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
- Conclusion

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

- 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 the basis of what's called *"Provably Security"* or *"Reductionist Security"*.

Basic Cryptographic Notions

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

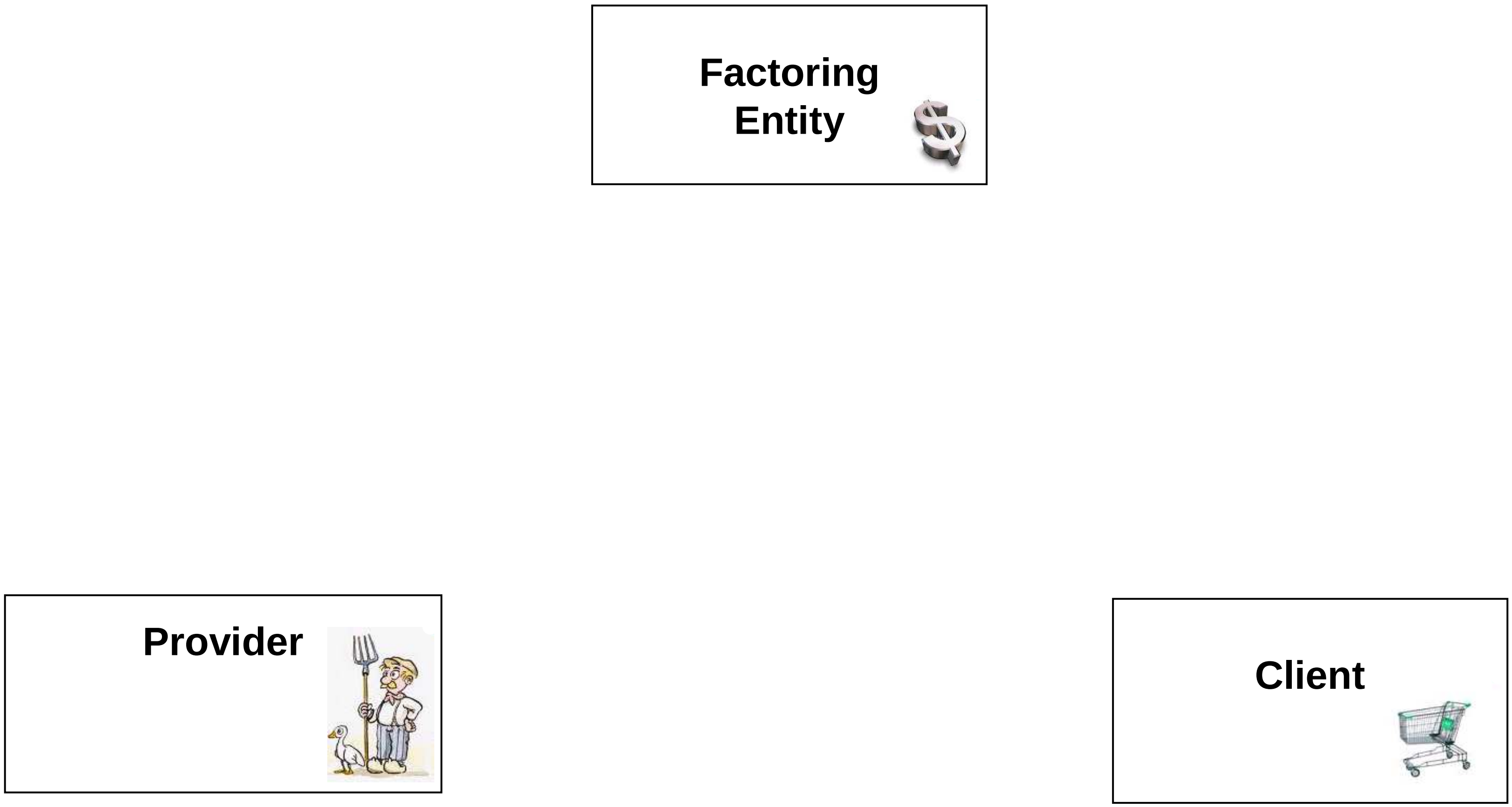
This definition is not formal. Just an intuition.

Basic Cryptographic Notions

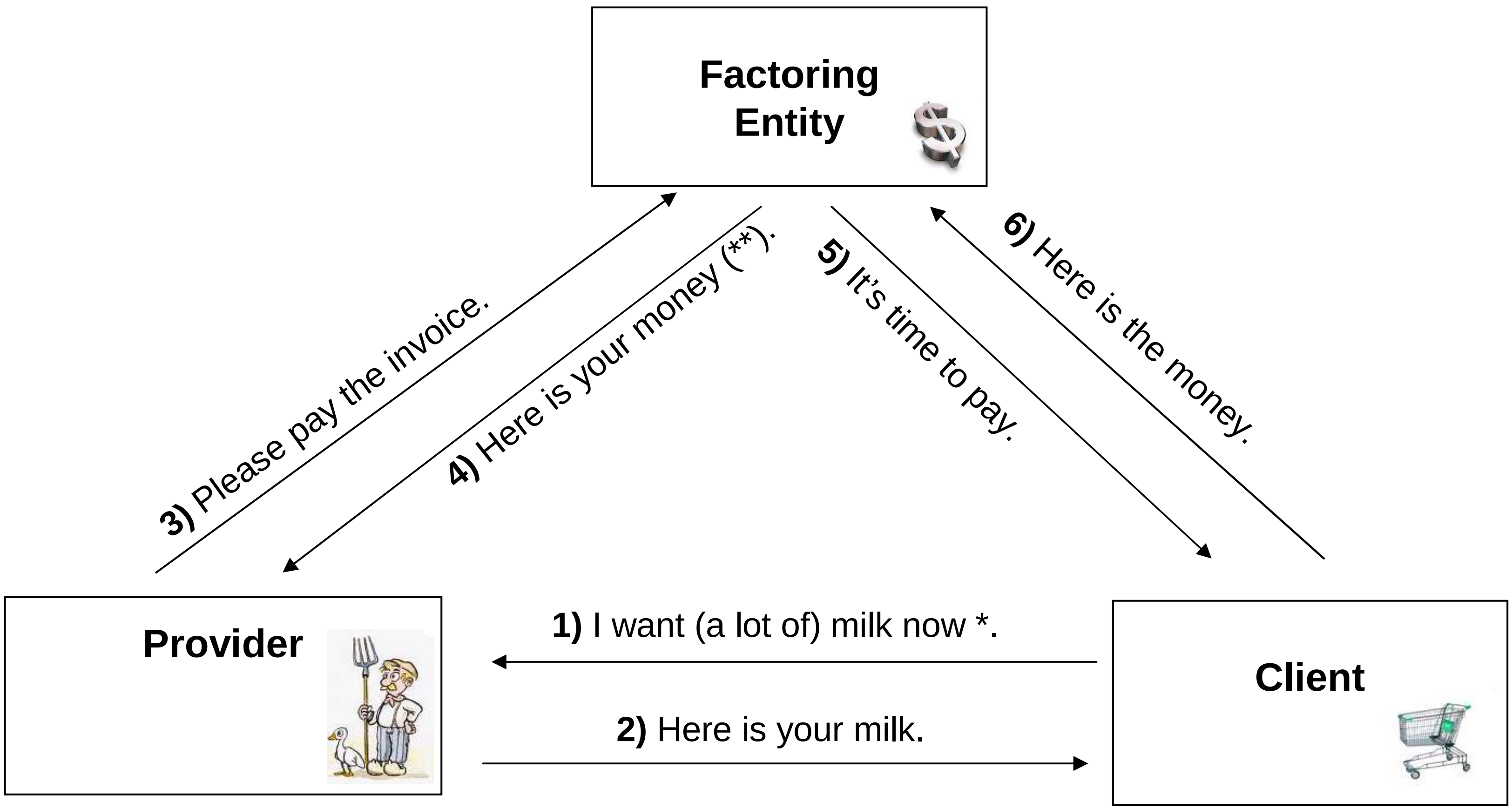
- **Assumption:**
  Collision-Resistant Hash Functions exist.

Not related to Number Theory!

Factoring Industry in Chile



Factoring Industry in Chile



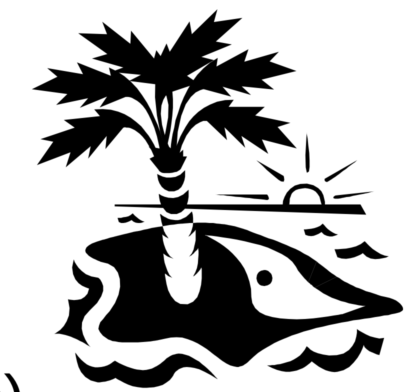
(*) but I do not want to pay yet.
(**) minus a fee.

The Problem

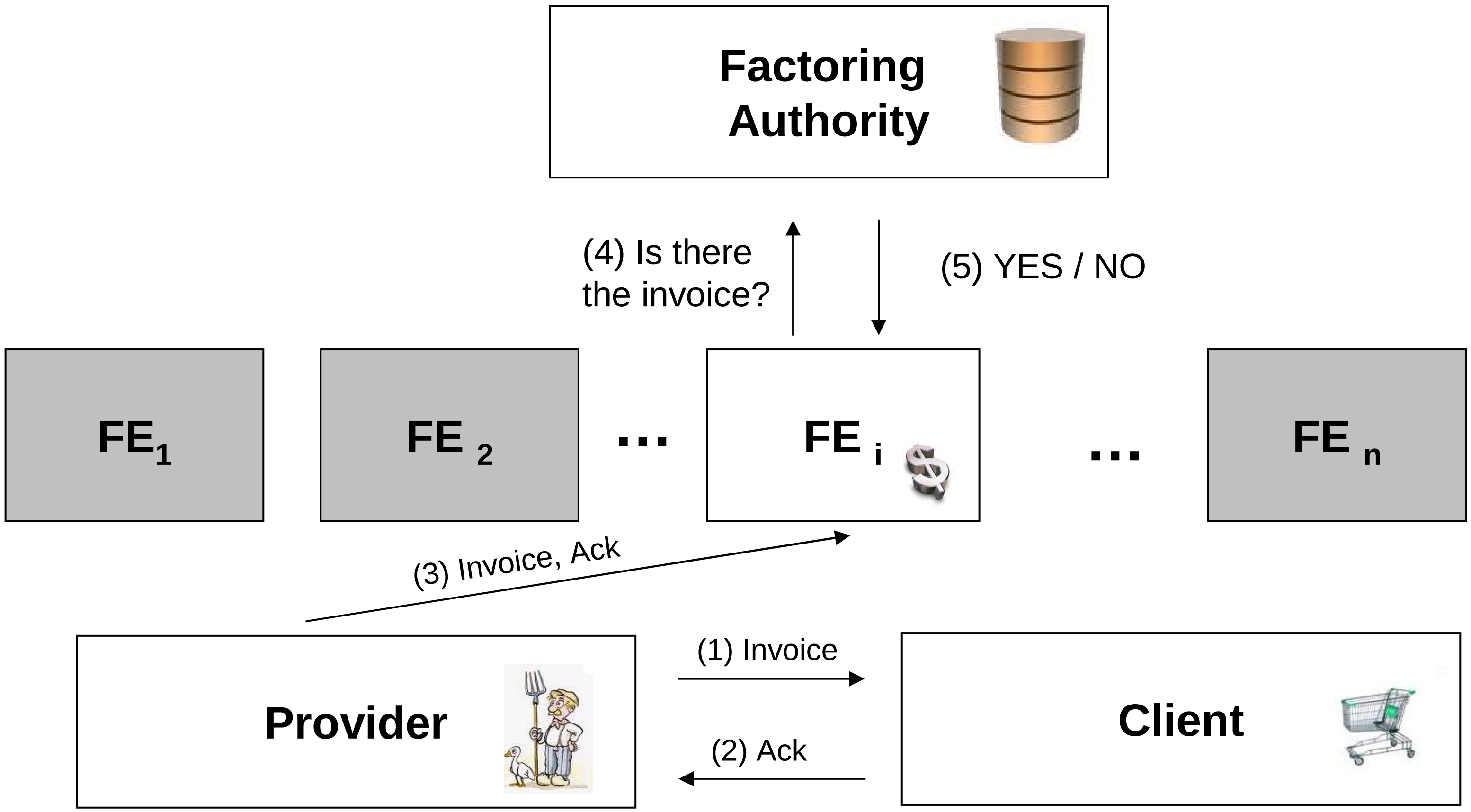
- A malicious provider could send the same invoice to various Factoring Entities.

- Then he leaves to a far away country with all the money (say, *southern France*)

- Later, several Factoring Entities will try to charge the invoice to the same client. Losts must be shared… (*do not count on government bailout though* ☺)



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.

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
[CHKO08]	✓	✓	✓	Collision-Resistant Hashing	O(ln(n))	Our work

Notation

- $H: \{0,1\}^* \rightarrow \{0,1\}^k$
  - Collision-resistant hash function.
- $x_1, x_2, x_3, \ldots \in \{0,1\}^k$
  - $x_1 < x_2 < x_3 < \ldots$ 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.

Public Data Structure

- Called “Memory”.
- Compute efficiently the accumulated value and the witnesses.
- In our construction the Memory M will be a binary tree.

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

Checking for (non-)membership

User

Accumulator Manager

Does x belong to X?

$\longrightarrow$

w = Witness(M,x)

$\xleftarrow{\quad w \quad}$

Belongs(x,w,Acc) = True ⇔ x є X

Update of the accumulated value

User

Accumulator Manager

Insert or Delete x →

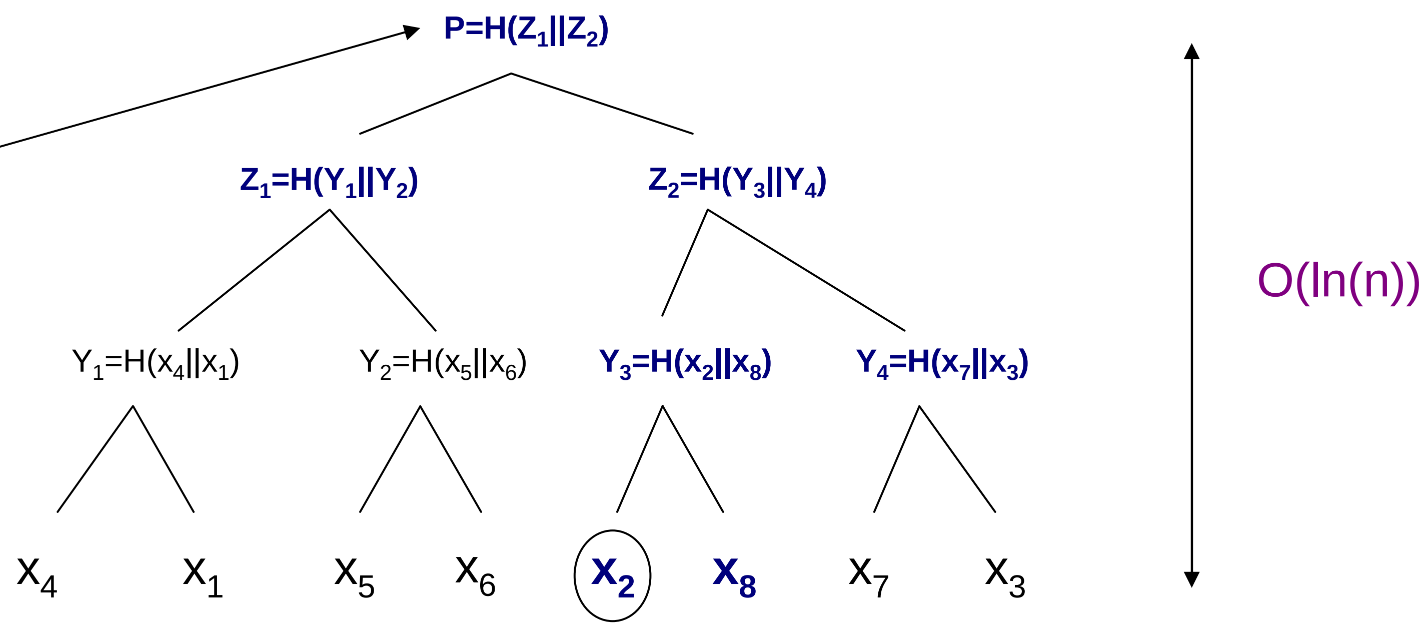
$Acc_{after}, M_{after}, w_{up} = Update_{Add/Del}(M_{before}, x)$

← Acc_{after}, w_{up}

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

Ideas

- Merkle-trees



Root value: Represents the set $\{x_1,\ldots,x_8\}$

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 (-∞,1) belongs to X'.
    - y=2 <u>does not</u> belong to X ⇔ (1,3) belongs to X'.

How to insert elements?

$(-\infty, \infty)$

$X = \emptyset$, next: x_1

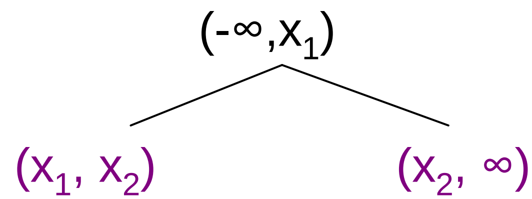
How to insert elements?

$(-\infty, x_1)$

(x_1, ∞)

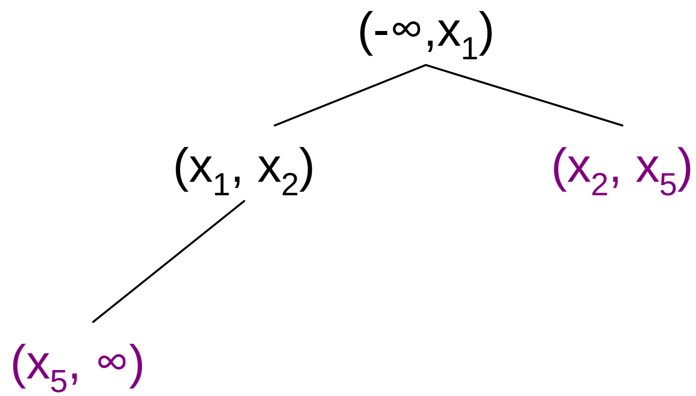
$X=\{x_1\}$, next: x_2

How to insert elements?



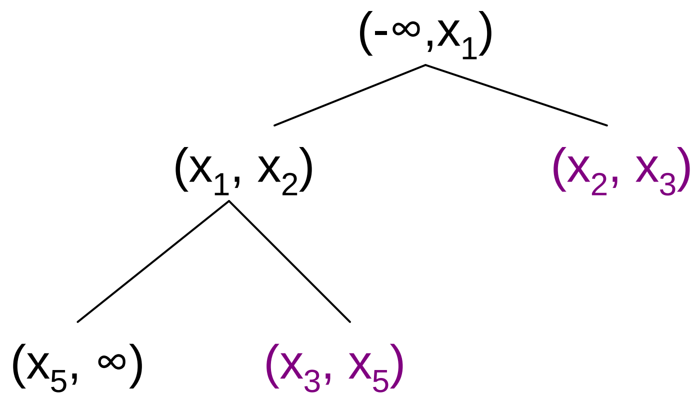
$X=\{x_1,x_2\}$, next: x_5

How to insert elements?



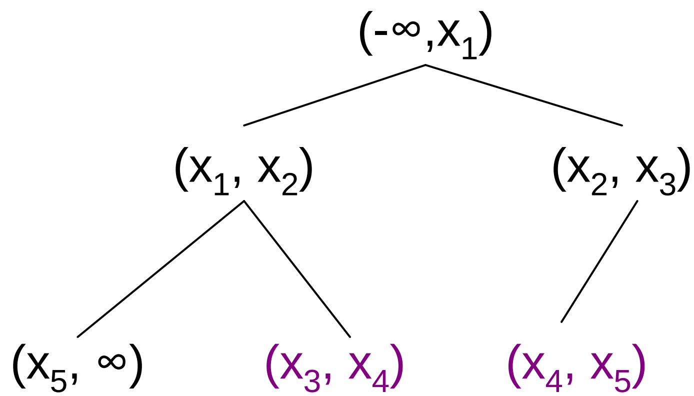
$X=\{x_1, x_2, x_5\}$, next: x_3

How to insert elements?



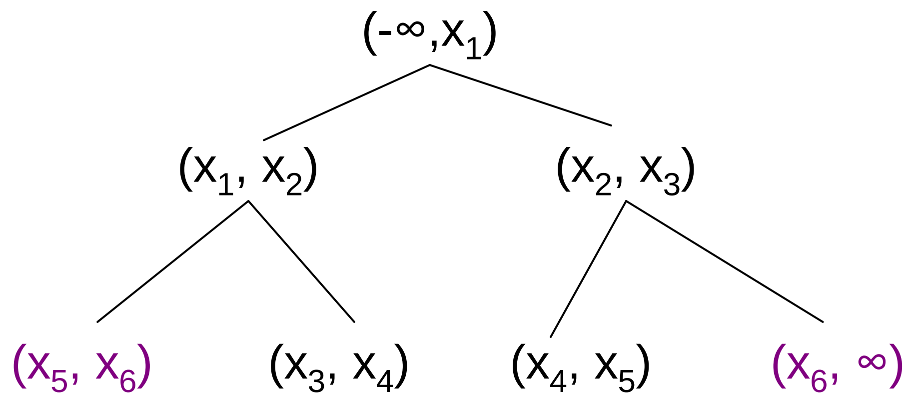
$X=\{x_1,x_2,x_3,x_5\}$, next: x_4

How to insert elements?



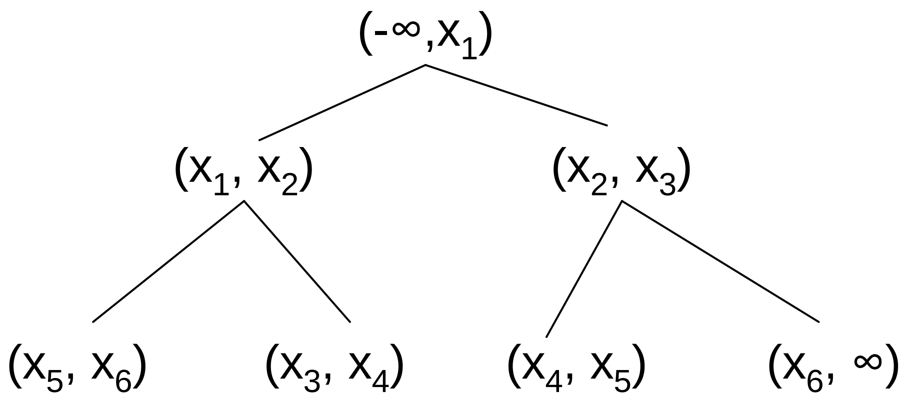
$X=\{x_1,x_2,x_3,x_4,x_5\}$, next: x_6

How to insert elements?



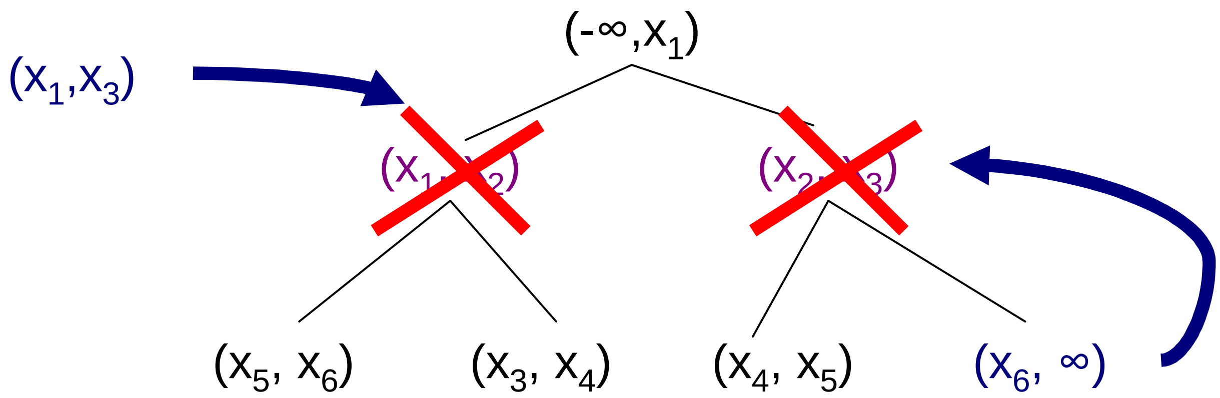
$X=\{x_1,x_2,x_3,x_4,x_5,x_6\}$

How to delete elements?

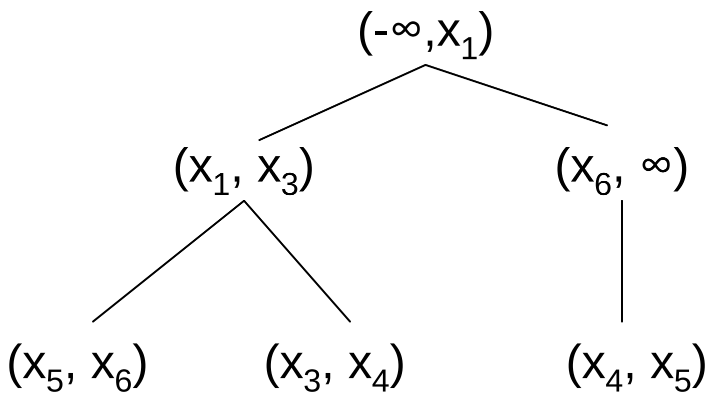


$X=\{x_1,x_2,x_3,x_4,x_5,x_6\}$
element to be deleted: x_2

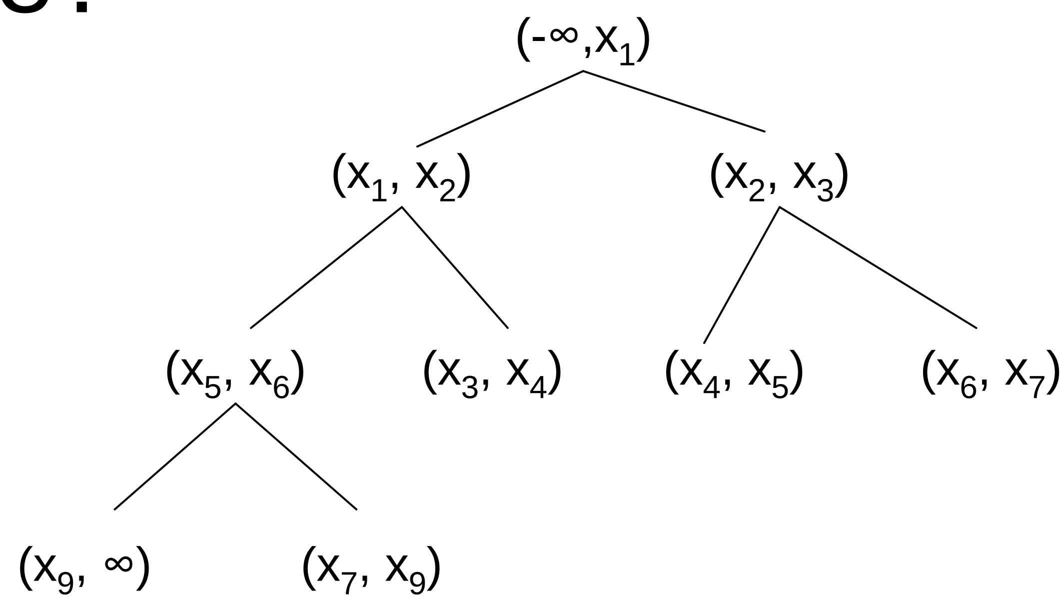
How to delete elements?



How to delete elements?



How to compute the accumulated value?



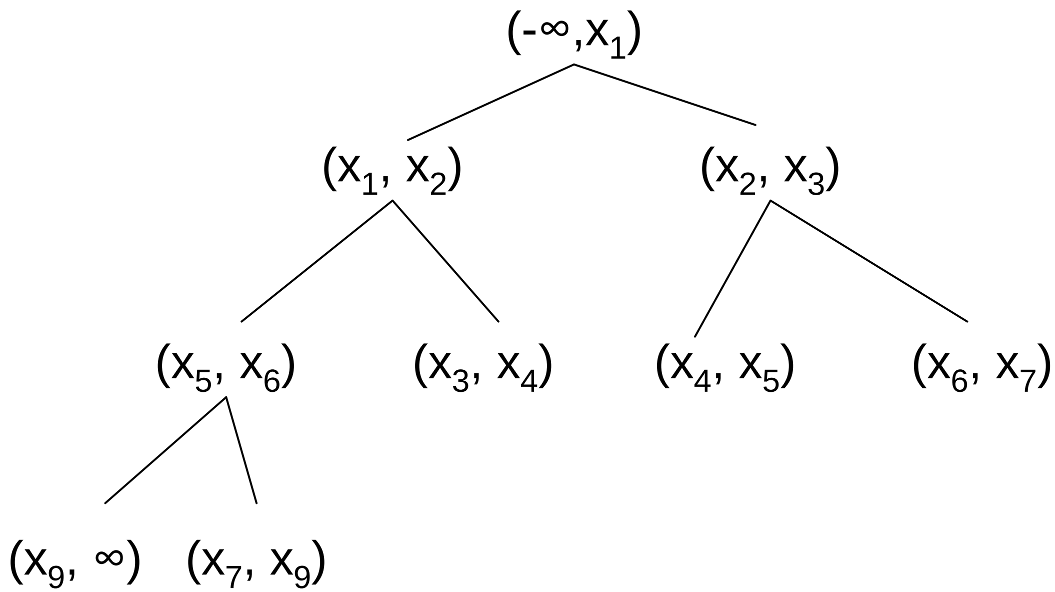
A pair (x_i, x_j)

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

$Proof_{Nil} =$ “”

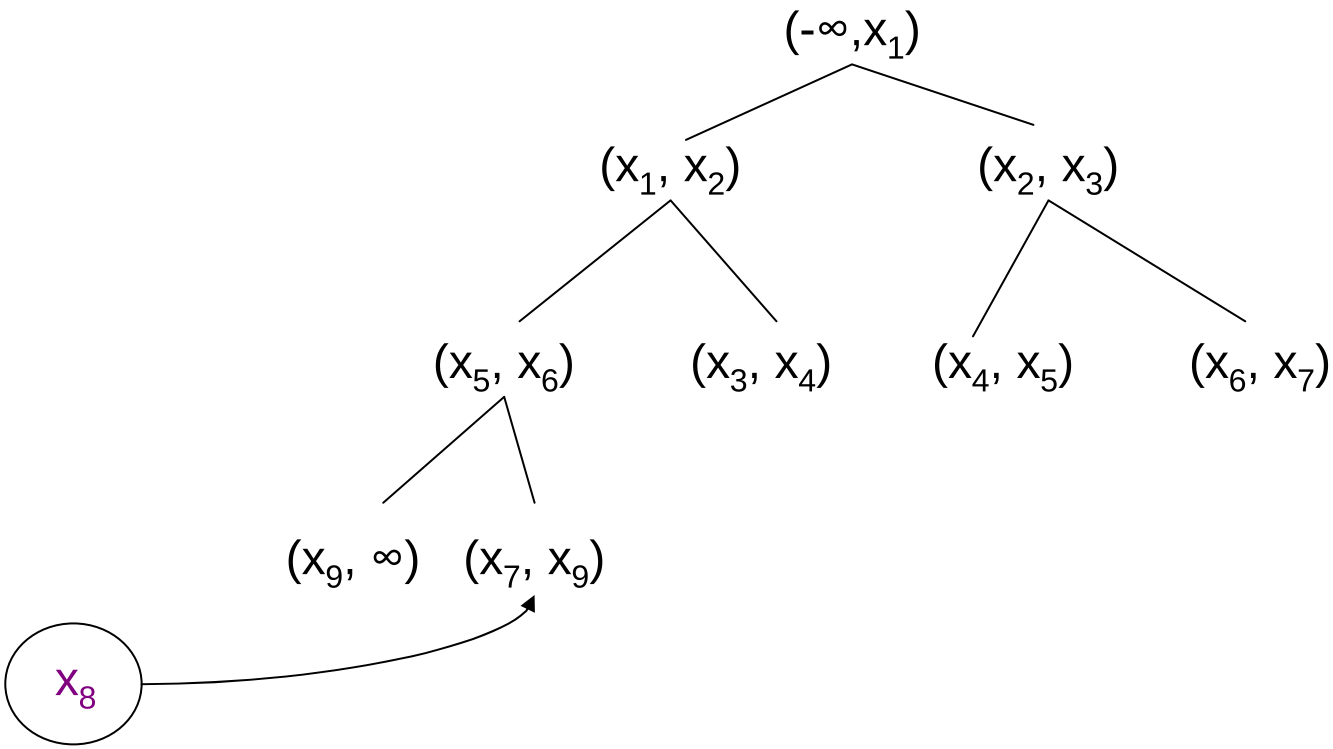
$Acc = Proof_{Root}$

How to update the accumulated value? (Insertion)



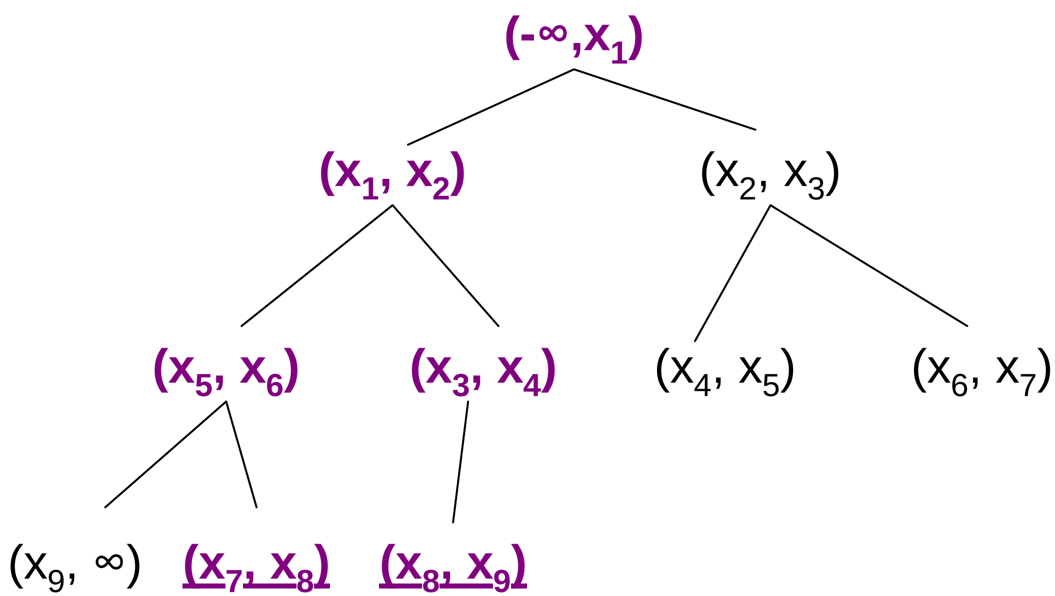
x_8 to be inserted.

How to update the accumulated value? (Insertion)



We will need to recompute proof node values.

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.

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.

Security (Informal)

- **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.

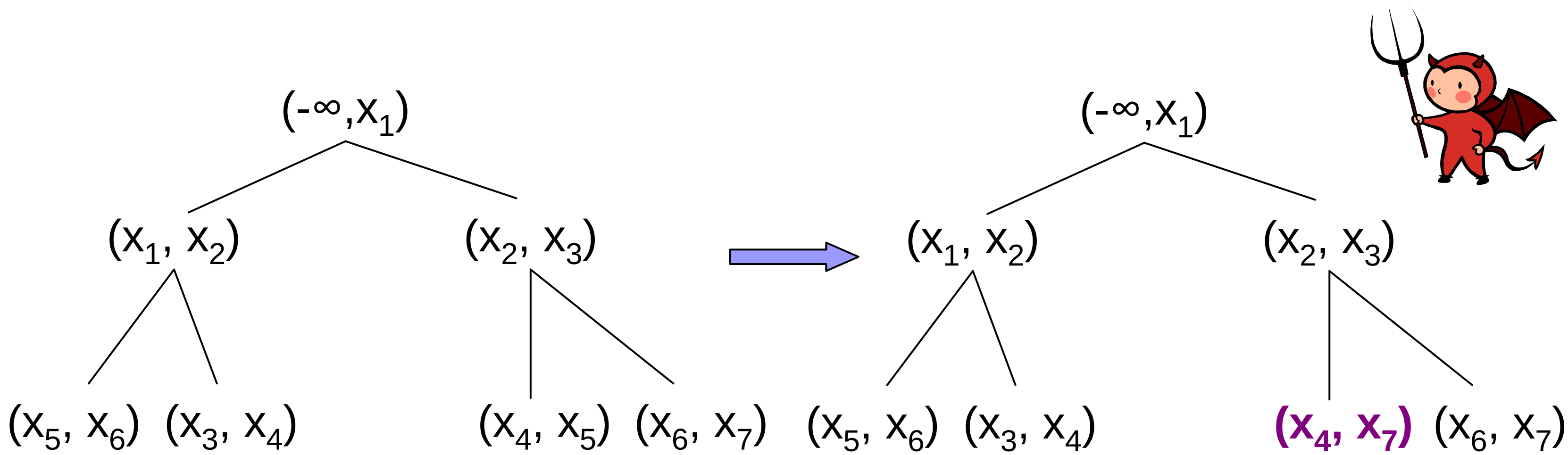
Security (Informal)

- **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.

Security

- **Lemma:** Given a tree T with accumulated value Acc_T, finding a tree T', T≠T' such that $Acc_T = Acc_{T'}$ is difficult.

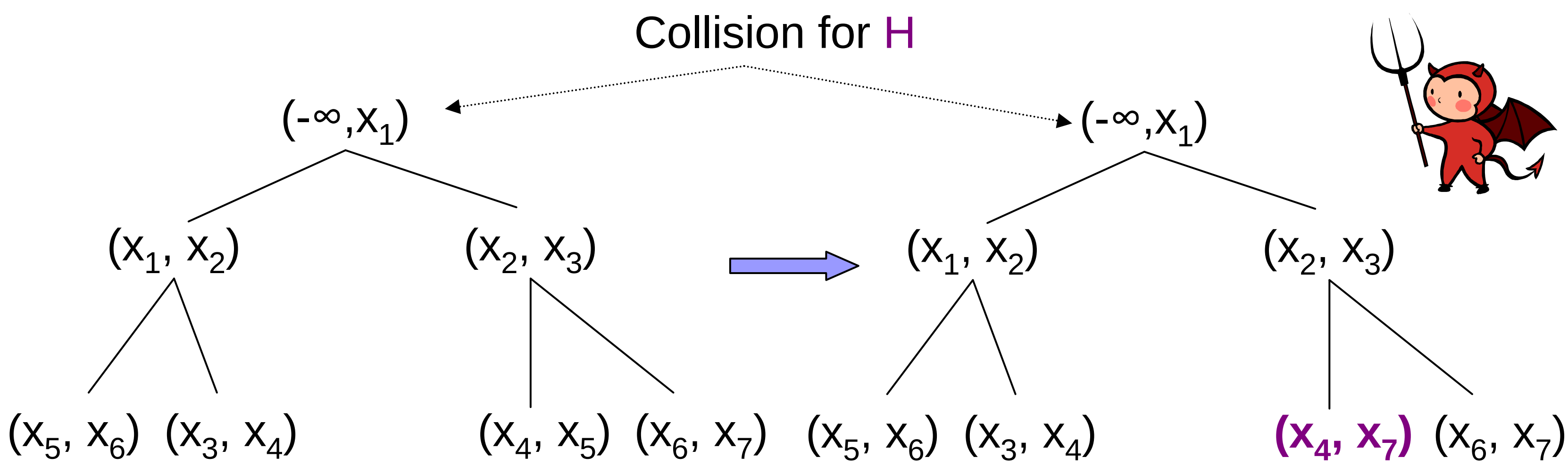
- *Proof (Sketch):* $Proof_N = H(Proof_{left}||Proof_{right}||value)$



Security

- **Lemma:** Given a tree T with accumulated value Acc_T, finding a tree T', $T \neq T'$ such that $Acc_T = Acc_{T'}$ is difficult.

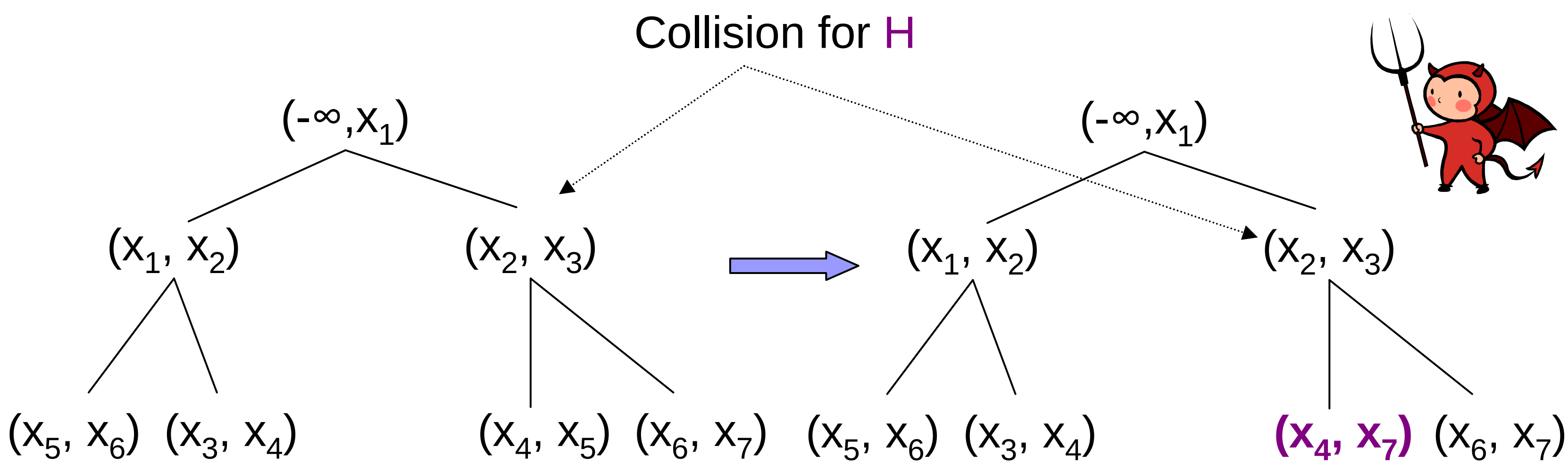
- *Proof (Sketch):* $Proof_N = H(Proof_{left} || Proof_{right} || value)$



Security

- **Lemma:** Given a tree T with accumulated value Acc_T, finding a tree T', $T \neq T'$ such that $Acc_T = Acc_{T'}$ is difficult.

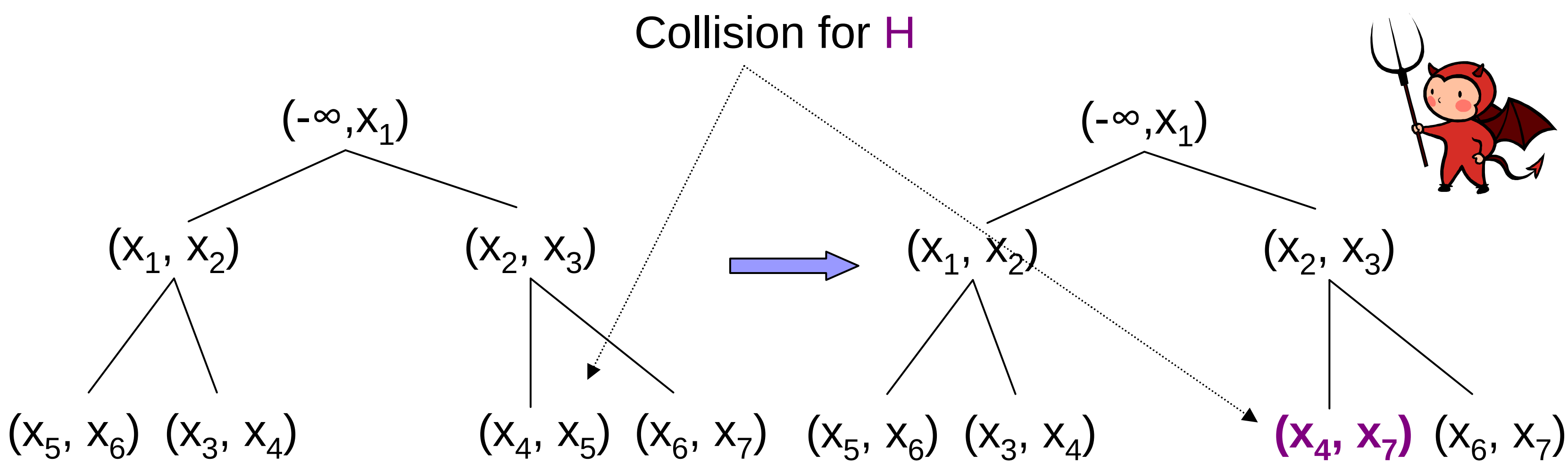
- *Proof (Sketch):* $Proof_N = H(Proof_{left} || Proof_{right} || value)$



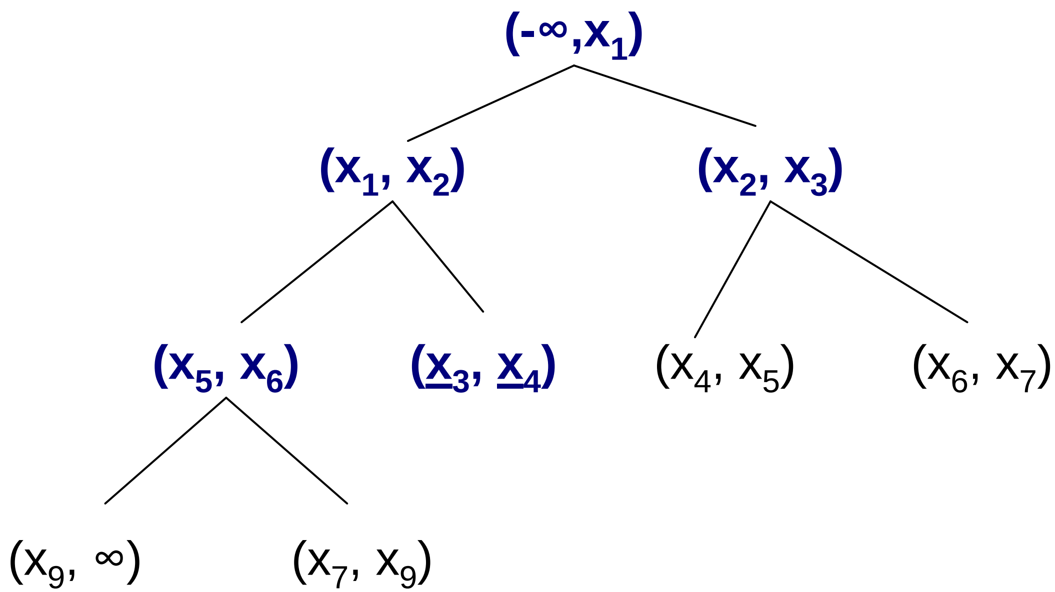
Security

- **Lemma:** Given a tree T with accumulated value Acc_T, finding a tree T', $T \neq T'$ such that $Acc_T = Acc_{T'}$ is difficult.

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



Security (Consistency)



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

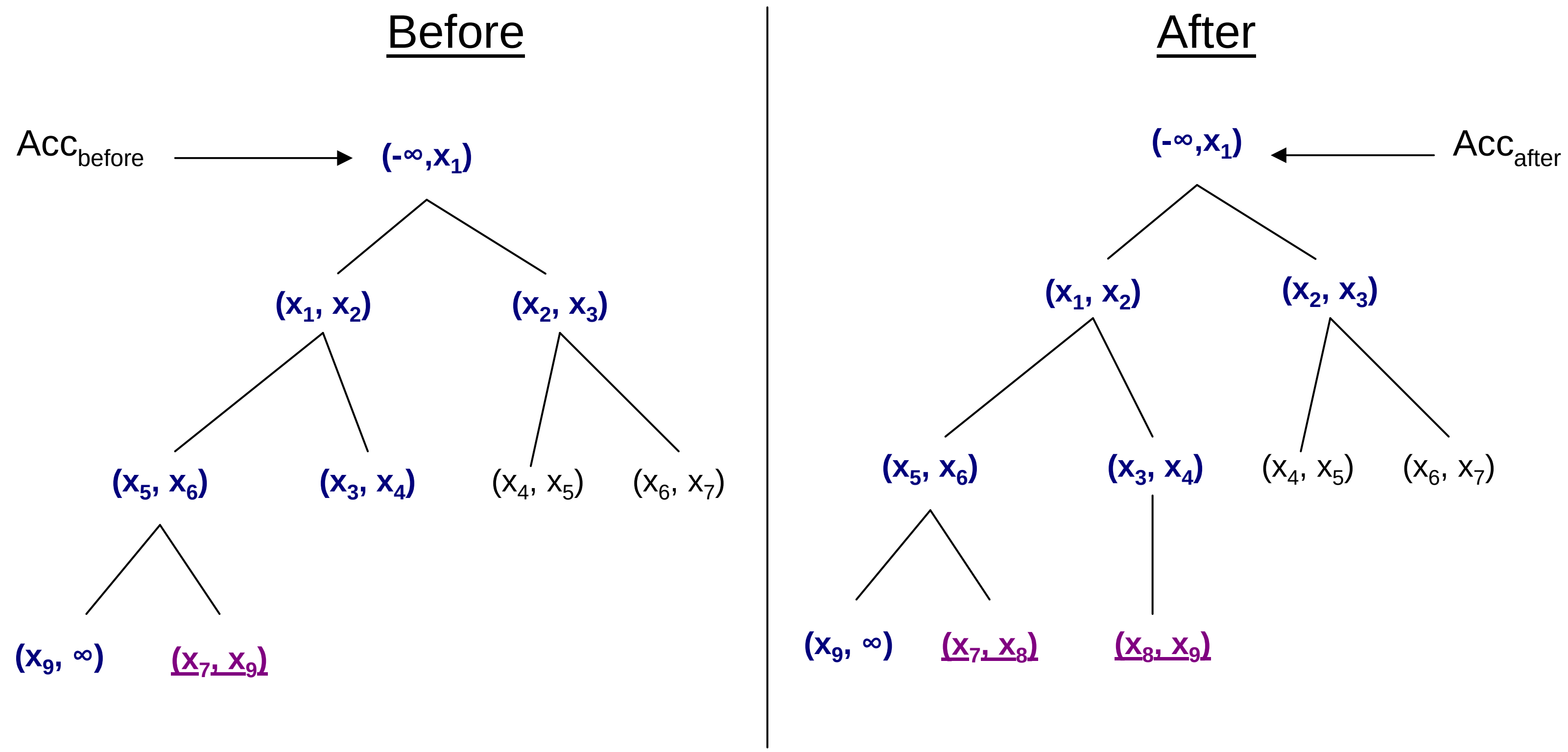
Checking that x belongs (or not) to X:

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)

Security (Update)



Insertion of x_8

Conclusion & Open Problem

- First *dynamic, universal, strong* accumulator.
- Simple.
- Security
  - Existence of collision-resistant hash functions.
- 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 a *strong*,*dynamic* and *universal* accumulator with witness size lower than O(ln(n))?"

Thank you!

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
- **[WWP08]** A new Dynamic Accumulator for Batch Updates *Peishun Wang, Huaxiong Wang and Josef Pieprzyk* 2008
- **[CKHO08]** Strong Accumulators from Collision-Resistant Hashing *Philippe Camacho, Alejandro Hevia, Marcos Kiwi, and Roberto Opazo* 2008