# Advanced Web Security

Electronic Payments
Part 2

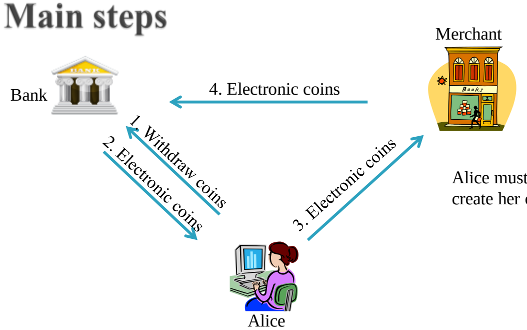


## Untraceable E-cash

- When using credit and debit cards, the issuing bank can track your shopping behaviour
- With cash, you are anonymous
  - Well...there is a serial number on bills...but it is quite useless for tracking
- Using anonymous electronic coins is one alternative
- Two main problems that need to be solved
  - Creation must be controlled by bank
  - Should not be possible to double spend a coin
- **Example:** Principles behind DigiCash



## Main steps

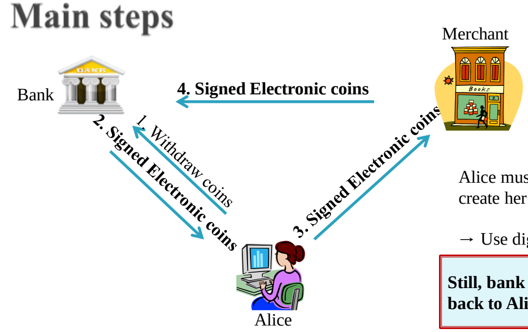



Alice must not be able to create her own coins!

1. Alice asks bank for electronic coins
2. Issue electronic coins
3. Send electronic coins to merchant upon buying something
4. Merchant deposits the electronic coins into his own account



## Main steps




Alice must not be able to create her own coins!

→ Use digital signature

**Still, bank can trace coin back to Alice**

1. Alice asks bank for electronic coins
2. Issue electronic coins
3. Send electronic coins to merchant upon buying something
4. Merchant deposits the electronic coins into his own account

## Blind Signatures

- Idea is to let someone sign a document without seeing the document.
  - ...or digitally sign a number without seeing the number
- Recall RSA:
  - Public modulus *n* and exponent *e*
  - Private exponent *d*.
  - Sign the value *x* by using hash function *h*() and computing

$$\sigma = h(x)^d \bmod n$$

  - Verify by computing

$$\sigma^e = h(x)' \bmod n$$

  - ...and check that

$$h(x) = h(x)'$$



## Blind Signatures

- Multiplicative property of (plain) RSA:

$$(x_1 x_2)^d = (x_1^d \bmod n)(x_2^d \bmod n)$$

- This is why we sign a hash (known redundancy)
- ...but it can also be used to *blind* the signature
  1. Pick random *r*
  2. Let signer sign $r^e \cdot h(x)$
  3. Signature is

$$r^e \cdot h(x) \overset{sign}{\Rightarrow} r^{ed} \cdot h(x)^d \bmod n = r \cdot h(x)^d \bmod n$$

  4. Multiply signature by inverse of *r* to get a signature on *x*

$$r^{-1} r \cdot h(x)^d \bmod n = h(x)^d \bmod n$$



## First protocol, withdrawal

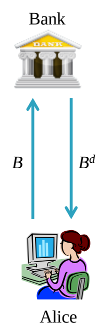
- Alice generates two random numbers
  - *x* is a coin
  - *r* is a blinding value
  - Let *e* = 3
- Alice computes

$$B = r^3 \cdot h(x) \bmod n$$

  and sends *B* to the bank
- Bank signs *B* and returns the signature

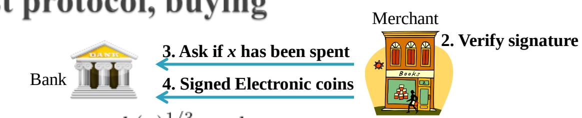
$$r \cdot h(x)^{1/3} \bmod n$$

  to Alice
- Withdrawal is complete!
  - *x* is a coin signed by bank, but bank has not seen *x*, or *h(x)*

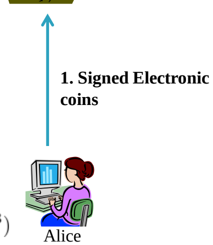





## First protocol, buying




- Alice computes $h(x)^{1/3} \bmod n$
- When buying something
  1. Alice sends $(x, h(x)^{1/3})$ to Merchant
  2. Merchant verifies the signature using the Bank's public key (*e* = 3)
  3. Merchant checks with bank that *x* has not been spent before
  4. Merchant deposits *x* by sending $(x, h(x)^{1/3})$ to the bank




- Bank knows it is a valid coin but it has not seen *x* before so it can not be traced to a specific person

## Adding more features

- Problems
  - Step 3 is used to prevent double spending, but it is not very practical
  - If Alice double spends, she is still anonymous and can not be punished
- The following two features will be added
  1. Merchant does not have to contact the bank for every transaction in order to check double spending
  2. If and only if Alice double spends, she will be identified by the bank
- Note that by solving the second problem, the first is implicitly solved



## Improved protocol, withdrawal

- Alice chooses $2k$ quadruples of random numbers

$$(a_i, c_i, d_i, r_i) \qquad 1 \le i \le 2k$$

- Let

$$x_i = h(a_i, c_i) \qquad y_i = h(a_i \oplus ID, d_i)$$

and compute

$$B_i = r_i^3 f(x_i, y_i) \bmod n$$

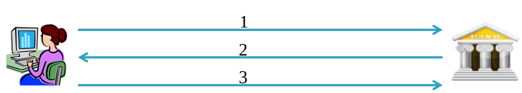
- These $B_i$ values are sent to the bank
- Bank uses cut-and-choose to verify that a random half of the $B_i$ correctly identifies Alice
- Rest are used to compute the blind signature, which is regarded as the coin.



## Cut-and-Choose

$$\begin{array}{ccccccc}
(a_1, c_1, d_1, r_1) & \to & x_1 = h(a_1, c_1) & y_1 = h(a_1 \oplus ID, d_1) & \to & B_1 = r_1^3 f(x_1, y_1) \\
(a_2, c_2, d_2, r_2) & \to & x_2 = h(a_2, c_2) & y_2 = h(a_2 \oplus ID, d_2) & \to & B_2 = r_2^3 f(x_2, y_2) \\
(a_3, c_3, d_3, r_3) & \to & x_3 = h(a_3, c_3) & y_3 = h(a_3 \oplus ID, d_3) & \to & B_3 = r_3^3 f(x_3, y_3) \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
(a_{2k}, c_{2k}, d_{2k}, r_{2k}) & \to & x_{2k} = h(a_{2k}, c_{2k}) & y_{2k} = h(a_{2k} \oplus ID, d_{2k}) & \to & B_{2k} = r_{2k}^3 f(x_{2k}, y_{2k})
\end{array}$$

1. Alice sends all $B_i$ to bank
2. Bank selects $k$ indices randomly and sends these to Alice
$$R = \{i_1, i_2, \ldots, i_k\}$$
3. Alice reveals how $B_i$ was computed for these indices. Sends
$$(a_i, c_i, d_i, r_i), \quad i \in R$$
4. Bank checks that $ID$ is ok for all






## Cut-and-Choose

- For all other indices, Bank computes

$$\prod_{i \notin R} B_i^{1/3} \bmod n \left( = \prod_{i \notin R} \left(r_i^3 f(x_i, y_i)\right)^{1/3} = \left(\prod_{i \notin R} r_i\right) \underbrace{\left(\prod_{i \notin R} f(x_i, y_i)\right)^{1/3} \bmod n}_{S} \right)$$

and sends this value to Alice

- Alice extracts $S$ which is the coin

$$S = \prod_{i \notin R} B_i^{1/3} \left(\prod_{i \notin R} r_i\right)^{-1} \bmod n$$

## Improved protocol, Purchase

- Alice sends the signature to Merchant
- Merchant generates random $z = (z_1, z_2, \dots, z_k)$ and sends to Alice
- Alice returns

$$\begin{cases} (x_j, a_j \oplus ID, d_j), & \text{if } z_j = 0 \\ (y_j, a_j, c_j), & \text{if } z_j = 1 \end{cases}$$

- Now, Merchant can verify the signature since

$$x_i = h(a_i, c_i) \qquad y_i = h(a_i \oplus ID, d_i)$$

- ...but not identify Alice
- Merchant can at any time send coin, *z* and Alice's responses to Bank
  - If Alice double spends, Bank can identify Alice since the new merchant will use another *z*



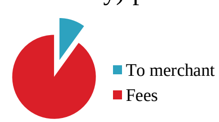
## Possible improvements

- Alice can use a signature together with ID so she can not be framed by bank
- Zero-knowledge proofs can be used instead of cut-and-choose
  - Alice proves that her ID is inside $B_i$ without revealing half of the $B_i$ values
  - See eVoting lecture for more info on this
- Minimize computations, storage space, amount of communication needed etc...

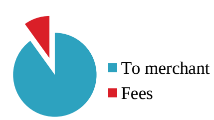


## Micropayments

- Card fees and interchange fees are sometimes large compared to purchase
- Buying/selling cheap items not (economically) possible
- **Micropayment:** payment where transaction fee is a substantial part of total transaction



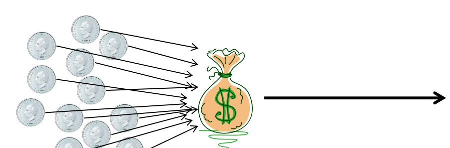

- **Macropayment:** Payment where transaction fee is a small part of total transaction






## Aggregation

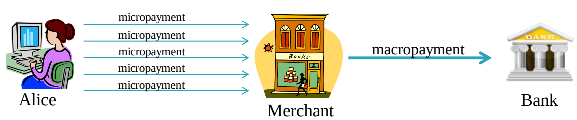
- All micropayment schemes are based on aggregation
  - Transform several *micropayments* to one *macropayment*



- Three types of aggregation
  - Session-level aggregation
  - Universal aggregation
  - Aggregation by intermediation

## Session-level aggregation

- Alice makes several purchases from the same merchant
  - Someone keeps track of total amount
  - After some period of time all purchases are collected into one macropayment
  - Phone bill is one example – but users can not control how much money the company can charge
    - We have to trust their system so they do not charge more than what we have authorized
    - We can easily fix this (at least mathematically)






## Payword, initialization

- Alice *(A)* has a certificate signed by the Bank *(B)*

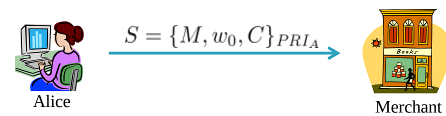
$$C = \{B, A, PUB_A\}_{PRI_B}$$

- When making purchases from a new Merchant, Alice computes a hash chain

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

$$w_n \xrightarrow{h} w_{n-1} \xrightarrow{h} w_{n-2} \xrightarrow{h} \ldots \xrightarrow{h} w_1 \xrightarrow{h} w_0$$

- Alice commits to $w_0$ by sending to Merchant *(M)*




- Merchant checks that Alice has account with Bank



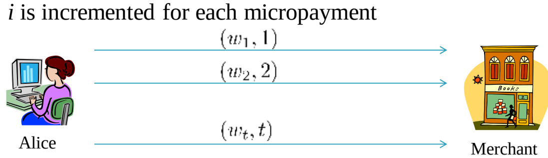
## Payword, Purchases

- When Alice buys something that costs 1 unit she sends

$$(w_i, i)$$

to Merchant

- *i* is incremented for each micropayment



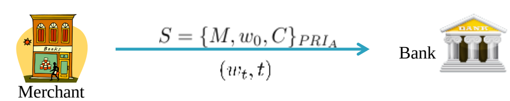

- If something costs *m* units, *i* is incremented by *m*
- Merchant can always check that it is a valid payment
  - But he can never compute

$$w_{t'} \text{ such that } t' > t$$



## Payword, Making the Macropayment

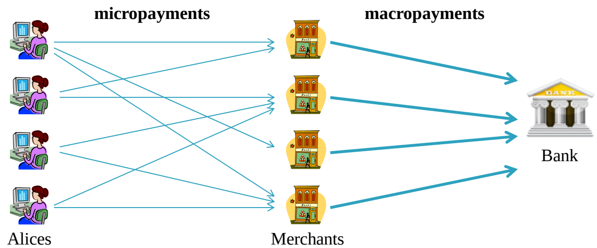
- Commitment *S* and $w_t$ is sent to the bank when *t* is large enough




- Bank verifies $w_t$ before crediting the Merchant's account and debiting Alice's account

## Universal Aggregation

- Session-level aggregation only aggregates between one customer and one merchant
- Universal aggregation is instead many-to-many






## Electronic Lottery Tickets

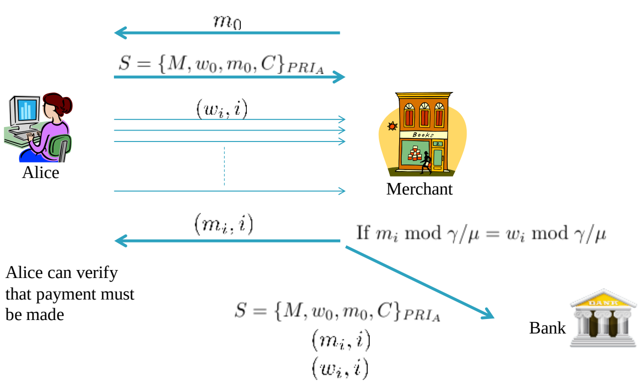
- Probabilistic payments
  - Micropayment is $\mu$ SEK
  - Macropayment is $\gamma$ SEK
  - A macropayment is paid with probability $s = \mu / \gamma$ SEK
- First time Alice buys from Merchant, Merchant creates his own hash chain

$$m_n \xrightarrow{h} m_{n-1} \xrightarrow{h} m_{n-2} \xrightarrow{h} \ldots \xrightarrow{h} m_1 \xrightarrow{h} m_0$$

- And sends $m_0$ to Alice, which is included in her commitment
- If $m_i \bmod \gamma/\mu = w_i \bmod \gamma/\mu$ then Alice pays $\gamma$ SEK



## Electronic Lottery Tickets






## Electronic Lottery Tickets

- Problems
  - Interaction
  - Psychological problem for Alice
    - She sometimes pays more than she has spent.
- Improvement: Peppercoin
  - Alice never pays more than she has actually spent and merchant always gets $\gamma$ SEK
  - Bank takes the psychological problem
  - Less, or no, interaction

## Peppercoin, Micropayment Purchase

**Basic principles**

- $T$ is info about purchase
- $S$ is a number that is incrementing for each micropayment
- $F$ is a function mapping a binary string to a number between 0 and 1
- Alice sends $\{T,S\}_{PRI_A}$ to Merchant

Alice

$$\xrightarrow{\{T,S\}_{PRI_A}}$$

Merchant

- Macropayment is made if

$$F(\{\{T,S\}_{PRI_A}\}_{PRI_M}) \leq s = \mu/\gamma$$



## Peppercoin, Macropayment Transaction

**Basic principles**

- If macropayment should be made, the data is sent to the bank

Merchant

$$\xrightarrow[F(\{\{T,S\}_{PRI_A}\}_{PRI_M})]{\{T,S\}_{PRI_A}}$$

Bank

- Bank keeps record of highest S that has been paid, $S_{max}$
- Bank verifies signatures
  - Credits merchant's account with $\gamma$ SEK
  - Debits Alice's account with $\mu(S - S_{max})$
  - $S_{max}$ updated as $S_{max} \leftarrow S$
- Need to make sure that $S$ is not reused with different merchants



## Aggregation by Intermediation

- A third party is placed inbetween users and merchants to keep track of all micropayments
  - When a user has paid enough, he/she will be charged by the intermediary
    - Or he/she will pre-pay a certain number of transactions
  - When merchant has received enough, he will get transaction from intermediary