Strong Accumulators from Collision-Resistant Hashing

Philippe Camacho *(University of Chile)*
Alejandro Hevia *(University of Chile)*
Marcos Kiwi *(University of Chile)*
Roberto Opazo *(CEO Acepta.com)*

Outline

- Basic Cryptographic Notions
- Motivation
  - e-Invoice Factoring
- Notion of Accumulator
- Our Construction
- Open Problem

Basic Cryptographic Notions

- How to define security?
  - This is one of the cryptographer's hardest task.
  - A good definition should capture intuition… … and more!
  - Community had to wait until 1984 with [GM84] for a satisfactory definition of (computational) *"secure encryption"*.

Basic Cryptographic Notions

- Adversary
  - With unlimited computational power
    - One Time Pad, Secret Sharing
  - Computationally Bounded *(Probabilistic Polynomial Time = PPT)*
    - Key Agreement, Public Key Encryption, Digital Signatures, Hash Functions, Commitments,…

Basic Cryptographic Notions

- Cryptographic Assumptions
  - Most of cryptographic constructions rely on **complexity assumptions**.
    - Factoring is hard.
    - Computing Discrete Logarithm is hard.
    - Existence of functions with “good” properties
      - One-way functions
      - Collision-Resistant Hash functions
    - …
  - All these assumptions require that $P \neq NP$.

Basic Cryptographic Notions

- How to prove security?
  - What we want:
    - Assumption X holds => protocol P is secure.
    - No *adversary* can break X => No *adversary* can break P.
  - What we do:
    - Suppose protocol P is insecure => X does not hold.
    - Let *A* the *adversary* that breaks P => We can build an *adversary* *B* that breaks X.
  - This method is sometimes called *"Provable Security"* or *"Reductionist Security"*.

Basic Cryptographic Notions

- **Let's get into the details…**
  - We need to quantify the probability that an *adversary* can compute some values.
  - **Asymptotic notion**
    - The running time of the adversary depends on the ***security parameter***.
    - **E.g:** size of the secret key in the case of encryption, size of the primes for the factoring assumption.
  - **Definition:** (negligible function)
    A function $\varepsilon : \mathbf{N} \rightarrow [0,1]$ is negligible if for *every polynomial* $q: \mathbf{N} \rightarrow \mathbf{N}$, for k sufficiently large:
    $$\varepsilon(k) < 1/q(k)$$

Basic Cryptographic Notions

- RSA
  - Initialization
    - $n=pq$, p,q safe primes , $\Phi(n) = (p-1)(q-1) = |Z_n^*|$
    - $e \in Z_{\Phi(n)}^*$ (encryption)
    - $d \in Z_{\Phi(n)}^*$ (decryption)
    - $ed = 1 \mod \Phi(n)$ (*Euclidian Algorithm*)
  - Encryption / Decryption
    - $x \in Z_n^*$ plaintext
    - Encrypt: $c = x^e \mod n$
    - Decrypt: $y = c^d \mod n = x^{ed} \mod n = x \mod n$

Basic Cryptographic Notions

- Assumptions
  - *RSA Instance generator*
    $(n,p,q,e,d) \leftarrow I(k)$
  - *Factoring Assumption*
    $\Pr[(p,q)\leftarrow A(n) : n=pq] < \varepsilon(k)$
  - *RSA Assumption*
    $\Pr[y \in_R Z_n^* ; x\leftarrow A(n,y,e) : y=x^e \bmod n] < \varepsilon(k)$
  - *Strong RSA Assumption [BarPfi97]*
    $\Pr[u \in_R Z_n^* ; (x,e)\leftarrow A(n,u) : u=x^e \bmod n, e \neq 1] < \varepsilon(k)$
  - *Strong RSA => RSA => Factoring*
    (note the direction <= is open)

Basic Cryptographic Notions

- Assumptions and efficiency
  - We know how to build encryption schemes based on
    - RSA Assumption
    - Factoring Assumption
  - However encryption algorithms based on the **RSA Assumption** are much *faster* than those based only on the **Factoring Assumption**.

Basic Cryptographic Notions

- Collision-Resistant Hash Functions
  - $H:\{0,1\}^* \rightarrow \{0,1\}^k$
    - Given x, it is *easy* to compute $H(x)$.
    - Given y, *hard* to compute x such that $H(x)=y$.
    - Given x, *hard* to compute $x' \neq x$ such that $H(x)=H(x')$.
    - *Hard* to compute $x \neq x'$ such that $H(x)=H(x')$.

This definition is not formal. Just an intuition.

Basic Cryptographic Notions

- Formal definition for Collision-Resistant Hash Functions
  - **Definition:** (1st attempt)
    A function H is collision-resistant iff:
    For all A: $\Pr[x,x' \leftarrow A():x \neq x' \text{ and } H(x)=H(x')] < \varepsilon(k)$
  - Why does the previous definition not work?
    - A():
      return (x,x') // Where (x,x') is a collision-pair

Basic Cryptographic Notions

- **Definition:**
  (family of collision-resistant hash functions)
  - $\{F_k\}_{k \epsilon N}$ where $F_k=\{Hj,j\ \epsilon J_k\}$ is a family of collision resistant hash functions iff:
    - For all j, H_j can be selected efficiently,
    - $Pr_{j\ \epsilon\ J_k}\ [x,x' \leftarrow A(j,k): x \neq x'\ ,\ H_j(x)=H_j(x')] < \varepsilon(k)$

Basic Cryptographic Notions

- **Assumption:**
  Collision-Resistant Hash Functions Families (CRHF) exist.

Not related to Number Theory!

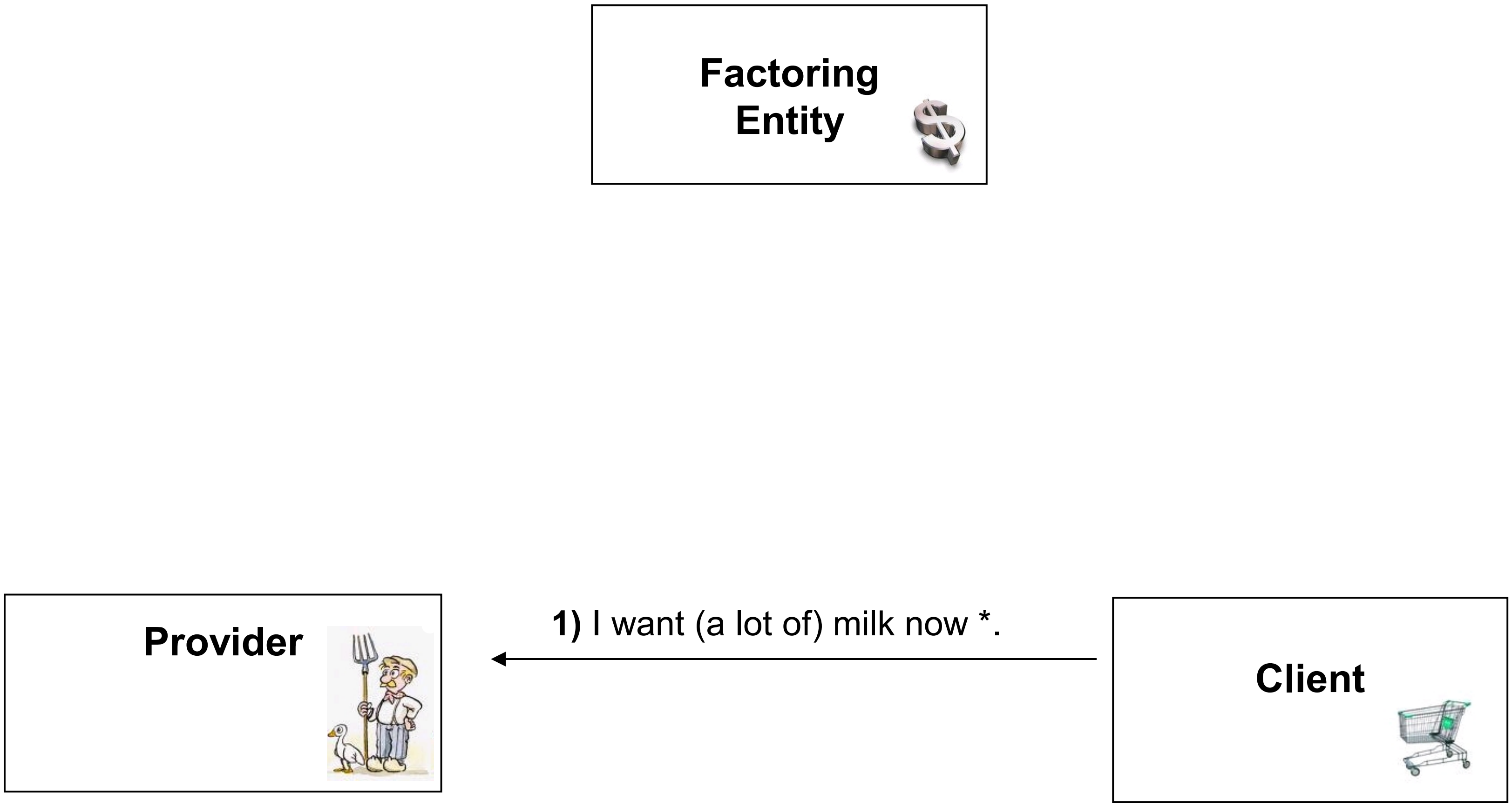
Factoring Industry in Chile

Factoring Entity

Provider

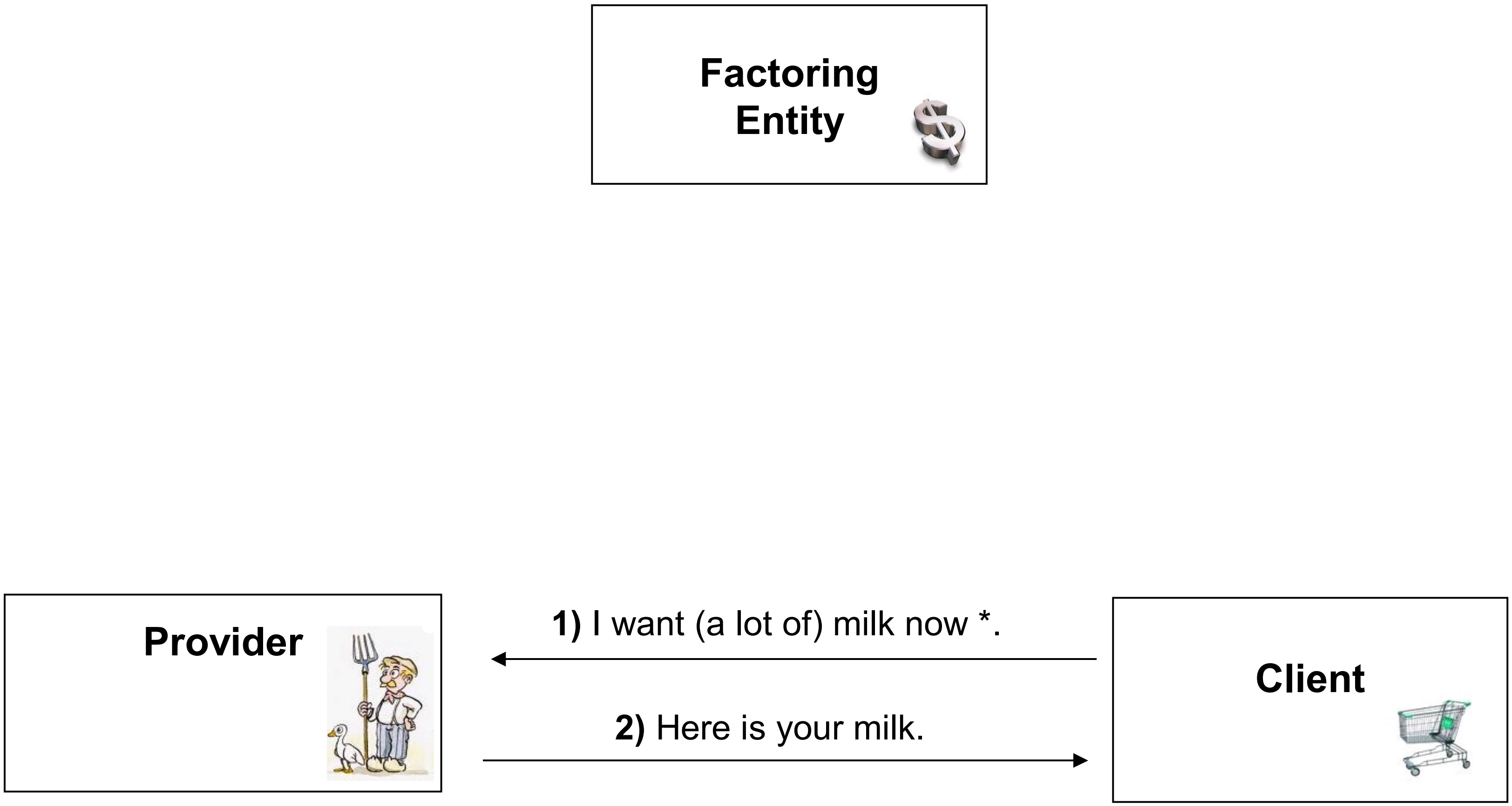
Client

Factoring Industry in Chile



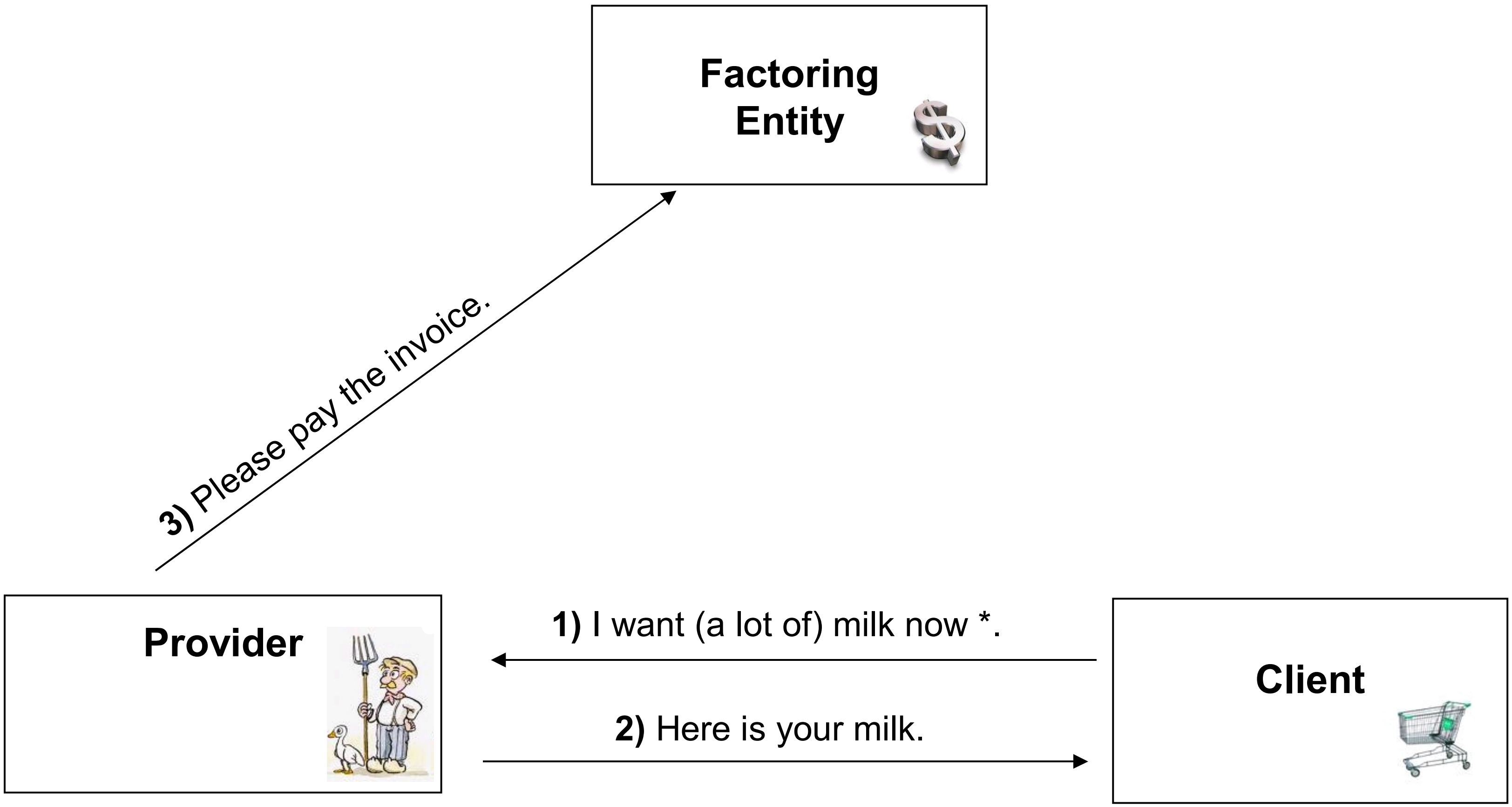
(*) but I do not want to pay yet.

Factoring Industry in Chile



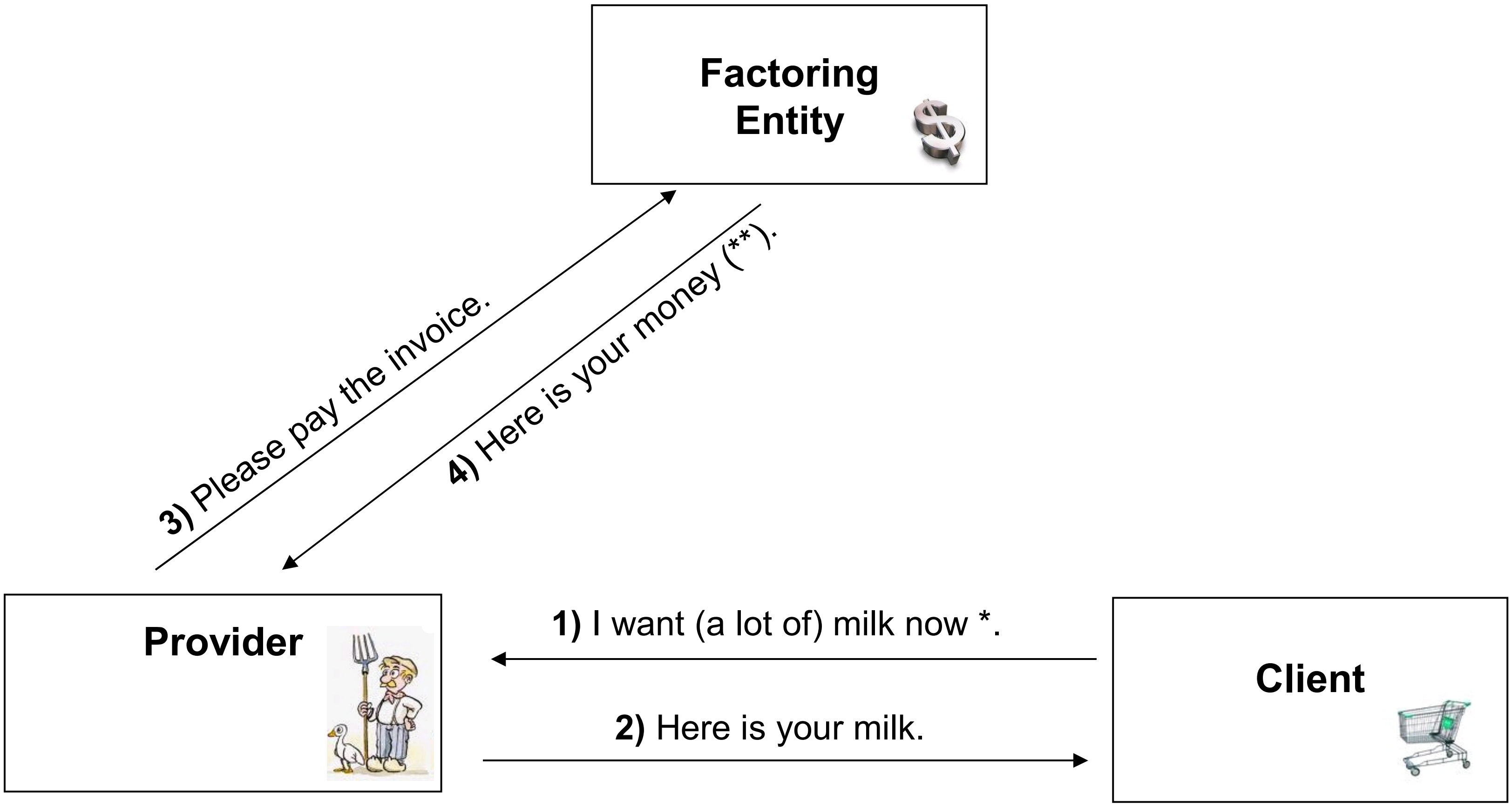
(*) but I do not want to pay yet.

Factoring Industry in Chile



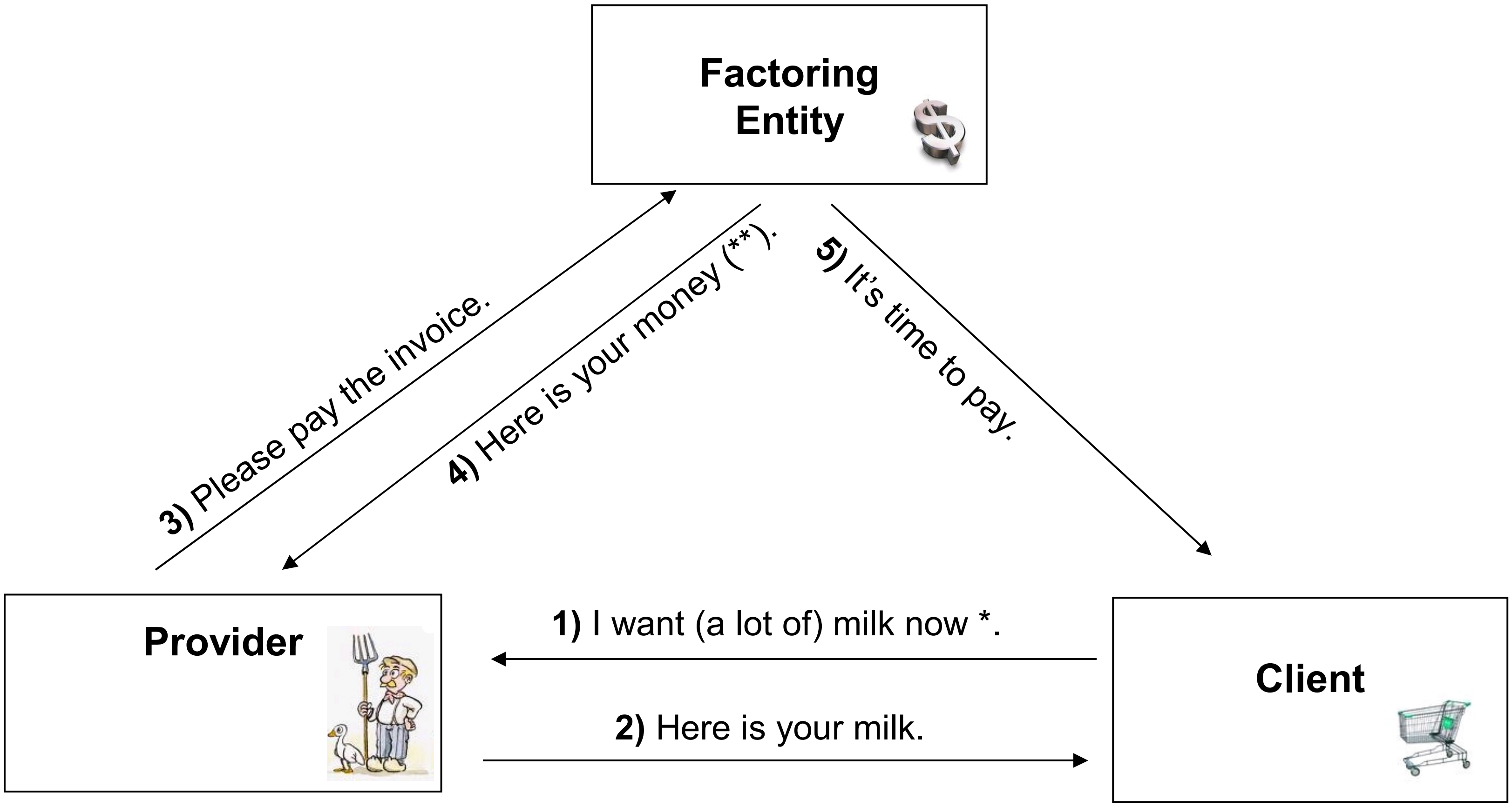
(*) but I do not want to pay yet.

Factoring Industry in Chile



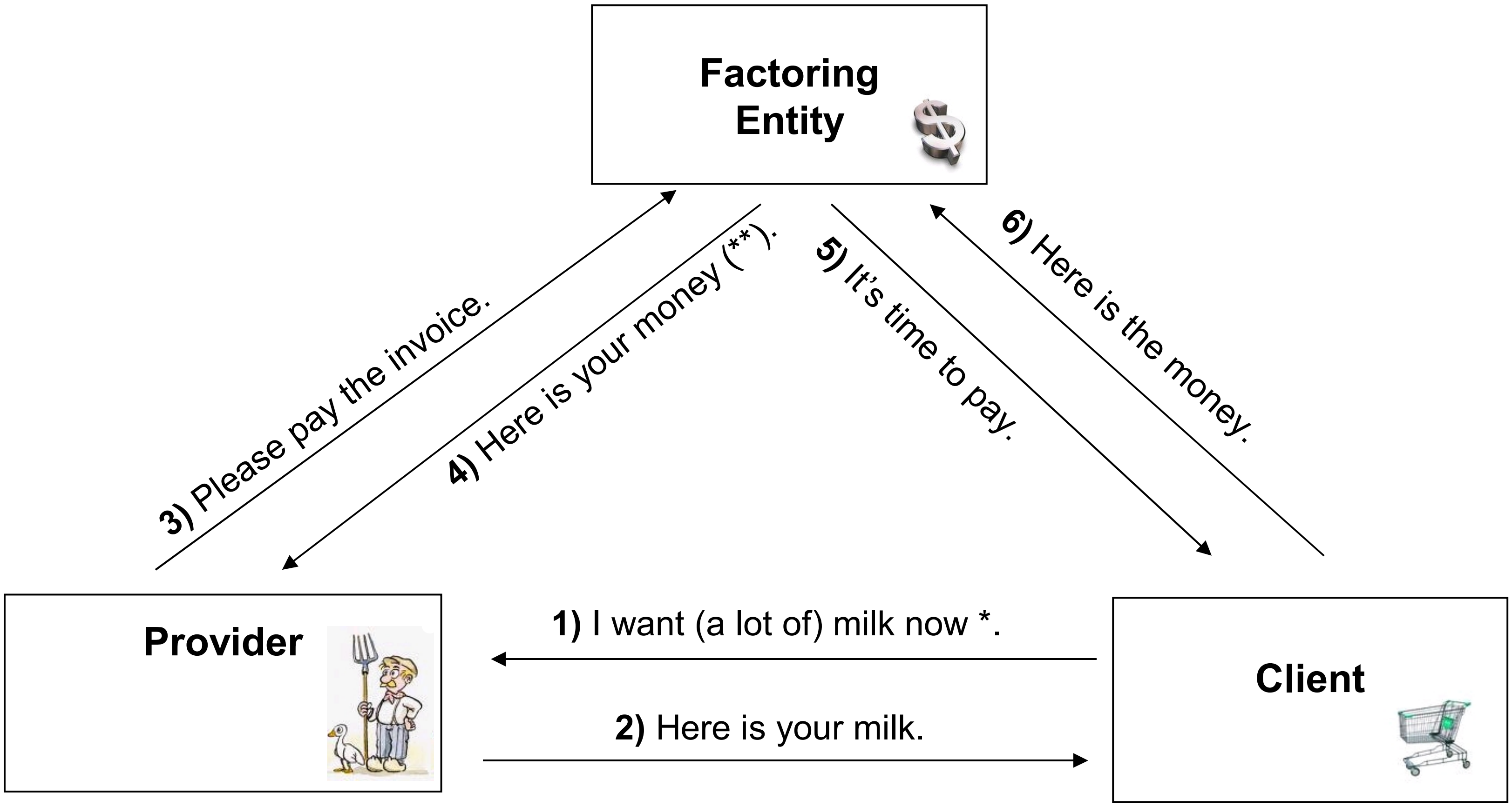
(*) but I do not want to pay yet.
(**) minus a fee.

Factoring Industry in Chile



(*) but I do not want to pay yet.
(**) minus a fee.

Factoring Industry in Chile



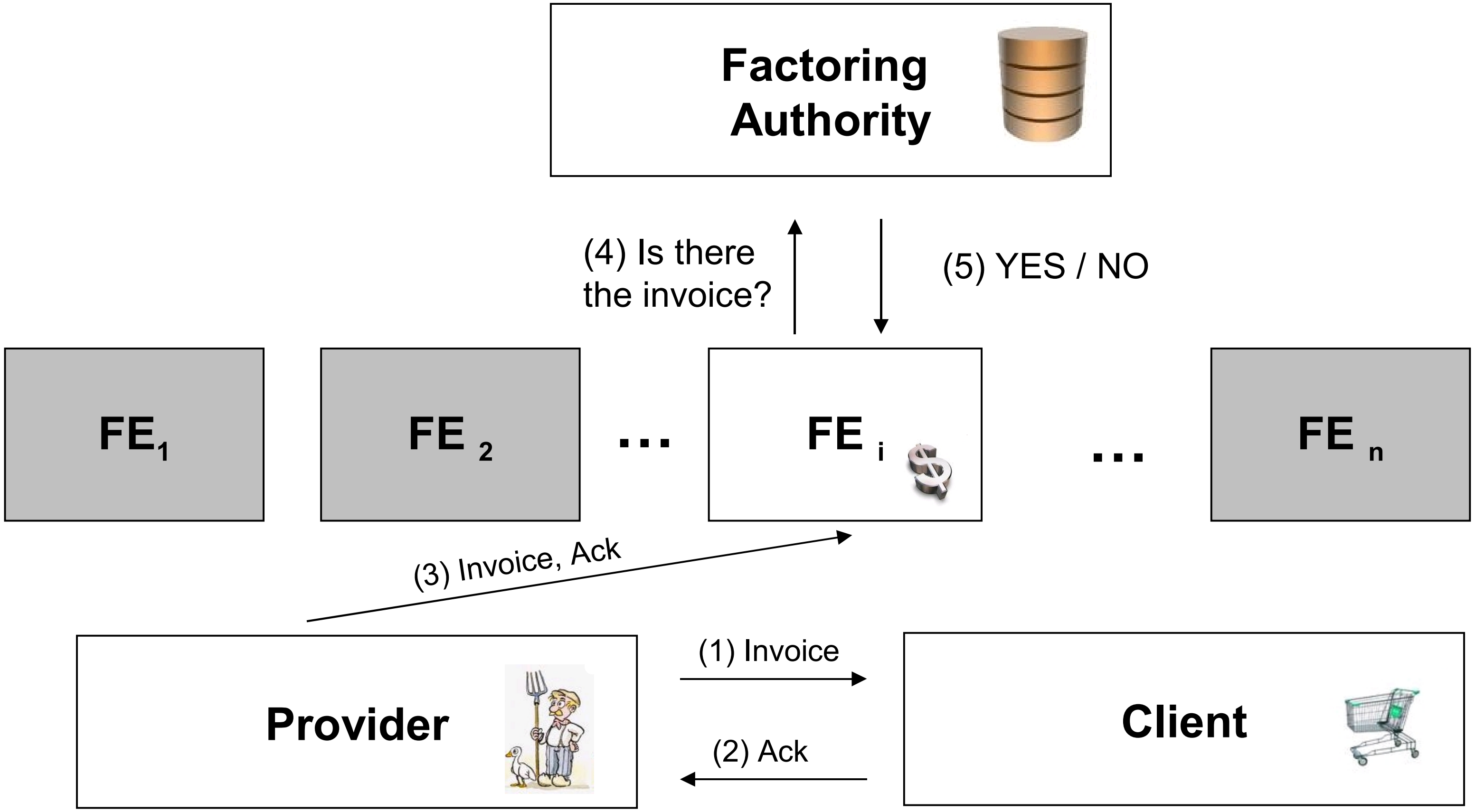
(*) but I do not want to pay yet.
(**) minus a fee.

The Problem

- A malicious provider could send the same invoice to various Factoring Entities.

- Then he leaves to a far away country with all the money.
- Later, several Factoring Entities will try to charge the invoice to the same client. Losts must be shared…

Solution with Factoring Authority



Caveat

- This solution is quite simple.
- **However**
  - Trusted Factoring Authority is needed.
- Can we remove this requirement?

Notion of accumulator

- Problem
  - A set X.
  - Given an element x we wish to prove that this element belongs or not to X.
- Let $X=\{x_1,x_2,\ldots,x_n\}$:
  - X will be represented by a short value Acc.
  - Given x and w (*witness*) we want to check if x belongs to X.

Notion of accumulator

- Participants
  - Manager
    - Computes the accumulated value …
    - … and the witnesses.
  - User
    - Tests for (non)membership of a given element using the accumulated value and a witness provided by the manager.

Properties

- Dynamic
  - Allows insertion/deletion of elements.
- Universal
  - Allows proofs of membership and nonmembership.
- Strong
  - No need to trust in the Accumulator Manager.

Applications

- Time-Stamping [BeMa94]
- Certificate Revocation List [LLX07]
- Anonymous Credentials [CamLys02]
- E-Cash [AWSM07]
- Broadcast Encryption [GeRa04]
- …

Prior work

	Dynamic	Strong	Universal	Security	Efficiency (witness size)	Note
[BeMa94]	✗	✓	✗	RSA + RO	O(1)	First definition
[BarPfi97]	✗	✓	✗	Strong RSA	O(1)	-
[CamLys02]	✓	✗	✗	Strong RSA	O(1)	First dynamic accumulator
[LLX07]	✓	✗	✓	Strong RSA	O(1)	First universal accumultor
[AWSM07]	✓	✗	✗	Pairings	O(1)	E-cash

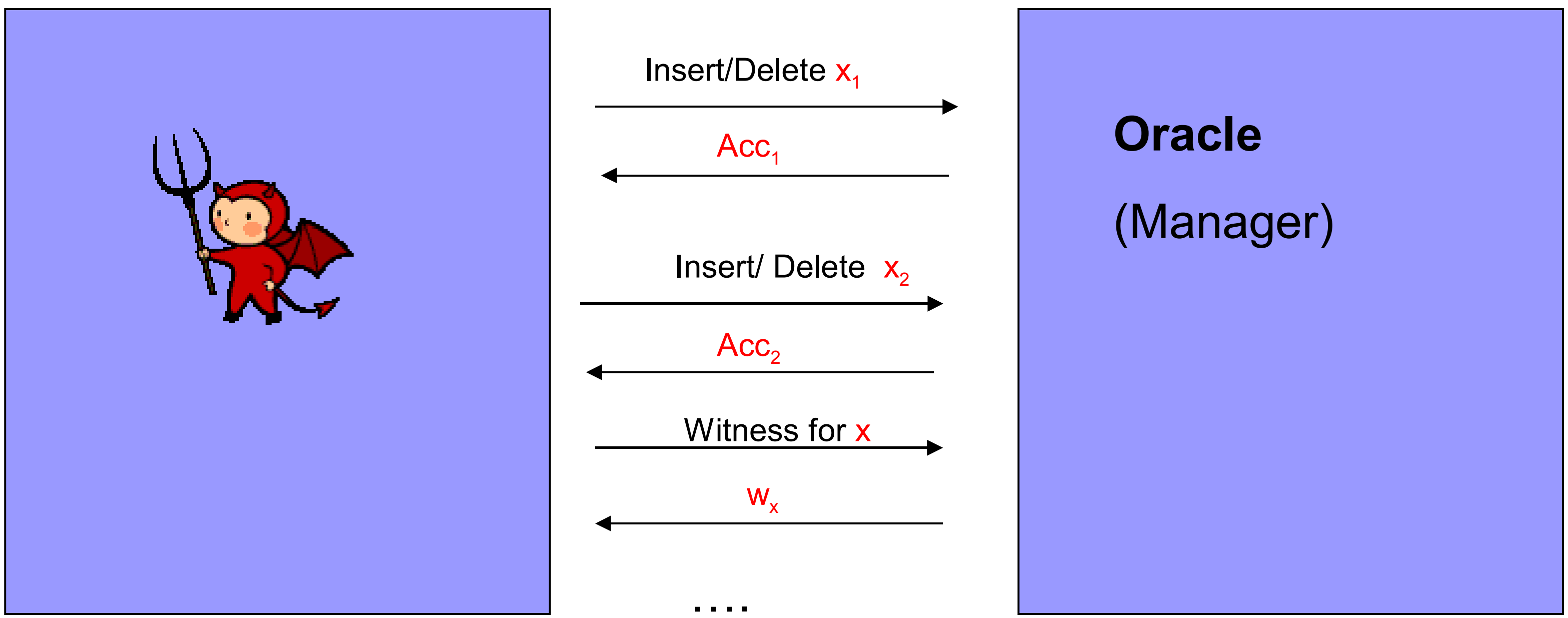
Prior work

	Dynamic	Strong	Universal	Security	Efficiency (witness size)	Note
[BeMa94]	✗	✓	✗	RSA + RO	O(1)	First definition
[BarPfi97]	✗	✓	✗	Strong RSA	O(1)	-
[CamLys02]	✓	✗	✗	Strong RSA	O(1)	First dynamic accumulator
[LLX07]	✓	✗	✓	Strong RSA	O(1)	First universal accumultor
[AWSM07]	✓	✗	✗	Pairings	O(1)	E-cash
[CHKO08]	✓	✓	✓	Collision-Resistant Hashing	O(ln(n))	Our work

Prior work

Dynamic Accumulators [CamLys02]

- Security Model



Scheme secure iff:

$\Pr[(w,x)\leftarrow A^{O}(): \text{Belongs}(w,x,Acc)=1 \text{ and } x \notin X] < \varepsilon(k)$

Prior work

Dynamic Accumulators [CamLys02]

- Initialization
  - $n = pq$, $u \in Z_n^*$
- Set
  - $X=\{x_1,x_2,\ldots,x_l\}$ (primes)
- Accumulated value
  - $Acc = u^{x_1 \cdot x_2 \cdots x_l} \bmod n$
- Witness for x_i
  - $w = u^{x_1 \cdots x_{i-1} \cdot x_{i+1} \cdots x_l} \bmod n$
- Membership test
  - $w^{x_i} \bmod n = Acc$

Prior work

Dynamic Accumulators [CamLys02]

- Adding elements
  - $Acc' := Acc^x \bmod n$
  - $w' := w^x \bmod n$

- To delete elements
  - Recompute the accumulated value with all the elements of the new set.
  - Doing the same for the witnesses (without the element we want to test).
  - $O(|X|)$ => not efficient.

- To delete elements efficiently
  - Manager knows $\Phi(n)$
    - We want to delete x:
      - $Acc = u^{x_1 \cdot x_2 \cdot \dots x \dots x_l} \bmod n$
      - Compute $y = x^{-1} \bmod \Phi(n)$
      - $Acc_{new} = Acc^{1/x} \bmod n = Acc^y \bmod n$
  - The manager **must be trusted** because he can compute fake witnesses for any x:
      - $w = Acc^{1/x} \bmod n$

Prior work

Dynamic Accumulators [CamLys02]

- **Theorem:** if the Strong RSA Assumption holds, the dynamic accumulator is secure.

Prior work

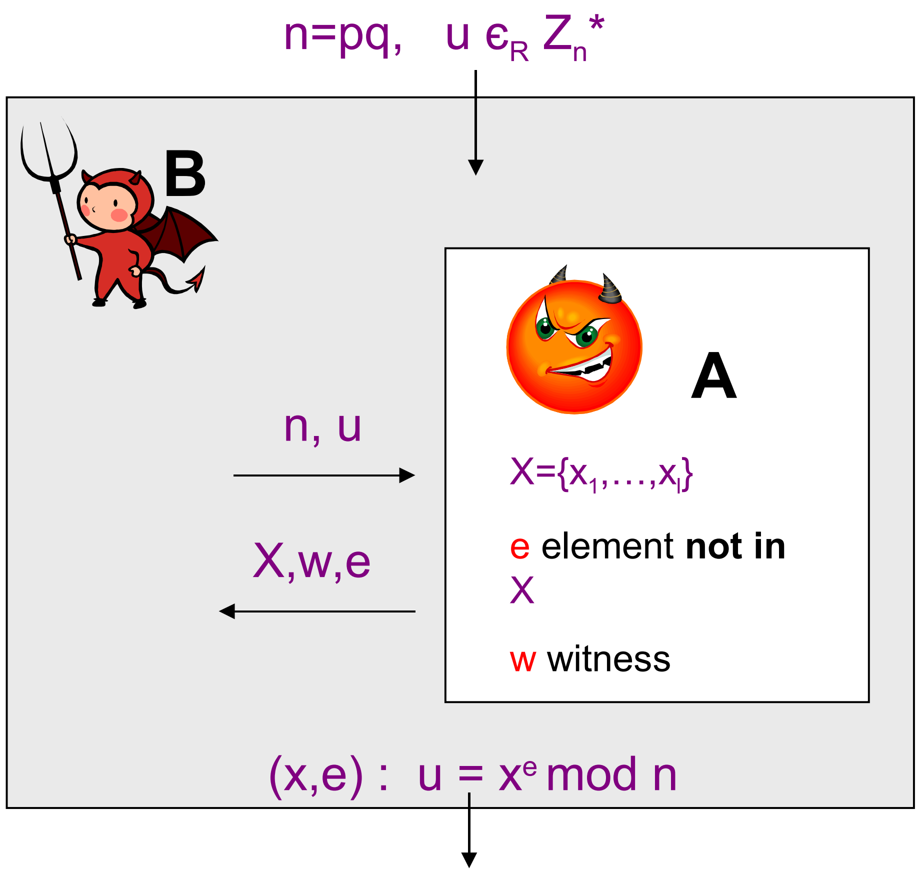
Dynamic Accumulators [CamLys02]

- **Lemma:** Let n be an integer, given $u,v \in Z_n^*$ and $a,b \in Z$ such that $u^a = v^b \bmod n$ and $\gcd(a,b) = 1$, we can compute efficiently $x \in Z_n^*$ such that $x^a=v \bmod n$.
- **Proof:**
  - $\gcd(a,b)=1 \Rightarrow bd = 1 + ac$
  - $x := u^d v^{-c} \Rightarrow x^a = u^{da} v^{-ca} = (u^a)^d v^{-ca} = v^{bd} v^{-ca} = v$

Prior work

Dynamic Accumulators [CamLys02]

- Proof of the theorem:



If there exists an adversary **A** that can break our scheme

We can build an adversary **B** that can break the Strong RSA Assumption

Prior work

Dynamic Accumulators [CamLys02]

- Proof of the theorem:
  - $X = \{x_1,\ldots,x_l\}$,
  - $Acc = u^{x_1 \cdots x_l} \bmod n = u^v \bmod n$
  - e does not belong to X
  - $w^e \bmod n = Acc = u^v \bmod n$
  - $gcd(v,e) = 1$ and $w^e = u^v \bmod n$

    => by the lemma we can conclude

  (we can find easily x s.t. $x^e = u \bmod n$)

Our Construction

Notation

- $H: \{0,1\}^* \rightarrow \{0,1\}^k$
  - Function randomly chosen from a *family of collision-resistant hash functions*.

- $x_1, x_2, x_3, \ldots \in \{0,1\}^k$
  - $x_1 < x_2 < x_3 < \ldots$ where $<$ is the lexicographic order on binary strings.

- $-\infty, \infty$
  - Special values such that
    - For all $x \in \{0,1\}^k$: $-\infty < x < \infty$

- $||$ denotes the concatenation operator.

Our Construction

Public Data Structure

- Manager owns a public data structure called “Memory”.
- Compute efficiently the accumulated value and the witnesses.
- In our construction the Memory M will be a binary tree.

Our Construction

Accumulator Operations

Operation	Who runs it?
$Acc_0, M_0 \leftarrow$ Setup(1^k)	Manager
$w \leftarrow$ Witness(M,x)	Manager
True,False,$\perp \leftarrow$ Belongs(x,w,Acc)	User
$Acc_{after}, M_{after}, w_{up} \leftarrow$ $Update_{add/del}$(M_{before},x)	Manager
OK, $\perp \leftarrow$ CheckUpdate(Acc_{before},Acc_{after},w_{up})	User

Our Construction

Checking for (non)membership

User

Accumulator Manager

Does x belong to X?

w = Witness(M,x)

w

Belongs(x,w,Acc) = True $\Leftrightarrow x \in X$

Our Construction

Update of the accumulated value

User

Accumulator Manager

Insert or Delete x →

$Acc_{after}, M_{after}, w_{up} = Update_{Add/Del}(M_{before}, x)$

← Acc_{after}, w_{up}

$CheckUpdate(Acc_{before}, Acc_{after}, w_{up})$

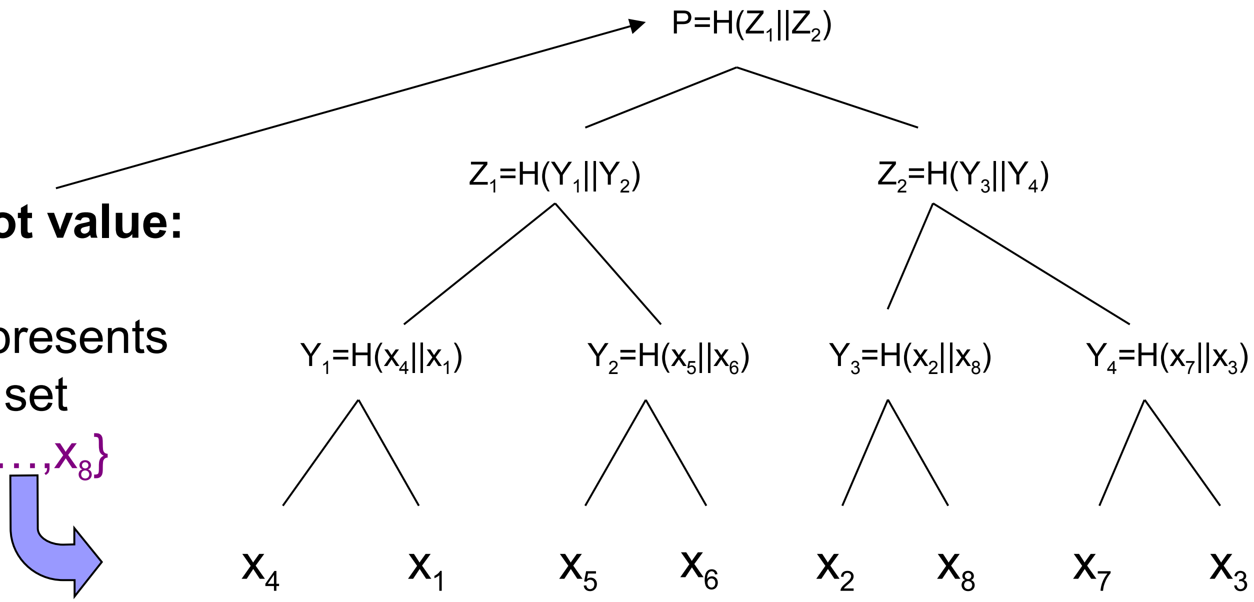
Our Construction

Ideas

- Merkle-trees

Root value:

Represents the set $\{x_1,...,x_8\}$



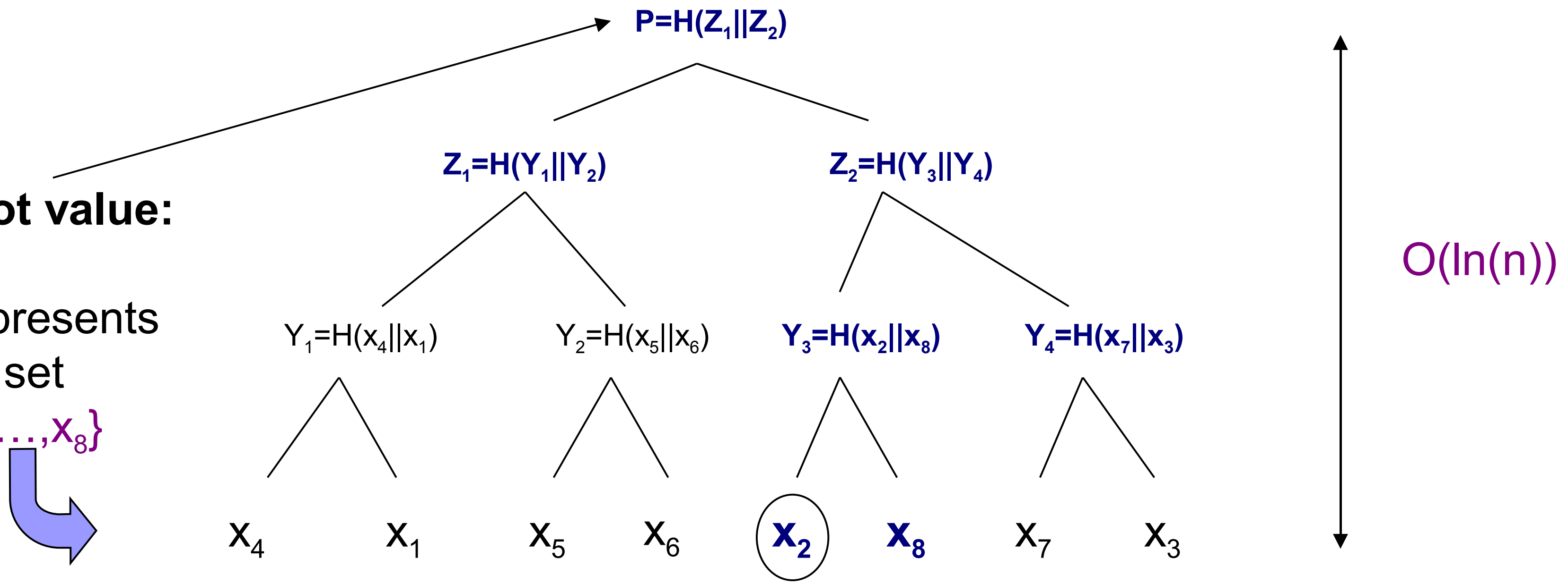
Our Construction

Ideas

- Merkle-trees

Root value:

Represents the set $\{x_1,\ldots,x_8\}$



Our Construction

Ideas

- How to prove nonmembership?
  - Kocher's trick [Koch98]: store pair of consecutive values
    - X={1,3,5,6,11}
    - X'={(-∞,1),(1,3),(3,5),(5,6),(6,11),(11, ∞)}
    - y=3 belongs to X ⇔ (1,3) or (3,5) belongs to X'.
    - y=2 does not belong to X ⇔ (1,3) belongs to X'.

Our Construction

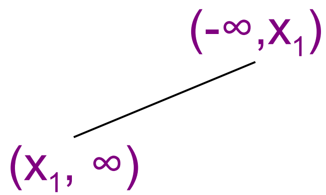
How to insert elements?

$(-\infty,\infty)$

$X=\emptyset$, next: x_1

Our Construction

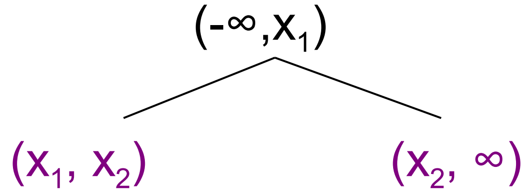
How to insert elements?



X={x_1}, next: x_2

Our Construction

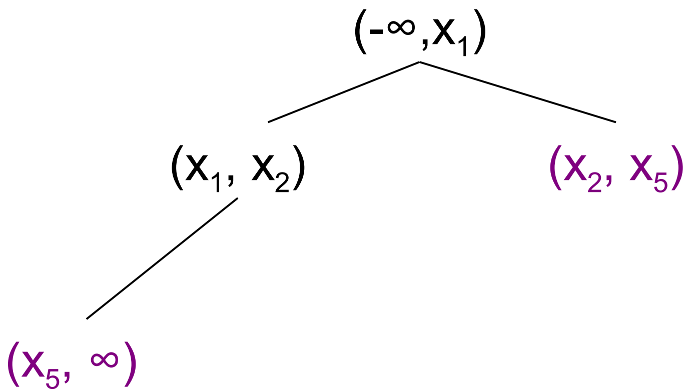
How to insert elements?



$X=\{x_1,x_2\}$, next: x_5

Our Construction

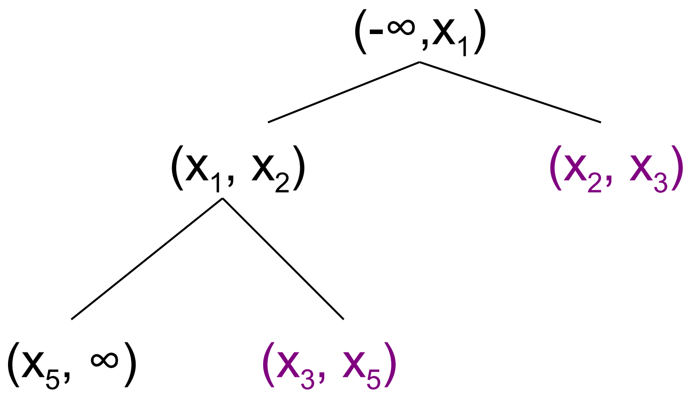
How to insert elements?



$X=\{x_1,x_2,x_5\}$, next: x_3

Our Construction

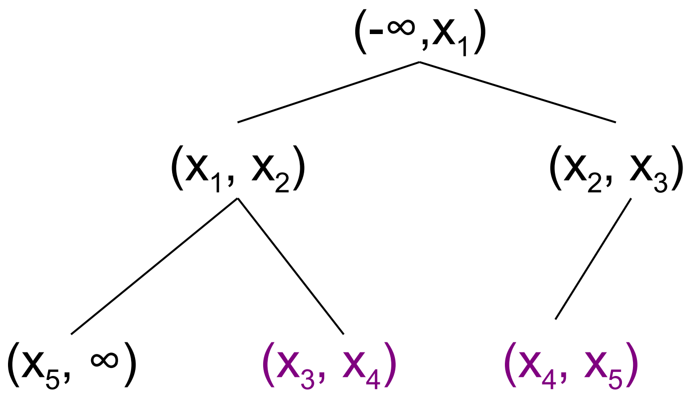
How to insert elements?



$X=\{x_1,x_2,x_3,x_5\}$, next: x_4

Our Construction

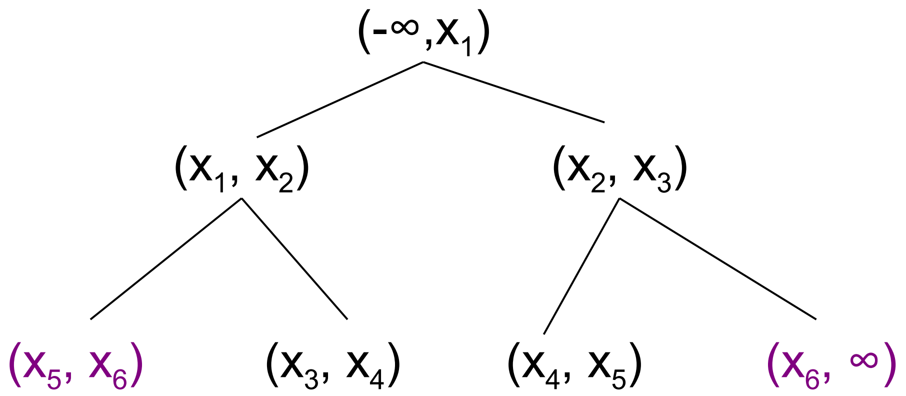
How to insert elements?



$X=\{x_1,x_2,x_3,x_4,x_5\}$, next: x_6

Our Construction

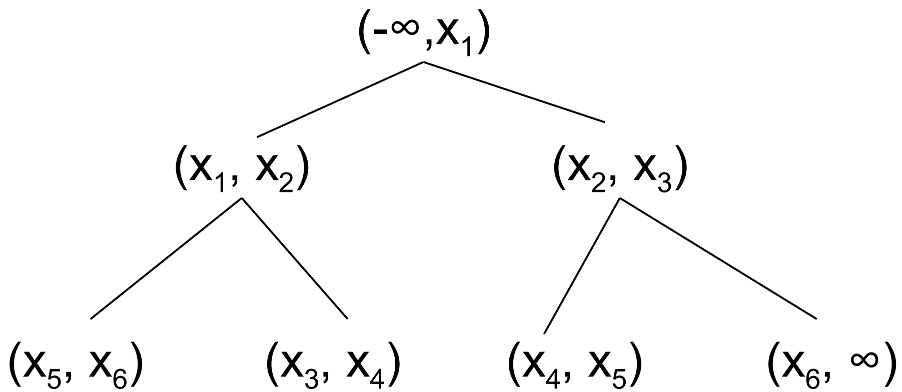
How to insert elements?



$X=\{x_1,x_2,x_3,x_4,x_5,x_6\}$

Our Construction

How to delete elements?

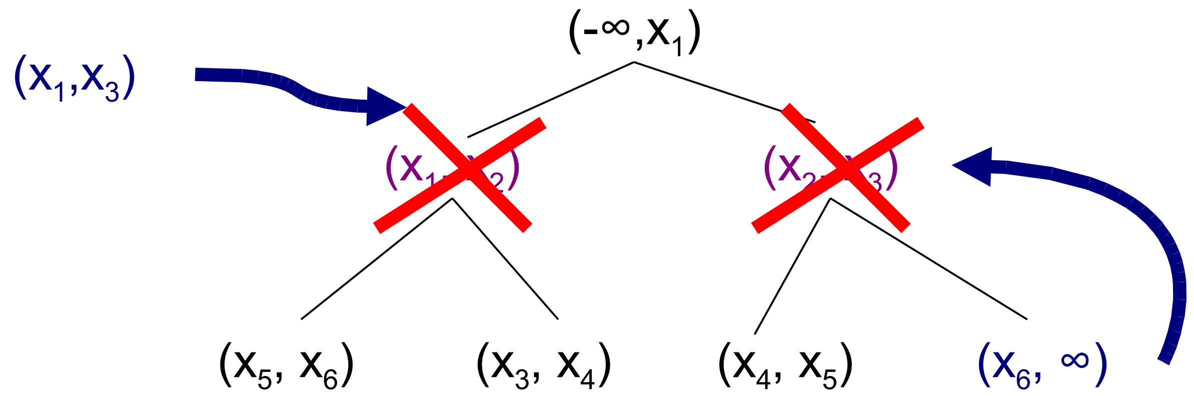


$X=\{x_1,x_2,x_3,x_4,x_5,x_6\}$

element to be deleted: x_2

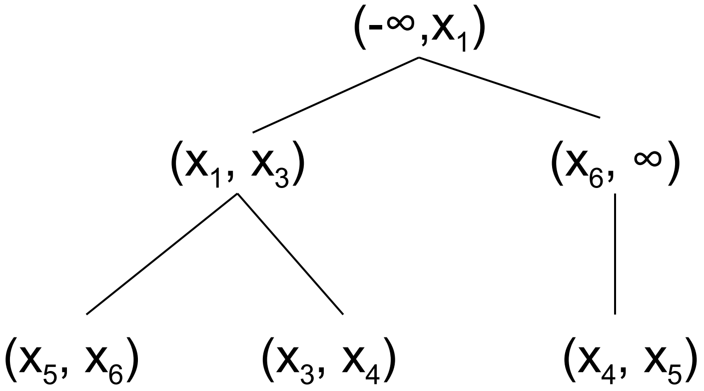
Our Construction

How to delete elements?



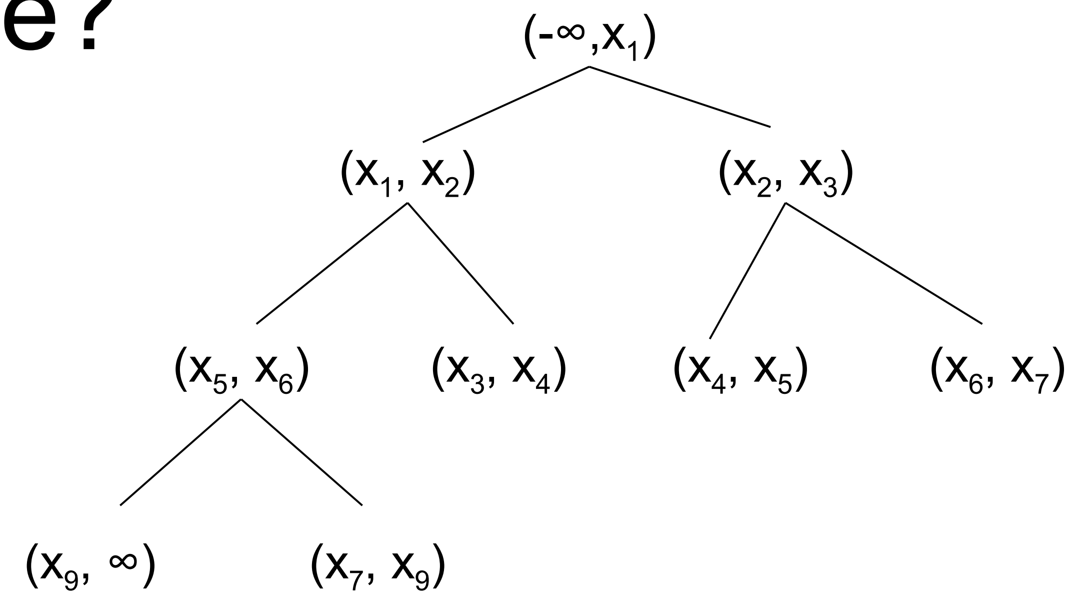
Our Construction

How to delete elements?



Our Construction

How to compute the accumulated value?



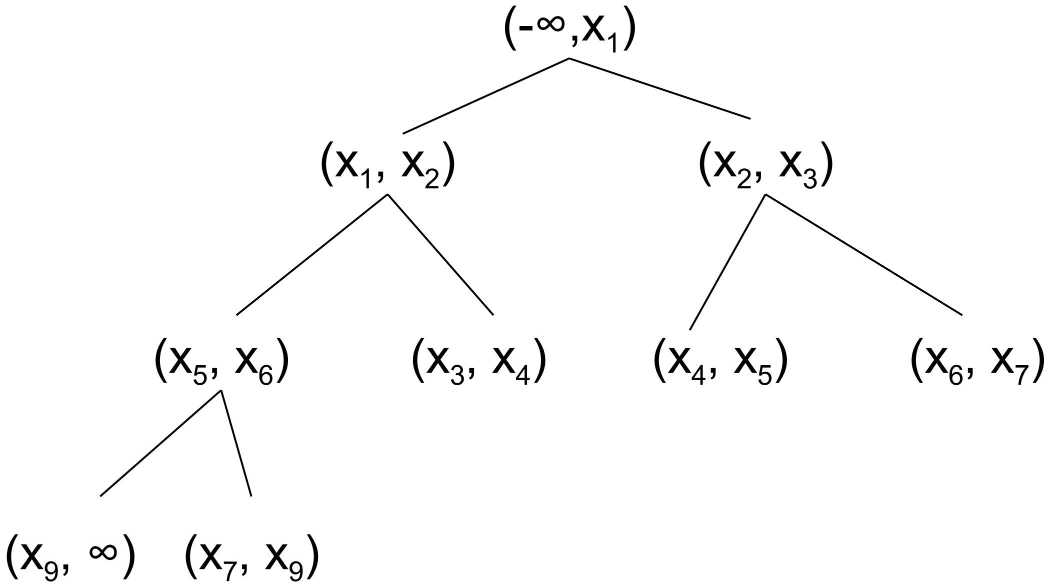
A pair (x_i,x_j)

$Proof_N=H(Proof_{left}||Proof_{right}||value)$

$Proof_{Nil}=$ “”

$Acc = Proof_{Root}$

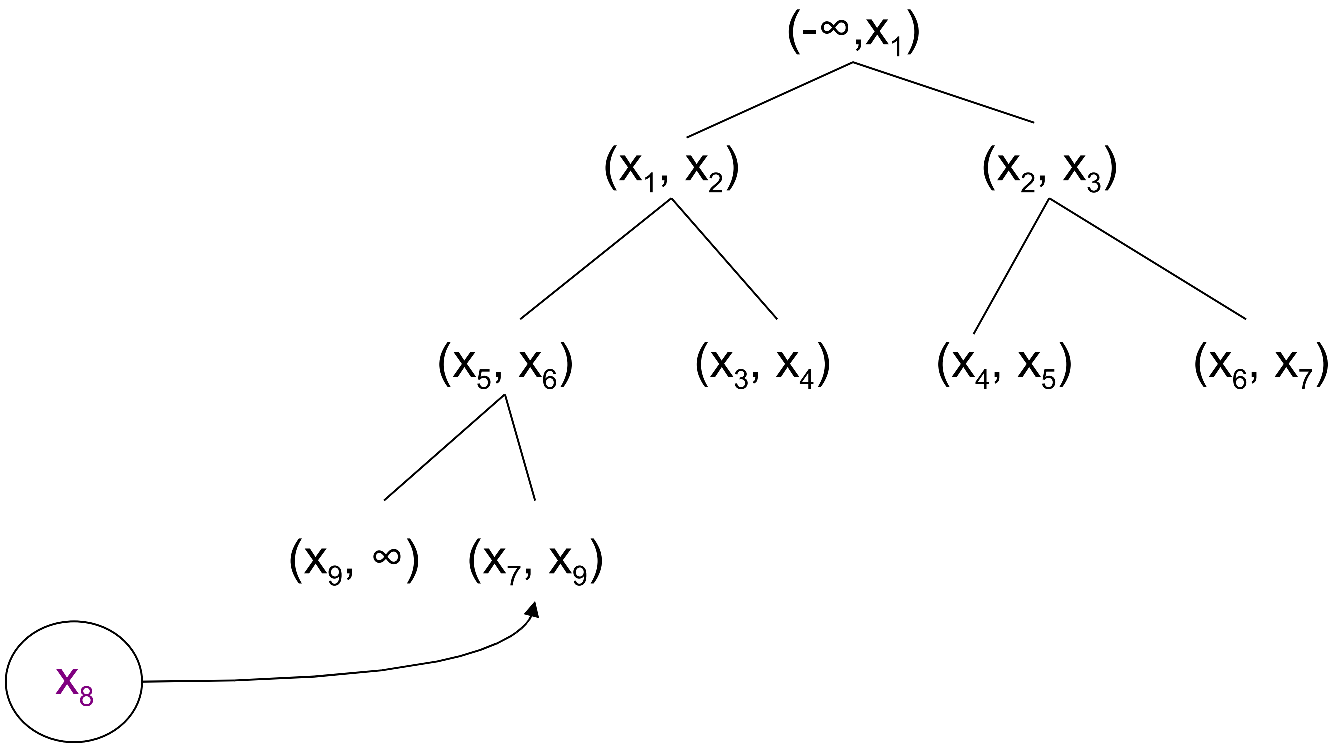
How to update the accumulated value? (Insertion)



x_8 to be inserted.

Our Construction

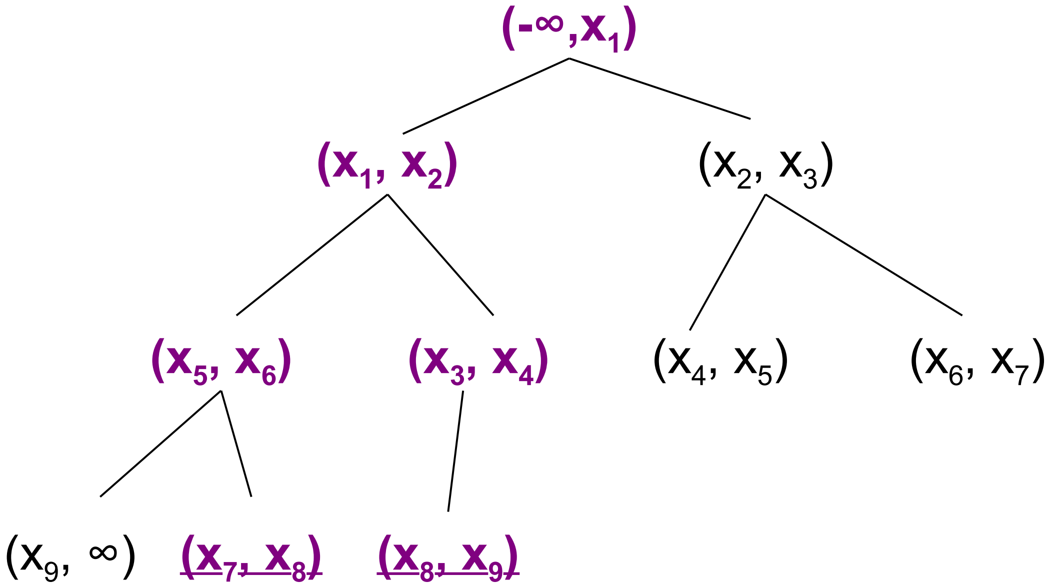
How to update the accumulated value? (Insertion)



We will need to recompute proof node values.

Our Construction

How to update the accumulated value? (Insertion)



New element: x_8.

Proof_N stored in each node.

Dark nodes do not require recomputing Proof_N.

Only a logarithmic number of values need recomputation.

Our Construction

Security

- **Definition:** an accumulated value Acc represents the set $X=\{x_1,x_2,\ldots,x_n\}$, if it has been computed from a tree T containing node values $\{(-\infty,x_1),(x_1,x_2),\ldots,(x_n,\infty)\}$, where each pair appears only once.

Our Construction

Security

- **Definition:** (Consistency)
  - Given Acc that represents X, it is hard to find witnesses that allow to prove inconsistent statements.
    - X={1,2}.
    - Hard to compute a *membership* witness for 3.
    - Hard to compute a *nonmembership* witness for 2.

Our Construction

Security

- **Definition:** (Update)
  - Guarantees that the accumulated value Acc represents the set X after insertion/deletion of x.
  - Every update must be checked by users but it is not needed to store the sequence of insertion/deletion.

Our Construction

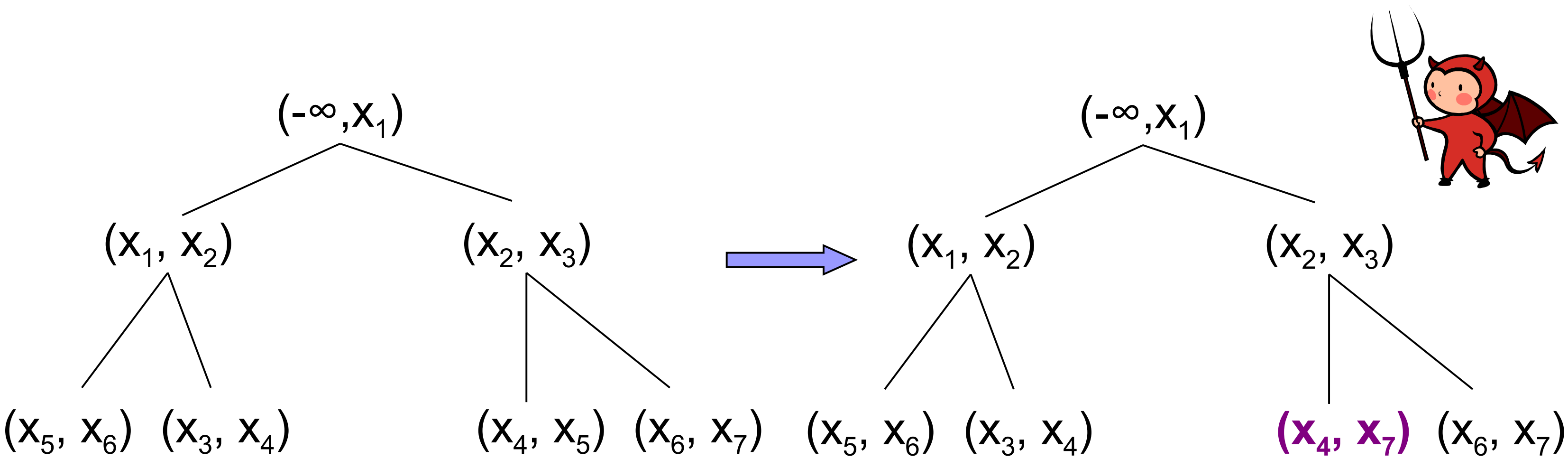
Security

- **Theorem:** if CRHF exist the accumulator is secure (i.e. satisfies consistency and update).

Our Construction

Security

- **Lemma:** Given a tree T with accumulated value Proof_T, finding a tree T', T≠T' such that $\text{Proof}_T = \text{Proof}_{T'}$ is difficult.
- *Proof (Sketch):* $\text{Proof}_N = H(\text{Proof}_{left} || \text{Proof}_{right} || \text{value})$

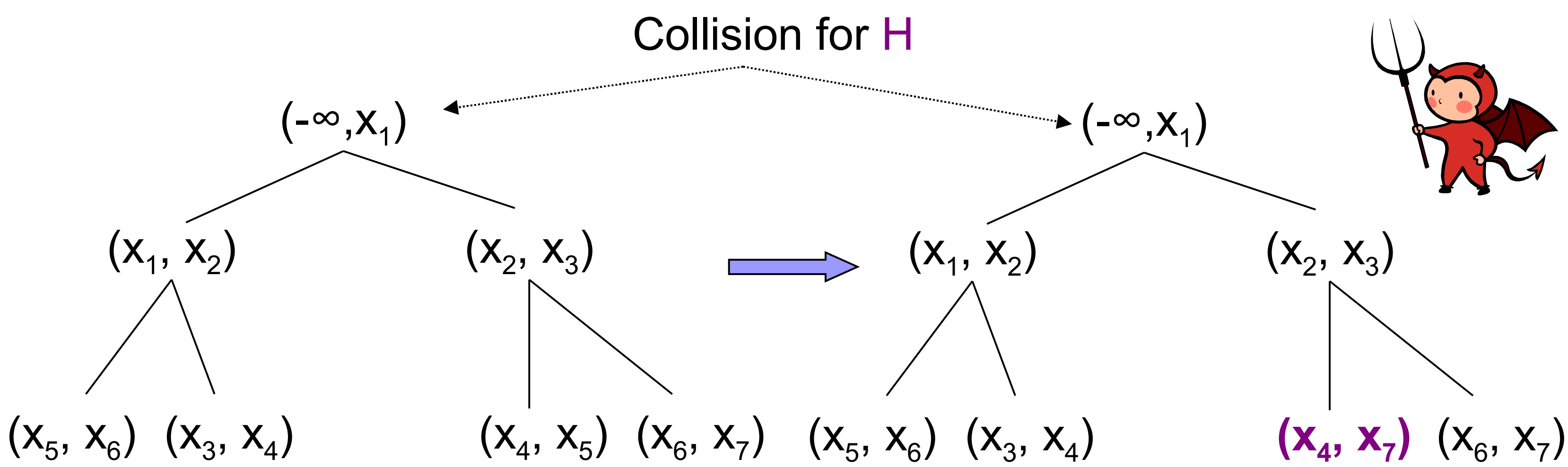


Our Construction

Security

Lemma: Given a tree T with accumulated value $Proof_T$, finding a tree T', T≠T' such that $Proof_T = Proof_{T'}$ is difficult.

- *Proof (Sketch):* $Proof_N = H(Proof_{left}||Proof_{right}||value)$

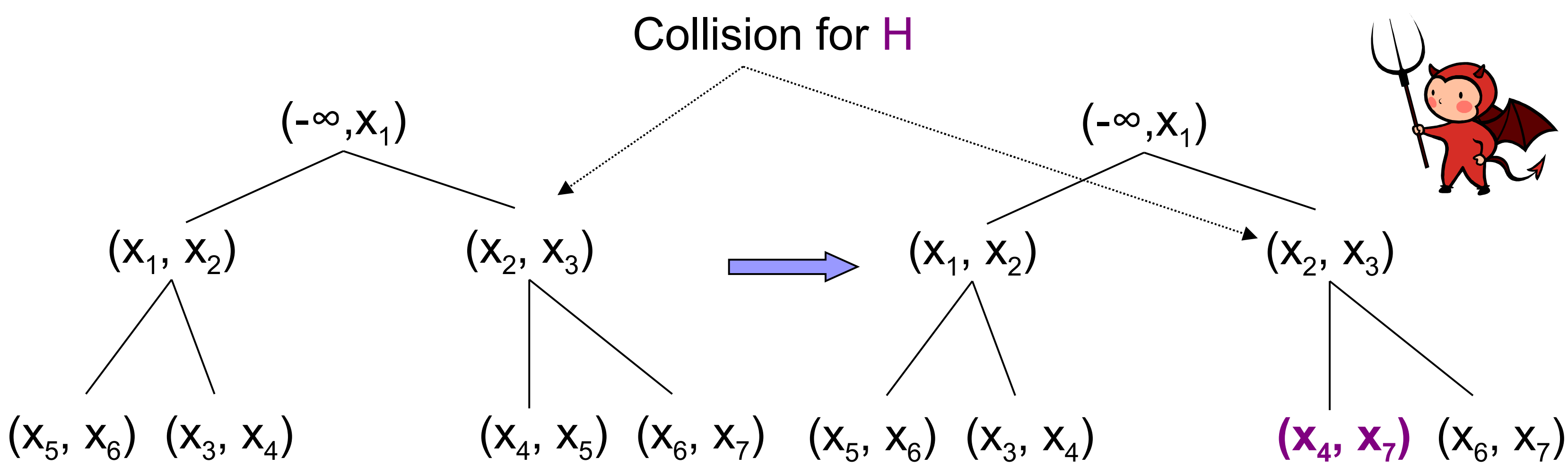


Our Construction

Security

Lemma: Given a tree T with accumulated value $Proof_T$, finding a tree T', $T \neq T'$ such that $Proof_T = Proof_{T'}$ is difficult.

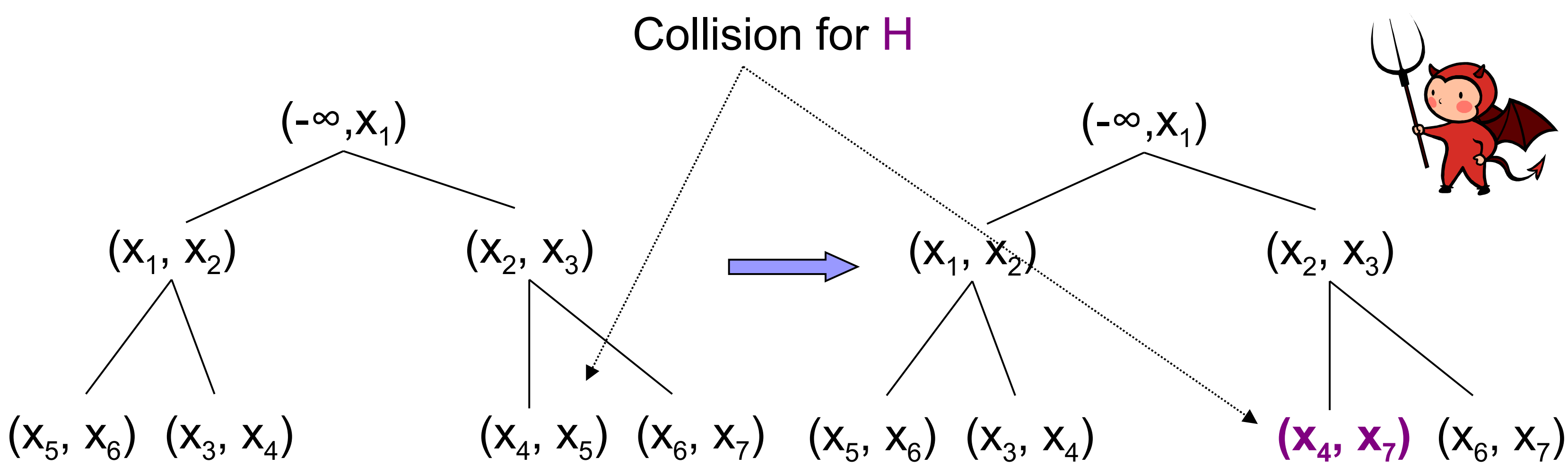
- *Proof (Sketch):* $Proof_N = H(Proof_{left} || Proof_{right} || value)$



Security

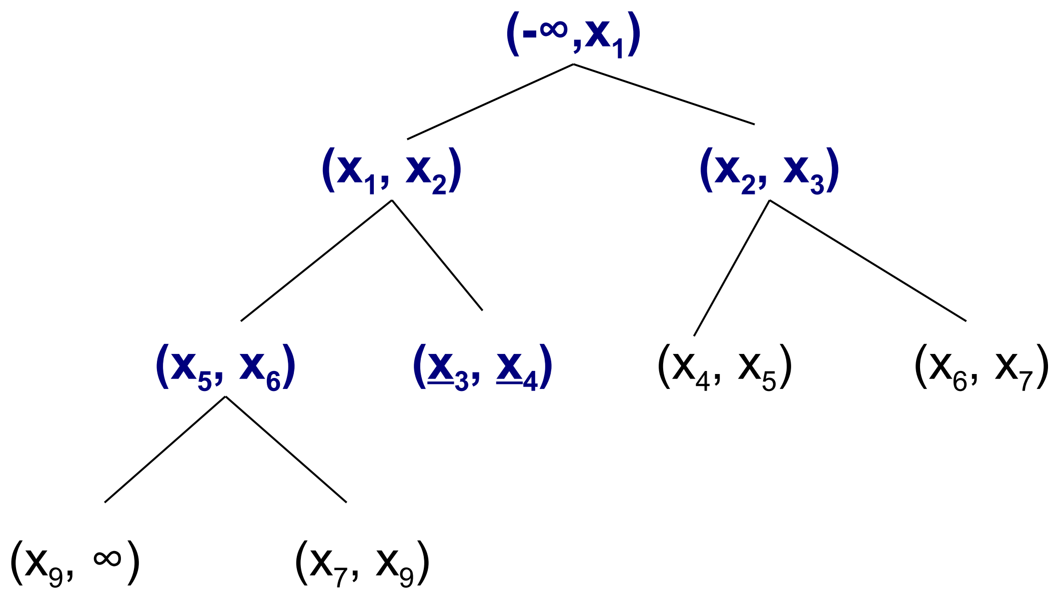
Lemma: Given a tree T with accumulated value $Proof_T$, finding a tree T', T≠T' such that $Proof_T = Proof_{T'}$ is difficult.

- *Proof (Sketch):* $Proof_N = H(Proof_{left}||Proof_{right}||value)$



Our Construction

Security (Consistency)



Witness: blue nodes and the **(x_3,x_4)** pair, size in O(ln(n))

Checking that x belongs (or not) to X:

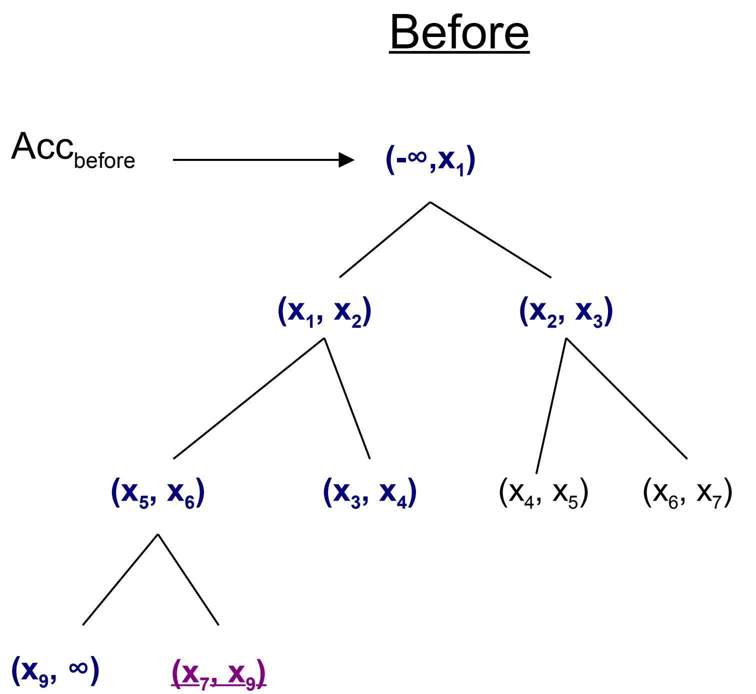
1) compute recursively the proof P and verify that P=Acc

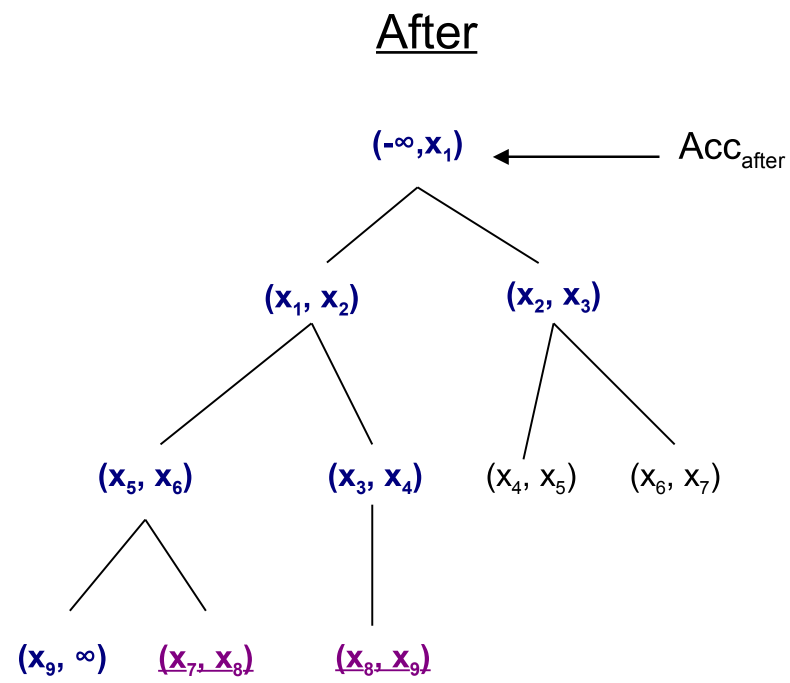
2) check that: $x=x_3$ or $x=x_4$ (membership)

$x_3 < x < x_4$ (nonmembership)

Our Construction

Security (Update)





Insertion of x_8

Conclusion & Open Problem

- First *dynamic, universal, strong* accumulator
- Simple
- Security
  - Existence of CRHF
- Solves the e-Invoice Factoring Problem
- Less efficient than other constructions
  - Size of witness in O(ln(n))
- Open Problem
  - "Is it possible to build an efficient *strong*,*dynamic* and *universal* accumulator with witness size lower than O(ln(n))?"

Thank you!

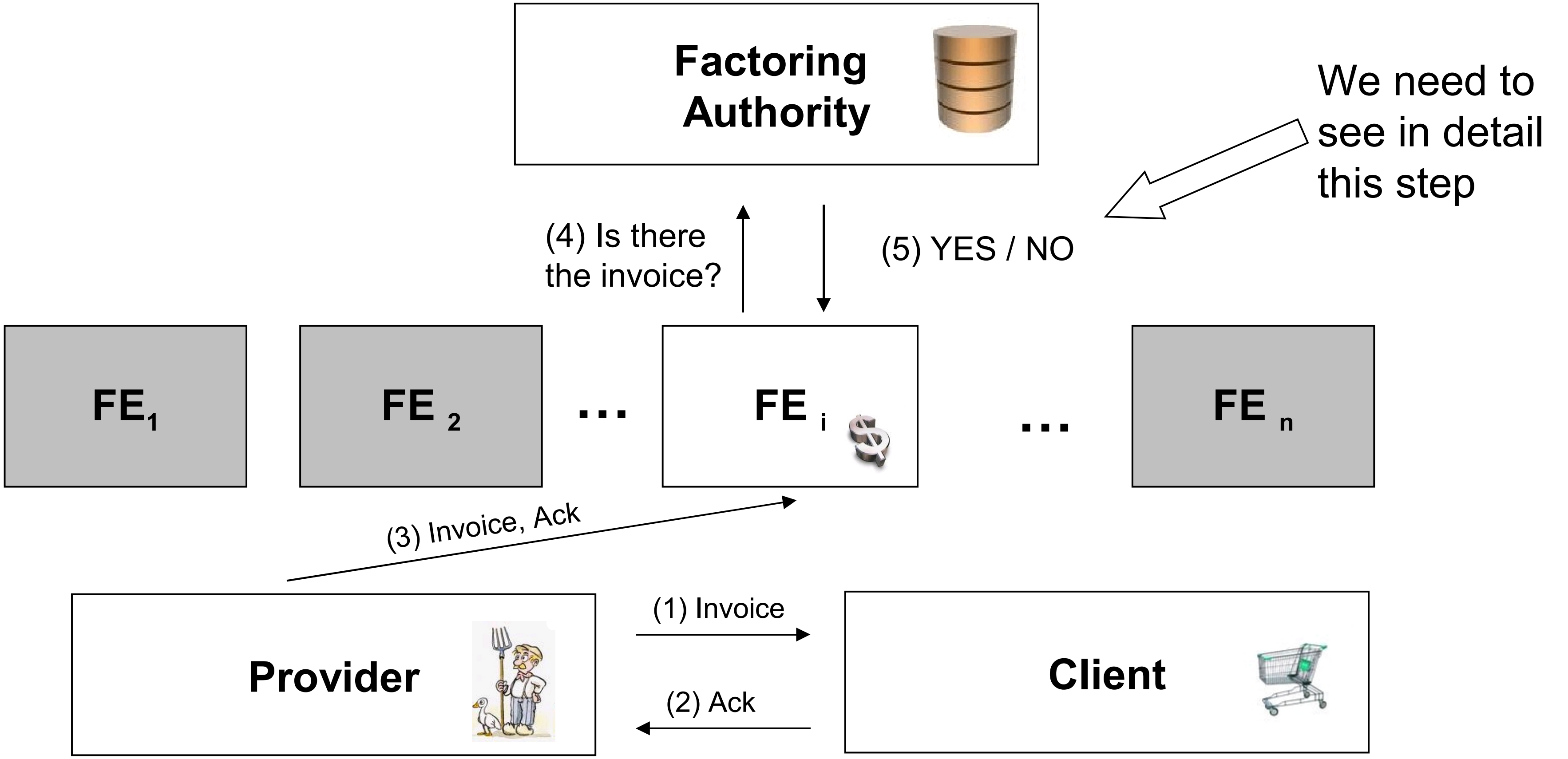
Distributed solutions?

- Complex to implement
- Hard to make them robust
- High bandwith communication
- Need to be online – synchronization problems
- **That’s why we focus on a centralized solution.**

Invoice Factoring using accumulator

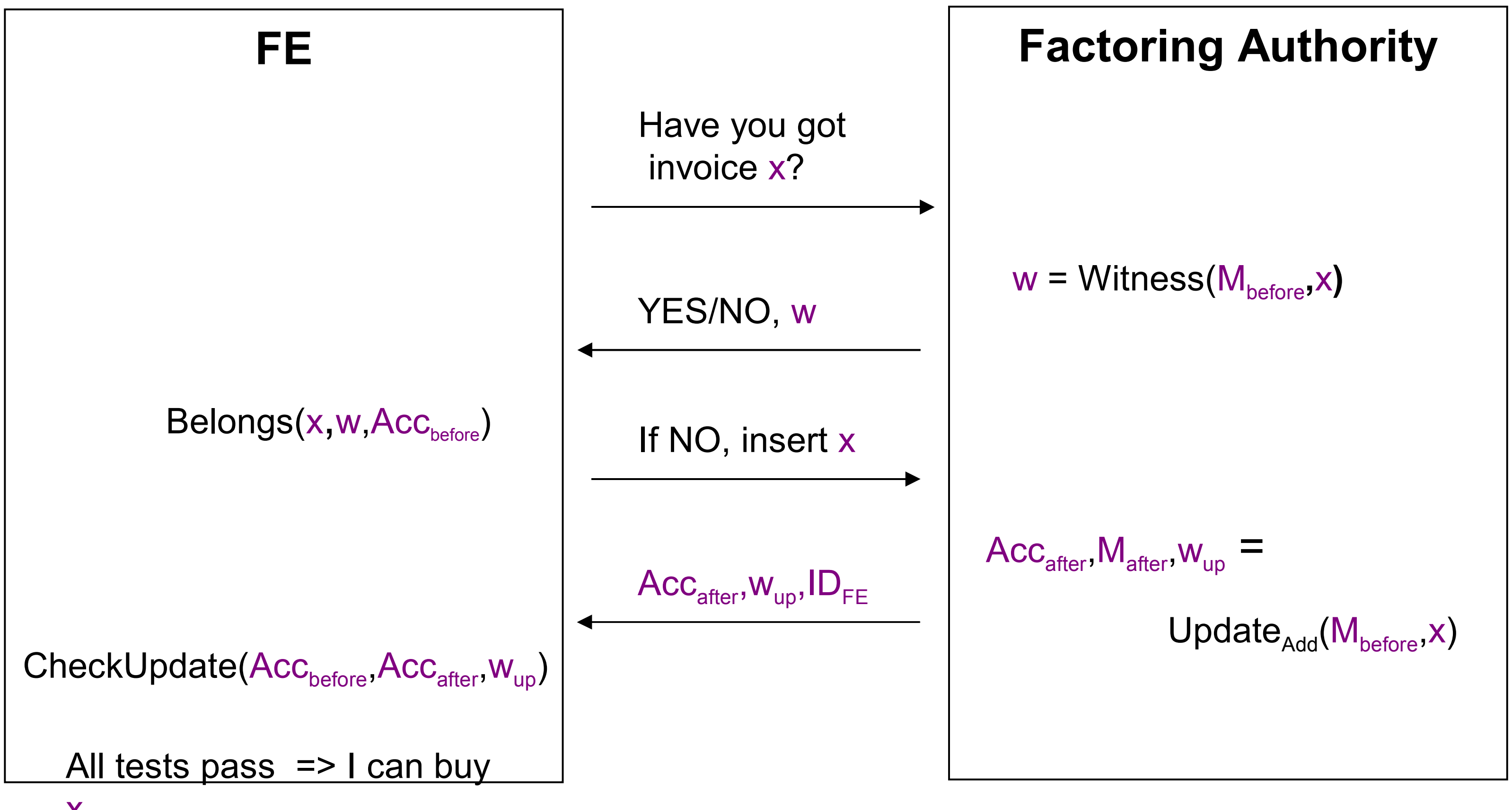
- We need a secure broadcast channel
  - If a message m is published, every participant sees the same m.
- Depending on the security level required
  - Trusted http of ftp server
  - Bulletin Board [CGS97]

Invoice Factoring using accumulator



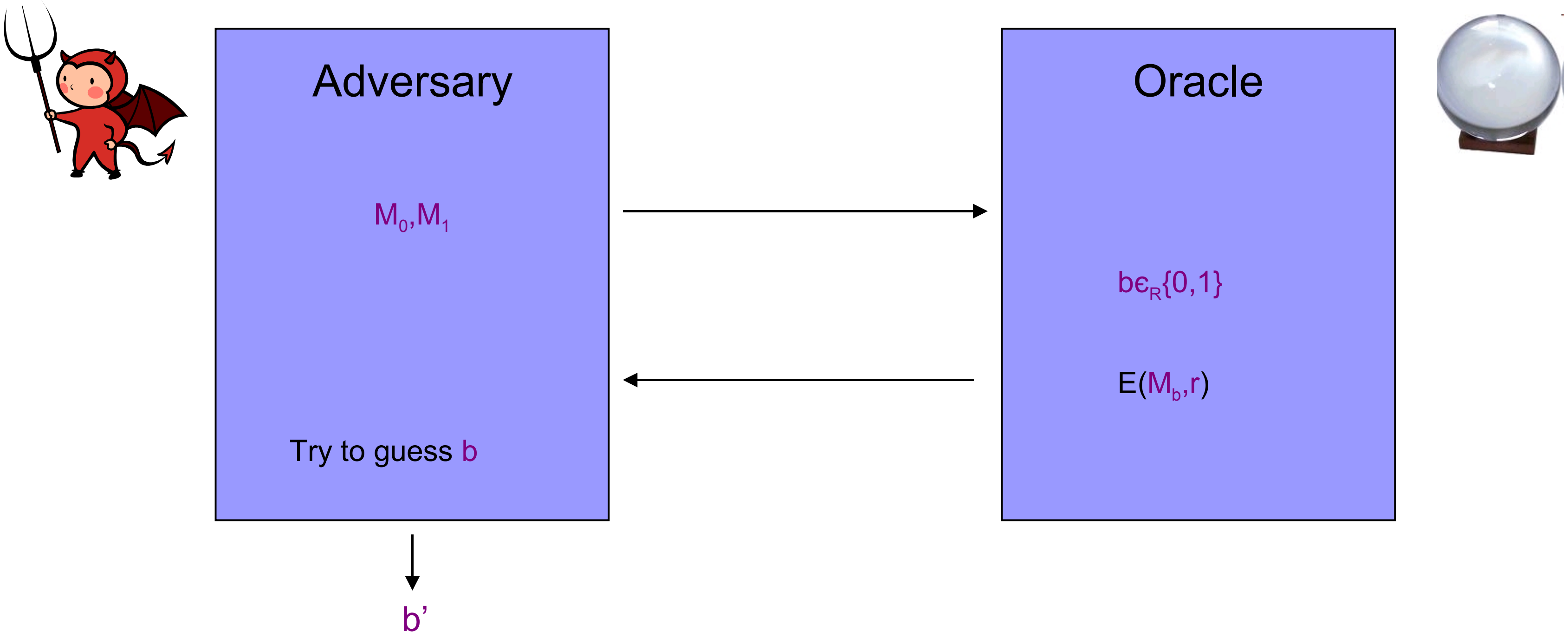
Invoice Factoring using accumulator

- Step 5 (Details)



Basic Cryptographic Notions

- Secure encryption [GM84]



Adversary wins if **Pr[b=b'] > ½ + 1/q(n)**

Bibliography

- **[GM84]** Probabilistic Encryption *Shafi Goldwasser and Silvio Micali* 1984
- **[BeMa92]** Efficient Broadcast Time-Stamping *Josh Benaloh and Michael de Mare* 1992
- **[BeMa94]** One-way Accumulators: A decentralized Alternative to Digital Signatures *Josh Benaloh and Michael de Mare ,* 1994
- [**BarPfi97]** Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees *Niko Barić and Birgit Pfitzmann* 1997
- **[CGS97]** A secure and optimally efficient multi-authority election scheme *R. Cramer, R. Gennaro, and B. Schoenmakers* 1997
- **[Koch98]** On certificate revocation and validation *P.C. Kocher* 1998
- **[CGH98]** The random oracle methodoly revisited R. Canetti, O. Goldreich and S. Halevi 1998
- **[Sand99]** Efficient Accumulators Without Trapdoor *Tomas Sanders* 1999
- **[GoTa01]** An efficient and Distributed Cryptographic Accumulator *Michael T. Goodrich and Roberto Tamassia* 2001
- **[CamLys02]** Dynamic Accumulators And Application to Efficient Revocation of Anonymous Credentials *Jan Camenisch Anna Lysyanskaya* 2002
- **[GeRa04]** RSA Accumulator Based Broadcast Encryption *Craig Gentry and Zulfikar Ramzan 2004*

Bibliography

- **[LLX07]** Universal Accumulators with Efficient Nonmembership Proofs *Jiangtao Li, Ninghui Li and Rui Xue* 2007
- **[AWSM07]** Compact E-Cash from Bounded Accumulator *Man Ho Au, Qianhong Wu, Willy Susilo and Yi Mu* 2007
- **[CKHO08]** Strong Accumulators from Collision-Resistant Hashing *Philippe Camacho, Alejandro Hevia, Marcos Kiwi, and Roberto Opazo* 2008