Articolo per MC Microcomputer - Intelligiochi

Crittografia visiva

di Francesco Stajano
fms@orl.co.uk

Più volte su queste pagine si è parlato di crittologia, sia per gli aspetti di sfida ludica intellettuale che per gli evidenti legami di questa disciplina con il calcolo automatico. Sebbene l'esigenza di mantenere segreti risalga all'antichità (è ben noto l'esempio del Cifrario di Cesare, in cui ogni lettera viene sostituita da quella tre posti più avanti nell'alfabeto) e sebbene una teoria formale della materia sia stata stabilita da Leon Battista Alberti sin dal '400 [Kahn 1967], rimane il fatto che è stato l'avvento dei computer e della loro crescente potenza elaborativa a rendere possibile l'adozione dei moderni, sofisticati codici solidamente basati sulla Teoria dei Numeri. Al giorno d'oggi, in cui anche in un normale programma di email è legittimo aspettarsi di trovare il supporto per complessi algoritmi crittografici quali RSA, fa effetto pensare che, nei tempi andati, ci fossero persone con la pazienza di mettersi a cifrare e decifrare pagine e pagine di messaggi con il solo ausilio di carta e matita.

Eppure, a pensarci bene, la dipendenza incondizionata da una procedura non eseguibile a mano può, in alcuni casi, rivelarsi un grave handicap. Non ci sarà difficile immaginare scenari in cui Alice e Bob, i convenzionali protagonisti degli esempi inventati dai crittologi, sarebbero ben felici di poter comunicare in segreto senza dover ricorrere ad un computer. Alice, dal quartier generale, deve mandare un messaggio a Bob che si trova in uno sperduto rifugio alpino o in una pensione di terza categoria dove è a malapena disponibile un fax; e Bob, in missione, non ha potuto caricarsi appresso anche un laptop. Oppure Bob è su un'isola tropicale dove non c'è nemmeno l'elettricità e Alice può solo mandargli una cartolina. Oppure Bob, pur essendo una valorosa spia abilissima nelle arti marziali e nella guida spericolata alla 007, davanti a un computer diventa ciò che poco rispettosamente si definisce una *grannie* (letteralmente "nonnetta", anche se l'esperienza indica che spesso "capufficio" potrebbe risultare molto più appropriato — ma non divaghiamo). Risulta chiaro, insomma, il vantaggio di disporre di un metodo di comunicazione segreta che non richieda al ricevente l'uso di apparecchiature informatiche.

Qualcuno potrebbe ipotizzare che il vincolo di non utilizzare strumenti informatici implichi necessariamente un codice crittografico più facilmente attaccabile, specie considerando un nemico che abbia invece a disposizione risorse informatiche di grande potenza: dopotutto, buona parte dei codici crittografici medievali e rinascimentali sono ormai crittanalizzabili con modesto sforzo su un normale personal. È logico aspettarsi che, a una procedura di decifrazione necessariamente semplificata perché eseguibile senza computer, corrisponda un codice con minori possibilità di resistere al massiccio fuoco di artiglieria dei nostri odierni microprocessori.

Ma per fortuna le cose stanno diversamente: esiste un codice semplicissimo a descriversi, inventato nel 1917 da Mauborgne e Vernam e noto come *one time pad* (letteralmente "blocchetto monouso" [Kahn 1967] [Schneier 1996]), che è anche l'unico di cui si possa dimostrare matematicamente l'assoluta inviolabilità persino da parte di un nemico dotato di risorse di calcolo illimitate. Si assume che Alice e Bob abbiamo ciascuno una copia di un blocchetto di appunti (*pad*) contenente una lunghissima serie di bit casuali. Alice, quando vuole trasmettere un messaggio M a Bob, trasforma M in binario e poi, per ogni bit di M, trasmette a Bob lo XOR di quel bit con il corrispondente bit del pad, che subito dopo viene scartato in modo da non essere più riutilizzato. Bob, in ricezione, esegue lo XOR di ogni bit ricevuto con il corrispondente bit del pad (che a sua volta poi scarta). È facile riconoscere che il doppio XOR con il medesimo bit del pad ricostruisce il bit originale. Per dimostrare ciò basta esaminare i due possibili casi (0 e 1) per il bit del pad. Dato un bit di messaggio *m*, se il corrispondente bit del pad era uno 0, *m* XOR 0 (eseguito da A) ridà *m*, che viene trasmesso come testo cifrato, e *m* XOR 0 (eseguito da B in ricezione) continua a dare *m*, come volevamo. Se invece il bit del pad era un 1, quando A esegue *m* XOR 1 si ottiene l'opposto di *m*, ossia il bit

NOT *m*, che viene trasmesso come testo cifrato; e quando B esegue (NOT *m*) XOR 1 si ottiene l'opposto dell'opposto di *m*, ossia *m*, come desiderato. Qualunque sia il bit del pad, dunque, i due XOR si annullano a vicenda e B ottiene lo stesso *m* che A aveva originariamente. Per l'eventuale ascoltatrice nemica Eve (da "eavesdropper", chi intercetta le comunicazioni), il bit cifrato, ossia quello in uscita dal primo XOR, è reso totalmente casuale dalla "contaminazione" col bit casuale proveniente dal pad e potrebbe essere un 1 o uno 0 con uguale probabilità. Questa entropia massima del messaggio cifrato non lascia il minimo spiraglio a grimaldelli criptanalitici di alcun genere visto che, dato il bit osservato, non c'è alcun appiglio per poter ritenere l'uno o l'altro bit in ingresso più o meno probabile dell'altro. Visto infine che tutti i bit vengono trattati indipendentemente, queste due conclusioni (della perfetta ricostruibilità e della perfetta casualità e dunque inattaccabilità del testo cifrato) possono essere estese all'intero messaggio.

Si potrebbe dunque applicare l'idea dello *one time pad* al nostro problema visto che l'algoritmo, seppur in assoluto laborioso data la mole di operazioni da svolgere per messaggi di lunghezza anche modesta, è di semplicissima esecuzione anche con carta e penna. Alice prepara due blocchetti di carta identici contenenti dei bit casuali e ne fornisce uno a Bob prima della partenza, eccetera. Questo metodo è stato di fatto ampiamente utilizzato dalle spie durante la guerra fredda: i blocchetti in dotazione agli agenti sul campo venivano stampati su acetato, altamente infiammabile, in modo da poter essere rapidamente distrutti se l'agente si vedeva scoperto. Nulla di nuovo, dunque.

Ma una interessantissima e, perché no, divertente novità viene introdotta nel 1994 da due crittologi israeliani, Moni Naor e Adi Shamir (Shamir, si noti, è la "S" di RSA—non si tratta del primo venuto) [Naor Shamir 1994]. Con una geniale idea essi mostrano come implementare lo *one time pad* semplicemente sovrapponendo due fogli per trasparenza! Alice spedisce il messaggio cifrato a Bob sotto forma di un foglio di carta su cui è stampata una immagine di punti bianchi e neri apparentemente casuali. Questa immagine può essere trasmessa per lettera, via fax o in qualunque altro modo e non succede nulla se Eve ne ottiene una copia, giacché da essa è del tutto impossibile (come avviene nel caso del messaggio cifrato col pad) risalire al messaggio originario. Quando Bob riceve l'immagine egli non fa altro che sovrapporre ad essa un altro foglio di puntini casuali che custodiva gelosamente (la chiave, ossia l'equivalente del pad) e, guardando in trasparenza, "legge" direttamente il messaggio in chiaro, senza nessun calcolo o manipolazione simbolica di sorta. L'operazione può essere resa più agevole se Bob utilizza un lucido anziché un foglio opaco come chiave. Fantastico, no? Il procedimento di Naor e Shamir consente di eseguire la procedura di *one time pad* in parallelo su tutti i bit del messaggio e in modo "visivo" anziché algoritmico-computazionale. Codificando i bit in maniera grafica, l'operazione di XOR che decodifica il testo cifrato viene eseguita semplicemente sovrapponendo testo cifrato e chiave.

A questo punto i più svegli di voi avranno già cominciato a scalpitare, notando giustamente che la combinazione di due pixel per trasparenza equivale a un OR e non a uno XOR. Difatti, assumendo per chiarezza di usare dei lucidi e codificando 0 con "trasparente" e 1 con "opaco", risulta chiaro che la combinazione di due trasparenti dà un trasparente (0 *op* 0 = 0), ma che la combinazione di qualunque cosa con un opaco dà un opaco (0 *op* 1 = 1, 1 *op* 0 = 1, 1 *op* 1 = 1). Queste quattro equazioni identificano univocamente l'operazione *op* come OR. Problemaccio, perché il cuore dell'idea dello *one time pad* è proprio il fatto che si possa, ricombinando il bit cifrato con il bit della chiave, ottenere il bit originale, qualunque esso sia: questo è chiaramente impossibile se la combinazione avviene con un OR, giacché se il bit cifrato è un 1, nessun pad sarà in grado di "compensarlo" per ottenere uno 0 in uscita.

È qui che entra in gioco la parte davvero geniale dell'invenzione di Naor e Shamir: un metodo di codifica che permette di ottenere in modo puramente visivo l'effetto dello XOR. Anzitutto i nostri eroi fanno osservare che, posto che ciascun pixel del nostro messaggio originale (supposto monocromatico) può rappresentare solo uno fra due valori (0 e 1), nulla però ci costringe a codificare questi valori rispettivamente con un pixel bianco (trasparente) ed un pixel nero (opaco). Possiamo utilizzare una qualunque altra codifica purché i due valori risultino chiaramente distinguibili: ad esempio possiamo rappresentare 0 con grigio 50% e 1 con grigio 25%. Oppure, smettendo di considerare come atomici gli elementi dell'immagine che usiamo per la codifica: striscette verticali per 0 e striscette orizzontali per 1; o addirittura, per assurdo: paperette per 0 e maialini per 1 [figura 1]. Ci siete? Bene. Con una buona dose di pensiero laterale essi poi aggiungono che non è nemmeno necessario che la codifica sia la stessa per le variabili di ingresso (i pixel da combinare fra loro) e quella di uscita (il pixel risultante dalla combinazione): l'importante è che, in ciascuno di questi due domini, sia ben chiaro quale valore corrisponde a 0 e quale a 1. Siete un po' confusi? Non vi preoccupate, l'esempio che segue chiarirà tutto.

A questo punto tutto è pronto per il favoloso gioco di prestigio che ci consentirà di eseguire visivamente lo XOR: si tratta solo di scegliere in modo opportuno la codifica dei pixel 0 e 1 per il caso di ingresso e per il caso di uscita in modo tale che sia possibile combinarli secondo la tabella di verità della funzione XOR. Già immaginiamo che lo XOR con 0, che lascia immutato l'altro ingresso, sarà facile: occupiamoci dunque del caso più interessante, lo XOR con 1, che equivale a un NOT (x XOR 1 = NOT x), ossia l'operazione che intuitivamente è impossibile realizzare per trasparenza ("non posso trasformare il nero in bianco, qualunque sia la cosa che decido di sovrapporre al nero"). Guardiamo, ad esempio, come ottenere l'effetto di 1 XOR 1 = 0.

Come codifica di ingresso rappresentiamo 0 con un pixel la cui metà sinistra sia trasparente e la metà destra nera e rappresentiamo 1 con un pixel la cui metà sinistra sia nera e quella destra trasparente [figura 2]. (Si noti che usiamo dei pixel "non atomici", ossia a loro volta composti da subpixel.) Per eseguire 1 XOR 1 sovrapponiamo due pixel con il nero a sinistra, ottenendone un altro uguale. Poniamo allora, come codifica di uscita, che questo pixel valga 0. Ad alcuni sembrerà che stiamo barando, ma viceversa stiamo procedendo in modo più che rigoroso. Ci serve ora far sì che anche 0 XOR 0 valga 0: con lo stesso procedimento notiamo che 0 XOR 0 produce un pixel la cui metà destra è nera. Bene: anche questo, per noi, dovrà valere 0. La nostra convenzione di codifica in uscita sarà dunque che tutti i pixel grigi al 50% (pixel mezzi bianchi e mezzi neri) valgono 0, a prescindere dal fatto che abbiano il nero a destra o a sinistra. Cosa avviene quando invece combiniamo uno 0 ed un 1 (oppure, il che è lo stesso, un 1 ed uno 0)? Lo 0 contribuisce con il nero di sinistra e l'1 col nero di destra, per cui il risultato è un nero pieno al 100%. Bene, noi allora dichiareremo che, in uscita, il nero 100% vale 1 [figura 3]. A questo punto abbiamo due sistemi di codifica: in quello di ingresso sia 0 che 1 sono rappresentati da dei pixel di grigio 50% (seppur con disposizione diversa), mentre in quello di uscita lo 0 è rappresentato da grigio 50% (in due possibili varianti, entrambe considerate equivalenti) mentre l'1 è rappresentato dal nero pieno. Dal punto di vista matematico siamo già a posto: le codifiche ora descritte ci permettono di soddisfare la tabellina di verità dello XOR tramite la semplice sovrapposizione visiva delle disposizioni descritte [figura 4]. L'altro punto di cruciale importanza da osservare è che la codifica di uscita (a differenza di quella di ingresso) è stata scelta in modo da essere immediatamente percettibile ad occhio nudo: 0 è grigio, 1 è nero pieno; in pratica, assumendo un'operazione di filtraggio che l'occhio realizza con facilità, si tratta della stessa codifica utilizzata normalmente (0 = bianco, 1 = nero) fatto salvo un dimezzamento del contrasto. Questo punto è fondamentale perché consente a Bob di leggere il messaggio in modo del tutto diretto semplicemente

"guardandolo", senza che si renda necessario il tramite di una qualsivoglia manipolazione sull'immagine.

Abbiamo dunque, come per magia, realizzato visivamente il sistema di *one time pad*. Alice può spedire un qualunque documento monocromatico (figura, testo stampato, cartina) eseguendone lo XOR con il pad e poi codificando il risultato secondo la codifica di ingresso (in cui sia 0 che 1 sono rappresentati da due diversi pattern di grigio 50%); questo documento grigio, all'apparenza solo rumore, è il documento cifrato. Bob può estrarre da esso il documento originario semplicemente sovrapponendogli il proprio lucido-chiave (anch'esso tutto grigio al 50% e di aspetto simile a rumore): a questo punto egli vedrà apparire immediatamente l'immagine codificata da Alice.

Il software: Visual Cryptography Kit

Per mettere in pratica tutto questo ci serve ora, manco a dirlo, un programma (dal lato di Alice, sia ben chiaro) che generi le immagini codificate. Nello spirito degli intelligiochi e per consentirne la massima flessibilità d'uso lo abbiamo realizzato sotto forma di kit componibile in modo che ciascuno possa adattarlo alle proprie esigenze e si senta libero di sperimentare nei modi più vari, con facoltà di curiosare ed osservare cosa accade a tutte le immagini ottenute nei passi intermedi. Abbiamo dunque deciso di adottare la potente libreria grafica Netpbm (vedi riquadro apposito) che ci consente di supportare praticamente tutti i formati grafici conosciuti. Seguendo la filosofia di Netpbm, che poi se vogliamo è quella di Unix, la funzionalità del nostro VCK (la sigla sta per "Visual Cryptography Kit") è suddivisa in piccole utility componibili (anche tramite pipeline) ciascuna delle quali svolge un compito ben preciso. La suite di programmi è disponibile, sia come sorgenti C++ che come eseguibili precompilati per win32, sul sito web dell'autore all'indirizzo http://www.cl.cam.ac.uk/~fms27/vck/. Di seguito presentiamo i componenti del kit, descrivendone sintassi e semantica.

```
vck-bittopbm bitfile.bit x y [offset] → bitfile.pbm
```

Prende x, y ed un opzionale offset (default 0), tutti numeri interi, ed un file binario bitfile.bit di contenuto arbitrario (usare il nome "-" per stdin). Genera su stdout un file grafico, qui indicato come bitfile.pbm, di dimensioni x*y, i cui pixel provengono ordinatamente dai bit di bitfile.bit a partire dall'offset specificato (in bit). Per la precisione, i pixel dell'immagine vengono scanditi per righe, dall'alto in basso e da sinistra a destra, mentre i bit del bitfile.bit vengono scanditi byte per byte iniziando con il bit più significativo.

Questo programma va usato per generare un'immagine di bit casuali (a partire da un file di bit casuali che si suppone di aver generato in precedenza con un metodo opportuno) la quale, passata attraverso vck-pixelcode (vedi), sarà la chiave che Alice fotocopierà su un lucido e darà a Bob affinché la porti con sé.

Osserviamo che la sicurezza di un sistema *one time pad* dipende in modo essenziale dalla qualità (entropia) del pad di bit casuali, che idealmente andrebbero generati a partire da un procedimento fisico quale decadimento radioattivo, rumore termico eccetera anziché con l'uso di un generatore algoritmico pseudocasuale. Se il pad, anziché essere casuale, ha una struttura, e se il nemico riesce a scoprirla, la dimostrazione di inviolabilità non è più valida e il codice risulta compromesso. Per questo motivo si è ritenuto più corretto fornire un convertitore anziché solamente un generatore, per quanto buono, cosicché l'utente sia libero di utilizzare la sorgente casuale che riterrà più opportuna.

```
vck-pixelcode file.pbm → file.pbm
```

Prende un file grafico da 1 bit per pixel (usare "-" per stdin) contenente un'immagine codificata con 0 = nero e 1 = bianco. Genera su stdout un file grafico di larghezza e altezza doppie in cui ogni pixel dell'originale corrisponde a un quadratino di quattro pixel. I pixel 0 vengono codificati con un quadratino la cui metà destra è nera e la sinistra bianca, mentre i pixel 1 vengono codificati con un quadratino la cui metà sinistra è nera e la destra bianca.

Questo programma va usato per convertire le immagini (sia il messaggio originale in chiaro, sia il pad casuale che farà da chiave) dal formato nativo a quello "di ingresso" (vedi figura 2) in cui possono essere combinate tramite XOR.

```
vck-boolean op clear.pbm pad.pbm → cipher.pbm
```

Prende due file grafici da 1 bit per pixel, entrambi delle stesse dimensioni, e genera su stdout un terzo file grafico ottenuto eseguendo pixel per pixel sui due file di ingresso l'operazione booleana binaria specificata dal parametro op, che può essere "and", "or" o "xor".

NB: siccome questo programma prende due file anziché uno, non lo si può utilizzare come filtro. Di conseguenza non è possibile usare "-" per denotare stdin.

```
vck-trivialrng l → bitfile
```

Prende un intero l che specifica il numero di bit da generare e produce su stdout un file contenente il numero richiesto di bit pseudocasuali, arrotondato ove necessario al successivo multiplo di 8.

Questo programma utilizza la normale funzione di libreria per la generazione di numeri pseudocasuali e come tale non è molto robusto da un punto di vista crittografico. Viene tuttavia fornito come "pezzo di Lego" per dare all'utente modo di sperimentare con gli altri programmi del kit, in attesa di procurarsi una migliore sorgente di numeri casuali.

Istruzioni per l'uso

Il procedimento completo da eseguire è pertanto il seguente.

Alice e Bob stabiliscono la dimensione, in pixel, delle pagine che intendono scambiarsi. Ciò viene fatto in base a considerazioni legate al rumore del canale di trasmissione (es. spurie sul fax) e alla ripetibilità di posizionamento dei pixel sul foglio di carta e particolarmente sul lucido, limitata in genere dal fatto che la fotocopiatura o stampa laser sul lucido deforma il lucido stesso in modo irregolare. Dato che, secondo la codifica di ingresso mostrata in figura 2, per ogni pixel sorgente produrremo quattro pixel nel messaggio cifrato, la larghezza ed altezza massime accettabili dal metodo di stampa e trasmissione vanno divise per due per ottenere la larghezza ed altezza massime delle pagine da scambiare. Siano x ed y queste ultime.

Alice prepara un pad di almeno x*y bit casuali. A titolo di prova ciò può essere fatto con

```
vck-trivialrng x*y > pad-raw.bit
```

che genererà il file pad-raw.bit. Come già osservato, in una applicazione reale il file pad-raw.bit va generato con un procedimento ad alta entropia. Su Linux si può utilmente impiegare /dev/random, che mantiene un “pool di casualità” alimentato da eventi esterni quali gli interrupt del mouse ed emette bit casuali solo fintantoché l’entropia di questo pool non sia esaurita, attendendo in tal caso l’arrivo di ulteriori eventi casuali prima di generare altri bit.

Ottenuto che sia il pad-raw.bit, Alice lo usa per generare un’immagine di dimensioni x * y...

```
vck-bittopbm x y pad-raw.bit > pad-raw.pbm
```

...che poi filtra secondo la codifica di ingresso, ottenendo una nuova immagine di dimensioni 2x * 2y:

```
vck-pixelcode pad-raw.pbm > pad.pbm
```

Questa nuova immagine, pad.pbm, viene stampata su un lucido e consegnata a Bob, che la conserverà gelosamente. Alice, da parte sua, conserverà invece una copia del file pad-raw.pbm (quello con i pixel originali, non raddoppiati) e distruggerà pad-raw.bit e pad.pbm.

La procedura finora descritta prepara *una* pagina del pad che Alice e Bob hanno come segreto comune. Essa va ripetuta per ciascuna nuova pagina che si desidera nel pad. (Si ricordi che ogni pagina può essere usata una sola volta, dopodiché va distrutta. Altrimenti Eve può ricostruire il pad, e dunque decifrare il traffico, confrontando fra loro successivi messaggi cifrati.) Quando Alice e Bob pensano di avere generato un numero sufficiente di pagine per i loro scopi, Bob può andarsene in missione.

Supponiamo ora che Alice voglia cifrare una pagina di testo (o un’immagine) per mandarla a Bob e sia clear-raw.pbm la pagina da trasmettere, che dovrà essere delle dimensioni originariamente stabilite, ossia x * y. Alice genera la pagina cifrata come XOR del testo e della chiave, prendendo come chiave la prima pagina disponibile del pad, che chiameremo pad-raw.pbm:

```
vck-boolean xor clear-raw.pbm pad-raw.pbm > cipher-raw.pbm
```

e filtra la pagina cifrata secondo la codifica di ingresso di figura 2, ottenendo una pagina di dimensioni doppie 2x * 2y e di densità pari al grigio 50%:

```
vck-pixelcode cipher-raw.pbm > cipher.pbm
```

A questo punto Alice distrugge pad-raw.pbm e cipher-raw.pbm e spedisce cipher.pbm a Bob.

Grazie a quanto visto in precedenza, cipher.pbm gode della proprietà per cui, sovrapponendola (ossia combinandola con OR visivo) a pad.pbm, essa genererà, secondo la codifica di uscita, una ricostruzione visiva di clear-raw.pbm, sebbene a grandezza doppia e contrasto dimezzato.

La figura 5 mostra il diagramma delle operazioni mentre la figura 6 presenta un cipher.pbm e un pad.pbm che potete provare a sovrapporre per sperimentare l'effetto ancor prima di avvicinarvi al software. Fotocopiate pad.pbm su un lucido da proiezione e poi immaginate di essere Bob che ha appena ricevuto cipher.pbm.

Conclusioni

Se quanto visto finora vi è piaciuto, sappiate che era solo un assaggio! Chi scrive ha avuto il privilegio di assistere nei mesi passati a due interessantissime conferenze rispettivamente di Adi Shamir e di Eli Biham, entrambi regolari ospiti del Computer Security Group dell'Università di Cambridge, nelle quali questo nuovo affascinante paradigma è stato esteso in direzioni ancora diverse: Shamir [Shamir 1997] ha mostrato una costruzione teorica con la quale usare la crittografia visiva come calcolatore altamente parallelo per decrittare il DES (!) mentre Biham [Biham 1997] ha mostrato come usare pixel fatti con filtri polarizzatori per implementare le primitive della crittografia visiva senza perdite di contrasto. Il tutto senza dimenticare che lo stesso articolo originale di Naor e Shamir [Naor Shamir 1994], del quale abbiamo qui esposto solo il risultato fondamentale, propone diverse estensioni di questo schema, fra cui una stupenda variante steganografica in cui sia il pad che il testo cifrato, anziché avere l'aspetto di distribuzioni casuali (gli agenti di frontiera possono sospettare che Bob sia una spia se lo trovano in possesso di questi pad casuali; Bob rischia di finire torturato anche se i pad sono di per sé illeggibili), contengono innocentissime figure che, una volta combinate, spariscono per cedere il posto al documento segreto. Ma di tutto questo non si poteva parlare in una sola puntata...

Riquadro: Netpbm

Chiunque abbia provato a scrivere un programma che manipoli delle immagini conosce bene il sentimento di frustrazione che nasce dalla sconfinata varietà di formati grafici esistenti, dalla complicazione delle strutture di dati da essi impiegate e dalla certezza che, per quanti formati di file si decida di supportare nel programma, gli utenti continueranno murphisticamente a lamentarsi che manca quello che essi ritengono più utile. Per venire incontro a questa necessità è nata, da una prima implementazione dovuta a Jef Poskanzer e poi grazie a vari perfezionamenti di un gruppo di appassionati, la libreria di utility che oggi si chiama Netpbm. L'idea centrale di Netpbm è quella di definire un formato grafico fondamentale dalla struttura molto semplice, ottimizzato dunque per la manipolazione da programma, e di corredarlo di tutte le utility necessarie per il filtraggio, il ridimensionamento eccetera. A monte e a valle di questo universo tutto compatibile ci sono poi convertitori unidirezionali da e per tutti gli altri formati grafici (TIF, JPG, BMP, GIF e via dicendo). È evidente il risparmio: solo sul fronte dei convertitori, basta realizzarne $2n$ anziché n^2. Ogni operazione complessa che l'utente desideri può poi essere espressa con una catena di operazioni elementari che trasformano il file originario nel formato universale, manipolano il file con le utility disponibili fino ad ottenere il risultato voluto e poi riconvertono il risultato nel formato di uscita desiderato. La libreria è liberamente disponibile sotto forma di

sorgente C da numerosi archivi ftp nonché sul sito dell'autore dove troverete i programmi del kit di crittografia visiva VCK descritto nell'articolo.

Riferimenti

[Naor Shamir 1994] Moni NAOR, Adi SHAMIR, *Visual Cryptography*, Advances in Cryptology - Eurocrypt '94 Proceedings, LNCS vol. 950, Springer-Verlag, 1995, pp. 1-12.

Il fondamentale articolo in cui Naor e Shamir introducono la crittografia visiva.

[Kahn 1967] David KAHN, *The Codebreakers: The Story of Secret Writing*, Macmillan, 1967.

Giustamente famoso, questo monumentale volume racconta in gran dettaglio la storia della crittografia dalle origini all'era moderna.

[Schneier 1996] Bruce SCHNEIER, *Applied Cryptography* (2nd Ed), Wiley, 1996.

Un'ottima e dettagliata enciclopedia che descrive con grande chiarezza i protocolli, le tecniche e gli algoritmi della crittografia moderna, illustrando il tutto con frammenti di codice in C.

[Shamir 1997] Adi SHAMIR, *A New Paradigm for Massively Parallel Random Search*, Security Seminar Series, Cambridge University Computer Laboratory, 1997-07-08.

In questo affascinante seminario Shamir ha presentato una costruzione in grado di decrittare il DES utilizzando operazioni "visive" (simili a quelle della crittografia visiva) per ottenere l'effetto di un multielaboratore a parallelismo massiccio.

[Biham 1997] Eli BIHAM, *Visual Cryptography with Polarisation*, Security Seminar Series, Cambridge University Computer Laboratory, 1997-08-29.

Un altro famoso crittologo israeliano presenta un'estensione del paradigma della crittografia visiva "uscendo dal sistema": nuove costruzioni, basate su filtri polarizzatori, offrono contrasto e risoluzione molto migliori delle costruzioni ottimali del paradigma tradizionale, specie nei casi di sovrapposizione di più di due "share".

L'autore

Francesco "Frank" Stajano (http://www.orl.co.uk/~fms/), ingegnere elettronico, è da cinque anni Research Scientist presso l'Olivetti-Oracle Research Laboratory (ORL) di Cambridge, Inghilterra. Attualmente è anche ricercatore presso il Computer Security Group dell'Università di Cambridge. Nel 1992 ha scritto *Manuale Modem*, un libro elettronico gratuito di grande diffusione, specie in ambito Fidonet, considerato da molti appassionati la bibbia delle BBS amatoriali. Nel 1994 ha creato i *DOOM Honorific Titles*, una competizione internazionale dedicata ai campioni del più grande videogioco della storia. Nel tempo libero è anche un esperto fumettologo (particolarmente in ambito Disney) ed ha al suo attivo diverse pubblicazioni sull'argomento fra cui, con L. Gori ed A. Becattini, il recente volume *Don Rosa e il rinascimento disneyano*.

Figure

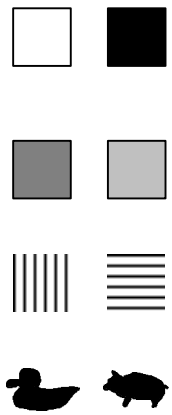

Figura 1: alcuni fra gli infiniti modi di codificare graficamente i bit 0 e 1.

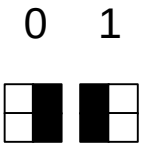

Figura 2: codifica di ingresso.

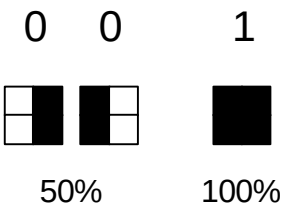

Figura 3: codifica di uscita.

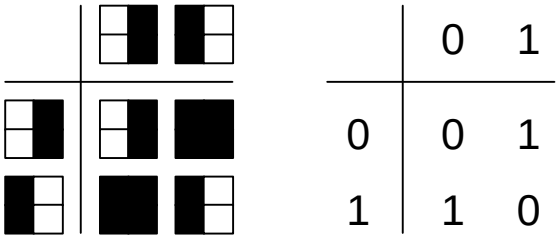

Figura 4: con le convenzioni scelte, l'operazione di sovrapposizione verifica la tabellina di verità dello XOR.

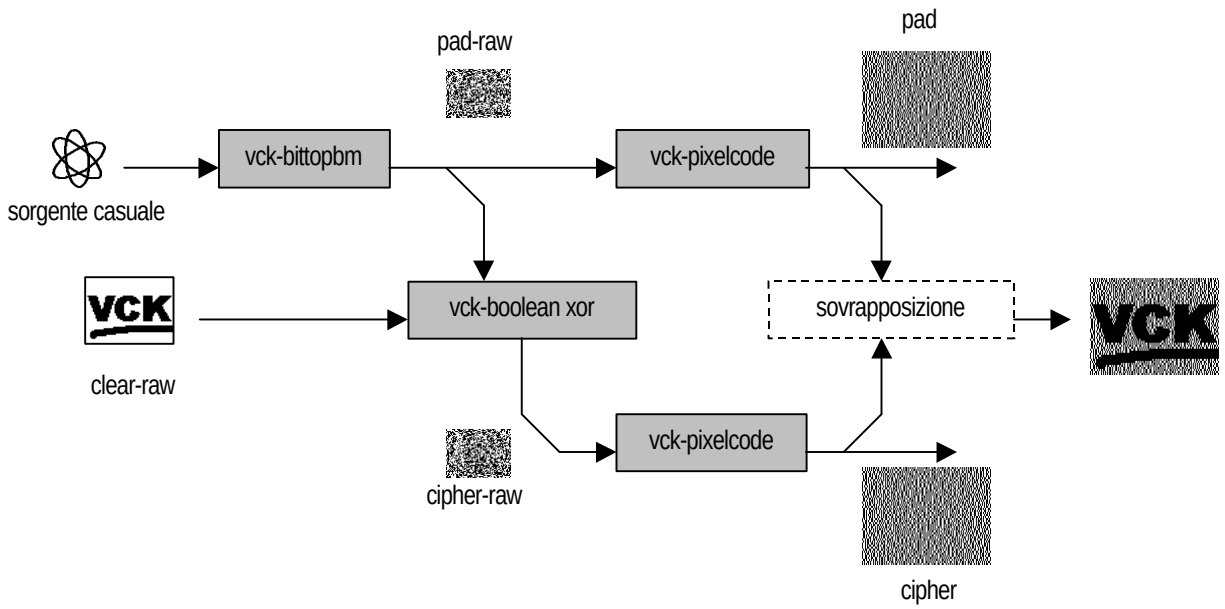

Figura 5: diagramma di flusso delle operazioni. Ciascun riquadro grigio equivale a una primitiva elementare implementata da uno specifico programma del kit VCK, mentre il riquadro tratteggiato è semplicemente l'operazione manuale di sovrapposizione del lucido "pad" al foglio "cipher", equivalente di fatto a "vck-boolean or" a meno di problemi di allineamento.

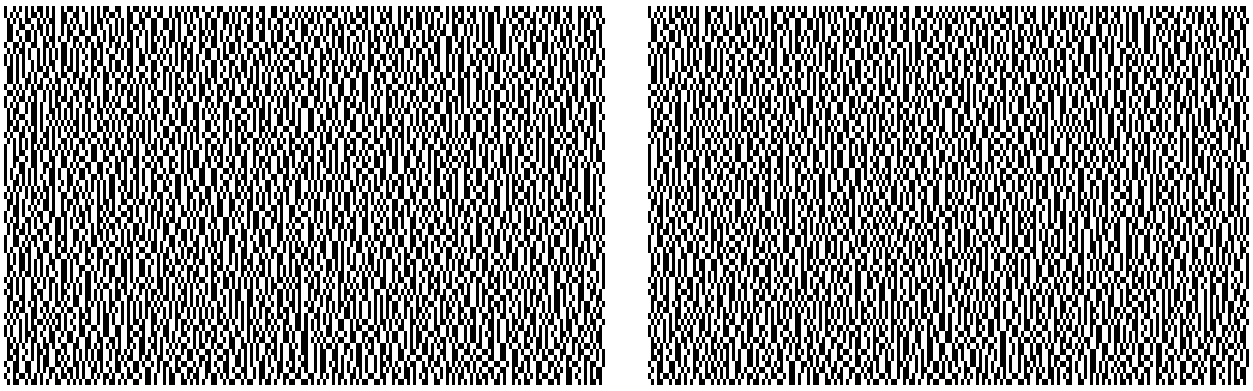

Figura 6: un cipher e un pad per una prova pratica immediata. Fotocopiate il pad su lucido e sovrapponetelo al cipher. I lettori di vecchia data dovrebbero a questo punto provare un irrefrenabile moto di nostalgia...