un asterisco * affianco alla voce di menu *Unreaded*. Questa è una cartella temporanea in cui i messaggi sono salvati così come vengono ricevuti. Da qui è possibile eliminare i messaggi prima di leggerli o aprirli. In questo caso il programma riconosce automaticamente il tipo di messaggio processando l'header.

Se il messaggio è in chiaro, ovviamente verrà aperto e mostrato all'utente senza elaborazioni aggiuntive. Se invece è cifrato, firmato, o entrambe le cose, la prima operazione che viene effettuata è controllare se il numero del mittente è già contenuto in rubrica e se si dispone della sua chiave pubblica.

In caso di risposta negativa, l'utente viene avvertito con un messaggio d'errore. Se il messaggio è cifrato non potrà neanche essere aperto, mentre se è solo firmato potrà essere letto ma la sua integrità non potrà essere verificata controllando la firma digitale.

In caso di risposta affermativa, invece, vengono avviate a seconda dei casi le operazioni di decifratura e di verifica della firma. Se questa non è verificata, ancora una volta, l'utente sarà avvertito con un messaggio di allerta.

L'ultimo caso possibile è la ricezione di una chiave pubblica. In tal caso, se il numero del mittente è già presente in rubrica, la chiave gli viene automaticamente associata (eventualmente sovrascrivendo quella precedente); altrimenti viene proposta all'utente una maschera per l'inserimento del nome e del numero del nuovo contatto, al quale sarà ovviamente associata la chiave.

Schema dettagliato dell'applicazione

Non potendo esplicitare approfonditamente tutti i dettagli implementativi dell'applicazione realizzata, si può fare riferimento al seguente schema, che riassume il ruolo di tutte le componenti principali del software.

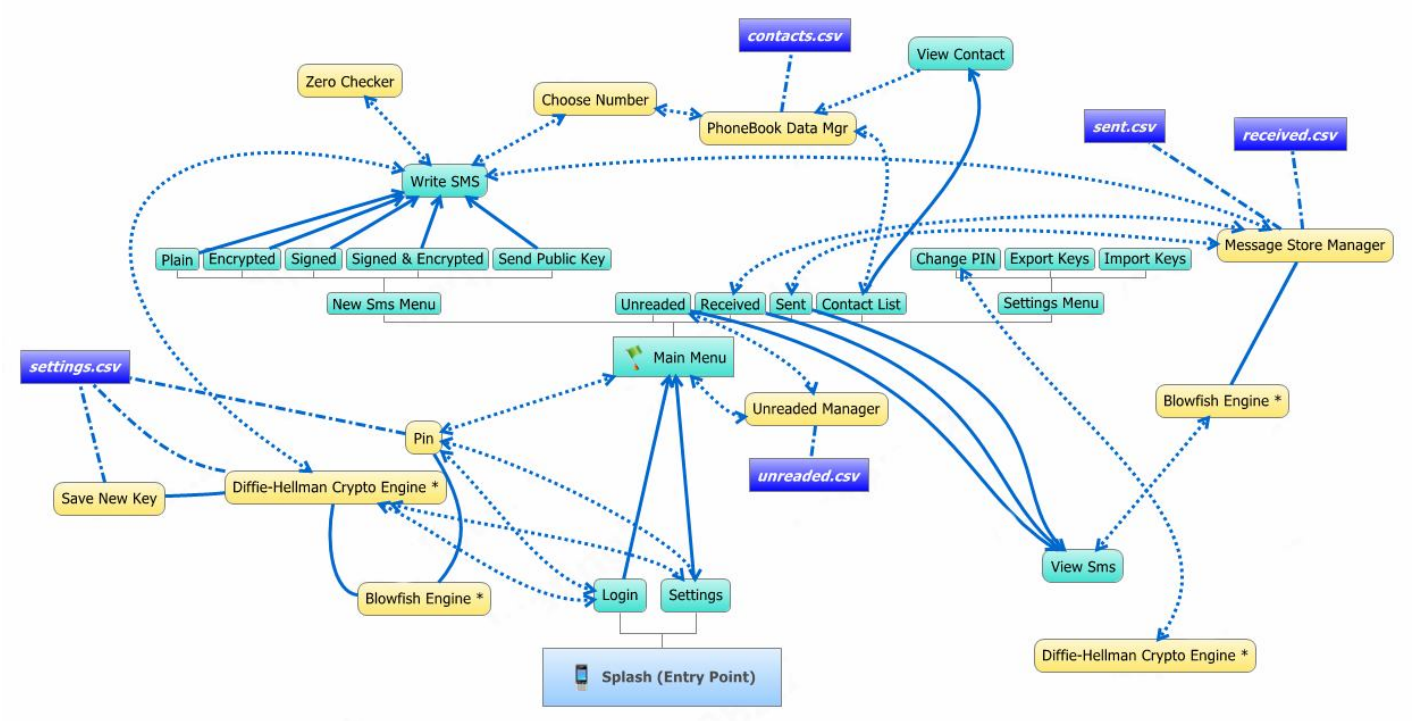


Fig. 15: schema generale dell'applicazione

I riquadri in azzurro indicano componenti dell'interfaccia grafica, corrispondono ciascuno ad un *dialog*, e sono connessi da linee solide; i riquadri gialli, invece, corrispondono a classi utilizzate solo per l'eleborazione, come quelle che si occupano ad esempio di mappare i dati persistenti dai relativi file (indicati nei riquadri blu scuro) ad adeguate strutture in memoria, e sono collegate tra loro mediante linee solide. Le frecce tratteggiate, infine, indicano l'utilizzo da parte di oggetti "grafici" delle classi ausiliarie. Si evidenzia tra queste l'importanza di *Blowfish Engine* (dedicato alla cifratura simmetrica), *Diffie-Hellman Cripto Engine* (generazione e gestione di chiavi e costruzione dei messaggi), *Pin* (gestione del codice di accesso al programma e della chiave di sistema), *PhoneBook Data Mgr* (gestione rubrica) e *MessageStore Manager* (gestione cartelle dei messaggi).

4 Sviluppi futuri

Nonostante il software prodotto sia solo alla sua primissima versione, è dotato comunque di un livello di funzionalità discreto. I test effettuati hanno rivelato una buona immunità agli errori ed un comportamento uniforme e conforme alle aspettative. Anche in termini di prestazioni i risultati sono da giudicare positivi.

Proprio in virtù di questo buon inizio, sarebbe possibile migliorare il prodotto con l'introduzione delle seguenti migliorie:

- Ottimizzazione della cifratura Blowfish;
- Riconoscimento dinamico di mittenti e destinatari dei messaggi salvati;
- Scambio di chiavi via Bluetooth;
- Azzeramento della chiave di sistema (attualmente residente in memoria) dopo un certo intervallo di tempo dalla sua generazione;
- Ottimizzazione dell'interfaccia grafica per la modalità orizzontale del display e per display di diverse risoluzioni;

5 Conclusioni

Il progetto oggetto di questa relazione ha raggiunto l'obiettivo prefissato, ossia la realizzazione di uno strumento software che consentisse di elevare notevolmente la soglia di sicurezza nello scambio di SMS.

Ha inoltre consentito di approfondire ed affrontare con profitto diversi aspetti della crittografia e della sicurezza in generale, anche cimentandosi con vari aspetti della progettazione software, dai linguaggi di programmazione (sempre con un occhio rivolto all'open source) alle risorse hardware, con le quali in ambiente PC risulta spesso più difficile o addirittura inutile confrontarsi.

Nonostante il prodotto finale non sia sufficientemente testato e non sia stato (ovviamente) sottoposto ad un benchmark di sicurezza, potrebbe probabilmente essere concretamente utilizzato, e l'autore si riserva di proseguire nello sviluppo del progetto.