Overview of basic concepts

- traditional forms of payment
  - cash
  - payment through bank
  - payment cards

- electronic payment systems
  - basic classification
  - security requirements
**Cash**

- most commonly used form of payment today
  - ~80% of all transactions
  - average transaction value is low

- advantages of cash
  - easy to transport and transfer
  - no transaction costs (no third party is involved directly)
  - no audit trail is left behind (that’s why criminals like it)

- disadvantages of cash
  - in fact, cash is not free
    - banknotes and coins need to be printed and minted
    - old bank notes and coins need to be replaced
    - this cost is ultimately borne by the tax payers
  - needs extra physical security when
    - transported in large quantities (e.g., from the mint to banks)
    - stored in large quantities (e.g., in banks)
    - vaults must be built and heavy insurances must be paid
  - risk of forgery

**Payment through banks**

- if both parties have accounts in a bank, then it is unnecessary for one party to withdraw cash in order to make a payment to the other party who will just deposit it again in the bank
Payment by check

- **Advantages**
  - No need for bank at the time of payment

- **Disadvantages**
  - Returned items
    - If funds are not available on the payer's bank account, the check is returned to the payee's bank
    - If the payee has already been credited, the bank loses money
    - Otherwise, the payee suffers
    - Problem: no verification of solvency of the payer at the time of payment
  - Processing paper checks is very expensive and time consuming
    - Checks must be physically transferred between banks
    - Authenticity of each individual check must be verified

- Still popular in some countries
  - E.g., in the US, ~80% of non-cash payment transactions are check payments with an average value of ~1000$.

Giro payment

- **Advantages**
  - The transaction cannot be initiated unless the payer has enough funds available
  - Can be fully electronic (using the existing banking networks)

- **Disadvantage**
  - The bank must be present at the time of payment

- Quite popular in Hungary
**Payment by card**

1. Present card
2a. Authorization
2b. Authorization
2c. Authorization
3. Prepare voucher
4. Sign voucher
5. Send vouchers
6. Clearing
7. Monthly statement

*Authorization is optional, depends on policy

---

**Payment cards – pros and cons**

- **Advantages**
  - Flexibility of cash and checks (assuming infrastructure is in place)
  - Security of checks (no need to carry cash in pocket)
  - Solvency of the customer can be verified before payment is accepted

- **Disadvantages**
  - Needs infrastructure to be deployed at merchants
    - E.g., card reader, network connection, etc.
  - Transaction cost
    - Covered by merchants
    - Paying with cards is not worth for very low value transactions (below 2$)
Payment card types

- debit card
  - the customer must have a bank account associated with the card
  - transaction is processed in real time: the customer’s account is debited and the merchant’s account is credited immediately

- charge card
  - the customer doesn’t need to pay immediately but only at the end of the monthly period
  - if she has a bank account, it is debited automatically
  - otherwise, she needs to transfer money directly to the card association

- credit card
  - the customer doesn’t need to pay immediately, not even at the end of the monthly period
  - the bank doesn’t count interest until the end of the monthly period

Classification of e-payment systems

- pre-paid, pay-now, or pay-later
  - pre-paid: customer pays before the transaction (e.g., she buys electronic tokens, tickets, coins, …)
  - pay-now: the customer’s account is checked and debited at the same time when the transaction takes place
  - pay-later (credit-based): customer pays after the transaction

- on-line or off-line
  - on-line: a third party (the bank) is involved in the transaction (e.g., it checks solvency of the user, double spending of a coin, …) in real-time
  - off-line: the bank is not involved in real-time in the transactions
General security requirements

- **authorization**
  - a payment must always be authorized by the payer
  - needs payer authentication (physical, PIN, or digital signature)
  - a payment may also need to be authorized by the bank

- **data confidentiality and authenticity**
  - transaction data should be intact and authentic
  - external parties should not have access to data
  - some data need to be hidden even from participants of the transaction
    - the merchant does not need to know customer account information
    - the bank doesn’t need to know what the customer bought

- **availability and reliability**
  - payment infrastructure should always be available
  - centralized systems should be designed with care
    - critical components need replication and higher level of protection

Further requirements

- **atomicity of transactions**
  - all or nothing principle: either the whole transaction is executed successfully or the state of the system doesn’t change
    - in practice, transactions can be interrupted (e.g., due to communication failure)
    - it must be possible to detect and recover from interruptions (e.g., to undo already executed steps)

- **privacy (anonymity and untraceability)**
  - customers should be able to control how their personal data is used by the other parties
  - sometimes, the best way to ensure that personal data will not be misused is to hide it
    - anonymity means that the customer hides her identity from the merchant
    - untraceability means that not even the bank can keep track of which transactions the customer is engaged in
Credit-card based systems

- motivation and concept:
  - credit cards are very popular today
  - use existing infrastructure deployed for handling credit-card payments as much as possible
  - enable secure transfer of credit-card numbers via the Internet

- examples:
  - MOTO (non-Internet based scheme)
  - First Virtual and CARI (non-cryptographic schemes)
  - SSL (general secure transport)
  - iKP (specific proposal from IBM)
  - SET (standard supported by industry including VISA, MasterCard, IBM, Microsoft, VeriSign, and many others)

TLS/SSL

- provides a secure transport connection between applications (typically between a web server and a web browser)

- implemented in all web browsers (e.g., Mozilla Firefox and MS Internet Explorer) and web servers and widely used on the Internet

- most of today’s credit-card based transactions on the Internet use TLS/SSL to protect the credit card number from eavesdropping
Credit-card payment with TLS/SSL

- the user visits the merchant’s web site and selects goods/services to buy
  - state information may be encoded in cookies or in specially constructed URLs
  - or state information may be stored at the merchant and referenced by cookies or specially constructed URLs
- the user fills out a form with his credit card details
- the form data is sent to the merchant’s server via a TLS/SSL connection
  - the merchant’s server is authenticated
  - transmitted data is encrypted
- the merchant checks the solvency of the user
- if satisfied, it ships the goods/services to the user
- clearing happens later using the existing infrastructure deployed for credit-card based payments

Pros and cons of TLS/SSL

- advantages:
  - TLS/SSL is already part of every browser and web server
    - no need to install any further software
    - users are used to it
    - this payment method can be used as of today
- disadvantages:
  - eavesdropping credit card numbers is not the only risk
  - another risk is that credit card numbers are stolen from the merchant’s computer
**SET – Secure Electronic Transactions**

- a protocol designed to protect credit card transactions on the Internet
- initiated and promoted by MasterCard and Visa
  - MasterCard (and IBM) had SEPP (Secure E-Payment Protocol)
  - VISA (and Microsoft) had STT (Secure Transaction Technology)
  - the two proposals converged into SET
- many companies were involved in the development of the specifications (IBM, Microsoft, Netscape, RSA, VeriSign, …)
- the SET specification is available on the web (Google)
- it consists of three books:
  1. Business Description
  2. Programmer’s Guide
  3. Formal Protocol Definition
     (around 1000 pages all together)

**SET participants**

- cardholder
  - wants to buy something from a merchant on the Internet
  - authorized holder of payment card issued by an issuer (bank)
- merchant
  - sells goods/services via a Web site or by e-mail
  - has a relationship with an acquirer (bank)
- issuer
  - issues payment cards
  - responsible for the payment of the dept of the cardholders
- acquirer
  - maintains accounts for merchants
  - processes payment card authorizations and payments
  - transfers money to the merchant account, reimbursed by the issuer
- payment gateway
  - interface between the Internet and the existing credit-card payment network
- CAs
## SET services

- cardholder account authentication
  - merchant can verify that the client is a legitimate user of the card
  - based on X.509 certificates
- merchant authentication
  - client can authenticate the merchant and check if it is authorized to accept payment cards
  - based on X.509 certificates
- confidentiality
  - cardholder account and payment information (i.e., her credit card number) is protected while it travels across the network
  - credit card number is hidden from the merchant too!
- integrity
  - messages cannot be altered in transit in an undetectable way
  - based on digital signatures

## Dual signature – basic concept

- **goal:**
  - link two messages that are intended for two different recipients (e.g., order info and payment instructions in SET)
  - link may need to be proven in case of disputes

```plaintext
data1  →  hash  →  sign  →  K^{-1}_X
       |           |            |         |
       +-----------+            +         +
       |           |            |         |
data2  →  hash  →  hash  (with hash key)
```
Dual signatures in SET

- goal:
  - same as in the basic case, but …
  - the two messages have the same signature

```
data1
hash

hash

hash

K^-1_x

sign

K^-1_x

data1 + data2
```

Overview of message flows

```
Cardholder
order info + payment instruction
ack + services

Internet
authorization request

authorization response + capture token
capture request

capture response

payment gateway

Issuer
authorizing processing
capture processing

Payment network
money transfer

Acquirer
```
Overview of message protection mechanisms

cardholder (C)  merchant (M)  acquirer (via payment gtw)

- **PReq**
- **OI**
- **signature**
- **dual signature**
- **PI**
- **digital envelop**

- **Auth.Req.**
- **KA**
- **M**
- **C**
- **PI**
- **KA**

- **Auth.Res.**
- **KA**
- **M**
- **PRes**
- **C**
- **KM**
- **A**
- **Cap.Token**

- **Cap.Req.**
- **KA**
- **M**
- **KA**
- **A**
- **Cap.Token**

- **Cap.Res.**
- **KM**
- **C**

Why did SET fail?

- less benefits than expected
  - merchants like to collect credit card numbers (they use it as indexes in marketing databases)
  - optionally, SET allows the merchant to get the credit card number from the acquirer → security improvements of SET are negated

- too high costs
  - SET requires a PKI

- no advantages for the customer!
  - the idea was that SET transactions would be handled as “cardholder present” transactions (due to the digital signature)
  - customers prefer MOTO-like systems where they can freely undo a transaction if they are unhappy (not only in case of fraud) → customers were much worse off
  - SET requires the download and installation of a special software, and obtaining a public-key certificate
Electronic cash

- motivation and concept:
  - people like cash (75-95% of all transactions in the world are paid in cash)
  - design electronic payment systems that have cash-like characteristics
  - it is possible to ensure untraceability of transactions (an important property of real-world cash)

- examples:
  - DigiCash (on-line)
  - CAFE (off-line)

E-cash – a naïve approach

- electronic coins: (value, Sig_{bank} (value))

- problem 1: double spending
  - a solution to problem 1:
    - coins can have a serial number: (sn, val, Sig_{bank} (sn, val))
    - the bank maintains a database of spent serial numbers
    - merchants deposit received coins before providing any service or goods
    - only coins that have never been deposited before are accepted by the bank

- problem 2: ever increasing database at the bank
  - a solution to problem 2:
    - coins have an expiration time: (sn, val, exp, Sig_{bank} (sn, val, exp))
    - bank needs to store deposited coins until their expiration time only
E-cash – a naïve approach (cont’d)

- problem 3: traceability

![Diagram of E-cash transactions]

- a solution to problem 3: DigiCash

The main idea of DigiCash

- blind RSA signatures
  - the bank’s public RSA key is (e, m), its private RSA key is d
  - user U generates a coin (sn, exp, val) and computes its hash value
    \( h = H(sn, exp, val) \)
  - user U generates a random number \( r \) (blinding factor), computes \( h \cdot r^e \), and sends it to her bank
  - the bank signs the blinded coin by computing
    \( (h \cdot r^e)^d = h^d \cdot r \)
  - when U receives the blindly signed coin, it removes the blinding:
    \( h^d \cdot r \cdot r^{-1} = h^d \)
  - U obtained a digital signature of the bank on the coin
  - the bank cannot link \( h^d \cdot r \) and \( h^d \) together (\( r \) is random)

- problem 4: How much should the user be charged?
  - the bank signs the blinded coin → it does not know the value of the coin

- a solution to problem 4:
  - the bank can use different signing keys for different denominations
Further mechanisms in DigiCash

- the user must authenticate herself to the bank when withdrawing money, so that the bank can charge her account
- the merchant must authenticate himself to the bank when depositing money, so that the bank can credit his account
- messages should be encrypted in order to prevent theft of money

Brands’ untraceable off-line cash

- most important outcome of European ESPRIT project called CAFE (1992-1995)
- no need for on-line checking of double spending
- the user is untraceable unless she cheats (double-spends)
- if a user spends the same coin twice, her identity will be revealed by the bank when the coins are redeemed
The representation problem

- **Preliminaries**
  - Let $G_q$ be a group of prime order $q$
  - A generator-tuple of length $k$ is a $k$-tuple $(g_1, g_2, \ldots, g_k)$ such that $g_i \in G_q \setminus \{1\}$ and $g_i \neq g_j$ if $i \neq j$
  - A representation of an element $h \in G_q$ with respect to a generator-tuple $(g_1, g_2, \ldots, g_k)$ is a tuple $(a_1, a_2, \ldots, a_k)$ such that $g_1^{a_1} g_2^{a_2} \cdots g_k^{a_k} = h$

- **The representation problem**
  - Given a group $G_q$, a generator-tuple $(g_1, g_2, \ldots, g_k)$, and an element $h \in G_q$, find a representation of $h$ with respect to $(g_1, g_2, \ldots, g_k)$

- **Complexity**
  - Assuming that it is infeasible to compute discrete log in $G_q$, the representation problem cannot be solved in polynomial-time

Protocols

- **Setup**
  - Public parameters: $G_q, g_1, g_2 \in G_q$, hash function $H$
  - The bank’s secret parameter: $x$

- **Opening an account**
  - The user identifies herself, selects a random number $u$, and computes an account number $g_1^u$
  - The bank stores $g_1^u$ together with U’s identifying information

- **Withdrawal**
  - The user generates random numbers $s, x_1, x_2$, and computes $A = g_1^{us} g_2^{x_1}$ and $B = g_2^{x_2}$
  - The user authenticates herself to the bank, and obtains a signature of the bank $\text{Sig}(A, B)$ in a blinded manner
  - The bank cannot link the signature to the identity of the user
  - The coin is the triplet: $(A, B, \text{Sig}(A, B))$
Protocols (cont’d)

- payment
  - the user sends the coin \((A, B, \text{Sig}(A,B))\) to the merchant
  - the merchant generates a challenge \(d = H(A, B, ID_M, \text{date/time})\), and sends it to the user
  - the user computes a response \(r_1 = d \cdot u \cdot s + x_1\) and \(r_2 = d \cdot s + x_2\), and sends \((r_1, r_2)\) to the merchant
  - the merchant verifies \(\text{Sig}(A, B)\) and checks if \(g_1^{r_1} g_2^{r_2} = A^d B\)

- notes:
  - \((r_1, r_2)\) is a representation of \(A^d B\) with respect to \((g_1, g_2)\)
  - computing such a representation is infeasible unless the user knows representations of \(A\) and \(B\)
  - in our case the user knows that \(A = g_1^{us} g_2^s\) and \(B = g_1^{x_1} g_2^{x_2}\), and thus, \(A^d B = g_1^{dus + x_1} g_2^{ds + x_2}\)
  - it can be proven that a user can spend a coin if and only if she knows a representation of both \(A\) and \(B\)

Protocols (cont’d)

- deposit
  - the merchant sends \((A, B, \text{Sig}(A, B)), ID_M, \text{date/time}, (r_1, r_2)\) to the bank
  - the bank re-computes the challenge \(d = H(A, B, ID_M, \text{date/time})\), verifies \(\text{Sig}(A, B)\), and checks if \(g_1^{r_1} g_2^{r_2} = A^d B\)
  - the bank looks up its database to see if this coin has been deposited before
  - if not, then it stores the transaction and credits the merchant
  - if the coin is found, then the bank has
    - \(r_1 = d \cdot u \cdot s + x_1\) and \(r_2 = d \cdot s + x_2\)
    - \(r_1' = d' \cdot u \cdot s + x_1\) and \(r_2' = d' \cdot s + x_2\)

\[
g_1^{(r_1-r_1')(r_2-r_2')} = g_1^{(d-d')us/(d-d')s} = g_1^u \quad \rightarrow \quad \text{the user is identified}
\]
Micropayment schemes

- motivation and concept:
  - many transactions have a very low value (e.g., paying for one second of a phone call, for one article in a newspaper, for one song from a CD, for 10 minutes of a TV program, etc.)
  - transaction costs of credit-card, check, and cash based payments may be higher than the value of the transaction
  - need solutions optimized for very low value transactions (perhaps by sacrificing some security)

- examples:
  - Millicent
  - PayWord
  - MicroMint
  - probabilistic micro-payment schemes

- the truth: micropayment schemes are not very successful so far
  - people are used to get these kind of things for free
  - if they have to pay, they prefer the subscription model

Millicent

- developed by DEC in the mid 90's (published in 1995)

- subscription-like, pre-paid system

- scales very well with the number of customers
  - decentralized
    - a Millicent payment can be validated at a vendor without contacting a third party
    - entirely based on symmetric key cryptography
    - payments can be processed very efficiently
High level overview

- start of a week:

  - user
  - credit card #
  - broker scrip of value V
  - broker
  - vendor
  - macropayment transaction

High level overview (cont’d)

- new day or new vendor:

  - user
  - broker scrip of value V
  - broker
  - request for vendor scrip of value v
  - vendor
  - Millicent protocol
  - vendor scrip of value v
High level overview (cont’d)

- purchase:

  - vendor scrip of value \( v \)
  - request for services of value \( s \)
  - vendor scrip of value \( v-s \)
  - services of value \( s \)

  

  **Millicent protocol**

  

  

  broker

Role of the broker

- provides all the different vendor scrips needed by the customer in return for a single macropayment
  - if the customer bought scrips from the vendors directly, then she would need to run a macropayment transaction with each of them
  - in Millicent, the macropayments are **aggregated** by the usage of the broker

- the broker can get vendor scrips in two ways:
  - scrip warehouse model:
    - the broker buys them from the vendors in large batches
    - vendor scrips are produced by the vendors
    - scrips are stored and re-sold piece by piece to different customers
  - licensed scrip production:
    - the broker generates the vendor scrip on behalf of the vendor
    - the license allows the broker to generate only a specific amount of vendor scrip
    - the license is enforced through normal business practices
    - the broker(s) are typically assumed to be trusted
Scrip properties

- a scrip represents a pre-paid value (like a phone card)
- a scrip is protected by using a one-way hash function and limited symmetric cryptography
  - a scrip can be efficiently produced and validated
  - it cannot be tampered with or its value changed without detection
  - it is computationally expensive to counterfeit a scrip
- each scrip is vendor specific
  - it has value at one vendor only
- a scrip can be used only once
  - double spending is detected by the vendor locally at the time of purchase
- a scrip can be used only by its owner
  - using a scrip requires the knowledge of a secret
  - a stolen scrip cannot be used without the secret
- scrips do not provide anonymity
  - scrips have visible serial numbers that can be traced

Scrip structure

- identifies the vendor
- value of the scrip
- unique serial number of the scrip
- used to compute a shared secret*
- expiration time of the scrip
- optional details about the customer
- manipulation detection code

* CustomerID:
  - unique to every customer
  - need not have any connection with the customer’s real identity
  - a scrip returned as a change will have the same CustomerID as the original scrip used to make the payment
**Double spending prevention**

- the vendor stores the ScripID of all used scrips
- before accepting a scrip, it looks up the database of used ScripIDs
- a scrip is accepted only if its ScripID is not found in the database
- a ScripID must be stored only until the expiration date of the corresponding scrip
  - when the scrip expires, it is not accepted anymore in any case
  - this ensures that the size of the database does not grow forever

**Scrip encryption**

- motivation: to prevent a scrip being stolen
- protocol:
  \[ C \rightarrow V: \text{VendorID}, \text{CustomerID}, \{\text{Scrip, Request}\}_K \]
  \[ V \rightarrow C: \text{VendorID}, \text{CustomerID}, \{\text{NewScrip, OldCert, Response}\}_K \]
- generation of \( K \):
  \[
  \text{CustomerID} \rightarrow \text{selects}
  \begin{cases}
  \text{master cust. sec. i-1} \\
  \text{master cust. sec. i} \\
  \text{master cust. sec. i+1}
  \end{cases}
  \rightarrow \text{master cust. sec. i} \rightarrow \text{hash e.g., MD5} \rightarrow K
  \]
- transport of \( K \) to the customer:
  - the scrip is sold together with \( K \) by the broker
  - buying the scrip needs a secure connection between the customer and the broker (e.g., based on SSL)
Other applications of Millicent

- authentication to distributed services
  - a scrip is similar to a Kerberos ticket
  - authorization can be given in a more dynamic way than in Kerberos
- metering usage
  - a scrip can keep track the number of accesses to a given service
- usage based charges
  - Millicent can be used for per-connection charging for services like e-mail, ftp, etc.
- discount coupons
  - further fields can be added to the scrip to provide discounts for certain contents (e.g., once the customer has bought half of an article, the change scrip can contain a discount for the second half)
- preventing subscription sharing
  - a scrip can be used as a capability to access a subscription service
  - the double spending detection mechanism prevents two users from using the same scrip for accessing the service (i.e., subscription sharing)

PayWord

- designed by Rivest and Shamir in 1996
- representative member of the big family of hash-chain based micropayment schemes
- check-like, credit based (pay later) system
  - payment tokens are redeemed off-line
- uses public key crypto, but very efficiently (in case of many consecutive payments to the same vendor)
  - the user signs a single message at the beginning
  - this authenticates all the micropayments to the same vendor that will follow
PayWord model

- players:
  - user (U)
  - vendor (V)
  - broker (B)

- phases:
  - registration (done only once)
  - payment
  - redemption

Registration phase

- U provides B with
  - account information in a real bank (e.g., her credit card number)
  - shipping address
  - public key

- B issues a certificate for U

\[ \text{Cert}_U = \{ B, U, \text{addr}_U, K_U, \exp, \text{more\_info} \}_K_B^{-1} \]

  more\_info: serial number, credit limit, contact information of B, broker terms and conditions, ...

- the certificate is a statement by B to any vendor that B will redeem authentic paywords (micropayment tokens) produced by U turned in before the expiration date.
Payment phase – commitment

- when U is about to contact a new vendor, she computes a fresh payword chain

\[ w_n, w_{n-1} = h(w_n), \ w_{n-2} = h(w_{n-1}) = h^2(w_n), \ \ldots, \ w_0 = h^n(w_n) \]

where
- n is chosen by the user
- \( w_n \) is picked at random

- U computes a commitment

\[ M = \{ V, \ Cert_U, \ w_0, \ date, \ more\_info \}_{K_U^{-1}} \]

- the commitment authorizes B to pay V for any of the paywords \( w_1, \ldots, w_n \) that V redeems with B before the given date

- paywords are vendor specific, they have no value to another vendor

Payment phase – micropayment

- the i-th micropayment from U to V consists of the i-th payword and its index: \((w_i, i)\)

- when V receives \( w_i \), it can verify it by checking that it hashes into \( w_{i-1} \) (received earlier, or in the commitment in case of \( i = 1 \))

- since the hash function is one-way (preimage resistant) the next payment \( w_{i+1} \) cannot be computed from \( w_i \)

- V needs to store only the last received payword and its index

- variable size payments can be supported by skipping the appropriate number of paywords
  - let’s assume that the value of each payword is 1 cent
  - and the last payword that U sent is \( (w_k, k) \)
  - if U wants to perform a payment of 10 cents, then she sends \( (w_{k+10}, k+10) \)
Redemption phase

- at the end of each day, the vendor redeems the paywords for real money at the broker

- V sends B a redemption message that contains (for each user that contacted V) the commitment and the last received payword $w_k$ with its index $k$

- B verifies the commitment and checks that iteratively hashing $w_k$ $k$ times results in $w_0$

- if satisfied, B pays V $k$ units and charges the account of U with the same amount

Efficiency

- user U
  - needs to generate one signature per "session"
  - needs to perform as many hash computation as the number of paywords needed (pre-computation of hash chains is possible)
  - needs to store the hash chain and her current position in the chain (time-memory trade-off is possible)

- vendor V
  - needs to verify one signature per "session"
  - needs to perform one hash computation per micropayment received
  - needs to store only the last received payword with its index, and the commitment

- broker B
  - needs to verify signatures and compute lot of hashes but all these are done off-line
Probabilistic micropayment

- motivation:
  - in traditional micropayment schemes, the vendor cannot aggregate micropayments of different users
  - if the user spent only a few cents, then the cost of redeeming the micropayment tokens may exceed the value of the payment
  - example: typical value of a payword is 1 cent, whereas processing a credit-card transaction costs about 25 cents

- main idea:
  - suppose that U wants to pay 1 cent to V
  - U sends to V a lottery ticket that is worth 10$ if it wins, and it wins with probability 0.001
  - the expected value of U’s payment is exactly 1 cent
  - if V conducts business with many users, then he approximately earns the value of the services/goods provided
  - advantage: only winning lottery tickets are redeemed at the bank
    - number of vendor-bank transactions is greatly reduced
    - value of lottery tickets surely exceeds the transaction cost

Micali-Rivest scheme

- check based, the user simply signs the transaction

- notation
  - $T$ – encoding of the transaction (IDs of user, merchant, bank, transaction time, etc.)
  - $F$ – fixed public function that maps an arbitrary bit string to a number between 0 and 1
  - $s$ – fixed selection rate of payable checks

- setup
  - everyone establishes his own public key and corresponding private key for a digital signature scheme
  - the merchants signature scheme must be deterministic
    - $\text{Sig}_M(x) = \text{Sig}_M(x')$ if $x = x'$
Micali-Rivest scheme (cont’d)

- payment
  - user U pays by sending $C = (T, \text{Sig}_U(T))$ to merchant M
  - M verifies if $C$ is payable by checking if $F(\text{Sig}_M(C)) < s$

- selective deposit
  - M sends only payable checks to the bank for deposit
  - after verification, B credits M’s account with $1/s$ cents and debits U’s account with the same amount

Some properties

- $\text{Sig}_M(C)$ is unpredictable for both U and M
  - practically, $F(\text{Sig}_M(C))$ is a random number with close to uniform distribution over $[0, 1]$
  - the probability that $F(\text{Sig}_M(C)) < s$ is $s$
  - expected value of a check is 1 cent

- the bank essentially processes macropayments of value $1/s$
  - e.g., if $s = 1/1000$, then the value is $10$\\

- potential “psychological” problem
  - possibility of user’s excessive payments (in the short term)
  - e.g., it has a positive probability that the first 10 checks sent by the user are all payable
    - value of the goods/services received by the user is 10 cent
    - but her account is debited $100$\\
  - in the long run it will work, but users may not tolerate the risk of short term overpaying
**Modified Micali-Rivest scheme**

- notation and setup
  - same as for the basic Micali-Rivest scheme

- payment
  - U pays by sending \( C = (T, \text{Sig}_U(T)) \) to M
  - T contains a serial number SN (assigned sequentially to transactions by U)
  - M verifies if C is payable by checking if \( F(\text{Sig}_M(C)) < s \)

- selective deposit
  - M sends only payable checks to the bank for deposit
  - \( \text{maxSN}_U \) denotes the highest serial number corresponding to U processed by B so far
  - if B receives a new payable check, then
    - B credits M’s account with \( 1/s \)
    - if \( SN > \text{maxSN}_U \), then it debits U’s account with \( SN - \text{maxSN}_U \) and sets \( \text{maxSN}_U \) to \( SN \)

**Illustration**

<table>
<thead>
<tr>
<th>Issued Checks with Serial Numbers</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 2 3 4 5 6 7 8 9]</td>
<td></td>
</tr>
</tbody>
</table>

- deposit (1): 3 = SN > maxSN = 0
  - SN - maxSN = 3
  - debit 3 units
  - total debit so far is 3
  - maxSN := SN = 3

- deposit (2): 5 = SN < maxSN = 8
  - 5 = SN - maxSN = 8

- deposit (3): 8 = SN > maxSN = 3
  - maxSN := SN = 8

**Note:** Total debit of the user is always less than or equal to the highest serial number signed by the user so far.
Some properties

- Cheating is possible
  - The same serial number may be used with different merchants
  - If only one of the two checks is payable than the cheating will not be detected

- However, large scale cheating can be detected with statistical auditing
  - Example:
    - Assume the user uses every serial number twice
    - Number of payments made by the user is N
    - Highest serial number used is N/2, user is charged at most N/2 cents
    - The joint credit of the merchants is approximately N
    - This can be detected by the bank!
    - In addition, the more the user cheats the higher the probability of two merchants depositing checks with the same serial number

MicroMint

- Designed by Rivest and Shamir in 1996

- Optimized for unrelated low-value payments
  (Recall: PayWord is optimized for repeated payments to the same vendor)

- Model:
  - MicroMint coins are produced by a broker
  - The broker sells coins to users (many coins in a single macropayment transaction)
  - The user gives coins to a vendor as payment (any coin can be used with any vendor)
  - The vendor redeems coins at the broker (many coins in a single transaction)
MicroMint coins

- basic requirements
  - coins should be difficult to generate by anyone but the broker
  - validity of coins should be easy to verify by anyone
  - digital signature of the broker?
    - would satisfy these requirements
    - would be costly in terms of computation compared to the value of a coin

- instead, MicroMint coins are represented by hash function collisions
  - let \( h: \{0, 1\}^m \rightarrow \{0, 1\}^n \) be a hash function
  - a pair \((x_1, x_2)\) is a two-way collision if \( h(x_1) = h(x_2) \)
  - a k-way collision is a k-tuple \((x_1, x_2, \ldots, x_k)\) such that \( h(x_1) = h(x_2) = \ldots = h(x_k) \) and all \( x_i \) are different

- each MicroMint coin \((x_1, x_2, \ldots, x_k)\) is worth 1 cent

Minting coins

- the broker generates \( x \) values at random, hashes them, and stores \((x, h(x))\) pairs (sorted by \( h(x) \) values)
- when \( k \) \( x \) values are found that have the same hash value, a coin has been minted
- analogue: throwing balls into \( 2^n \) bins

- the broker should produce at most one coin from each bin (why?)
Minting costs

- finding the first k-way collision needs processing \( \sim 2^{n(k-1)/k} \times \) values
- however, further coins are found easier (there are already many balls in the bins): after processing \( c2^{n(k-1)/k} \times \) values \((1 \leq c \leq 2^{n/k})\), one expects \( c^k \) k-way collisions
- \( k > 2 \) has two advantages
  - increases the effort to find the first collision
  - accelerates minting once the threshold is passed
- problem: computation is much cheaper than storage
  - the number of \( x \) values that can be processed in a month far exceeds the number of \( x \) values that can be stored on a reasonable hard disk
  - how to balance the computation and memory requirements?

Computation-storage trade-off

- solution
  - let \( n = t + u \), and let \( z \) be a bit string of length \( t \) specified by the broker
  - \( x \) is a “good” value if the high order \( t \) bits of \( h(x) \) are equal to \( z \)
  - a coin is valid only if it consists of “good” \( x \) values
- why is this good?
  - \( x \) values that are not “good” need not be stored
  - but they need to be processed in order to know that they are not “good”
- trade-off
  - computation cost is increased (minting process is slowed down) by a factor of \( 2^t \)
  - storage requirement is \( \sim k2^u \) (there are only \( 2^u \) bins)
A detailed scenario

**business plan**
- the broker wants 1M $ profit per month
- the broker charges 10% brokerage fee
  - he sells each coin for 1 cent, but redeems it for 0.9 cent only
- thus, the broker needs to sell $2^{30}$ coins per month
- if each user buys 2500 coins (25 $) per month, then the broker needs to have a customer base of 0.4 million customer

**example parameters**
- $k = 4$, $t = 21$, $u = 31 \Rightarrow n = 52$
- the broker needs to process $\sim k^2 n = 2^{54}$ x values
- only one in $2^{21}$ will be “good” $\Rightarrow$ only $\sim 2^{33}$ “good” x values need to be stored ($2^{31}$ bins, on average 4 values in each bin)
- around half of the bins will contain 4 or more “good” x values $\Rightarrow \sim 2^{30}$ coins are generated

A detailed scenario (cont’d)

**the broker will invest in special hardware that gives him computational advantage over potential forgers**
- $2^{54}$ hash/month = $2^{33}$ hash/sec
- 256 special chips $\Rightarrow 2^{25}$ hash/sec/chip
- such a chip costs a few hundred dollars
- if x values are 16 byte long, then the broker needs $2^{33} \times 16$ byte = $2^{37}$ byte = 128 GB storage
Preventing large-scale forgery

- short coin validity period
  - coins are valid for a short time (e.g., one month)

- coin validity criterion
  - there is a new coin validity criterion in each month
  - the coin validity criterion can be the value of z or the hash function itself
  - the broker keeps the new coin validity criterion secret while minting coins
    for the next month; it is made public at the beginning of the month when the
    new coins are started to be used

- unused bins
  - only half of the bins contain valid coins
  - the broker can detect forgery by noting when he receives coins
    corresponding to bins that he didn’t produce coins from
  - a bit array of size $2^u$ is enough to keep track of unused bins

- special hardware
  - the broker can increase t, and invest in more expensive hardware to keep
    its advantage over attackers

Double spending

- MicroMint coins can be spent several times

- double spending will be detected only when the vendor
  wants to redeem the doubly-spent coin

- the broker knows to which user the coin was sold, and he
  knows which vendor wants to redeem it

- thus, the broker can keep track of how many doubly-spent
  coins are associated with each user and each vendor → a
  large-scale cheater can be identified and expelled from the
  system

+ coins can be made user and vendor specific (see later)
Extensions

- hidden predicates
  - x values are required to satisfy a number of hidden predicates
  - example:
    - x[1..n] are selected randomly
    - x[j] = f(x[1..n]) for n < j ≤ m
  - the hidden predicates should be difficult to learn from random examples
  - the broker can have a hidden predicate for each day of the month (m-n = 32); he would reveal them day by day

- user specific coins
  - the broker sells coins to user U such that for each coin (x₁, ..., x_k)
    h'(x₁, ..., x_k) = h'(U), where h' is another hash function that produces short outputs (e.g., 2 bytes)
  - the vendor authenticates the user and checks that the coins she uses belong to group h'(U)
  - in order to reuse a stolen coin, the cheater must be member of h'(U)

Summary

- credit-card based
  - TLS/SSL
  - SET

- e-cash
  - DigiCash: untraceable, on-line
  - CAFÉ: untraceable, off-line

- micropayments
  - MilliCent: symmetric crypto, change
  - PayWord: hash chains
  - MicroMint: hash collisions
  - probabilistic: lottery tickets