Tecniche di Progettazione Automatica **Anno Acc. 1995-96**
PROF. ALFREDO DE SANTIS

DIGITAL CASH

a cura di Antonio Fabrizio e Rosa Perillo *8 Ottobre 1996*

Indice

1 Introduzione

Nella societá moderna é sentito il problema della sicurezza e della portabilitá della moneta contante; per questo motivo si cerca di sostituirla con strumenti che la rappresentino e che siano piú comodi e sicuri (ad esempio gli assegni). Negli ultimi anni si sono diffusi sistemi che gestiscono elettronicamente la moneta (ad esempio, mediante carte di credito) e, da quando é aumentata la possibilitá di connessione tra i computer sono nate molte societá che forniscono servizi a pagamento come accesso on-line a documenti, messaggierie, prenotazione posti, etc... che prima richiedevano pagamenti con moneta contante, mentre oggi possono essere pagati con moneta elettronica. Ciò é reso possibile con l'introduzione di protocolli che consentono di effettuare transazioni; molti di essi permettono transazioni anonime, (tra questi quelli di Chaum [6]) richiedendo tipicamente la partecipazione di una banca centrale coinvolta in tutte le transazioni.

1.1 Requisiti per sistemi Digital Cash

Tra le proprietà desiderabili per un sistema di moneta elettronica troviamo:

- **Sicurezza:** la falsificazione della moneta cartacea é difficile. Sfortunatamente la moneta elettronica, essendo rappresentata da stringhe di bit, è facilmente duplicabile. Tale operazione può essere evitata. Teoricamente la creazione illegale, la copia e il riutilizzo della moneta digitale è computazionalmente impossibile;

- **Anonimato:** l'identitá di un individuo che usa moneta elettronica deve essere protetta impedendo il monitoraggio del percorso della moneta, fino a risalire alla persona che l'ha utilizzata. Nei sistemi di transazione tradizionali, basati su assegni o carte di credito, gli utenti sono reperibili con grande facilità. Alcuni protocolli consentono l'irreperibilità assoluta, anche in caso di collusione tra i partecipanti alla transazione [5, 6]. In un gran numero di proposte di cash elettronico, invece, è garantito un anonimato più debole: la reperibilità dei partecipanti è possibile, ma risulta sufficientemente difficile, rendendo il costo di una ricerca superiore ai benefici che essa può portare;

- **Scalabilitá:** un sistema é scalabile se puó gestire l'aumento degli utenti e delle risorse, senza subire una notevole perdita di prestazioni. L'esistenza di un server centrale, attraverso il quale le transazioni devono avvenire, limita la scalabilitá del sistema, così pure l'esistenza di meccanismi per individuare il riutilizzo della moneta. Molti protocolli di moneta elettronica propongono che il server memorizzi tutte le monete già spese, e controlli questa lista, al verificarsi di una transazione [6, 9, 10].
  Questo database crescerá progressivamente, incrementando quindi il costo della ricerca; anche se la vita di una moneta é limitata nel tempo, non c'é limite superiore alla quantitá di memoria richiesta per un tale database, in quanto esso dipende, oltre che dal numero di monete circolanti, dal periodo di validità della monete stesse;

- **Accettabilitá:** molti sistemi di moneta elettronica usano una singola banca [6, 9, 10], ma, per ottenere una maggiore scalabilitá, oltre che per favorire la diffusione di tali sistemi, sono necessarie in pratica molte banche, anche perché non tutti gli utenti sono clienti di una stessa banca.
  In un simile ambiente, é importante che la moneta emessa da una banca sia accettata

dalle altre. Quando tra le banche si effettua uno scambio di valuta, la loro riconciliazione avviene in modo automatico.

- **Operazioni Off-line:** la possibilità che le parti, durante una transazione, possano essere indipendenti dall'autoritá che ha emesso o che assicura la moneta (la banca);
- **Trasferibilitá:** è desiderabile che la moneta elettronica sia accettata da terzi, senza prima contattare la banca. In questo modo, viene valorizzata la proprietá di anonimato, anche se ciò complica il meccanismo che garantisce la sicurezza;
- **Indipendenza dall'hardware:** per impedire il riutilizzo delle monete durante le operazioni off-line, alcuni protocolli si poggiano su un hardware contro le manomissioni e le intrusioni non autorizzate [7]. Questo approccio ha il difetto di non garantire la sicurezza a quegli utenti che non possiedono un hardware di questo tipo.

1.2 Blind Signature

I nuovi sistemi di pagamento elettronico si basano su un'estensione delle firme digitali dette *blind signature*.Uno schema blind signature é un protocollo per ottenere una firma da un firmatario in modo che egli non possa vedere ció che firma, ad esso partecipano un mittente ed un firmatario. L'anonimato del mittente viene garantito, e, per questo motivo, tali schemi vengono utilizzati nei protocolli Digital Cash, ove assicurano, in particolare, l'anonimato di chi usa moneta digitale. Sfortunatamente questo anonimato puó essere usato da un malintenzionato per ottenere una firma che altrimenti non potrebbe avere. Descriveremo tali protocolli con l'aiuto di due partecipanti immaginari: Alice e Bob. Come accade usando del contante, Alice dovrá poter trasferire moneta digitale a Bob senza che una terza persona riesca a conoscere l'identitá di Alice. Come conseguenza di ció Bob potrá depositare la moneta elettronica sul suo conto bancario senza che la banca sappia chi sia Alice. Ma se Alice tentasse di dare a due persone lo stesso denaro elettronico con un programma che copia i bit, verrebbe individuata dalla banca; se Bob cercasse di depositare la stessa moneta digitale su due differenti conti, sarebbe individuato, ma Alice resterebbe anonima.
La nozione di blind signature fu inventata da D. Chaum [3] che propose anche una prima implementazione [4]. Lo schema usa l'algoritmo RSA.
Bob ha una chiave pubblica e, una privata d ed un valore pubblico n. Alice vuole che Bob firmi il messaggio m senza vederlo.
Alice sceglie un valore casuale k compreso tra 1 ed n. Allora maschera m calcolando

$$t \equiv mk^e \mod n$$

Bob firma t:

$$t^d \equiv (mk^e)^d \mod n$$

Alice smaschera t^d calcolando:

$$s \equiv \frac{t^d}{k} \mod n$$

che é la firma di Bob su m, infatti:

$$s = \frac{t^d}{k} = \frac{(mk^e)^d}{k} = \frac{km^d}{k} = m^d$$

da cui:

$$s \equiv m^d \mod n$$

2 Alcuni protocolli per l'implementazione del Digital Cash

Analizziamo piú in dettaglio alcuni protocolli.

2.1 Protocollo 1

Questo protocollo ha delle analogie con alcuni protocolli crittografici. Esso é una semplificazione dell'assegno al portatore:

1. Alice prepara 100 assegni da \$1000;
2. Alice mette ciascun assegno insieme ad un pezzo di carta carbone in cento differenti buste e le dá tutte alla banca;
3. la banca apre 99 di queste buste scelte a caso e verifica che ciascuna contiene un ordine di \$1000;
4. la banca firma l'unica busta rimasta non aperta e mediante la carta carbone la firma si trasferisce sull'assegno. La banca restituisce ad Alice la busta non aperta e detrae \$1000 dal suo conto;
5. Alice apre la busta e spende l'assegno da un commerciante;
6. il commerciante controlla la firma della banca e si assicura che l'assegno sia valido;
7. il commerciante incassa l'assegno dalla banca;
8. la banca verifica la sua firma e accredita i \$1000 sul conto del commenciante.

Questo protocollo é realizzabile: la banca non vede l'assegno che ha firmato, cosí quando il commerciante lo riporta, essa non è in grado di determinare che l'assegno era di Alice. La banca é certa della validitá dell'assegno in quanto su esso è posta la propria firma. Presumibilmente l'assegno firmato nella busta di Alice varrà \$1000 (e non \$100,000 o \$1,000,000) in quanto attraverso il protocollo *cut-and-choose* esiste l' 1% di probabilitá di poter imbrogliare da parte di Alice. Certamente la banca punirá l'eventuale tentativo di imbroglio in modo da scoraggiare chiunque dal tentare la frode. Se così non fosse un imbroglione qualsiasi, puó tentare la truffa fino ad avere fortuna.

2.2 Protocollo 2

Il precedente protocollo impedisce ad Alice di scrivere un'assegno superiore a quello dichiarato, ma non evita che Alice possa copiare la moneta per spenderla piú volte; con qualche accorgimento possiamo ottenere ciò:

1. Alice prepara i cento assegni anonimi di \$1000 ciascuno. Ogni assegno include una stringa unica scelta casualmente, lunga abbastanza da renderla insignificante per un'altra persona che la volesse utilizzare;
2. Alice prende ognuno di questi assegni, e unitamente ad un pezzo di carta carbone li chiude in 100 differenti buste, dandole alla banca;
3. la banca apre 99 buste e verifica che in ognuna di esse ci sia un assegno di \$1000, e che su ognuno di essi sia presente una stringa differente da quella degli altri;

4. la banca firma la busta rimanente, restituisce la busta ad Alice e detrae \$1000 dal suo conto;
5. Alice apre la busta e spende l'assegno in un negozio;
6. il commerciante verifica la firma della banca e si assicura che l'assegno sia valido;
7. il commerciante incassa l'assegno;
8. la banca verifica la sua firma e cerca nel suo data-base in modo da essere sicura che l'assegno con quella stringa non sia stato precedentemente depositato, ed in questo caso accredita i \$1000 al commerciante.

Se a questo punto Alice cercasse di spendere o se il commerciante tentasse di depositare una copia dell'assegno, la banca si accorgerebbe di questa operazione illecita.

2.3 Protocollo 3

Il precedente protocollo protegge la banca dagli imbrogli, ma non identifica il falsificatore. La banca non conosce se la persona che ritira l'assegno (la banca non ha idea di chi sia Alice) cerca di imbrogliare il commerciante o se é il commerciante che vuole imbrogliare la banca. Il protocollo seguente evita questo problema.

1. Alice prepara cento assegni anonimi di \$1000 ciascuno. Ogni assegno include una stringa di riconoscimento unica, scelta casualmente e lunga abbastanza da renderla insignificante per un'altra persona che la volesse utilizzare;
2. Alice pone ogni assegno unitamente ad un pezzo di carta carbone in una busta, consegnando poi le cento buste sigillate alla banca;
3. la banca apre 99 di queste buste scelte a caso e verifica che ognuna di essa contenga un assegno di \$1000, e che ogni assegno sia caratterizzato da una stringa di riconoscimento differente;
4. la banca firma la busta ancora chiusa, la restituisce ad Alice e detrae \$1000 dal suo conto;
5. Alice apre la busta e spende l'assegno in un negozio;
6. il commerciante verifica la firma della banca e si assicura che l'assegno sia valido;
7. il commerciante chiede ad Alice di scrivere una stringa casuale di identificazione sull'assegno;
8. Alice accetta;
9. il commerciante incassa l'assegno dalla banca;
10. la banca verifica la sua firma e cerca nel suo data base in modo da essere sicura che l'assegno con quella stringa di riconoscimento non sia stato precedentemente depositato; in questo caso accredita i \$1000 al commerciante e memorizza la stringa di riconoscimento dell'assegno e la stringa identificativa in un database;
11. se la stringa di riconoscimento dell'assegno é nel database la banca si rifiuta di accettarlo. Poi confronta la stringa identificativa sull'assegno con quella memorizzata nel database se sono le stesse allora capisce che il commerciante ha copiato l'assegno; se sono differenti, la banca capisce che la persona che ha ritirato l'assegno per la prima volta lo ha copiato.

Questo suppone che il commerciante non possa cambiare la stringa identificativa una volta che Alice ha firmato l'assegno. L'assegno puó avere una serie di piccoli quadrati che il commerciante puó richiedere ad Alice di riempire con una sequenza di X e di 0. L'assegno puó essere fatto in modo da danneggiarsi a seguito di un tentativo di contraffazione. Implementazioni pratiche di questo protocollo possono richiedere ad Alice di attendere alla cassa durante l'interazione tra il commerciante e la banca, come accade con le carte di credito. Il successivo protocollo elimina questo inconveniente.

2.4 Protocollo 4

Se risulta che la persona che ha ritirato la prima volta l'assegno ha cercato di imbrogliare il commerciante, é doveroso per la banca scoprire chi sia questi. Per fare ció dobbiamo addentrarci nel mondo della crittografia. Tecniche di scomposizione segreta possono essere usate per nascondere il nome di Alice in un assegno digitale.

1. Alice prepara 100 assegni anonimi di \$1000 ciascuno. Ognuno di essi contiene:

$$\begin{aligned}
\text{Ammontare}: &\ \$1000 \\
\text{Stringa di riconoscimento assegno}: &\ \text{X} \\
\text{Stringhe identitá}: &\ I_1 = (I_{1_L}, I_{1_R}) \\
&\ I_2 = (I_{2_L}, I_{2_R}) \\
&\ \quad . \\
&\ \quad . \\
&\ \quad . \\
&\ I_{100} = (I_{100_L}, I_{100_R})
\end{aligned}$$

   Ogni assegno è caratterizzato da una differente stringa di riconoscimento X. Su ogni assegno ci sono anche 100 coppie di stringhe identitá I_n ognuna delle quali é generata come segue: Alice crea una stringa che contiene il suo nome, l'indirizzo, ed ogni altro tipo di informazione identificativa che la banca vuole avere; scompone poi il tutto in due pezzi usando un protocollo segreto di scomposizione ed invia il risultato alla banca attraverso un protocollo *bit-commitment.* Per esempio, I_{37} consiste di due parti:I_{37_L} e I_{37_R} ogni parte é un pacchetto *bit-committed* la cui apertura può essere chiesta in ogni momento ad Alice. La combinazione di ciascuna coppia, I_{37_L} e I_{37_R}, ma non I_{37_L} e I_{38_R} rivela l'identitá di Alice;
2. Alice nasconde tutti i 100 assegni usando un protocollo di blind signature e consegna tutto alla banca;
3. la banca chiede ad Alice di scoprire 99 dei 100 assegni e conferma che essi sono tutti conformi. Controlla poi l'importo, la stringa di riconoscimento, e chiede ad Alice di rivelare tutte le stringhe identitá;
4. se Alice non ha tentato di imbrogliare,la banca firma l'assegno non scoperto, e riconsegna l'assegno ad Alice detraendo \$1000 dal suo conto;
5. Alice scopre l'assegno e lo spende da un commerciante;
6. il commerciante verifica la firma della banca e si assicura che l'assegno sia valido;
7. il commerciante chiede ad Alice di rivelare la parte sinistra o quella destra di ciascuna delle stringhe identitá poste sull'assegno. In effetti il commerciante dà ad

Alice una stringa selezionatrice di 100 bit a caso, $b_1, b_2, \ldots, b_{100}$. Alice apre la parte sinistra o destra di I_k, a seconda se b_k sia rispettivamente 0 o 1;

8. Alice accetta;
9. il commerciante incassa l'assegno dalla banca;
10. la banca verifica la firma e cerca nel suo database assicurandosi che l'assegno con la stessa stringa di riconoscimento non sia stato precedentemente depositato. Se é cosí la banca accredita \$1000 sul conto del commerciante. La banca memorizza la stringa di riconoscimento e tutte le informazioni identificative nel database;
11. se la stringa di riconoscimento é nel database, la banca rifiuta di accettare l'assegno. Confronta poi la stringa identitá posta sull'assegno con quella memorizzata nel database: se é la stessa, capisce che l'assegno è stato duplicato dal commerciante, se invece risulta differente, deduce che la truffa è stata tentata dalla persona che ha ritirato l'assegno. Poichè un commerciante che accetta a propria insaputa un assegno già speso altrove, dà ad Alice una stringa selezionatrice di bit b_i sicuramente diversa almeno in un bit da quella dell'altro commerciante , la banca ricerca proprio uno di tali bit, poniamo sia l'$n - esimo$, mette insieme le due parti I_{n_L} ed I_{n_R} e scopre l'identitá di Alice.

Alice puó imbrogliare?
Il suo assegno elettronico non é nient'altro che una stringa di bit che puó essere copiata. Spendere l'assegno la prima volta non sará un problema. Il commerciante le dará una stringa selezionatrice casuale di 100 bit b_i (passo 7), e Alice aprirá la parte sinistra o destra di ogni I_i, (passo 8). La banca memorizzerá tutti questi dati insieme a quelli relativi alla stringa di riconoscimento dell'assegno (passo 10). Quando Alice cerca di usare per la seconda volta lo stesso assegno elettronico, il commerciante, lo stesso o anche un secondo, le dará una differente stringa di bit b_i (passo 7). Alice deve accettare (passo 8); non facendo questo, insospettisce il commerciante. Quando il commerciante porta l'assegno in banca (passo 10), la banca noterá immediatamente che esso ha la stessa stringa di riconoscimento di un altro precedentemente depositato. La banca confronta le mezze stringhe identitá. La probabilitá che due stringhe casuali di 100 bit siano le stesse é $\frac{1}{2^{100}}$. Ora la banca ricerca una coppia che, combinando i risultati delle due differenti aperture, abbia entrambe le metà svelate.; lo XOR delle due metá rivela l'identitá di Alice. In questo modo la banca é venuta a conoscenza di chi ha cercato di spendere due volte lo stesso assegno. Non è necessario che Alice attenda in negozio che il commerciante e la banca abbiano terminato la loro parte di protocollo. Solo nel caso in cui Alice imbrogli, la banca la identificherà; ella non puó evitare di essere scoperta poichè non è in grado di cambiare nè la stringa di riconoscimento, nè le stringhe identificative. Il commerciante sará avvisato immediatamente (passo 6). Alice potrebbe tentare la truffa inserendo un assegno, fra i 100 che consegna alla banca, nel quale la sua identitá sia falsa. La probabilitá che Alice riesca a farsi firmare tale assegno é $\frac{1}{100}$, che é evidentemente bassa. Per aumentare ulteriormente la sicurezza si puó aumentare il numero di assegni che Alice deve presentare alla banca.
Il commerciante puó imbrogliare?
Le possibilitá di imbrogliare da parte del commerciante non sono migliori di quelle di Alice. Infatti egli non puó depositare l'assegno due volte; la banca gli fará sapere che ha usato nuovamente la stessa stringa $b_1, b_2, ..., b_{100}$. Egli non puó tentare di far incolpare Alice in quanto lei è la sola a poter aprire le stringhe identitá. Qualsiasi collusione fra Alice ed il

commerciante non permette di imbrogliare la banca. Se la banca firma un assegno con la stringa di riconoscimento, è sicura di pagere tale assegno solo una volta.
La banca può ottenere informazioni sull'intestatario dell'assegno, quando questi si comporta correttamente?
Alice é protetta dal protocollo di blind signature nei passi da 2 a 5. Non c'é nessun modo per la banca di rintracciarla pur tenendo delle registrazioni complete di tutte le transazioni. Addirittura, anche collaborando con il commerciante, la banca non può identificare chi sia Alice. Alice, quindi, puó effettuare i suoi acquisti in modo del tutto anonimo.
Questi protocolli potrebbero essere attuati mediante l'emissione da parte delle banche di *computer-card* personali in grado di collegarsi al sistema centrale attraverso punti di accesso distribuiti. Presentiamo ora un protocollo di pagamento digitale che prevede l'utilizzo della computer-card, pur operando in maniera analoga al sistema con le buste sigillate.

1. La computer card dell'individuo sceglie, attraverso un processo random, una sequenza di cifre, e la duplica per creare il numero m della banconota, il quale avrà la speciale proprietà della "metà ripetuta". (Questo passo corrisponde alla scelta casuale di un foglio di carta).

2. La card crea un numero totalmente random k. (corrispondente alla scelta di una busta corredata di carta carbone).

3. La computer card eleva k alla potenza pubblica e della banca (a valore di \$1), moltiplica il risultato per m (cioè $k^e \cdot m$) e lo invia alla banca. (come il sigillo sulla busta).

4. La banca preleva dal conto il valore della banconota e, dopo averla firmata con la corrispondente potenza privata d, la restituisce alla computer card.

5. La card verifica l'esattezza di ciò che la banca le ha inviato ed ottiene la banconota firmata dividendo per k. (Ciò corrisponde all'estrazione del foglio firmato dalla busta sigillata).

6. Quando il negoziante riceve la banconota in pagamento si accerta che questa sia firmata digitalmente e ne invia una copia alla banca come deposito.

7. La banca verifica la propria firma e accetta il deposito se la banconotavalida non è stata già depositata.

La computer-card potrebbe assomigliare ad un calcolatore con le dimensioni di una carta di credito ed includere un display di caratteri, una tastiera, capacità di comunicazione a corto raggio (come quella di un telecomando per televisori), e deve essere semplice da usare. Durante l'acquisto in un negozio, ad esempio, l'apparecchio del punto vendita trasmette una descrizione dei beni acquistati ed il loro costo alla card, la quale mostra queste informazioni al suo proprietario. Egli consentirà la transazione semplicemente digitando, sulla tastiera della card, un numero segreto di autorizzazione . Questo stesso numero sarà usato dal proprietario per consentire ogni transazione; senza di esso una computer-card perduta o rubata sarebbe di scarsa utilità. Una card perduta, può essere sostituita con un'altra nella quale saranno inseriti i dati di backup salvati in una forma codificata sicura sul proprio personal computer a casa o altrove.

2.5 Digital cash ed il crimine perfetto

La moneta elettronica ha un suo lato debole: l'anonimato potrebbe essere usato a scopi illeciti. Vediamo come Alice commette il crimine perfetto.

1. Alice rapisce un bambino;
2. Alice prepara 10000 assegni anonimi di \$1000 ciascuno (o di qualsiasi cifra lei voglia);
3. Alice nasconde tutti gli assegni attraverso un protocollo di blind signature. Manda tutto alle autoritá con la richiesta che ucciderá il bambino se non verranno eseguite le seguenti istruzioni:
   - Una banca deve firmare tutti gli assegni.;
   - Pubblicare i risultati su di un giornale;
4. Le autoritá accettano;
5. Alice compra il giornale, scopre gli assegni e va a spenderli. Non c'é modo per le autoritá di rintracciare Alice attraverso gli assegni;
6. Alice rilascia il bambino.

3 NetCash

Uno dei sistemi di cash elettronico, implementato per Internet é *NetCash*, sviluppato presso l'Istituto di Scienze dell'Informazione dell'Università della California del Sud. NetCash supporta pagamenti elettronici in tempo reale, prevedendo per essi l'eventuale anonimato su una rete insicura. Tale sistema è stato progettato per abilitare nuovi tipi di servizi su Internet, non attuati fino ad oggi a causa dell'assenza di un metodo di pagamento sicuro e scalabile, che consentisse il potenziale anonimato degli utenti. Possiamo affermare dunque che gli scopi principali per cui è stato realizzato sono due:

- facilitare pagamenti elettronici anonimi su di una rete;
- garantire una certa sicurezza senza richiedere l'uso di un hardware particolare contro le alterazioni e le intromissioni.

NetCash realizza un compromesso tra la moneta elettronica assolutamente anonima, ed altri titoli firmati più scalabili, ma che identificano i partecipanti ad una transazione (ad esempio, assegni elettronici). I protocolli di moneta elettronica in esso proposti, difatti, possono essere utilizzati in una infrastruttura di deposito bancario elettronico non anonimo, adeguatata per transazioni di routine, ove l'identità delle parti non viene protetta. In realtà è auspicabile che tale identità venga salvaguardata per evitare che dalle transazioni possano essere estratte e memorizzate informazioni relative alle abitudini degli individui (per esempio ciò che essi leggono, ciò che acquistano o ancora l'ammontare di denaro che essi spendono o ricevono).
NetCash dunque è progettato per consentire transazioni anonime ed anche sicure, in un ambiente dove la creazione illegale, la copia ed il riutilizzo della valuta elettronica sono possibili. Al fine di proteggere la privacy delle parti di una transazione, NetCash fornisce strumenti finanziari che evitano la rintracciabilità e conservano l'anonimato degli utenti. Inoltre, con

NetCash, i fornitori di servizi ed i loro utenti possono selezionare dei meccanismi di pagamento basati sul livello di anonimato desiderato, a partire da strumenti non anonimi o debolmente anonimi, fino a giungere a strumenti incondizionatamente anonimi, (l'anonimato assoluto è ottenuto integrando alcuni protocolli per transazioni anonime, esterni a NetCash, come quello di Chaum [6], i quali, tipicamente, richiedono una banca centrale coinvolta in ogni transazione). NetCash fornisce valuta elettronica scalabile accettata in molti domini amministrativi; grazie ad esso, clienti di banche differenti, possono accettare l'uno la moneta dell'altro.
L'interoperabilità fra i vari server di valuta è garantita da protocolli che consentono il trasferimento di fondi all'interno dell'infrastruttura elettronica globale di deposito bancario non anonimo, su cui il sistema poggia.

3.1 Struttura

L'infrastruttura di NetCash si basa su una rete di server di valuta elettronica, gestiti indipendentemente, i quali forniscono punti di scambio fra valuta elettronica anonima ed altri strumenti finanziari non anonimi.
Il trasferimento di fondi fra i vari server avviene tramite assegni elettronici, dando vita, in questo modo, ad una federazione finanziaria, dove la valuta coniata da server differenti viene universalmente accettata. Un'organizzazione che desideri avviare e dirigere un nuovo server di valuta ottiene l'autorizzazione da un'agenzia simile alla United States Federal Deposit Insurance Corporation (che indicheremo brevemente con FIC). La valuta sarà sostenuta dai fondi registrati a nome del server nell'infrastruttura globale di deposito bancario. Un certificato di assicurazione consentirà alle monete coniate dal server di essere accettate nei vari domini amministrativi.
Più in dettaglio la procedura seguita è la seguente:

- il nuovo server crea una coppia di chiavi pubbliche e le invia al FIC su di un canale sicuro (la chiave privata corrispondente viene usata per firmare le monete);
- ricevute le chiavi pubbliche il FIC emetterà un certificato di assicurazione che abilita il nuovo server a produrre e gestire indipendentemente la valuta.

Il certificato di abilitazione é composto da:

$$(Certif\_id, CS\_name, K_{CS}, issue\_date, exp\_date)K_{FIC}^{-1}$$

Il certificato include l'identificatore unico $Certif\_id$ del server, il suo nome CS_name, la sua chiave pubblica K_{CS}, la data di emissione $issue\_date$ e quella di scadenza exp_date. Tutte queste informazioni sono cifrate con la chiave privata del FIC K_{FIC}^{-1}. Basandosi su questo certificato, gli altri server e le istituzioni finanziarie accetteranno la moneta del nuovo server come legale. Sarà l'utente finale a scegliere il proprio server in base alla vicinanza od alla fiducia.
Un server offre i seguenti servizi:

- verifica della moneta relativa ad eventuali copie illegali;
- cambio di moneta;
- cambio di assegni in moneta elettronica e viceversa.

I fondi possono essere trasferiti fra i server di valuta quando, uno, il pagante emette un assegno che autorizza l'altro, il pagato, a trasferire valuta dal conto del primo sul proprio. In questo modo i trasferimenti consentono transazioni anonime fra utenti in domini amministrativi differenti. Per verificare le validità delle monete emesse da un altro server di valuta, il server di valuta locale di un utente contatta quello remoto per convertire le monete, ricevendo in cambio un assegno pagabile. Questo assegno sarà poi liquidato attraverso l'infrastruttura globale di account. Nuove monete, emesse dal server locale, saranno inviate all'utente.

3.2 Funzionalità e struttura dei componenti di NetCash

La struttura di una moneta nel nostro protocollo é la seguente:

$$(CS\_name, s\_addr, exp\_date, serial\_num, coin\_val)K_{CS}^{-1}, Certif\_id$$

Il valore della moneta é specificato nel campo $coin\_val$, s_addr é l'indirizzo su Internet del server e CS_name il suo nome; la data di scadenza della moneta exp_date indica per quanto tempo quella moneta sarà valida. $Serial\_num$ è il numero di serie della moneta che associa in modo univoco la moneta al server che l'ha emessa. Il server tiene un elenco dei numeri di serie appartenenti alle monete emesse ed ancora valide, per poter individuare eventuali tentativi di truffa. Se, durante una transazione monetaria, uno dei partecipanti gli invia una moneta desiderandone una verifica, egli non deve fare altro che controllare se il numero di serie è presente nell'elenco: in caso affermativo la moneta è valida, il suo numero di serie viene cancellato dalla lista ed al cliente sarà emessa una moneta nuova, il cui numero di serie sarà aggiunto all'elenco delle monete valide. Trovandosi, invece, di fronte ad una moneta il cui numero di serie non è presente nell'elenco, si è scoperto un tentativo di riutilizzo della moneta, ed essa viene rifiutata. Se il server non è raggiungibile all'indirizzo indicato nella moneta, il nome del server sarà usato per trovarne l'indirizzo in un elenco di server. Tutte le informazioni contenute in una moneta sono cifrate con la chiave privata del server K_{CS}^{-1}. Un cliente che vuole decifrare la moneta puó usare $Certif\_id$, ottenendo cosí la chiave pubblica K_{CS} e solo a questo punto può inviarla al server per effettuarne la verifica di validità.

3.3 Protocolli di scambio

Definiamo ora i protocolli di scambio. Assumiamo che gli utenti siano clienti di differenti server, ognuno dei quali emette la propria moneta valida ed universalmente riconosciuta.
I partecipanti ai nostri protocolli sono:

- clienti;
- commercianti;
- server di valuta.

3.3.1 Notazioni

Indicheremo clienti e commercianti rispettivamente con A e B, i server con CS, ed un partecipante qualsiasi con X. Le banche nei nostri protocolli vengono considerate semplicemente come commercianti.
Il termine *transazione* in questo contesto indicherà esclusivamente traffico di moneta fra i

partecipanti; un generico pagamento avverrà da A a B. Le parentesi graffe {} indicheranno la codifica di ciò che contengono, la lettera K indicherà una chiave pubblica ed il suo pedice il "padrone" della chiave; *un pedice che termina con la lettera* N indica una chiave generata di recente, senza alcuna associazione con l'identità del titolare. Una chiave privata è l'inversa della chiave pubblica e sarà rappresentata come K_{key}^{-1}; infine, le chiavi per un sistema crittografico simmetrico saranno indicate da SK seguito da un pedice.

3.3.2 Protocolli di base

Descriviamo i protocolli di scambio fondamentali che servono come base di partenza per i successivi. L'accesso all'infrastruttura globale è garantito dai server, i quali emettono monete ed accettano monete di altri server ed altri strumenti finanziari (ad esempio, gli assegni).
In un semplice scambio di valuta con il *CS*, l'anonimato di un cliente è protetto per diverse ragioni:

- il *CS* scambia monete con un cliente fornendogli in cambio nuove monete aventi differenti numeri di serie e non memorizza alcun record ove le monete accettate vengono poste in relazione con quelle emesse;
- lo scambio avviene in modo del tutto anonimo; il *CS* non conosce chi sta inviando le monete, potrebbe essere un suo cliente o un commerciante anonimo che ha accettato queste monete e le vuol cambiare per essere sicuro che esse non siano già state spese.

Lo scambio di base prevede la verifica delle monete (individuazione delle monete già utilizzate), lo scambio delle monete ai fini della non rintracciabilità e la conversione degli assegni elettronici in moneta e viceversa. Quest'ultima operazione espone l'identità di X. Si assume che X conosca inizialmente la chiave pubblica K_{CS} del server, acquisita da transazioni precedenti o ottenuta direttamente da *CS* in un certificato. Analizziamo in dettaglio questa semplice transazione:

3.3.3 Scambio Utente-Server

Supponiamo che X voglia eseguire uno scambio di valuta con *CS*. I passi che X effettua sono (vedi Figura 1):

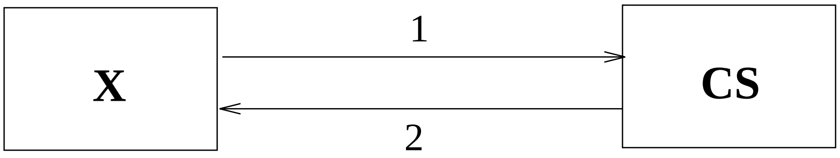

Figura 1: Schema Utente-Server

1. X spedisce una tripla $\{valuta, SK_X, tipo\ di\ transazione\}\ K_{CS}$, dove il campo *valuta* rappresenta una moneta o un assegno, SK_X é la chiave pubblica di X, e *tipo di transazione* indica il tipo di valuta che X desidera (ad esempio, nuove monete o un assegno ed, in caso di assegno, il nome del beneficiario), il tutto codificato con la chiave pubblica del server.
   Se la *valuta* é una moneta emessa dal *CS* stesso, allora *CS* verifica se é giá stata

spesa (controllando se il numero di serie della moneta è presente nella lista che egli conserva). Se la *valuta* é una moneta emessa da un'altro server, *CS* contatta quest'ultimo per convertirla, accettando in cambio un assegno di cui egli stesso è beneficiario. Se la *valuta* é un assegno allora *CS* lo liquida versandone l'ammontare sul suo conto.

2. *CS* ritorna ad X $\{valuta\}\,SK_X$, ovvero il tipo di valuta desiderata cifrata con la chiave SK_X; la cifratura con SK_X, prova l'identità del *CS* e protegge il messaggio dagli attacchi di un nemico.

3.3.4 Un semplice protocollo di scambio utente-utente

La Figura 2 mostra un semplice protocollo di pagamento in cui A rimane anonimo. Quando il protocollo è ultimato, B non è protetto contro il riutilizzo delle monete e ad A non è garantita una ricevuta valida. Inizialmente si assume che A possieda l'indirizzo di B. I passi fondamentali sono:

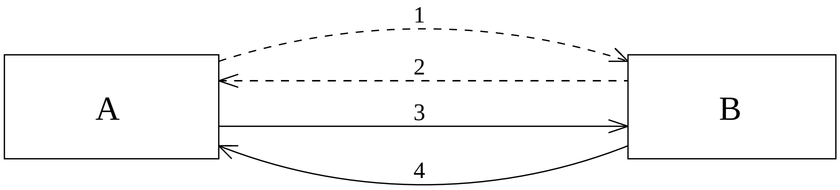

Figura 2: Semplice scambio Utente-Utente

1. A invia a B la propria chiave pubblica K_{AN} appositamente generata in modo da rimanere anonimo;

2. B può scegliere di rimanere anonimo o meno:
   nel primo caso, invierà il messaggio $\{K_{BN}\}\,K_{AN}$, nel secondo $\{K_B\}\,K_{AN}$.
   (Questi primi due passi consentono ad A di ottenere la chiave pubblica di B, nel caso in cui A già la conosca possono essere omessi);

3. A invia a B:

   $$\{moneta, SK_{AN}, K_{ses}, S\_id\}\,K_{BN}, \{Certif\_id, K_{CS}, issue\_date, exp\_date\}\,K_{FIC}^{-1}$$

   dove S_id è l'identificatore del servizio, SK_{AN} e K_{ses} sono due chiavi. B usa K_{CS} per verificare che le monete sono state coniate da un server autorizzato. Al fine di mettere in relazione le monete con una connessione, B conserva la chiave di sessione K_{ses}; nel momento in cui il servizio deve essere fornito, B controlla che A conosca la chiave di sessione;

4. B restituisce la ricevuta $importo, T_id, dataK_{BN}^{-1}SK_{AN}$ firmata con la sua chiave privata K_{BN}^{-1} e codificata con SK_{AN}, proteggendo in questo modo il contenuto del messaggio dagli attacchi del nemico; essa include l'importo pagato, la data ed un identificatore unico T_id usato insieme alla chiave di sessione per ottenere il servizio.

Notiamo che se i passi 1 e 2 sono usati per ottenere una chiave pubblica anonima, il protocollo resisterà solo ad attacchi passivi, nel senso che è mantenuta la privacy della transazione, ma sarà vulnerabile da un attacco attivo qualora, ad esempio, un anonimo nemico si sostituisce al fornitore di servizio.

3.4 Combinazione dei protocolli di base

Presentiamo i protocolli che proteggono l'anonimato del creditore e del debitore; essi utilizzano combinazioni di protocolli fondamentali, per fornire garanzie contro il riutilizzo delle monete, la mancata emissione della ricevuta o l'emissione di una ricevuta non valida, la mancata fornitura del servizio.

3.4.1 Scambio Utente-Utente

Supponiamo che A sia un utente anonimo che voglia usufruire di un servizio fornito da un commerciante B, e che conosca la chiave pubblica K_B. Il protocollo consiste nei seguenti passi:

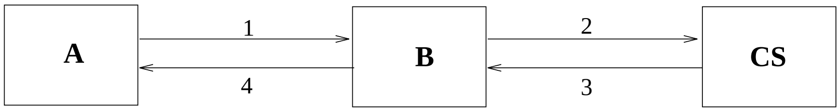

Figura 3: Schema Utente-Utente

1. A manda la quadrupla $\{moneta, SK_{AN}, K_{ses}, S\_id\}\, K_B$ dove SK_{AN} é la chiave pubblica di A, K_{ses} é la chiave di sessione, S_id é l'identificativo del servizio;
2. B verifica la legalitá della *moneta* contattando il server CS tramite la sua chiave pubblica K_{CS}, inviandogli $\{moneta, K_{BN}, tipo\ di\ transazione\}\, K_{CS}$ (questo passo é stato definito nella sessione precedente);
3. CS restituisce $\{nuova\_moneta\}\, SK_{BN}$ a B;
4. B invia ad A la ricevuta $\left\{\{importo, T\_id, data\}\, K_B^{-1}\right\} SK_{AN}$, dove *importo* é il costo del sevizio, T_id é un identificativo univoco che viene usato insieme a K_{ses} per accedere al servizio, e *data* é la data della transazione.

Questo tipo di protocollo protegge il commerciante B dal fatto che la moneta possa giá essere stata spesa, ma non protegge A dall'eventuale frode di B; B, difatti, potrebbe spendere le monete ricevute da A senza emettere alcuna ricevuta valida o addirittura senza fornire il servizio. La seguente variante risolve questo problema.

3.4.2 Scambio che protegge entrambe le parti dalle frodi

Il protocollo precedente puó essere completato al fine di eliminare la possibilità di frode da parte del commerciante B, ottenendo un protocollo che protegga l'anonimato di A, protegga B dal ricevere monete già spese e garantisca che A ottenga, a seguito del pagamento, il servizio richiesto con una ricevuta valida o la restituzione delle proprie monete. Per rendere possibile ciò estendiamo la definizione di moneta: una moneta può essere caratterizzata con un dato rilevante, ad esempio un *periodo di validità*. Così il pagante A può contattare il server per

ottenere una tripla di monete $<\mathbf{C}_B, \mathbf{C}_A, \mathbf{C}_X>$, da spendere presso B. Ogni moneta nella tripla ha lo stesso numero di serie e lo stesso valore. Durante un primo intervallo di tempo, $\mathbf{C}_B$ è l'unica moneta che può essere usata con il server. In un secondo intervallo potrà essere usata solo $\mathbf{C}_A$, mentre nell'ultimo periodo potrà essere usata solo $\mathbf{C}_X$. Ciò può essere implementato inserendo una finestra di tempo in ogni moneta nella tripla; le prime due monete hanno chiavi nascoste in esse.
Se ogni parte vuole usare le sue monete in una transazione con *CS1*, deve provare di conoscere la chiave nascosta. Ad esempio, la chiave pubblica di B K_B, è nascosta in $\mathbf{C}_B$. Durante la transazione con *CS1* egli dovrà provare di conoscere K_B^{-1}. La terza moneta $\mathbf{C}_X$, non ha una chiave nascosta in essa e può essere spesa presso chiunque. Informazioni addizionali nascoste in $\mathbf{C}_A$ potrebbero rappresentare la chiave pubblica di B, ciò per ridurre la quantità di informazioni che il server deve memorizzare per poter emettere una ricevuta ad A. E' necessario, inoltre, un bit addizionale da associare al numero di serie della moneta nel database di *CS1*, per mantenere traccia di quando essa viene spesa da A o B. Nella transazione con B, A tratterrà le monete $\mathbf{C}_A$ e $\mathbf{C}_X$ inviandogli $\mathbf{C}_B$. Se B, a seguito di ciò, non restituisce una ricevuta ad A, A potrà chiedere al server di controllare se B ha speso la moneta, ed eventualmente dove. Se B ha speso la moneta, il server emetterà ad A una ricevuta specificando in essa il valore della moneta e la chiave pubblica di B. A può ottenere ciò solo durante il periodo di tempo durante il quale $\mathbf{C}_A$ è valida; B dovrebbe tenere traccia di $\mathbf{C}_B$ prima che essa scada, nel caso che A tenti di spenderla più volte presso di lui. $\mathbf{C}_X$ è data per garantire maggiore flessibilità nelle transazioni monetarie, quando, ad esempio, A non avesse necessità, o non volesse, spendere tutta la terna di monete presso B.

Schematizziamo con l'ausilio di una figura il funzionamento del protocollo, dividendolo in passi (vedi Figura 4):

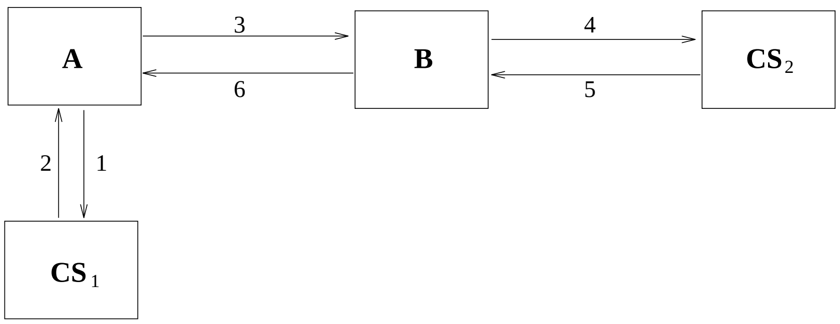

Figura 4: Schema Utente-Utente 2

1. A invia monete al proprio server per ottenere una tripla di monete con il messaggio $\{moneta, SK_{AN1}, K_B, data_A, data_B, importo\}\ K_{CS1}$; $data_A$ e $data_B$ denotano la scadenza entro cui le operazioni rispettivamente di A e di B devono essere completate;

2. Il server *CS1*, crea la tripla di monete $<\mathbf{C}_B, \mathbf{C}_A, \mathbf{C}_X>$; in esse nasconde delle informazioni come descritto in precedenza e restituisce la tripla ad A, accompagnata da $\langle, ,\mathbf{C}_X\rangle$, possibile differenza tra il costo del servizio fornito da B e l'ammontare totale delle monete inviate al passo 1. Il messaggio completo è $\{<\mathbf{C}_B, \mathbf{C}_A, \mathbf{C}_X>, \langle, ,\mathbf{C}_X\rangle\}\ SK_{AN1}$;

3. A invia $\mathbf{C}_B$ a B con $\{\mathbf{C}_B, SK_{AN2}, K_{ses}, S\_id\}\ K_B$;

4. B invia $\mathbf{C}_B$ al proprio server *CS2* per ottenernene la verifica (entro il periodo di validità specificato in essa);

5. *CS2* controllata la validità di $\mathbf{C}_B$ la cambia, inviando a B il tipo valuta desiderato;

6. B fornisce il servizio e restituisce una ricevuta valida ad A:

$$\left\{\{importo, T\_id, data\}\, K_B^{-1}\right\} SK_{AN2}.$$

Nel caso non gli venga recapitata alcuna ricevuta, durante il secondo intervallo di tempo, A invia $\mathbf{C}_A$ al server *CS1*; *CS1* controlla quindi se $\mathbf{C}_B$ è stata spesa nel primo periodo di tempo, ed in caso affermativo restituisce ad A una ricevuta, in cui specifica la chiave pubblica di B ed il valore della moneta, il tutto cifrato con la propria chiave privata. Nel caso in cui la moneta non è stata spesa, *CS1* invierà ad A una nuova moneta.
L'anonimato di chi paga può essere ottenuto combinando i passi 1 e 2 della Figura 4 con il protocollo appena descritto. Nel protocollo risultante la ricevuta fornita a chi paga non è molto funzionale, essendo egli anonimo.

3.4.3 Transazioni off-line

In una transazione off-line è auspicabile evitare il riutilizzo delle monete e preservare l'anonimato delle parti. Le transazioni condotte in modo off-line, dove le parti non contattano mai il server durante lo scambio, possono essere gestite da NetCash in molti modi. Il protocollo, essenzialmente basato su quello illustrato in Figura 4, potrebbe essere il seguente:

- Se A conosce a priori di essere in procinto di trattare un affare con B, i passi 1 e 2 possono essere eseguiti in anticipo.

- A e B procedono allo scambio, effettuando i passi 3 e 6, dove viene evitato il riutilizzo delle monete e viene mantenuto l'anonimato del pagante (notare l'assenza di *CS2*).

Un difetto di questo protocollo è che il pagante deve conoscere in anticipo con chi perfezionerà la transazione.
Un'altro approccio alle transazioni off-line è utilizzare il protocollo mostrato in Figura 2 su di un hardware contro le intromissioni e le alterazioni; il riutilizzzo delle monete, in questo caso, è evitato dalle proprietà dell'hardware.
Attualmente si sta cercando di incorporare gli schemi di Chaum [6]di punizione post-fatto in NetCash, in quanto il riutilizzo delle monete rende statisticamente possibile l'individuazione di un utente disonesto. Il vero problema di questo approccio è che la punizione post-fatto non può essere accettata dalle istituzioni finanziarie, in quanto per poter punire un malfattore bisognerebbe prima individuarlo con certezza e poi provarne giuridicamente la colpevolezza, operazioni per cui si incontrano notevoli difficoltà sia di carattere tecnico che legale.

3.5 Considerazioni finali

- NetCash combina i vantaggi delle transazioni anonime con la scalabilità dei protocolli di pagamento on-line non anonimi. E' un sistema sicuro, scalabile, valido in differenti domini amministrativi e fornisce una certa garanzia di anonimato alle parti di una transazione.

- Netcash risulta completamente sicuro quando almeno uno dei partecipanti, durante una transazione, ha la possibilità di interagire con il server: le monete riutilizzate vengono individuate al momento, facendo in modo che colui che le accetta possa controllarle avvalendosi dell'aiuto del server stesso. Il riutilizzo può essere evitato semplicemente stabilendo che le monete abbiano solo un certo intervallo di validità, in genere molto breve, entro cui possano essere girate od incassate.

- Con NetCash è possibile scambiare monete tra utenti senza coinvolgere il server, ma, operando in questo modo, non si ha nessuna certezza sulla loro validità; pertanto è consigliabile effettuare questo tipo di operazione solo tra persone che si conoscano e nutrano fiducia reciprocamente.

- Dal momento che esistono più server di valuta indipendenti tra loro, NetCash consente una maggiore scalabilità rispetto ad altre proposte esistenti di cash elettronico; tale scalabilità, possibile grazie all'infrastruttura globale di deposito bancario, consente di effettuare ,a mezzo assegni elettronici non anonimi, transazioni fra server remoti, e di conseguenza anche fra utenti geograficamente distanti. In questo tipo di transazioni è sempre fatto salvo l'anonimato degli utenti finali, mentre i server vengono identificati [8].

- L'anonimato fornito da NetCash è più debole di quello garantito dai protocolli di Chaum: in particolare, quando un cliente acquisisce monete da un server, quest'ultimo ha sempre la possibilità di memorizzare quali (numero di serie, valore,etc.) e quante monete sono state inviate a quel particolare utente; naturalmente ci si aspetta che un server corretto non lo faccia. Anche per questo motivo è preferibile che ogni utente scelga il proprio server, oltre che per motivi geografici, soprattutto per una questione di fiducia. Una volta acquistate, le monete entrano in circolo senza l'identificazione dei vari intermediari e, nonostante il server venga coinvolto tutte le volte che la valuta passa di mano, e potrebbe conservare traccia di ogni operazione effettuata, egli non conoscerà l'identità degli intermediari se non quando uno di essi sceglie volontariamente di palesarsi (ad esempio, per poter effettuare, un controllo sulle monete).

- NetCash supporta parzialmente operazioni off-line: in esse le parti risultano off-line solo durante lo scambio finale.

- Qualora fosse richiesto l'anonimato assoluto, oppure si rendessero necessarie operazioni completamente off-line, la struttura di NetCash potrà integrare e supportare i protocolli di Chaum o altri protocolli di valuta elettronica garantendo ancora la scalabilità, l'accettabilità e l'interoperabilità proprie del sistema originale.

Esistono altri protocolli progettati per consentire pagamenti elettronici su Internet [6], rispettanti le stesse proprietá viste per NetCash, essi assicurano anche transazioni totalmente off-line; fra gli altri presentiamo ora Payword e MicroMint, nati per effettuare micropagamenti.

4 PayWord e MicroMint

PayWord e *MicroMint*, basati sugli studi per lo schema *Millicent* di DEC [20], consentono di effettuare piccoli acquisti su Internet.

Il loro scopo principale è quello di minimizzare il numero di operazioni a chiave pubblica richieste per il pagamento, usando, invece, operazioni hash ogni qualvolta sia possibile. Approssimativamente, le funzioni hash sono circa 100 volte più veloci della verifica di firma con RSA, e circa 10.000 volte più veloci della generazione della firma con RSA: in un secondo, su una tipica workstation, è possibile firmare due messaggi oppure verificare 200 firme o ancora calcolare 20.000 valori di una funzione hash.
Per consentire micropagamenti, è richiesta massima efficienza, altrimenti i costi di gestione supereranno il valore dei pagamenti. Di conseguenza tali schemi di micropagamento sono "leggeri" da questo punto di vista, se paragonati agli schemi completi di macropagamento. Considerata la bassa entità dei pagamenti trattati, non si è ritenuto opportuno prevedere per questi due schemi, costosi meccanismi per l'individuazione di monete false e/o falsari. Nonostante ciò, essi possono funzionare efficientemente garantendo mediamente una giusta distribuzione dei pagamenti e bassi livelli di abuso.
In questi due schemi i partecipanti sono *broker*, *utente* e *venditore*. I broker autorizzano gli utenti ad effettuare micropagamenti ai venditori, e riscattano i pagamenti riscossi dai venditori. Mentre i rapporti utente-venditore sono transitori, quelli broker-utente e broker-venditore sono a lungo termine. In una tipica transazione un venditore vende l'accesso ad una pagina world wide web per un cent.
Il primo schema, Payword, è uno schema basato su credito, fondato su serie di monete dette *payword* (valori hash). Simili serie sono state precedentemente proposte per scopi differenti da Lamport [19] e Haller (in S/Key) per il controllo d'accesso [17], e da Winternitz [21] come uno schema di firma one-time. L'applicazione di questa idea ai micropagamenti è stata anche utilizzata da Anderson [12] e da Pederson [24]. L'utente autentica una serie di payword completa al venditore con una singola firma a chiave pubblica e successivamente rivela al venditore ogni payword della serie per effettuare un micropagamento. Il costo incrementale di un pagamento è così un calcolo di funzione hash per parte. PayWord è ottimizzato per sequenze di micropagamenti, ma è abbastanza sicuro e flessibile da sostenere bene anche pagamenti di taglio maggiore.
Il secondo schema, MicroMint, è stato progettato per eliminare del tutto le operazioni a chiave pubblica; ha una sicurezza inferiore, ma velocità elevata. Introduce un nuovo paradigma di rappresentazione delle monete, individuate come collisioni di funzioni hash a k-vie. Proprio come una vera zecca consente al broker di produrre grandi quantità di monete a costi molto bassi per moneta, mentre i tentativi di falsificazione su piccola scala possono solo produrre monete ad un costo superiore al loro valore.

4.1 Generalità e notazioni

In questo contesto è utilizzata la crittografia a chiave pubblica (per esempio, RSA con un esponente pubblico corto). Le chiavi pubbliche del broker B, l'utente U e il venditore V sono denotati PK_B, PK_U e PK_V, rispettivamente; le loro chiavi segrete sono denotate SK_B, SK_u, SK_v. Un messaggio M con la sua firma digitale prodotta da una chiave segreta SK è denotato $\{M\}_{SK}$. Questa firma può essere verificata usando la corrispondente chiave pubblica PK.
Poniamo che h denoti una funzione hash crittograficamente forte, così come MD5 [25] o SHA [23]. L'output (designato a 128 o 160 bit) può essere troncato a lunghezze minori. L'aspetto importante è che h è una funzione one-way e resistente alle collisioni; sarebbe necessaria una ricerca molto vasta per trovare un singolo input che produca un dato output, o trovare due input che producano lo stesso output. La lunghezza dell'input in alcuni casi potrebbe essere

uguale a quella dell'output.

4.2 PayWord

PayWord è uno schema basato sul credito. L'utente apre un conto presso un broker, il quale emette un suo "certificato payword" con firma digitale contenente il nome del broker, il nome dell'utente, l'indirizzo *IP*, la chiave pubblica dell'utente, la data di scadenza, ed altre informazioni. Il certificato viene rinnovato dal broker (per esempio, mensilmente), il quale lo farà solo se il conto dell'utente è in buona consistenza. Questo certificato autorizza l'utente a creare serie di payword, e assicura i venditori che esse siano riscattabili presso il broker. Assumiamo che ogni payword valga esattamente un cent (questa cifra potrebbe essere variata). In una tipica applicazione quando U effettua un collegamento ad una pagina web non libera di un venditore V, il suo browser determina se questa è la prima richiesta a V di quel giorno. Per una prima richiesta U calcola e firma un "ordine" e crea una nuova serie di payword $w_1, w_2, ..., w_n$ specifica per il venditore.
L'utente crea la serie di payword in ordine inverso, scegliendo l'ultima payword w_n a caso, e calcolando poi

$$w_i = h(w_{i+1}) \qquad per\ i = n-1, n-2, ..., 0.$$

w_0 è la radice della serie di payword ma non è di per sè una moneta. L'ordine contiene la radice w_0 ma non le payword w_i per $i > 0$. Successivamente U fornisce quest'ordine ed il suo certificato a V il quale ne verifica le firme.
L'i-esimo pagamento (per $i = 1, 2, ...$) di U a V, consiste nella coppia (w_i, i), che il venditore può verificare usando w_{i-1}; ogni pagamento simile non richiede calcoli da parte di U, ma solo una singola operazione hash da parte di V.
Alla fine di ogni giorno, V Comunica A B l'ultimo pagamento (quello con indice maggiore) (w_l, l) ricevuto da ogni utente quel giorno, insieme ad ogni ordine corrispondente. B addebbita sul conto di U l cent e accredita l cent sul conto di V. (Il venditore potrebbe anche addebbitare tasse di sottoscrizione e/o di transazione che per comodità ignoriamo).
Uno scopo fondamentale di PayWord è minimizzare la comunicazione (in particolare di tipo on-line) con il broker. Immaginiamo che ci sia un piccolo numero di broker; per impedire che rallentino il sistema è importante che il loro carico computazionale sia allo stesso tempo ragionevole ed "off-line". PayWord è uno schema "off-line": V non ha bisogno di interagire con B quando U lo contatta per la prima volta e non ha bisogno di interagire con B per nessuno dei pagamenti effettuati. Notiamo che B non riceve tutte le payword esaurite, ma solo l'ultima spesa da ogni utente ogni giorno, presso ogni venditore.
PayWord è estremamente efficiente quando l'utente effettua ripetute richieste allo stesso venditore, comunque risulta piuttosto efficace in ogni situazione. Le operazioni a chiave pubblica richieste da V sono solo verifiche di firma, le quali sono relativamente veloci.
Notiamo che le tecniche probabilistiche di Shamir circa la protezione della firma [27] possono essere usate per ridurre ulteriormente il carico computazionale. Un'altra applicazione per cui PayWord risulta indicato è l'acquisto a tempo di pagine; l'utente può pagare pochi cent per ogni minuto di navigazione.

4.2.1 Utente-Broker: rapporti e certificati

L'utente U inizia un rapporto con il broker B, aprendo presso di lui un conto e richiedendogli un "certificato payword". U invia a B su un canale sicuro: il suo numero di carta di credito, la sua chiave pubblica PK_V ed il suo indirizzo di consegna A_U. I costi relativi alle payword gli saranno addebbitati sulla carta di credito. Il suo indirizzo di consegna sarà il suo email su Internet o il suo indirizzo postale; il suo certificato autorizzerà solo pagamenti da parte sua per acquisti da consegnare presso il suo indirizzo A_U. Il certificato dell'utente ha una data di scadenza E; ad esempio, i certificati possono scadere mensilmente. Agli utenti che non pagano i loro certificati non ne saranno emessi di nuovi. Il broker può inserire un'altra informazione I_U (possibilmente specifica per l'utente) nel certificato, come ad esempio un numero di serie, limiti di credito presso ogni venditore, informazioni su come contattare il broker, termini ed accordi broker-venditore, etc.
Quindi il certificato C_U dell'utente ha la seguente forma:

$$C_U = \{B, U, A_U, PK_U, E, I_U\}_{SK_B}.$$

Il "certificato payword" è una dichiarazione da parte di B a qualche venditore V, che B estinguerà payword autentiche distribuite ad U e consegnate prima della data di scadenza stabilita (con un giorno di tolleranza ammessa). PayWord non garantisce l'anonimato. Anche se i certificati possono contenere i numeri dei conti degli utenti, invece che i loro nomi, l'inclusione di A_U elimina effettivamente l'anonimato di U.
Se U perde la sua chiave segreta dovrebbe riferirlo subito a B. Le sue responsabilità dovrebbero essere minime, in casi del genere, come accade per la perdita di una carta di credito. Se ciò accadesse ripetutamente il broker potrebbe rifiutare di potrarre ulteriormente il rapporto. A tale scopo egli potrebbe conservare una *hot-list* di certificati i cui intestatari hanno riferito di aver smarrito le chiavi o che presentano qualche altro tipo di problema.
Come alternativa alle *hot-list*, si potrebbe provvedere quotidianamente all'autenticazione del certificato dell'utente, usando le serie hash in modo differente, come proposto da Micali [22]. Il certificato dell'utente dovrebbe contenere addizionalmente la radice w'_0 di una serie hash di lunghezza 31. Il giorno $(j-1)-esimo$ del mese, il broker manderà all'utente (per esempio, via posta) il valore w'_j se e solo se il conto dell'utente è ancora in buona consistenza. I venditori, quindi, prima di accettare un pagamento richiederanno l'appropriato valore w'.

4.2.2 Utente-Venditore: rapporti e pagamenti

I rapporti utente-venditore sono di tipo transitorio.

Ordini Quando un utente U sta per contattare un nuovo venditore V, egli calcola una nuova serie di payword $w_1, ..., w_n$ con radice w_0. Il valore n è scelto dall'utente a proprio piacimento; potrebbe essere 10 oppure 10.000. Egli poi calcola il suo ordine per quella serie:

$$M = \{V, C_U, w_0, D, I_M\}_{SK_U}.$$

V identifica il venditore, C_U è la certificazione dell'utente, w_0 è la radice della serie delle payword, D è la data corrente ed I_M è qualche informazione ulteriore che può essere richiesta (ad esempio la lunghezza n della serie di payword).
M viene firmato da U ed inviato a V. (Dal momento che la firma deve avvennire necessariamente on-line, poichè contiene il nome del venditore, l'utente potrebbe usare uno schema di

firma di tipo "on-line-off-line" [15]).
Quest'ordine autorizza B a pagare V per ognuna delle payword $w_1, ..., w_n$, che V riscatta da B, dopo la data D (più un giorno di tolleranza). Notiamo che queste payword saranno specifiche per tale coppia utente-venditore.
Evidenziamo che U deve firmare un ordine per ogni venditore V da pagare. Il venditore verifica la firma di U sull'ordine M e la firma del broker su C_U (contenuto in M), registrandone le date di scadenza.
Il negoziante deve acquisire gli ordini effettuati, prima che essi scadano alla fine della giornata. Diversamente, se riscattasse le payword ricevute dopo la data di scadenza dell'ordine, U potrebbe imbrogliarlo, sostituendo al più presto ordini e payword. (Allo stato attuale, per evitare ciò, il venditore V necessita di memorizzare una piccola parte di ogni ordine che egli ha comunicato a B in giornata). L'utente dovrebbe preferibilmente annullare il suo ordine prima di aver finito di effettuare ordini da V o, prima che esso scada.

Pagamenti L'utente ed il venditore devono essere d'accordo sull'ammontare del pagamento. Nella nostra applicazione d'esempio il costo di una pagina web è tipicamente di un cent, ma potrebbe essere anche diverso. Un pagamento P da U a V consiste di una payword e del suo indice

$$P = (w_i, i).$$

Il pagamento è costituito da una stringa lunga solo 20 o 30 bit. (Il primo pagamento a V del giorno dovrebbe normalmente accompagnare il corrispondente ordine di U; i successivi pagamenti sono composti solo dalla payword e dal suo indice, fin quando la serie corrente non si esaurisce e deve esserne ordinata un'altra). Il pagamento non è firmato da U, dal momento che è auto-autenticato (usando l'ordine).
L'utente spende le sue payword in ordine sequenziale w_1, poi w_2 e così via.
Ciò conduce alla politica di pagamento PayWord :*ogni ordine, un venditore V é pagato l cent, dove* (w_l, l) *é il pagamento corrispondente ricevuto avente l'indice maggiore.* Ciò significa che V ha necessità di conservare solo un pagamento per ogni utente: quello con l'indice maggiore. Una volta che un utente spende w_i, non potrà spendere w_j con $j < i$. Il broker può verificare il valore da pagare per w_l, determinando quante applicazioni di sono richieste per ottenere (to map) w_l partendo da w_0.
PayWord supporta pagamenti di taglia variabile in modo semplicee naturale. Se U salta alcune payword di una serie, e spende direttamente w_7 dopo w_2, darà a V un valore quintuplo rispetto a quello della generica w_i. Quando U salta delle payword, durante la verifica, V deve solo applicare h un numero di volte proporzionale al valore del pagamento fatto.
Un pagamento non specifica per cosa viene effettuato, dunque il venditore può truffare il cliente, inviandogli un articolo sbagliato oppure nulla. I venditori che imbrogliano in questo modo i loro clienti saranno evitati. Se U non paga, V lo può notificare a B e/o rifiutare ad U ulteriori servizi.

4.2.3 Venditore-Broker: rapporti e retribuzioni

Un venditore V non deve avere necessariamente un rapporto prioritario con B, ma deve necessariamente ottenere PK_B, in modo da poter autenticare i certificati firmati da B. Egli deve anche accordarsi con il broker B, per stabilire in che modo questi gli verserà l'ammontare dei pagamenti ricevuti. (I broker pagano i venditori con mezzi esterni al sistema PayWord).

Alla fine di ogni giornata, (o di un periodo conveniente), V invia a B un messaggio consuntivo che contiene, per ogni utente di B che ha pagato V quel giorno, l'ordine C_U ricevuto da U accompagnato dall'ultimo pagamento da questi effettuato: $P = (w_l, l)$.
Il broker ha necessità di verificare ognuno di questi ordini (controllando solo la firma dell'utente, dal momento che riconoscerà con facilità le sue stesse certificazioni), nonchè ogni pagamento (w_l, l) (ciò richiede l'applicazione di l funzioni hash) per poter onorare tutte le richieste valide di riscatto da parte dei venditori.
Dal momento che le computazioni di funzioni hash sono economiche, e le verifiche delle firme sono solo moderatamente costose, il peso di computazione dovrebbe essere ragionevole, essendo proporzionale al volume dei pagamenti che avvengono; il broker può caricare le transazioni o sottoscrizioni di onorari, adeguati a coprire i costi di computazione. Notiamo, infine, che B non deve mai rispondere in tempo reale; può prendere in carico le sue computazioni ed effettuarle off-line durante la notte.

4.2.4 Efficienza

Sintetizziamo i requisiti di calcolo ed immagazzinamento del sistema PayWord:

- il broker ha necessità di firmare ogni certificato dell'utente e calcolare un'applicazione di funzione hash per ogni pagamento (tutti questi calcoli avvengono off-line).
  Egli conserva le copie dei certificati degli utenti ed tiene i conti di utenti e venditori;

- l'utente deve verificare i suoi certificati, firmare ogni suo ordine, calcolare un'applicazione di funzione hash per ogni payword commissionata. (Il solo firmare gli ordini è un'operazione on-line).
  Egli deve memorizzare la sua chiave segreta SK_U, i suoi ordini attivi, le corrispondenti serie di payword e la sua posizione corrente in ogni serie;

- Il venditore verifica ogni certificato ed ogni ordine ricevuto e calcola un'applicazione di funzione hash per ogni payword ricevuta o saltata. (Tutti i suoi calcoli sono on-line).
  Egli deve memorizzare ogni giorno tutti gli ordini e l'ultimo pagamento ricevuto per ogni ordine.

4.2.5 Varianti ed estensioni

In una possibile variante, $h(\cdot)$ è rimpiazzato da $h_s(\cdot) = h(s, \cdot)$, dove s è un valore random specificato nell'ordine. Ciò può richiedere l'uso di funzioni hash più veloci o aventi un output più breve (fra i 64 e gli 80 bit).
Il valore di ogni payword può essere fissato di un cent oppure può essere specificato in C_U o in M. In una variante, M potrebbe autenticare più serie, le cui payword hanno differenti valori (un penny, un nichelino, etc.).
Potrebbe essere necessario specificare il nome dell'utente in un pagamento, se non risultasse chiaro dal contesto generale. Se U ha più di una serie di payword autorizzata da V, allora il pagamento dovrebbe specificare quella più rilevante.
Le payword potrebbero essere vendute a debito, piuttosto che a credito, solo se l'utente interagisse col broker nel formulare ogni ordine: il certificato potrebbe richiedere che il broker, e non l'utente, firmi ogni ordine. Il broker potrebbe rimborsare l'utente per le payword non usate, una volta che il venditore abbia riscattato le payword incassate.

L'ordine dovrebbe specificare sia il venditore che l'ammontare della cifra da pagare.
Il broker potrebbe specificare nel certificato dell'utente altre condizioni e termini per ridurre i propri rischi: ad esempio, limitare la cifra massima che U può spendere, ogni giorno, presso ogni venditore, rifiutando di pagare se U ha speso oltre il limite consentito, oppure rifiutare il pagamento se il nome di U è nella *hot-list* all'inizio del giorno. (I venditori possono leggere le *hot-list* ogni mattina).
Questi epedienti proteggerebbero B da ulteriori responsabilità se ad esempio la chiave segreta SK_U venisse perduta oppure scoperta ed usata abusivamente da persone non autorizzate, (sebbene, dal momento che C_U autorizza la distribuzione ad A_U, il rischio è ridotto); in questi casi il venditore dividerebbe i rischi con B.

4.3 MicroMint

MicroMint è progettato per dare una sicurezza ragionevole a costi molto bassi ed è ottimizzato per pagamenti indipendenti e di basso valore; usa completamente operazioni a chiave non pubblica.
Le monete in MicroMint sono prodotte da un broker, che le distribuisce agli utenti. Gli utenti girano queste monete ai venditori come pagamento. I venditori restituiscono le monete al broker che ne rimborsa l'ammontare attraverso altri mezzi.
Una moneta è una stringa di bit la cui validità può essere facilmente constatata da ognuno, ma che è difficile da produrre. Ciò somiglia ai requisiti per la firma a chiave pubblica, certamente onerosa, a causa della complessità, per transazioni del valore di un cent. (PayWord usa le firme, ma non su tutte le transazioni).
In MicroMint generare più monete risulta più conveniente di generarne poche; è necessario un cospicuo investimento iniziale per coniare la prima moneta, ma le successive vengono prodotte con molta convenienza. Ciò è simile all'economia per una coniatura reale, per la quale si investe molto per acquisire macchinari costosi che consentono poi di produrre monete in modo economico. (Non avrebbe senso produrre monete aventi un costo di produzione superiore al loro valore reale).
Il broker tipicamente emetterà nuove monete all'inizio di ogni mese; la loro validità scadrà a fine mese: le monete inutilizzate saranno restituite al broker alla fine di ogni mese e potranno essere acquistate quelle nuove, valide per il mese successivo. I venditori possono restituire le monete al broker quando risulta loro più comodo (ad esempio, alla fine di ogni giorno).
Descriviamo ora la versione base di MicroMint; sono possibili, comunque, molte estensioni e varianti di cui ne vedremo alcune in seguito.

Monete come collisioni di funzioni hash Le monete di MicroMint sono rappresentate da *collisioni di funzioni h ash*, per alcune specifiche funzioni hash one-way h che trasformano stringhe x di m bit in stringhe y di n bit. Diciamo che x è una pre-immagine (antimmagine) di y se $h(x) = y$. Una coppia di stringhe distinte di m bit, (x_1, x_2) è detta collisione a due vie se $h(x_1) = h(x_2) = y$, per qualche$y$ di n bit.
Se h agisce in maniera random, l'unico modo per produrre ancora una collisione accettabile a due vie, è effettuare l'hash di circa $\sqrt{2^n} = 2^{\frac{n}{2}}$ valori di x, e ricercare gli output ripetuti. (Ignoriamo le costanti piccole).
Effettuando l'hash c volte su tanti valori x quanti sono necessari a produrre la prima collisione, si generano approssimativamente in c^2, con $1 \leq c \leq 2^{\frac{n}{2}}$, altrettante collisioni, in questo modo, una volta superata la soglia per trovare le prime, produrre collisioni può essere fatto con

efficienza crescente, per ogni moneta generata.

Monete come collisioni a k-vie Un problema con collisioni a $2-vie$ è che scegliere un valore di n, piccolo abbastanza da rendere agevole il lavoro del broker, porta a situazioni in cui le monete possono essere falsificate molto facilmente da un malintenzionato. Per raggiungere la soglia massima contro le falsificazioni, proponiamo di usare collisioni a $k-vie$, in sostituzione di quelle a $2-vie$.
Una collisione a $k-vie$ è un insieme di k distinti valori x: $x_1, x_2, ..., x_k$ che hanno lo stesso valore hash y. Il numero di valori x che deve essere esaminato, prima che ci si aspetti di ottenere la prima collisione a $k-vie$, è approssimativamente $2^{n\frac{(k-1)}{k}}$. Se si esaminano c volte questi valori x, ove $1 \leq c \leq 2^{\frac{n}{k}}$, ci si aspetta di vedere circa c^k collisioni a $k-vie$. Scegliendo $K > 2$, si ha l'effetto duale di ritardare la soglia dove appare la prima collisione, ma anche di accelerare la velocità di generazione delle collisioni, una volta che tale soglia è stata oltrepassata.
Conviene dunque rappresentare una moneta con una collisione a $k-vie$ $(x_1, ..., x_k)$. La validità di questa moneta può essere facilmente verificata da chiunque, accertando che le x_i siano tali che $k(x_1) = k(x_2) = ... = k(x_k) = y$ per qualche stringa y di n bit.

La coniatura delle monete Il processo di calcolo $h(x) = y$ è analogo al lancio casuale di una biglia (x), in uno fra 2^n recipienti; quello dove la biglia (x) andrà a finire avrà proprio indice y. Una moneta, analogamente, è un insieme di k biglie lanciate nello stesso recipiente. Centrare k biglie nello stesso recipiente richiede di lanciarne, globalmente, un grosso numero, dal momento che esse non possono essere indirizzate ad un recipiente particolare. Per coniare monete, il broker creerà 2^n recipienti, lancerà approssimativamente $k \cdot 2^n$ biglie, e genererà una moneta per ogni recipiente contenente almeno k biglie. Con questa scelta di parametri, ogni biglia ha una probabilità di circa $\frac{1}{2}$ di far parte di una moneta.
Nel caso in cui uno dei 2^n recipienti avesse k o più biglie in esso, k di esse possono essere estratte, per formare una moneta. Notiamo che se un recipiente contiene più di k biglie, in linea di principio, il broker potrebbe estrarne k sottoinsiemi in molti modi, per produrre così più monete. Comunque, un nemico che ottiene 2 monete differenti dallo stesso recipiente, potrebbe essere in grado di combinarle, al fine di produrne altre nuove. Per questo motivo *un broker MicroMint deve produrre al più una moneta per ogni recipiente;* ciò semplifica il compito del broker nell'individuare monete riutilizzate, dal momento che dovrà memorizzare una tavola di soli 2^n bit, per indicare o meno se una moneta con un particolare valore hash ad n bit è stata già riscattata.
Un piccolo problema, in questa descrizione di base, comunque, è che la memorizzazione risulta più onerosa della computazione. Il numero di biglie che possono essere lanciate durante una computazione lunga un mese, supera di molto sia quante di esse possano essere memorizzate su di un numero ragionevole di hard disk, sia il numero di monete che il broker, realisticamente, avrebbe necessità di coniare. Si potrebbe tentare di bilanciare le richieste di memoria e di computazione utilizzando un algoritmo hash molto lento, come il *DES* iterato più volte; sfortunatamente, questo metodo rallenta anche il processo di verifica.
Un approccio migliore adottato, è rendere molte biglie inutili allo scopo di coniare monete. Per far ciò, diciamo che una biglia è "buona" se i bit di maggior peso del valore hash y hanno un valore z specificato dal broker. Più precisamente, sia $n = t + u$ ove t ed u sono due interi non negativi. Se i t bit di maggior peso di y sono uguali al valore z specificato, allora il valore y sarà "buono", e gli u bit di peso minore di y, determineranno l'indice del recipiente nel quale

la biglia "buona" x risulta lanciata. (In generale i valori x sono considerati semplicemente come "biglie", e quelli non buoni, possono essere pensati come concettualmente lanciati in recipienti virtuali inesistenti che sono "fuori range").
Una scelta appropriata di t consente di bilanciare le esigenze di calcolo e di memorizzazione del broker, senza rallentare il processo di verifica. Si rallenta il processo di generazione (produzione) di un fattore 2^t, mentre si limitano le richieste di memorizzazione del broker ad un piccolo multiplo del numero di monete da generare. Il broker in questo modo lancia approssimativamente $k \cdot 2^n$ biglie, memorizza circa $k \cdot 2^u$ biglie "buone" che lancia nei 2^u recipienti e genera da essi approssimativamente $\frac{1}{2} \cdot 2^u$ monete valide.
Nota: con funzioni hash standard come *DES* e *MD5*, il numero di bit di output prodotti può superare il numero n di bit specificato nei parametri del broker. Un'appropriata funzione per il broker, può essere ottenuta scegliendo gli n bit di peso minore dell'output dato dalla funzione hash standard di output. Scartare i bit ad eccezione degli n di peso minore, costituisce un processo differente da quello di specificare un particolare valore per i t bit di peso maggiore fuori dagli n precedentemente descritti.

Uno scenario dettagliato Segue una presentazione dettagliata di come, tipicamente, un broker può procedere nella scelta dei parametri per la propria coniatura valida per un dato mese. I calcoli sono approssimati ma istruttivi (i valori vengono arrotondati alla più vicina potenza del 2); essi possono essere facilmente modificati ad altre assunzioni di valori.
Il broker investirà in un hardware solido che gli dia vantaggio computazionale sui probabili contraffattori, e lo fa funzionare continuamente per un mese, per produrre monete valide il mese successivo; è preferibile che questo hardware includa alcuni chip specifici per la computazione efficiente di h.
Supponiamo che il broker abbia un profitto netto di un milione di dollari al mese (circa 2^{27} cent al mese). Egli carica una tariffa di mediazione pari al 10%, quindi, per ogni moneta inviata, pagherà al venditore 0.9 cent quando la stessa sarà riscattata. In questo modo il broker deve distribuire un bilione di monete al mese (circa 2^{30}) per raccogliere il suo milione di dollari. Se un cliente medio acquista 2.500 monete al mese (\$25.00) egli deve avere un clientela base di 500.000 utenti.
Il broker sceglie $k = 4$; una moneta sarà così una collisione a $4 - vie$.
Per coniare 2^{30} monete, il broker sceglie $u = 31$, in questo modo egli crea un array di 2^{31} recipienti (approssimativamente due bilioni) ognuno dei quali può contenere dai 4 valori di x in su, che portano ad un valore di n bit risultante dalla concatenazione di un fissato campione, lungo t bit, con l'indice del recipiente, lungo u bit.
Il broker lancerà una media di 4 biglie in ogni recipiente generando, quindi, $4 \cdot 2^{31} = 2^{33}$ (circa 8 bilioni) di valori x che producono y valori "buoni".
Operando in questo modo, la probabilità che un recipiente contenga 4 o più valori x è circa $\frac{1}{2}$ (così potrà produrre una moneta). Dal momento che ognuno dei 2^{31} recipienti produce una moneta con probabilità $\frac{1}{2}$, il numero delle monete generate sarà $2^{31} \cdot \frac{1}{2} = 2^{30}$, come si desiderava.
Al fine di massimizzare il vantaggio su avversari che tentino di contraffare le monete, il broker investe in un hardware specifico che gli consente di calcolare valori hash molto rapidamente. In questo modo potrà scegliere un valore di t relativamente grande, tale che i valori hash "buoni", risultino relativamente rari. Questo aumenta la quantità di lavoro per un ipotetico falsario (ma anche per il broker) di un fattore 2^t. Il broker sceglie la sua funzione hash h come gli n bit di peso minore della codifica di qualche valore fissato v_0 con chiave x sotto *DES*:

$$h(x) = [DES_x(v_0)]_{1...n}$$

Il broker acquista un numero di chip (FPGA), ognuno dei quali è capace di applicare una funzione hash approssimativamente su 2^{25} (circa 30 milioni) valori x al secondo (vedi [13]). Ognuno di questi chip costa circa \$200; si é stimato che al broker ogni chip non costi più di \$400, considerati i costi di progettazione, di mantenimento e dell'hardware associato. Egli acquista 2^8 (256) di questi chip che gli costano circa \$100.000. Questi chip possono collettivamente calcolare 2^{33} valori hash al secondo (circa 8,6 bilioni). Poichè ci sono approssimativamente 2^{21} secondi in un mese, i chip possono calcolare circa 2^{54} valori al mese (approssimativamente 18 milioni di bilioni).
Basandosi su queste stime, il broker sceglie $n = 52$ e $t = 21$ e conia monete per un mese. Dei $k \cdot 2^n = 2^{54}$ valori calcolati, solo 1 su 2^{21} sarà "buono", in modo da trovare circa 2^{33} valori di x "buoni", cioè quanti necessari a produrre 2^{30} monete.
Memorizzare una coppia $(x, h(x))$ "buona", richiede meno di 16 byte. Lo spazio totale richiesto per memorizzare tutte le coppie buone è minore di 2^{37} byte (128 Gbyte), ed il costo totale per la loro memorizzazione è minore di \$40.000. Il costo totale per l'hardware del broker risulta quindi minore di \$150.000, cifra inferiore al 15% del profitto del primo mese. Piuttosto che scrivere ogni coppia in un recipiente accessibile in maniera casuale, il broker può memorizzare sequenzialmente le 2^{33} coppie "buone" in un array su disco, ordinandole poi in maniera crescente per valori di y, per determinare quelle che si trovano nello stesso recipiente. Con un ragionevole algoritmo di sort, il tempo di riordino dovrebbe essere inferiore ad un giorno.

Distribuzione delle monete Verso la fine di ogni mese, il broker inizia la distribuzione agli utenti delle monete da usare il mese successivo e, solo quando questo inizia, ne rivela il criterio di validità. Queste vendite possono essere effettuate indifferentemente a credito o a debito, dal momento che B avrà la possibilità di individuare le monete quando gli saranno restituite per essere riscattate. In una situazione tipica, un utente potrebbe acquistare monete equivalenti a \$ 25.00 (2.500 monete), e caricare l'acquisto sulla sua carta di credito. Il broker può memorizzare un record delle monete acquistate da un utente; quelle non utilizzate gli saranno restituite alla fine di ogni mese.

Pagamenti Ogni volta che un utente acquista una pagina web, paga il venditore con una moneta $(x_1, x_2, ..., x_k)$ non spesa precedentemente. Il venditore verifica che essa sia una collisione valida a $k - vie$ calcolando $h(x_i)$ per $1 \leq i \leq k$, e controllando che i valori siano uguali e validi. Notiamo che, mentre il processo di coniatura delle monete è stato intenzionalmente rallentato di un fattore 2^t, il lavoro di verifica da parte del venditore si mantiene estremamente efficiente, richiedendo solo k computazioni hash e pochi confronti (nell'esempio, $k = 4$).

Riscatto delle monete Il venditore, alla fine di ogni giorno, restituisce al broker le monete accumulate. Il broker controlla se tra esse ve ne siano alcune già stata restituite in precedenza (esaminandole tutte), e, in caso contrario, paga al venditore 1 cent a moneta, dopo aver sottratto la sua percentuale di mediazione. Se il broker riceve una particolare moneta più di una volta, egli la pagherà una volta sola. Potrà decidere arbitrariamente, o anche in maniera puramente casuale, quale venditore pagare. Ciò è penalizzante per i venditori, ma elimina ogni motivazione di carattere finanziario ad imbrogliare da parte di un venditore, nel ridistribuire monete ricevute da altri venditori.

4.3.1 Proprietà di sicurezza

Distinguiamo tra attacchi su piccola scala e attacchi su larga scala. Riteniamo che utenti e venditori non tentino truffe allo scopo di guadagnare solo pochi cent poichè, se anche lo facessero, non ne ricaverebbero grossi guadagni. Dunque i meccanismi di sicurezza sono progettati in primo luogo per scoraggiare attacchi su larga scala come la forte contraffazione o il persistente riutilizzo delle monete.

Contraffazione La contraffazione su piccola scala, essendo molto onerosa, non può essere conveniente per un ipotetico falsario: con la scelta consigliata di $k = 4$, $n = 54$ ed $u = 31$, la generazione della prima moneta falsa richiede circa 2^{45} operazioni hash. Poichè una workstation standard può effettuare solo 2^{14} operazioni hash al secondo, un tipico utente avrà bisogno di 2^{31} secondi (circa 80 anni) per generare, in proprio, appena una moneta falsa.
La contraffazione su larga scala può essere scoperta e contrastata come segue:

- tutte le monete false automaticamente vengono invalidate alla fine del mese;
- le monete false non possono essere generate fin quando il broker non ha annunciato il nuovo criterio di validità delle monete per il prossimo mese;
- l'uso di predicati nascosti fornisce un intervallo di tempo più adeguato per respingere monete false senza intaccare la validità delle monete legali già in circolazione;
- il broker può individuare la presenza di un falsario selezionando, fra quelle ricevute, le monete provenienenti da recipienti dai quali egli non ne ha estratto. Ciò funziona bene nel nostro esempio poichè solo la metà dei recipienti utilizzati produce monete; per implementare questo espediente, il broker deve lavorare unicamente con un array di bit, avente un bit per recipiente;
- il broker può in ogni momento dichiarare il periodo corrente scaduto, ritirare tutte le monete, ed immettere in rete nuove monete, comunicando un nuovo criterio di validità;
- il broker può simultaneamente generare monete per più mesi futuri con una lunga computazione; ciò rende più difficile ad un falsario competere col broker.

Furto di monete Il furto di monete è un problema principalmente relativo alla fase iniziale di distribuzione agli utenti ed alla fase di riscatto da parte del venditore; tuttavia sottolineamo che, durante queste operazioni, è facile trasmettere monete in forma cifrata. I rapporti utente-broker e venditore-broker sono relativamente stabili, quindi, tra i due, può essere adottata una chiave crittografica a lungo termine.
Per proteggere le monete quando sono trasferite su Internet da utente a venditore, e, al fine di avere una comunicazione sicura, si può usare una tecnica a chiave pubblica. Comunque, nel tentativo di eliminare completamente, o per lo meno minimizzare le operazioni a chiave pubblica, esiste un'altra variante, che utilizza monete personali per un determinato utente, rendendone più difficile il riutilizzo, senza richiedere una crittografia a chiave pubblica. Altro aspetto da considerare è che due venditori possano colludere e che ognuno di essi cerchi di riscattare le stesse monete. La raccomandazione è che un broker non paghi più di una volta la stessa moneta, come detto in precedenza. Dal momento che ciò potrebbe penalizzare i venditori onesti che ricevono inconsapevolmente monete già utilizzate, si possono rendere le

monete specifiche anche per un determinato venditore, allo stesso modo di quanto fatto per gli utenti.

Riutilizzo di monete Dal momento che MicroMint non è uno schema anonimo, il broker può individuare una moneta riutilizzata, e può individuare da quali venditori ha ricevuto le diverse copie. Egli conosce il cliente cui la moneta era stata distribuita e, con l'aiuto del venditore onesto, può anche individuare quale cliente ha speso le diverse copie della moneta. Basandosi su queste informazioni, il broker può determinare quante monete riutilizzate sono associate ad ogni utente e venditore. Un imbroglione incallito (utente o venditore) può essere identificato dal grande numero di monete duplicate associate ai suoi acquisti o riscatti. Un piccolo imbroglione può essere identificato con maggiore difficoltà, ma, dato lo scarso valore delle singole monete, individuarlo non ha molta importanza.
MicroMint non prevede alcun meccanismo per prevenire alterazioni fatte per pura malizia, (senza alcun beneficio economico per il contraffattore) dal momento che i meccanismi conosciuti per evitare tali comportamenti sono eccessivamente onerosi. Poichè MicroMint non usa firme digitali reali, potrebbe essere difficile provare legalmente chi è il colpevole delle monete duplicate. Così un broker non potrà perseguire legalmente un sospetto imbroglione, ma potrà sempre escluderlo dal sistema.

4.3.2 Varianti

Monete personalizzate per gli utenti Descriviamo due proposte per rendere le monete personalizzate per gli utenti in modo da essere facilmente individuabili dai venditori. Queste monete se rubate non hanno alcun valore per gli altri utenti e ciò riduce molto l'incentivo al furto.
Nella prima proposta, il broker divide gli utenti in "gruppi", e distribuisce ad ogni utente monete la cui validità dipende dall'identità del gruppo. Per esempio, il broker può dare all'utente U monete che soddisfino l'ulteriore condizione $h'(x_1, x_2, ..., x_k) = h'(U)$, dove la funzione hash h' produce un valore output corto (ad esempio, 16 bit) che indica il gruppo di U. Un venditore può facilmente verificare questa condizione e rifiutare una moneta presentata da un membro del gruppo non corretto.
Il problema con questo approccio è che, se i gruppi sono molto grandi, allora, un ladro può facilmente trovare utenti del gruppo appropriato disposti ad acquistare monete rubate. D'altro canto, se i gruppi sono troppo piccoli (ad esempio, un utente per gruppo), il broker può essere costretto a pre-calcolare una grande quantità di monete in eccesso, per essere sicuro di poter soddisfare l'imprevedibile fabbisogno di ogni utente.
Nella seconda proposta, generalizziamo la nozione di collisione a strutture combinatoriali più complesse: formalmente, una moneta $(x_1, ..., x_k)$ sarà valida per un utente U se le immagini $y_1 = h(x_1)$, $y_2 = h(x_2)$, ..., $y_k = h(x_k)$, soddisfano la condizione:

$$y_{i+1} - y_i = d_i \ (mod\ 2^u)$$

per $i = 1, 2, ..., k-1$, dove

$$(d_1, d_2, ..., d_{k-1}) = h'(U)$$

per un'adeguata funzione hash ausialiaria h' (la proposta originale di rappresentare le monete come collisioni può essere vista come il caso speciale in cui tutte le distanze d_i fra i k contenitori

hanno valore zero).
Per coniare monete di questo tipo, il broker riempe la maggior parte dei suoi contenitori, lanciando biglie in essi in maniera casuale, con la differenza che, in questo caso, non è necessario avere più di una biglia per contenitore. Questa pre-computazione non è specifica per ogni utente ed il broker non deve avere, a priori, alcuna conoscenza del numero di monete che saranno richieste da ogni utente, dal momento che ogni biglia "buona" può essere usata in una moneta per ogni utente. Dopo questa lunga pre-computazione il broker potrà rapidamente creare una moneta per ogni utente U

- calcolando $(d_1, d_2, ..., d_{k-1}) = h'(U)$;
- prendendo a caso un indice y_1 di un recipiente. (Questo recipiente non dovrebbe essere stato precedentemente usato come un y_1 per un'altra moneta, in modo che y_1 può essere usato come identità della moneta quando il broker usa un array di bit per determinare le monete già riscattate);
- calcolando $y_{i+1} = y_i + d_i \ (mod\ 2^u)$, per $i = 1, 2, ..., k-1$;
- estraendo una biglia x_1 dal recipiente x_1 ed una copia di una biglia da ogni recipiente $y_2, ..., y_k$. (Se ogni recipiente y_i è vuoto si riparte da y_1. Notiamo che in questo schema le biglie possono essere riutilizzate);
- producendo le k tuple $(x_1, ..., x_k)$ come monete output.

Una buona caratteristica di questo schema è che è facile produrre un grosso numero di monete per un dato utente, quando il dispositivo di memorizzazione del broker è un disco magnetico con un tempo di accesso relativamente lento. L'idea è basata sull'osservazione che se i valori di y_1 per successive monete, sono consecutivi, allora così saranno i valori y_i per ogni i, $1 < i \leq k$. Quindi, una richiesta di 2.500 monete nuove ponendo $k = 4$ richiede solo 4 accessi al disco piuttosto che 10.000: a 10 millisecondi ad accesso il tempo di accesso totale sarà ridotto da 100 secondi a soli 40 millisecondi.
Notiamo che in linea di principio le monete prodotte per utenti differenti possono riutilizzare la stessa biglia x_i. In modo concepibile, un falsario potrebbe falsificare una nuova moneta combinando insieme i pezzi di altre monete viste; difficilmente avrà buoni risultati così facendo, a meno che non abbia potuto vedere le biglie di una significante parte di un contenitore. Ad esempio, supponendo che vi siano 2^{31} contenitori dei quali il falsario abbia visto una frazione di 2^{-10} (per esempio, egli ha collezionato 2^{21} biglie, prendendole da monete spese da altri utenti), il numero di monete che potrà comporre da queste biglie, è pari solo a $2^{31}(2^{-10})^3 = 2$. Persino se egli avesse 1.000 acquirenti per queste monete, potrebbe fare al più 2.000 monete totali o, in media, 2 monete a cliente. Per tali motivi gli autori non hanno dedicato particolare attenzione a questo tipo di falsificazione "taglia e incastra".

Monete personalizzate per i venditori Per ridurre ancor più la probabilità di furto delle monete, l'utente può darle ai venditori, facendo in modo che ogni moneta possa essere riscattata solo da un piccolo gruppo di venditori. Questa tecnica rende il furto delle monete ancor meno meno conveniente, in quanto saranno difficilmente accettate da un venditore che non sia uno di quelli autorizzati al riscatto. Un'ulteriore verifica di validità può essere effettuata sia dal venditore che dal broker. (Avere monete specifiche per il venditore è una delle caratteristiche principali dello schema Millicent[20]).

La difficoltà ovvia è che nè l'utente nè il broker possono sapere a monte con quale venditore l'utente avrà rapporti, e non è ragionevole costringere l'utente ad acquistare in anticipo monete specifiche per ogni possibile venditore. (Millicent adotta la strategia alternativa secondo cui l'utente contatta il broker in tempo reale, quando ha necessità di avere monete specifiche per un nuovo venditore. Egli necessita anche di contattare il broker per restituire le monete non spese specifiche per quel venditore). Possiamo scavalcare questi problemi con un estensione dello schema personalizzato per l'utente, descritto prima, nel quale l'utente acquista un blocco di monete consecutive di MicroMint.
Intuitivamente l'idea è la seguente: scegliamo un valore $v < u$; sia y indice di un recipiente ad u bit, diviso in una parte superiore y' di v bit e in una parte inferiore y'' di v bit. Consideriamo che y' specifichi un indice un "super-recipiente" e che y'' specifichi un recipiente interno ad esso. Un utente, ora, acquista biglie all'ingrosso e crea le proprie monete. Egli acquista biglie dal super-recipiente, ottenendone 2^v per super-recipiente con una sola in ogni recipiente interno al super-recipiente. Egli compra k super-recipienti di biglie per 2^v cent. Una moneta dall'utente U è valida per essere riscattata dal venditore V se:

$$y^{'}_{i+1} = y^{'}_{i} + d^{'}_{i} \ (mod\ 2^{u-v}), \quad per\ i = 1, 2, ..., k-1$$

e

$$y^{''}_{i+1} = y^{''}_{i} + d^{''}_{i} \ (mod\ 2^{u}), \quad per\ i = 1, 2, ..., k-1$$

dove

$$h'(U) = (d^{'}_{1}, ..., d^{'}_{k-1})$$

e

$$h^{''}(V) = (d^{''}_{1}, ..., d^{''}_{k-1}).$$

Il broker sceglie il prossimo super-recipiente disponibile da dare all'utente; gli altri sono poi unicamente determinati dalle differenze di $\left\{d^{'}_{i}\right\}$, dall'identità dell'utente e dalla scelta di quello iniziale. Analogamente, per fare una moneta per un particolare venditore, l'utente sceglie una biglia dal prossimo recipiente del suo primo super-recipiente, e deve usare biglie dei recipienti negli altri super-recipienti che sono poi unicamente determinate dalle differenze $\left\{d^{''}_{i}\right\}$, definite dall'identità del venditore e dal primo recipiente scelto. Notiamo che le biglie del primo super-recipiente vengono usate solo una volta, per consentire l'individuazione delle monete riutilizzate, mentre quelle provenienti dagli altri super-recipienti possono apparire più di una volta (in monete pagate a diversi venditori), o non apparire affatto. Può essere difficile per un broker creare super-recipienti pieni anche se egli lancia molte biglie. Egli potrebbe vendere super-recipienti abbastanza pieni, ma in seguito, un utente, potrebbe avere difficoltà nel produrre monete per più venditori. Per compensare ciò, il broker può ridurre il prezzo di 1 cent per ogni recipiente pieno venduto.

Produzione anticipata di monete per più mesi La principale linea di difesa contro la falsificazione di monete in larga scala, è il fatto che il broker può produrre con anticipo le monete, mentre un tentativo da parte di un falsario può avvenire solo dopo che sono stati annunciati i nuovi criteri di validità per il mese corrente. Una tecnica con la quale calcolare le biglie per le monete di un singolo mese, richiede otto mesi di tempo, ma il broker non viene

danneggiato, perchè egli può contemporaneamente generare biglie per otto mesi consecutivi. Il falsario così avrà il duplice problema di partire in ritardo e di essere più lento, anche se egli disponesse delle stesse risorse computazionali del vero broker.
In questo metodo il broker cambia il criterio di validità mensile non cambiando la funzione hash h, ma, svelando ogni mese un nuovo valore z tale che la biglia x si valida quando i t bit di peso maggiore di $h(x)$ sono uguali a z. Il broker, segretamente ed in modo del tutto casuale, sceglie in anticipo i valori z che saranno usati per ognuno dei prossimi otto mesi. Lanciare una biglia, significa ancora implementare la computazione di una funzione hash, ma la biglia lanciata è potenzialmente "buona" per ognuno dei prossimi otto mesi, ed è futile per il broker determinare se questo è il caso. Di contro il falsario conosce solo il valore corrente di z e non può permettersi di memorizzare tutte le biglie che lancia, poichè la memoria è relativamente costosa ed alla fine solo una piccola frazione delle biglie lanciate, (nel nostro esempio 2^{-21}) sono considerate "buone", ogni dato mese.
Assumiamo che all'inizio del mese j il broker abbia tutte le biglie che gli occorrono per codesto mese, $\frac{7}{8}$ di quelle necessarie per il mese $j+1$, $\frac{6}{8}$ per il mese $j+2...$, ed $\frac{1}{8}$ per il mese $j+7$. Durante il mese j il broker lancia biglie scegliendo in maniera random x valori, calcolando $y = h(x)$ e verificando se i t bit di peso maggiore sono uguali a qualcuno dei valori z da essere usati nei mesi $j + 1, ..., j + 8$. Per rallentare l'intervallo al quale egli genera biglie valide per ogni mese che si approssima, egli incrementa di 3 sia n che t. Dopo la lunga computazione di un mese ci aspettiamo che egli abbia tutte le monete necessarie per il mese $j + 1$, $\frac{7}{8}$ per il mese $j + 2$, e così via. Questo è ciò che si desiderava. Il broker ha bisogno di uno spazio di memorizzazione quattro volte superiore, per poter conservare le biglie generate per i mesi futuri, ma esse potranno essere temporaneamente memorizzate su un economico nastro magnetico, perchè egli non ha necessità di rispondere urgentemente alle richieste degli utenti relative a queste monete.

Predicati nascosti La tecnica dei predicati nascosti, usata per contrastare i falsari, funziona così: scegliamo $m > n$, richiediamo che ogni pre-immagine di un bit soddisfi un certo numero di predicati nascosti. Questi predicati dovrebbero essere tali che generare le pre-immagini soddisfacenti i predicati risulti facile (se si conoscono i predicati). Per generare un x_i, si possono scegliere i suoi ultimi n bit in modo casuale, e definire il j-esimo bit di x_i, per $j = m - n, ..., 1$ in modo che sia il j-esimo predicato nascosto applicato ai bit $j + 1, ..., m$ di x_i. I predicati nascosti devono essere equilibrati e difficili da capire da campioni casuali. Con $m - n = 32$, il broker può avere un predicato nascosto per ogni giorno del mese. Egli potrebbe rivelare un predicato al giorno, e chiedere ai venditori di verificare che le monete che essi ricevono, soddisfino questi predicati (in caso contrario le monete non saranno accettare dal broker). Ciò non influirà sulla validità delle monete leggittime già in circolazione, ma rende la falsificazione estremamente difficoltosa, dal momento che il probabile falsario dovrebbe scartare molto del suo lavoro di pre-computazione, ogni volta che viene rivelato un nuovo predicato. Riteniamo che questa tecnica sia decisamente consigliabile per MicroMint.

Altre estensioni Peter Wayner ha proposto una variazione al MicroMint nella quale le monete di valore differente sono distinte da predicati sui valori x di pubblico dominio.

4.4 Paragoni con altri schemi di micropagamento

Paragoniamo *PayWord* e *MicroMint* ad altri schemi di pagamento: *Millicent* [20], *NetBill* [11], *NetCard* [12] e Pedersen [24].
NetBill offre un numero di caratteristiche avanzate (come gli ordini di acquisto elettronici e la crittografia di informazioni acquistate), ma è relativamente costoso: le firme digitali sono usate con difficoltà ed il server di *NetBill* è coinvolto in ogni pagamento.
Millicent usa le funzioni hash in maniera estensiva, ma il broker deve essere on-line ogni volta che l'utente desidera interagire con un nuovo venditore. L'utente acquista un certificato specifico per il venditore dal broker. Per applicazioni come la lettura delle pagine web, (web browsing), dove nuovi rapporti utente-venditore nascono con continuità, *Millicent* colloca in tempo reale un pesante fardello computazionale sul broker. Confrontati con *Millicent,* sia *PayWord* che *MicroMint* consentono all'utente di generare certificati specifici per il venditore, senza alcuna interazione con il broker e senza il peso ulteriore di restituire i certificati specifici per il venditore inutilizzati. Inoltre, *PayWord* è uno schema basato sul credito piuttosto che sul debito.
Anderson, Manifavas, e Sutherland [12] hanno sviluppato un sistema di micropagamento, *NetCard*, che è molto simile a *PayWord* nell'usare serie di valori hash con una radice firmata digitalmente, comunque, nella loro proposta, è la banca, piuttosto che l'utente, a preparare la serie ed a firmare la radice, il che si aggiunge al carico di lavoro della banca. Questo approccio impedisce all'utente di creare nuove serie di monete, benchè un utente di *NetCard* potrebbe spendere una singola serie più volte. Comparato a *PayWord*, *NetCard* è un sistema basato sul debito, piuttosto che sul credito.
Torben Pedersen abbozza una proposta di micropagamento [24] che è pure basata sulle serie hash. Il progetto *CAFE,* i cui dettagli non sono disponibili, è un'elaborazione dell'idea di Pedersen. Similmente seguendo la strada di Pedersen, i ricercatori Hauser, Steiner e Waidner hanno adottato un approccio simile [18].
Con la diffusione su Internet di sistemi di navigazione con interfaccia grafica, basati sul protocollo World Wide Web (tipo Mosaic, NetScape, etc.), si sono sviluppati dei siti che offrono servizi di tipo click-and-pay ove é possibile acquistare, giocare, leggere riviste, ed altro ancora, con un semplice click del mouse. Questo fa pensare che, in un futuro non lontano, l'uso della moneta elettronica su Internet diverrá una realtá.

Riferimenti

[1] G. Medvinsky, B. Newmann. "NetCash: a design for practical electronic currency on the Internet", *Proceedings of the First ACM Conference on Computers and Communications Security*, November 1993.

[2] S. Brands. "Electronic Cash on the Internet", *Proceedings of the Internet Society 1995 Symposium on Network and Distributed System Security*, San Diego, California, February 1995.

[3] D. Chaum. "Blind Signatures for Untraceable Payments", *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983.

[4] D. Chaum. "Security Without Identification: Transaction Systems to Make Big Brother Obsolete", *Communication of the ACM*, October 1985.

[5] D. Chaum, B. Boer, E. Heyst, S. Mjolsnes, and A. Steenbeek. "Efficent off-line electronic checks", *Proceedings of Eurocrypt '89*, 1989.

[6] D. Chaum, A. Fiat, and N. Naor. "Untraceable electronic cash", *Proceedings of Crypto '88*, 1988.

[7] S. Even, O. Goldreich, and Y. Yacobi. "Electronic wallet", *Proceedings of Crypto '83*, 1983.

[8] B. Clifford Neuman. "Proxy-based authorization and accounting for distributed systems", *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993.

[9] T. Okamoto and K. Ohta. "Universal electronic cash", *Proceedings of Crypto '91*, 1991.

[10] B. Pfitzmann and M. Waidner. "How to break and repair a 'provably secure' untraceable payment system", *Proceedings of Crypto '91*, 1991.

[11] "The NetBill Electronic Commerce Project", 1995.
http: //www.ini.cmu/NETBILL/home.html.

[12] R. Anderson, H. Manifavas, and C. Sutherland. "A practical electronic cash system 1995". Available from author: Ross.Anderson@cl.cam.ac.uk.

[13] M. Blaze, W. Diffie, R. L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener. "Minimal key lengths for symmetric ciphers to proviede adequate commercial security: A report by an ad hoc group of cryptographers and computer scientists", January 1996. Available at http: //www.bsa.org.

[14] A. Blum, M. Furst, M. Kearns, and R. J. Lipton. "Cryptographic primitives based on hard learning problems", Douglas R. Stinson, editor, *Proc. CRYPTO 93*, pages 278-291. Springer, 1994. Lecture Notes in Computer Science No. 773.

[15] S. Even, O. Goldreich, and S. Micali. "On-line/off-line digital signatures", G. Brassard, editor, *Proc. CRYPTO 89*, pages 263-277. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[16] P. Hallam-Baker. "W3C payments resources", 1995.
http: //www.w3.org/hypertest/WWW/Payments/overview.html.

[17] N. M. Haller. "The S/KEY one-time password system", *ISOC*, 1994.

[18] R. Hauser, M. Steiner, and M. Waidner. "Micro-Payments based on iKP", December 17, 1995. Available from authors.sti@zurich.ibm.com.

[19] L. Lamport. "Password authentication with insecure communication". *Communications of the ACM*, 24(11):770-771, November 1981.

[20] M. S. Manasse. "Millicent (electronic microcommerce)", 1995.
http://www.research.digital.com/SRC/personal/Mark-Manasse/uncommon/ucom.html.

[21] R. C. Merkle. "A certified digital signature", G. Brassard, editor, *Proc. CRYPTO 89*, pages 218-238. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[22] S. Micali. "Efficient certificate revocation", Technical Report TM-542b, MIT Laboratory for Computer Science, March 22, 1996.

[23] National Institute of Standards and Tecnology (NIST). *FIPS Publication 180: Secure Hash Standard (SHS)*, May 11, 1993.

[24] T. P. Pedersen. "Electronic payments of small amounts", Technical Report DAIMI PB-495, Aarhus University, Computer Science Department, Arhus, Denmark, August 1995.

[25] R. L. Rivest. "The MD5 message-digest algorithm. Internet Request for Comments", April 1992. RFC 1321.

[26] B. Schneier. *Applied Cryptography (Second Edition)*, John Wiley and Sons, 1996.

[27] A. Shamir. "Fast signature screening", CRYPTO '95 rump session talk; to appear in RSA Laboratories' *CryptoBytes*.

[28] P. Wayner. *Digital Cash: Commerce on the Net*, Academic Press, 1996.