Merkle Hash Tree

CS 5158/6058 Data Security and Privacy

Spring 2018

Instructor: Boyang Wang

Crypto Currency

- 1545 Different crypto currency
  - As of 03/08/2018, based on coinmarketcap.com
  - Major ones: Bitcoin, Ethereum, etc.

#	Name	Symbol	Market Cap	Price
1	Bitcoin	BTC	$167,557,694,218	$9,910.31
2	Ethereum	ETH	$73,453,471,840	$749.06
3	Ripple	XRP	$33,722,246,441	$0.862639
4	Bitcoin Cash	BCH	$18,424,896,270	$1,083.40

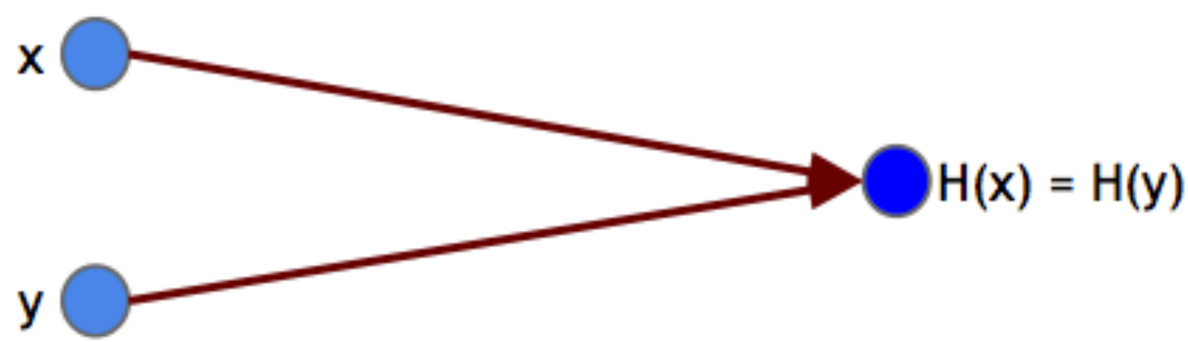
Crypto Currency

- Bitcoin and blockchain are <u>different</u>!
  - Bitcoin is crypto currency
  - Blockchain is a chain of hash values

- We will focus on crypto building blocks in Bitcoin
  - Blockchain (hash chain, merkle hash tree)
  - Mining (Proof of Works)

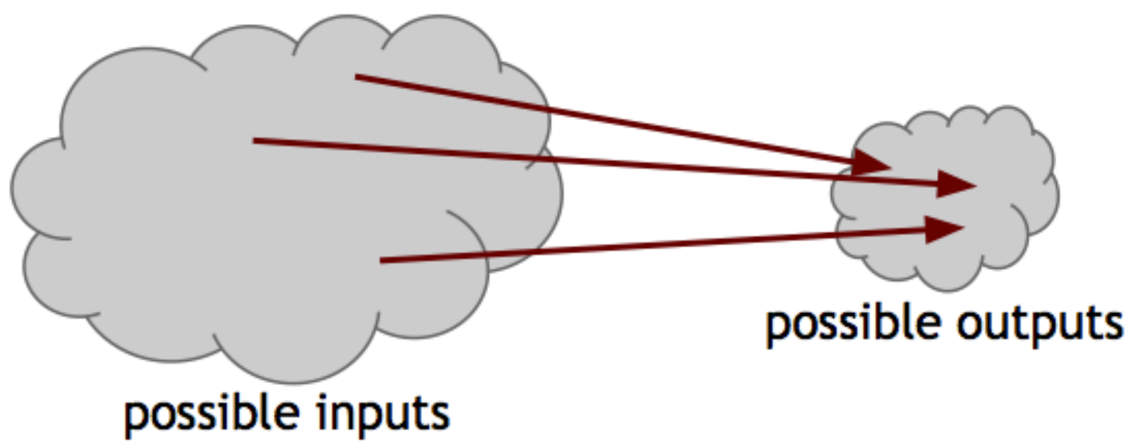
Hash Function Review

- Hash Function (e.g., SHA256):
  - An arbitrary-length input —> a fixed-length output
  - Deterministic function
  - Efficient to compute, but hard to invert
  - A collision: x != y, but H(x) == H(y)



Hash Function Review

- Hash Function (e.g., SHA256):
  - A collision must exist
    - No. of inputs >> no. of outputs
  - A collision is hard to find
    - happens with a negligible prob. for PPT algo.



Trans. ID	Sender	Receiver	Amount	Time
t1	Alice	Bob	$20	03/10/2018
t2	Alice	David	$100	03/15/2018
t3	Bob	David	$50	03/21/2018

- A bank has a number of transactions
- Bank wants to ensure all the trans are correct.
  - Trans could be changed by hacker on server
  - Trans could be changed by attacker on channel

Use Hash Function

- Assume a bank has transactions: {t1, t2, t3}
  - Can use a hash function (e.g., SHA256)
  - Take all trans. as input, compute one hash value
  - E.g., H(t1, t2, t3) = h

- Bank publishes **h**, anyone can verify {t1, t2, t3}
  - Given {t1, t2, t3}, Alice checks H(t1,t2, t3) ?= h
    - If identical, all trans are correct;
    - otherwise, some trans is not correct

Use Hash Function

- Assume a bank has transactions: {t1, t2, t3}
  - Can use a hash function (e.g., SHA256)
  - E.g., H(t1, t2, t3) = h
  - Bank publishes **h**, anyone can verify {t1, t2, t3}

- Attacker changes t3 to t3'
  - H(t1, t2, t3') != h, change can be detected
  - H(t1, t2, t3') == h, change will not be detected
    - A collision happens with negligible prob.

Use Hash Function

- Assume a bank has transactions: {t1, t2, t3}
  - H(t1, t2, t3) = h
  - Bank publishes **h**, anyone can verify {t1, t2, t3}

- Attacker changes the order of trans
  - {t1, t2, t3} —> {t1, t3, t2}
  - H(t1,t3,t2) != h, can be detected
  - H(t1,t3,t2) == h, will not be detected
    - A collision happens with negligible prob.

Use Hash Function

- Assume a bank has transactions: {t1, t2, t3}
  - H(t1, t2, t3) = h
  - Bank publishes **h**, anyone can verify {t1, t2, t3}

- Attacker appends 1 trans
  - {t1, t2, t3} —> {t1, t2, t3, t4}
  - H(t1,t2,t3,t4) != h, can be detected
  - H(t1,t2,t3,t4) == h, will not be detected
    - A collision happens with negligible prob.

Use Hash Function

- Assume a bank has transactions: {t1, t2, t3}
  - H(t1, t2, t3) = h
  - Bank publishes **h**, anyone can verify {t1, t2, t3}

- Can attacker add 1 trans without being detected?
  - {t1, t2, t3} —> {t1, t4, t2, t3}
  - H(t1,t4,t2,t3) != h, can be detected
  - H(t1,t4,t2,t3) == h, will not be detected
    - A collision happens with negligible prob.

Use Hash Function

- Assume a bank has transactions: {t1, t2, t3}
  - H(t1, t2, t3) = h
  - Bank publishes **h**, anyone can verify {t1, t2, t3}

- It is (computationally) hard for attacker to
  - change any trans
  - change order of trans
  - add new trans
- This approach can protect data integrity

Trans. ID	Sender	Receiver	Amount	Time
t1	Alice	Bob	$20	03/10/2018
t2	Alice	David	$100	03/15/2018
t3	Bob	David	$50	03/21/2018
t4	David	Alice	$80	03/22/2018
t5	Charlie	David	$10	03/23/2018

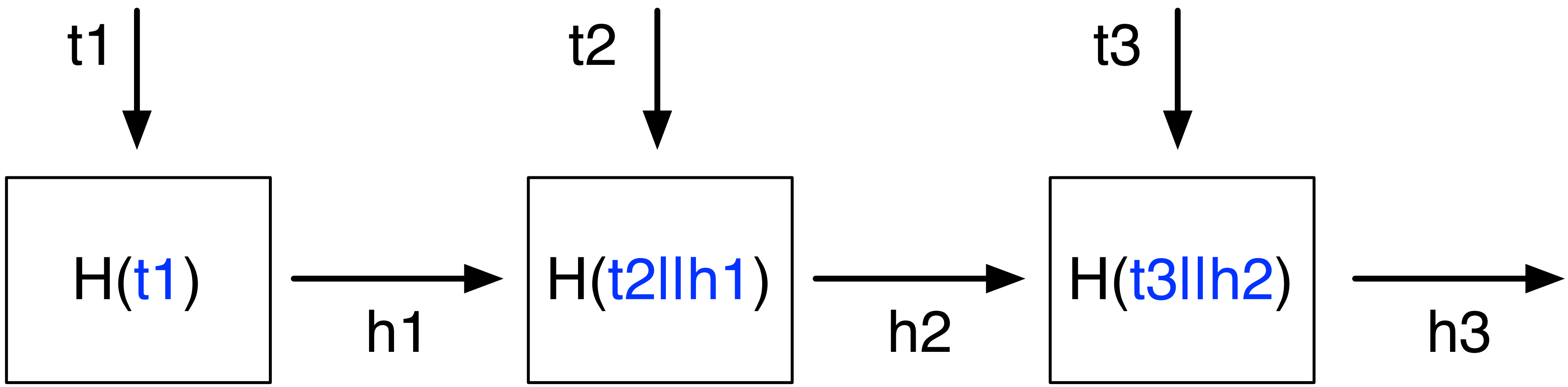
- Bank has new transactions every day
- Bank wants to ensure all the trans are correct.

Update Hash Value

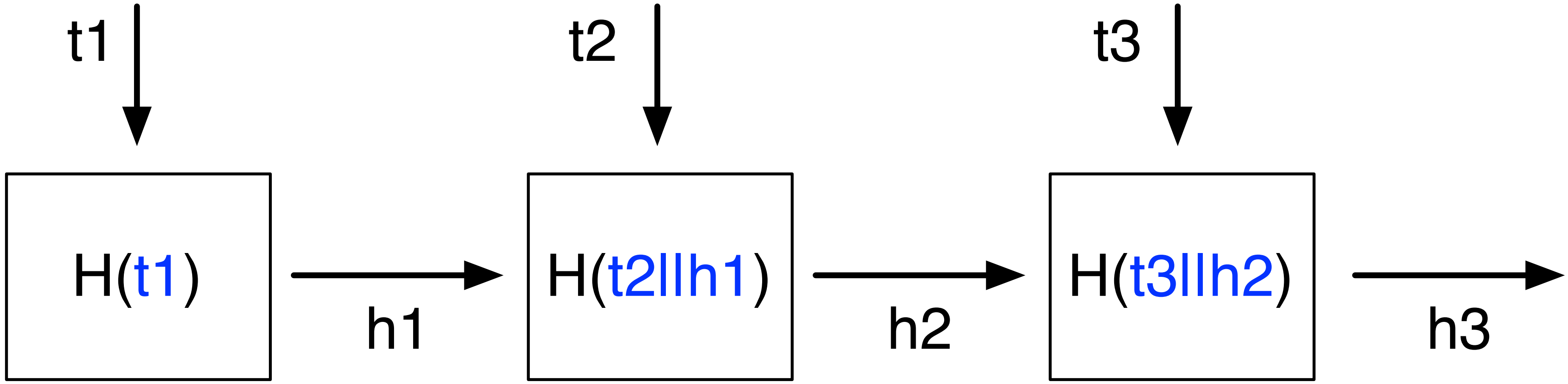
- Bank has new transactions: {t1, t2, t3, t4, t5}

- Recompute/Update a hash value
  - Take all trans. as input, compute one hash value
  - E.g., H(t1, t2, t3, t4, t5) = h
  - Bank publishes **h**, anyone can verify all trans

- Inefficient: bank needs to take all the trans as input
  - E.g., reading 1,000,000 trans. takes long time

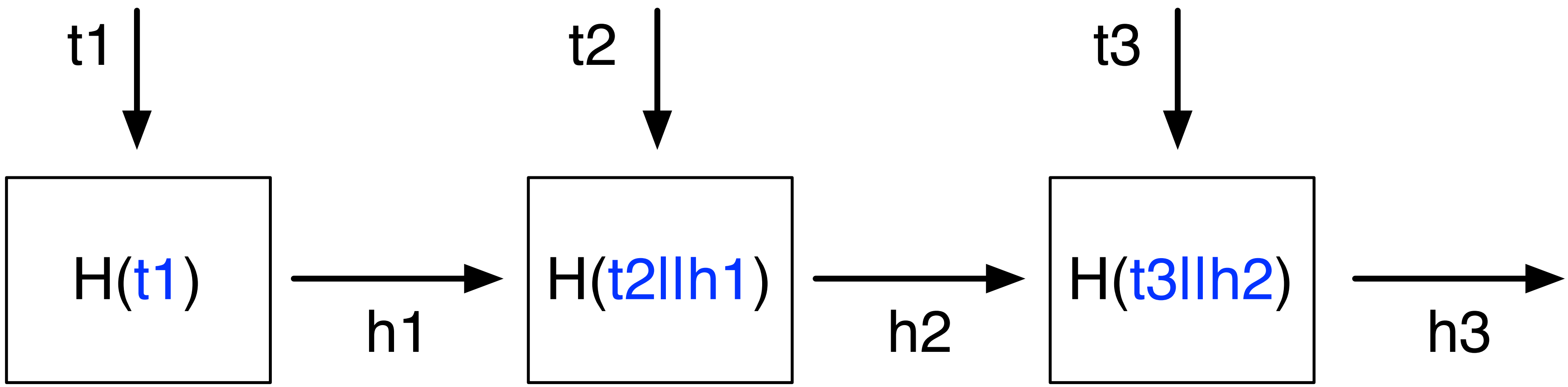


- Hash chain:
  - Given a new trans, bank computes a hash value of new trans with the current hash value
    - h1 can prove {t1} is correct
    - h2 can prove {t1, t2} are correct
    - h3 can prove {t1, t2, t3} are correct
  - Bank always updates the latest hash value

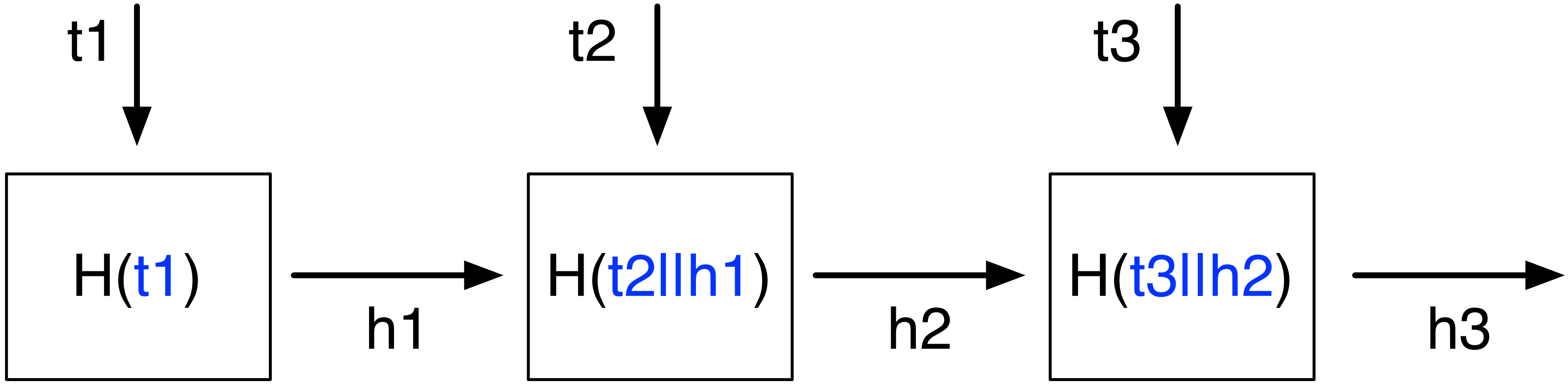


- Hash chain:
  - Bank publishes the latest hash value (e.g., **h3**)
  - Given {t1, t2, t3} and **h3**, Alice computes

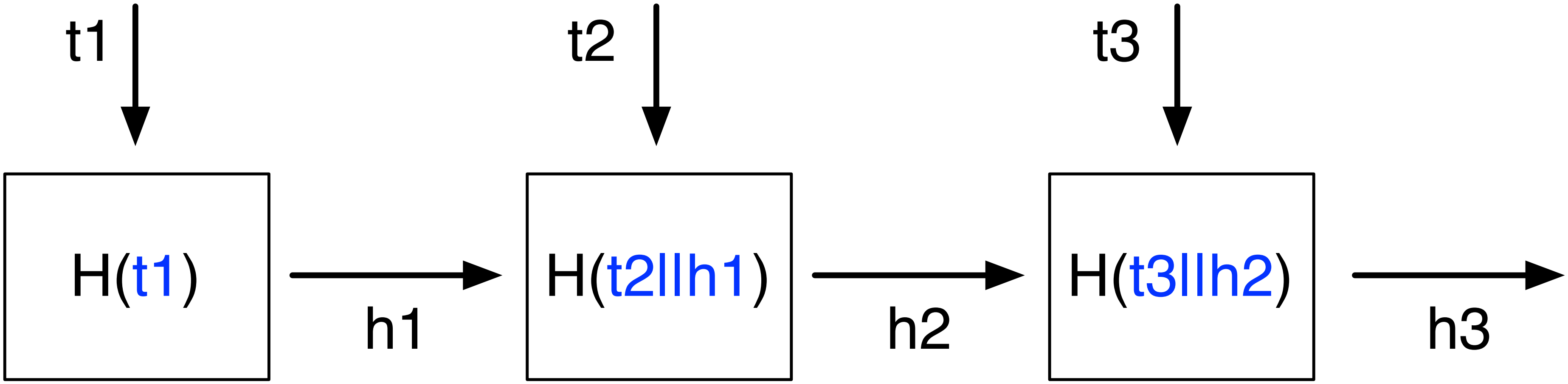
    H(t1) = h1; H(t2||h1) = h2; H(t3||h2) ?= **h3**
    - If identical, all trans are correct;
    - otherwise, some trans is not correct



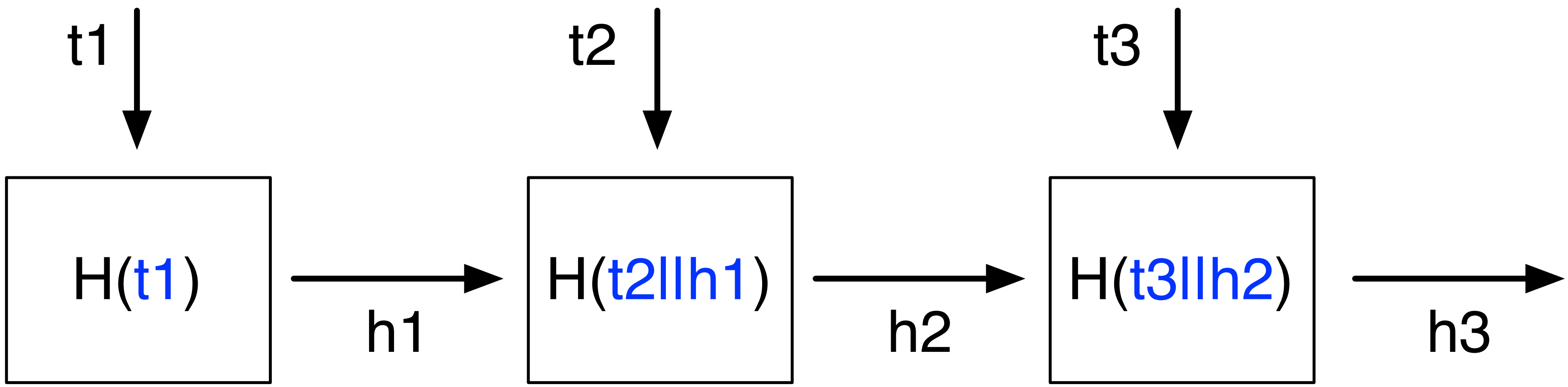
- Attacker changes t3 to t3’
  - H(t3’||h2) = h3’ != h3,
    - change can be detected
  - H(t3’||h2) = h3’ == h3,
    - change will not be detected
    - A collision happens with negligible prob.



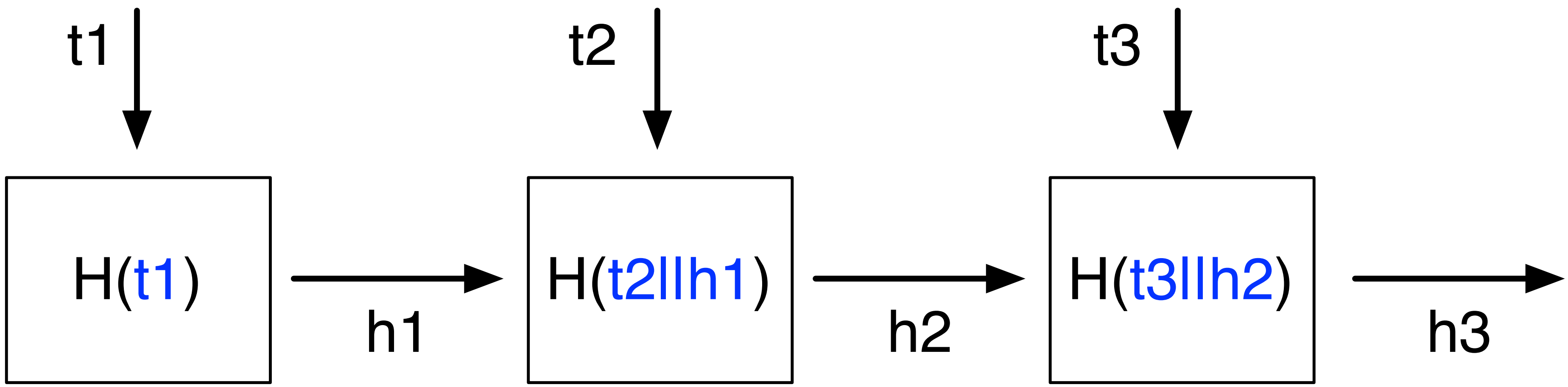
- Attacker changes t2 to t2'
  - H(t2'||h1) = h2' != h2
    - Does not find a collision at block 2 for t2'||h1
    - Since h2' != h2, then H(t3||h2') = h3' != h3
    - Change can be detected



- Attacker changes t2 to t2'
  - H(t2'||h1) = h2' != h2,
    - Does not find a collision at block 2 for t2'||h1
  - h2' != h2, H(t3||h2') = h3' == h3
    - Finds a collision at block 3 for t3||h2'
    - Change will not be detected
    - A collision happens with a negligible prob.

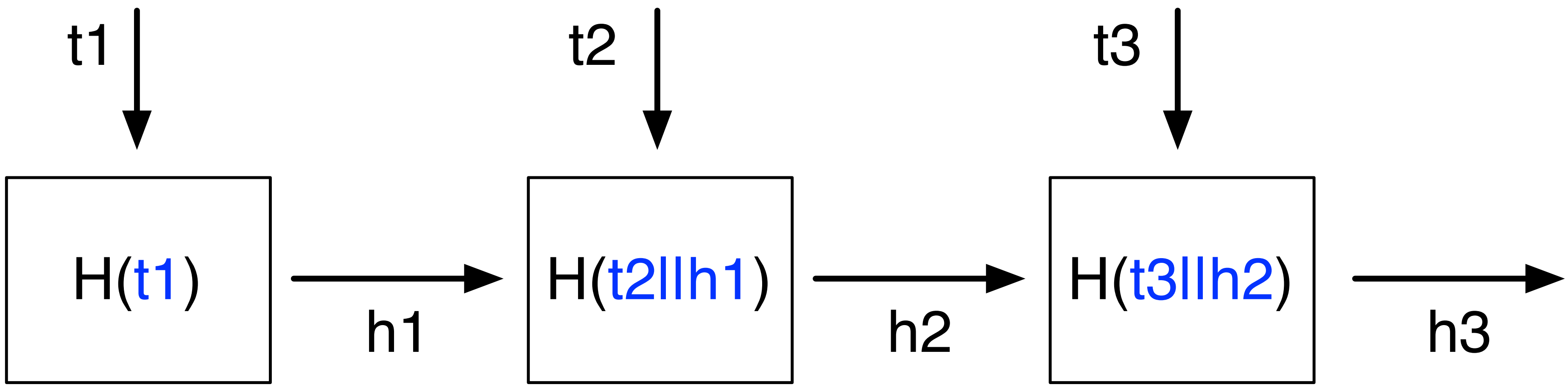


- Attacker changes t2 to t2’
  - H(t2’||h1) = h2’ == h2,
    - Finds a collision at block 2 for t2’||h1
    - Since h2’ == h2, then H(t3||h2’) = h3’ == h3
    - Change will not be detected
    - A collision happens with a negligible prob.



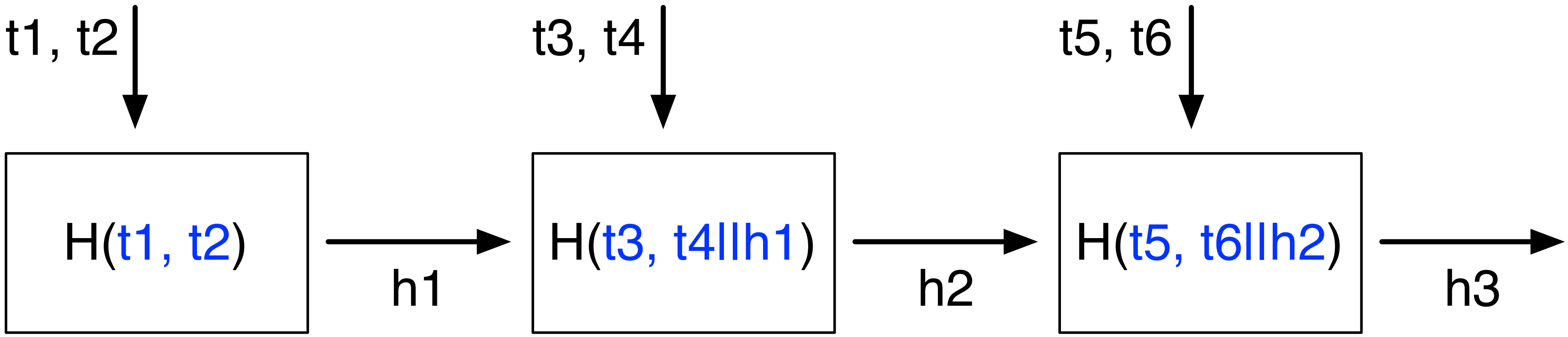
- Bank publishes the latest hash value (e.g., **h3**)

- Practice 1: Attacker changes the order to {t2, t3, t1}
  - Given **h3**, will Alice accept {t2, t3, t1}?
    - No, finding a collision is hard
- Practice 2: Attacker appends 1 trans {t1, t2, t3, t4}
  - Given **h3**, will Alice accept {t1, t2, t3, t4}?
    - No, finding a collision is hard

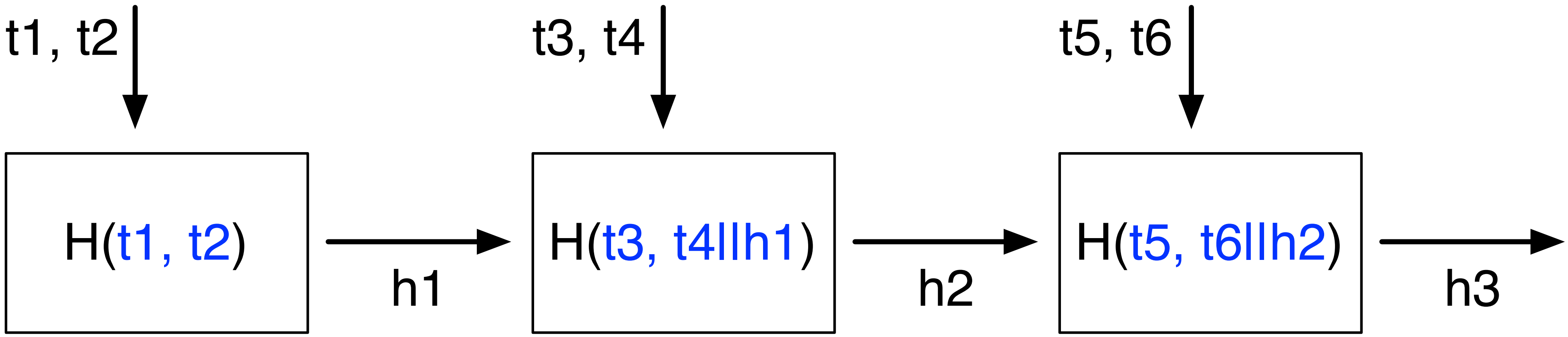


- Bank can efficiently update the latest hash value
  - 1 hash each new trans
- It is (computationally) hard for attacker to change trans, add trans or change the order of trans

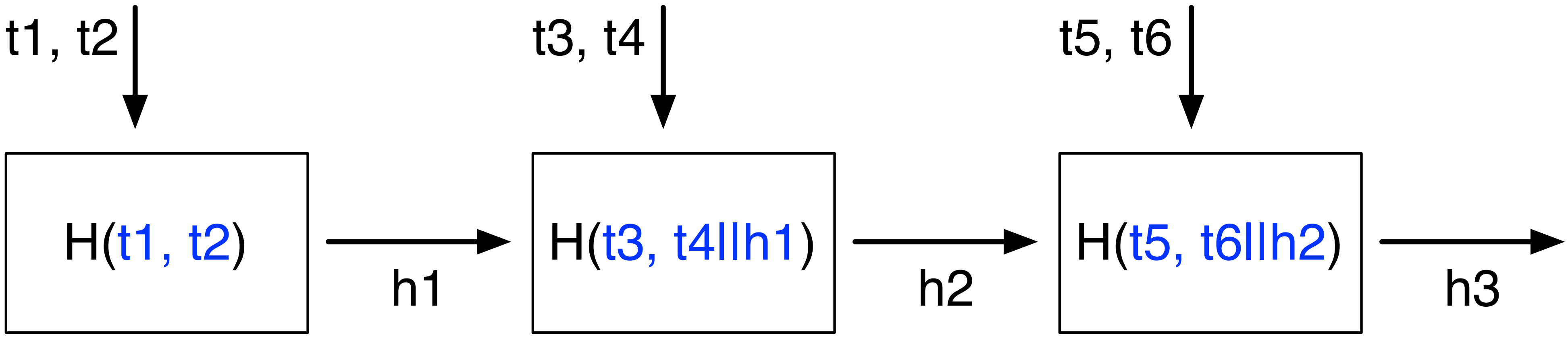
- Alice can still verify
  - Reads n trans in total, & computes n hashes (v.s. 1 hashes in previous method)



- Bank takes multiple trans each block, can reduce the total number of hash operations
  - h1 can prove {t1, t2} are correct
  - h2 can prove {t1, t2, t3, t4} are correct
  - h3 can prove {t1, t2, t3, t4, t5, t6} are correct
  - 3 hashes v.s. 6 hashes (in previous case)



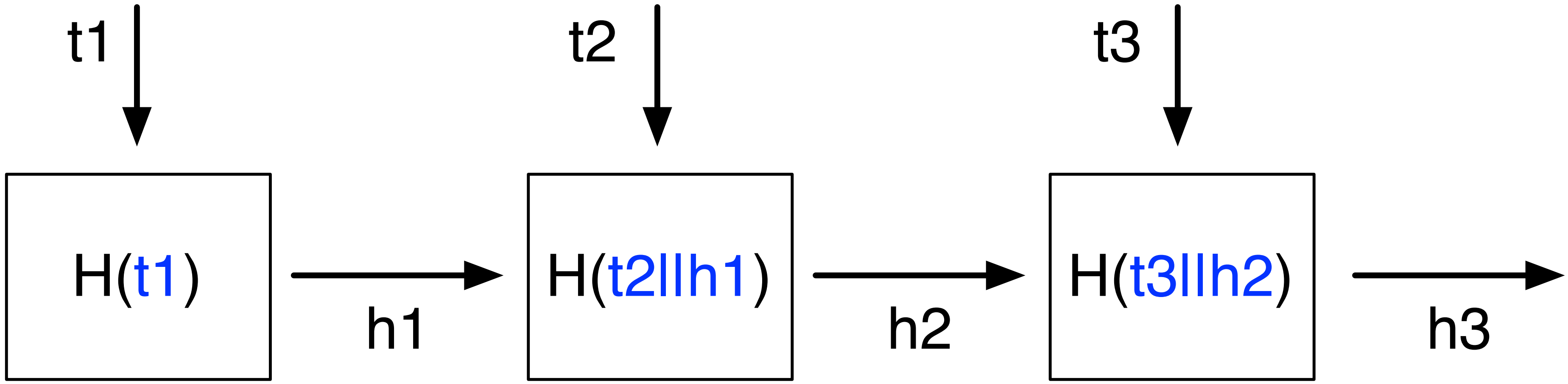
- Tradeoff:
  - If t5 is ready, but t6 is not available yet
  - Bank needs to wait to generate next hash value
  - t5 cannot be confirmed immediately
- If Bank increases no. of trans in each block
  - less hash operations, more pending trans.



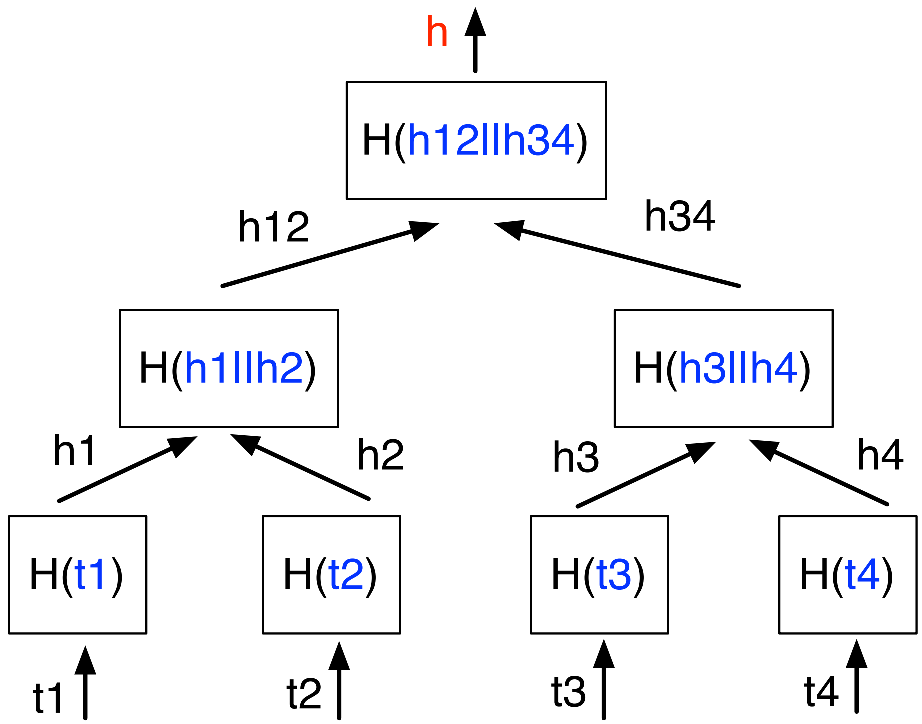
- Attacker changes t5 to t5’
  - Change can be detected
    - H(t5’,t6||h2) = h3’ != h3,
  - Change will not be detected
    - H(t5’,t6||h2) = h3’ == h3,
    - A collision happens with a negligible prob.

Trans. ID	Sender	Receiver	Amount	Time
t1	Alice	Bob	$20	03/10/2018
t2	Alice	David	$1000	03/15/2018
t3	Bob	David	$50	03/21/2018
t4	David	Alice	$80	03/22/2018
t5	Charlie	David	$10	03/23/2018

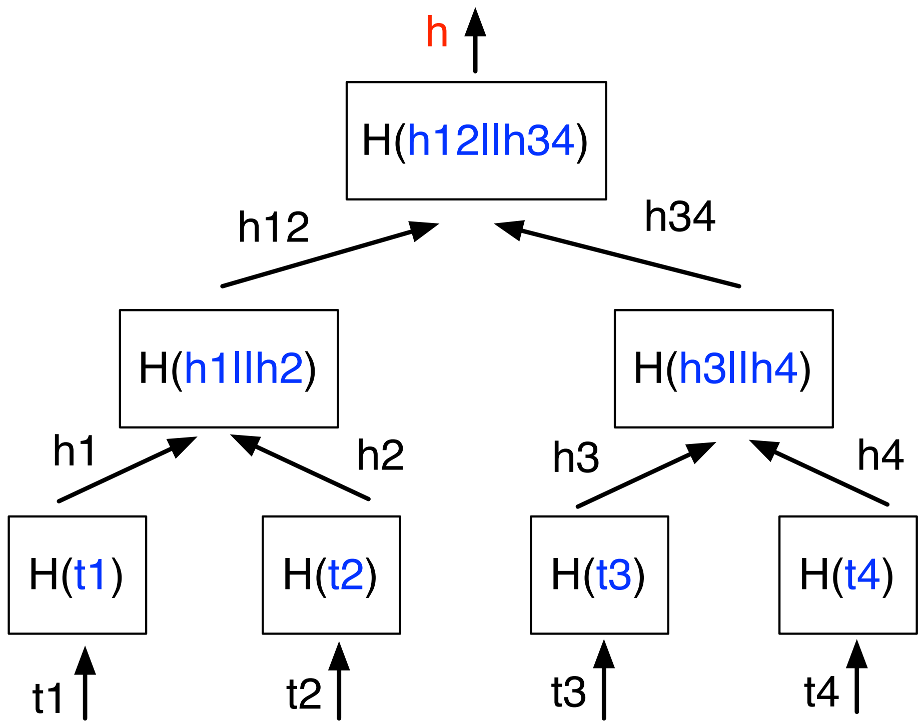
- Bank has new transactions every day.
- Bank updates old transactions.
- Bank wants to ensure all the trans are correct.



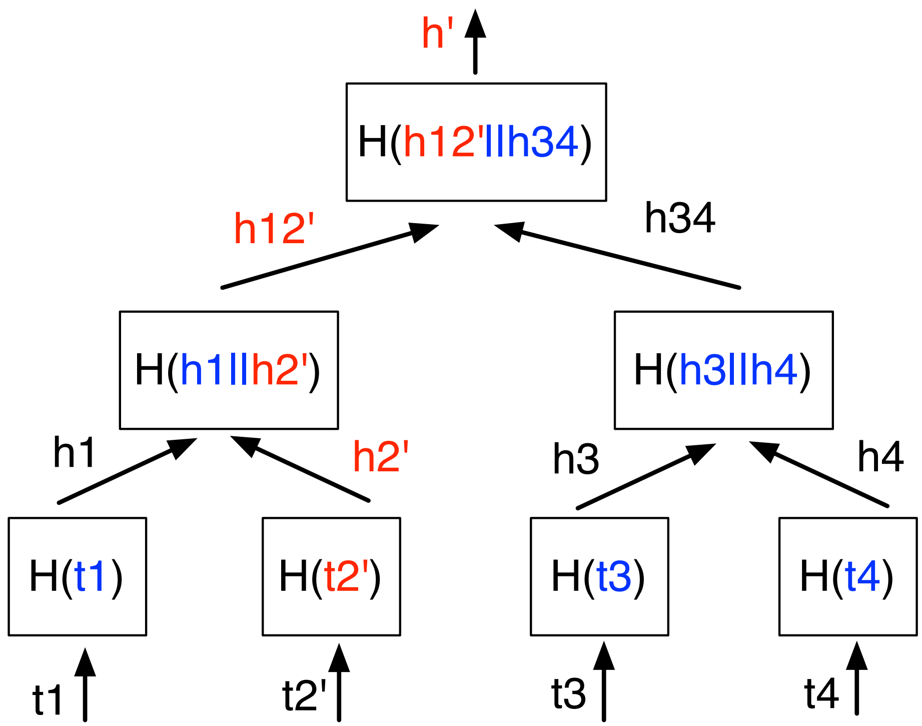
- Bank updates using Hash chain:
    - If t3 is updated to t3'
        - recompute h3, if still has a copy of h2; otherwise, recompute h1, h2, h3
    - If t1 is updated to t1',
        - recompute h1, h2, h3
- Recompute O(N) hashes, not very efficient
    - E.g., N=1,000,000



- Merkle hash tree:
  - Bank has {t1, t2, t3, t4}
  - Bank computes a root hash h based on tree
  - Each trans is a leaf
  - Root hash h proves {t1, t2, t3, t4}
  - Bank publishes **h**, anyone can verify all trans

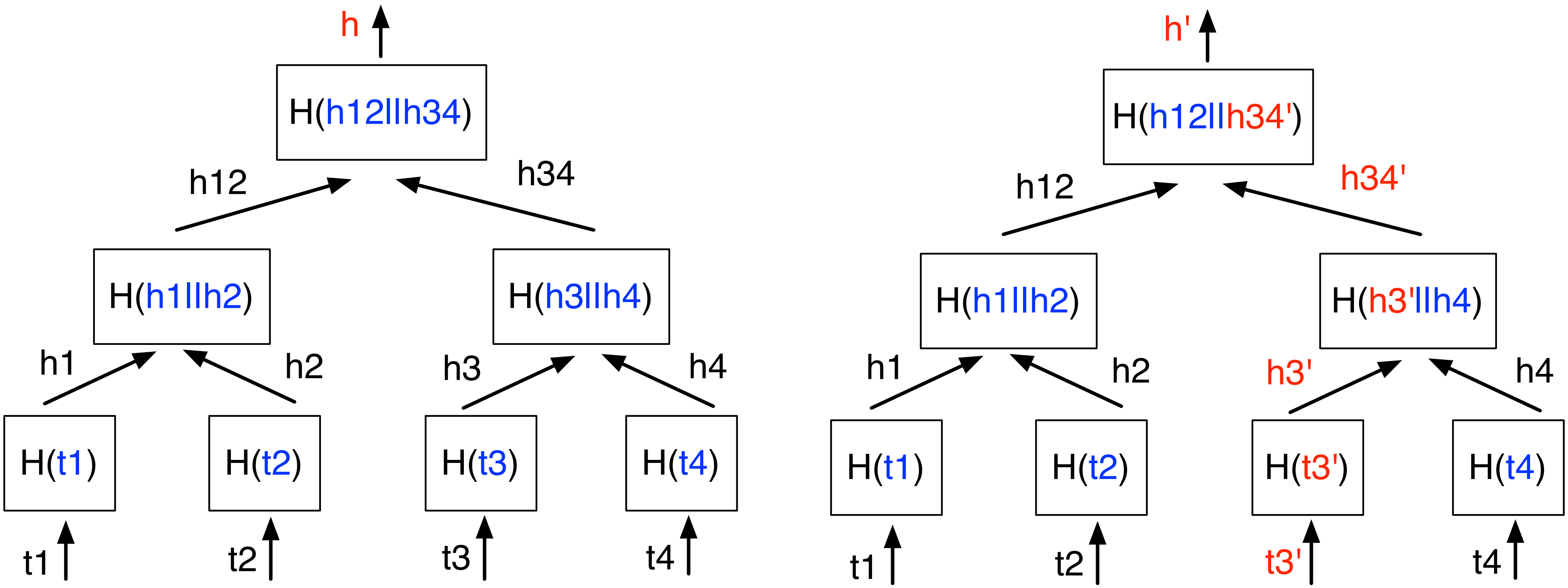


- Bank publishes **h**, anyone can verify all trans
- Given h and {t1, t2, t3, t4}, Alice recomputes root hash h', and compare it with **h**
  - 2N-1 hashes v.s. N hashes in hash chain
  - N = 4: 7 hashes v.s 4 hashes



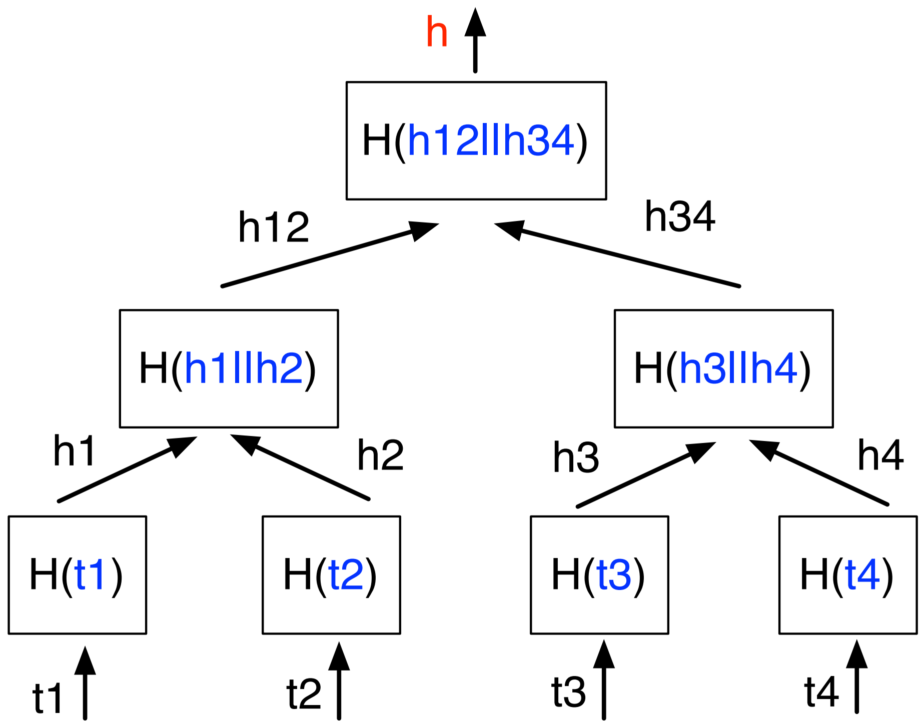
- Bank updates t2 to t2’
  - Recomputes h2’, h12’, h’ in the tree
    - Assume bank maintains all the non-leaf nodes
    - O(logN) hashes v.s. O(N) in hash chain

- Bank publishes **h’**, anyone can verify all trans



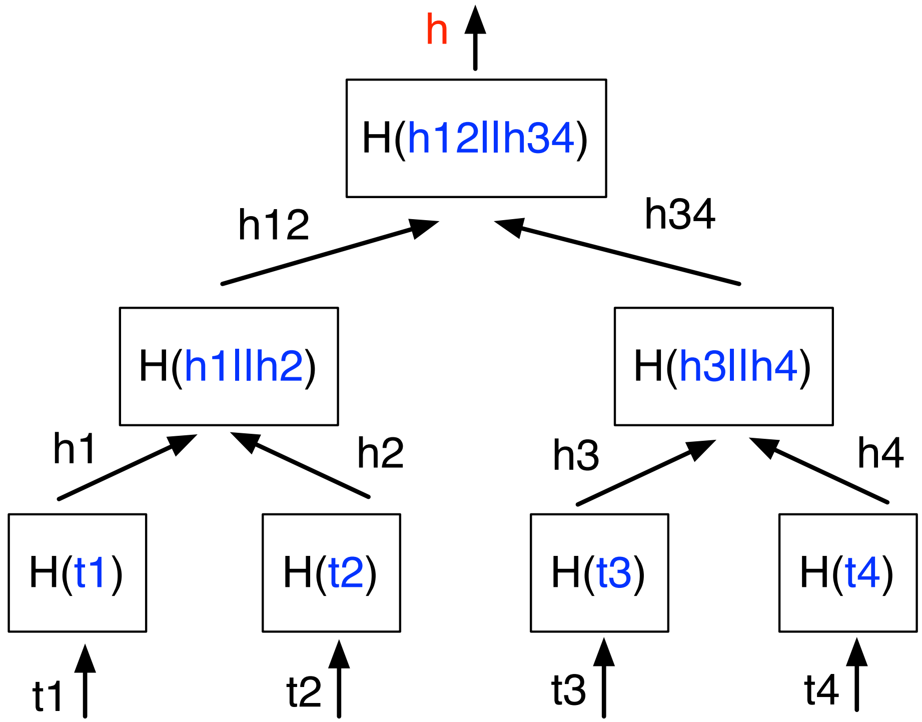
- <u>Practice:</u> if bank updates t3 to t3', which hash values in the tree need to be recomputed?

- Bank recomputes h3', h34', h'



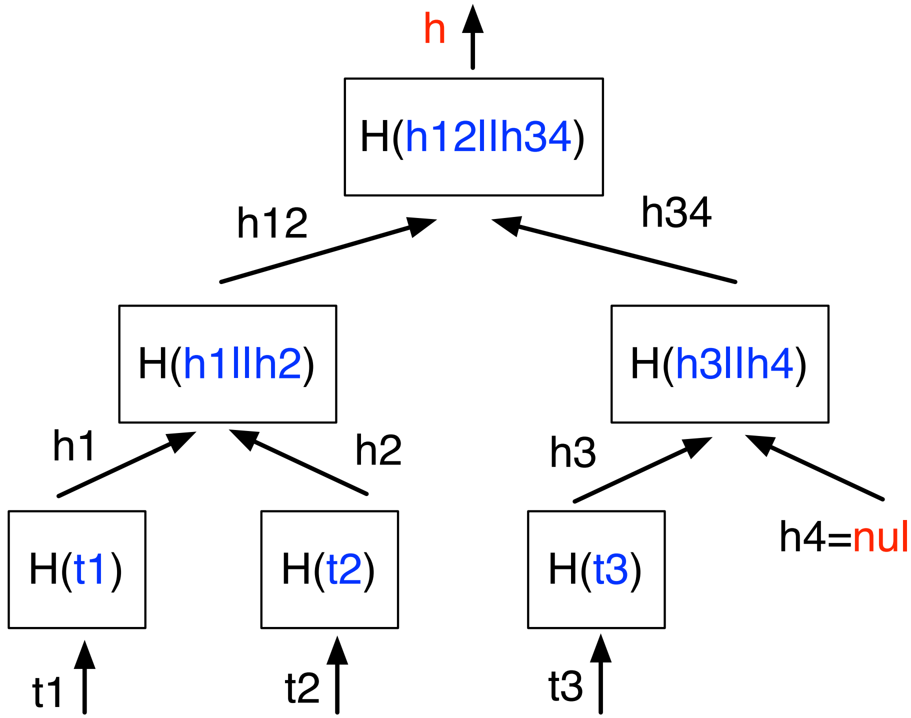
- Bank publishes **h**, anyone can verify all trans

- Attacker changes t2 to t2' without being detected
  - Attacker needs to find 1 collision, i.e., H(t2') == H(t2), s.t. root h is still correct
  - Happens with a negligible probability



- Bank publishes **h**, anyone can verify all trans

- Similarly, it is hard for attacker to add new trans or change the order of trans
  - Need to find collisions
  - Happens with a negligible probability



- If the no. of trans is not a power of 2, how does bank build this tree?
- Example: if bank only has 3 trans, {t1, t2, t3}
  - Only 3 leaf nodes in tree
  - Bank leaves hash value h4 as null

- Practice: if bank has 5 trans, {t1, t2, t3, t4, t5}
    - How to compute the root hash?
    - Leave h6, h7, h8 as null