Practical Divisible E-Cash From Bounded Accumulator

Man Ho Au
Joint Work with Willy Susilo and Yi Mu
Centre for Computer and Information Security Research,
School of Computer Science and Software Engineering,
University of Wollongong, Australia
Email: mhaa456@uow.edu.au

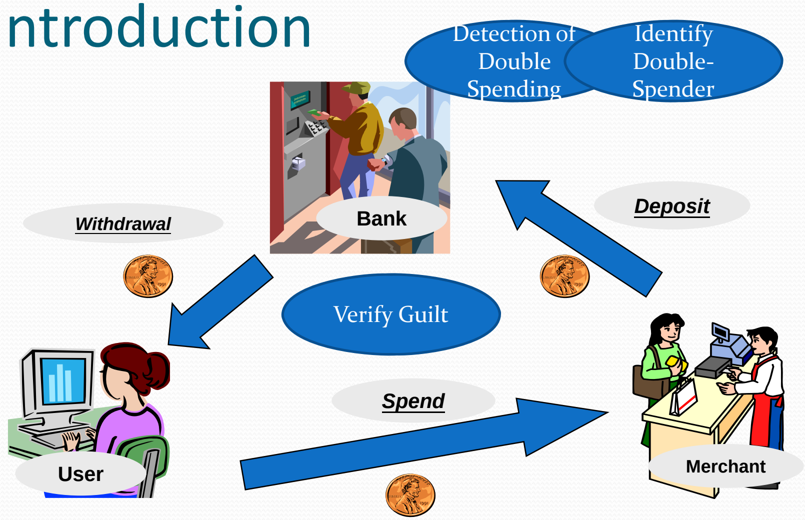
Outline

- Introduction
- Requirements
- Useful Tools
- Our Construction
- Conclusion

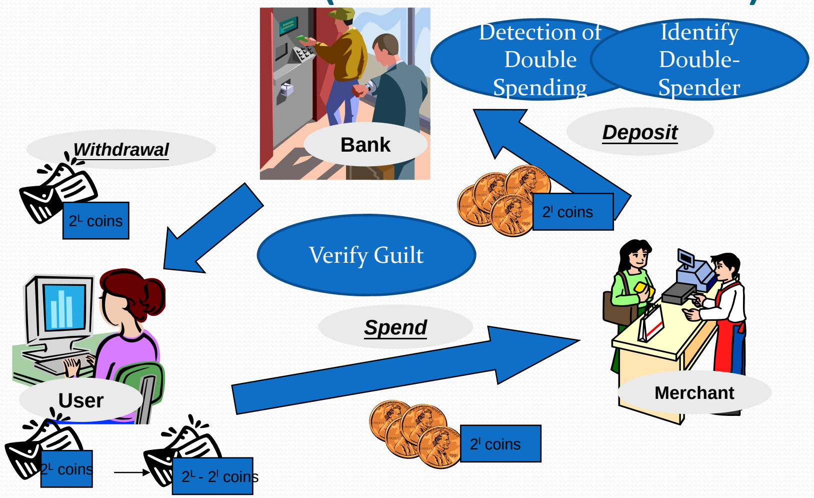
Introduction

- Electronic Cash
  - Introduced by D. Chaum in 1982
  - 3 parties: Bank, User, Merchant
  - 4 main operations: Account Establishment, Withdrawal, Spend, Deposit

Introduction



Introduction (Divisible E-Cash)



Requirements

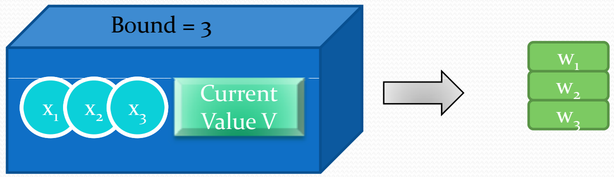
- Offline
- Balance
  - No collusion of users and merchants can deposit more than what they have withdrew without being detected
- Anonymity
  - Weak – anonymity of spender
  - Strong – spending of the user cannot be linked
- Identification of Double-Spender
- Exculpability
  - No collusion of bank, users and merchants can prove an honest user to have double-spent

Useful Tools

- Bounded Accumulator [Au et al 07]
  - An accumulator accumulates multiple values into one single value such that, for each value accumulated, there is a witness proving that it has indeed been accumulated.
  - A bounded accumulator is an accumulator with the constraint that at most *k*, called the bound of the accumulator, values could be accumulated into the accumulator.

Useful Tools

- Bounded Accumulator



Since the bound is 3, no more elements can be accumulated.

Useful Tools

- Bounded accumulator
  - With x_i, w_i and v (i=1,2,3), everyone can be assured that x_i has been accumulated.
  - On the other hand, it is hard to compute a witness-value pair (x*, w*) for a accumulator value, say v, if x* is not accumulated (that is, $x^* \neq x_1, x_2, x_3$)

Useful Tools

- Bounded Accumulator
  - Setting: Let G_1, G_2 be a bilinear group pair such that there exists a pairing e: $G_1 X\ G_1 \rightarrow G_2$. Let g_1, g_2, h be generators of G_1. Assume order of G_1 and G_2 is a prime p. Let α be a secret random number and set $h_1=h^\alpha$, …, h_k=h^{α ^k}.
  - To accumulate a set $\{s_1, \ldots, s_k\}$, compute
    - V = h^{(α+s_1) (α+s_2)…(α+s_k)}
  - The witness w_i of s_i being accumulated is
    - w_i = h^{(α+s_1)… (α+s_k) / (α+s_i)}

Useful Tools

- Bounded Accumulator
  - Given w_i , s_i and v, one can tell if s_i is accumulated in v by testing if $e(w, h_1 h^{si}) = e(v, h)$
  - Using zero-knowledge proof of knowledge technique (ZKPoK), one can prove that he is in possession of a pair (s,w) such that the above relationship holds without revealing s and w.
  - One can also conduct a ZKPoK such that one is in possession of the triple (s, w, v) without revealing them.

Useful Tools

- Signature Scheme with Efficient Protocols
  - Allow signing on message in commitment
  - Allow zero-knowledge proof-of-knowledge of possession of signature and message pair
  - Example: CL, CL+
  - However, the message space of the above scheme does not match with the need of our construction.
  - Modified Extended special signature (ESS+)
    - Message space is a cyclic group equipped with a bilinear map

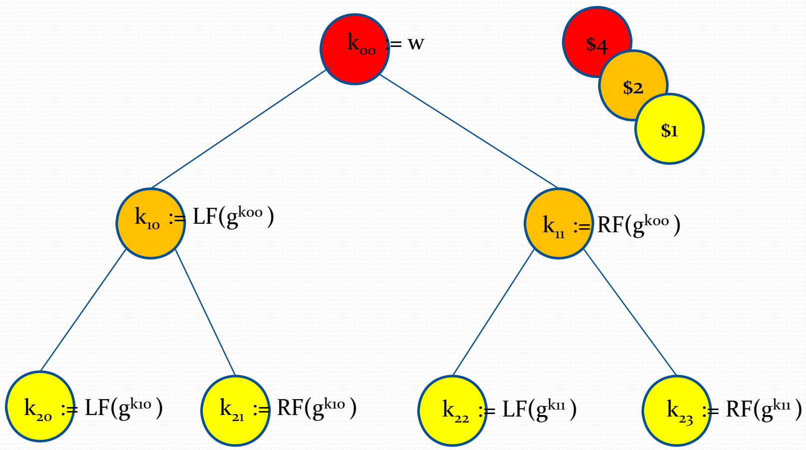
Our Construction

- Our construction is motivated by the first divisible E-Cash with Strong Anonymity due to Canard and Gouget
- The main trick of our construction is the use of a bounded accumulator, in combination with the classical binary tree approach

Our Construction

- User Alice is equipped with a DL type key pair, (pk, sk):= (u^x , x).
- To withdrawal a 2^L-spendable coin (or a wallet of value 2^L), construct a binary tree of L+1 level.
- In the following, we consider an example with L = 2.
- Alice choose a wallet secret w and construct a binary tree of 3 levels as shown. This tree corresponds to a wallet of 4 coins.

Our Construction



*LF, RF are hash functions chosen by the bank

Our Construction

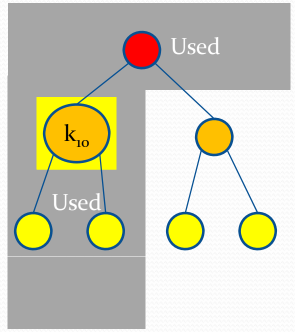
- Alice then "compresses" the binary tree using bounded accumulator:
  - Compute V_0 as the accumulation of $k_{00,}$
  - V_1 as the accumulation of k_{10} , k_{11} and
  - V_2 as the accumulation of k_{20} , k_{21} , $k_{22,}$, k_{23}
- The triple (V_0 , V_1 , V_2) is used by Alice to represent the binary tree.

Our Construction

- Withdrawal Protocol:
  - Alice first identifies herself to the bank (by demonstrate the knowledge of x in the public key u^x)
  - Using a signature scheme with efficient protocols, Alice obtains 3 signatures from the bank, namely,
    - σ_0 := Sign (V_0, x)
    - σ_1 := Sign (V_1, x)
    - σ_2 := Sign (V_2, x)
  - Alice wallet of 4 coins consist of the binary tree and these 3 signatures (σ_0 ,σ_1, σ_2)

Our Construction

- Now suppose Alice want to spend 2 dollars…
- She first chooses a node (k_{10}) of suitable level and marked the sub-tree from this node as used.
- Compute Serial Number S:= g^{k10}
- Compute Security Tag T:=PKh^{Rk10} where R is a random challenge from the Merchant.



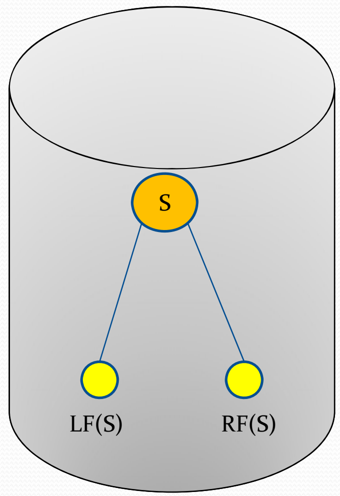
Our Construction

- Spend Protocol:
  - Alice sends S, T to the merchant, along with a non-interactive zero-knowledge proof-of-knowledge Π on x, σ_1, V_1,k_{10} such that
    - σ_1 is a valid signature on (x, V_1)
    - k_{10} is a value accumulated in V_1
    - S= g^{k10}
    - T=$u^x h^{Rk10}$

**The above is in fact proving that the node being used is in the 1st level of the binary tree, note also that the bank/merchant doesn't know whether Alice is using k10 or k11*

Our Construction

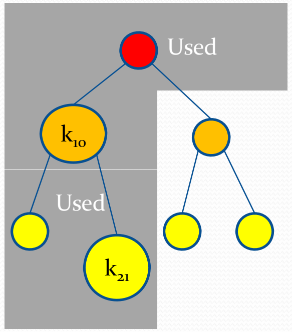
- Deposit:
  - Merchant gives the bank S, T, along with Π to the bank.
  - From S, the bank recovers the sub-tree from the node being spent and store it in its database of spent coins.



Bank's Database

Our Construction

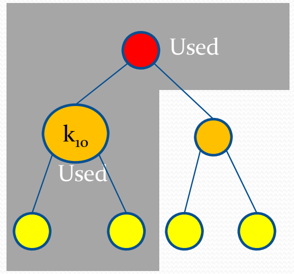
- What if Alice try to over-spend by using one of the descendant of a spent node?
  - Suppose Alice try to use k_{21}
  - $S' = g^{k21}$
  - $T' = u^x h^{R'k21}$
  - As $k_{21} = RF(g^{k10})$ and g^{k10} is in the bank's database already, Alice will be detected
  - Now from $T' / h^{R'k21}$, public key of Alice is revealed



*The case of spending an ancestor is the same

Our Construction

- What if Alice reuses the same node?
  - $S' = g^{k10}$
  - $T' = u^x h^{R'k10}$
  - Now S in the bank's database is equal to S', so Alice will be detected
  - The bank computes
  - $(T^{R'} / T'^{R})^{1/(R' - R)}$ and obtains the public key of Alice.



Our Construction

- One remaining problem:
  - During the withdrawal protocol, how can the bank ensure Alice generate the accumulator values honestly?
  - Trivial Solution: zero-knowledge proof of knowledge...

Our Construction

- Our approach: Statistical Balance...
  - With certain probability, the bank asks Alice to reveal w.
  - The bank then checks if Alice computes the accumulator values correctly.
  - Cheating is possible... However, as the accumulator is bounded, the gain of Alice is limited. (In our paper, the gain of a cheater is shown to be at most $L2^L$)
  - Thus, a fine of $2L2^L$ is enough to discourage cheating if the bank checks every two withdrawal requests.

Our Construction

- Theorem: Our scheme is secure in the random oracle under the q-SDH assumption and the AWSM assumption.

Analysis

	User		Bank
	With Pre-Processing	W/O Pre-Processing	
Withdrawal Protocol**			
Multi-EXP	2L + 2	2^{L+1} + 9L +5	2^{L+1} + 8L +6
Pairing	2L+2	2L+2	0
	User		Merchant
Spend Protocol			
Multi-EXP	1	21	13
Pairing	0	6	8

*Time Complexities
** Assume the bank checks the withdrawal protocol every 2 requests

Analysis

Withdrawal Protocol**	
G Element	7L + 7
Z_p Element	7L+8
Spend Protocol	
G Element	9
Z_pElement	21

*Space Complexities
** Assume the bank checks the withdrawal protocol for every 2 requests

Conclusion

- We present a practical divisible e-cash with full anonymity.
- This is the second scheme of this kind in the literature while the first scheme is quite inefficient (spending of a single coin requires more than 800 Multi-EXP for normal parameters)
- However, our scheme requires a strong assumption.

Thank You!

- Questions and Comments are Welcome!