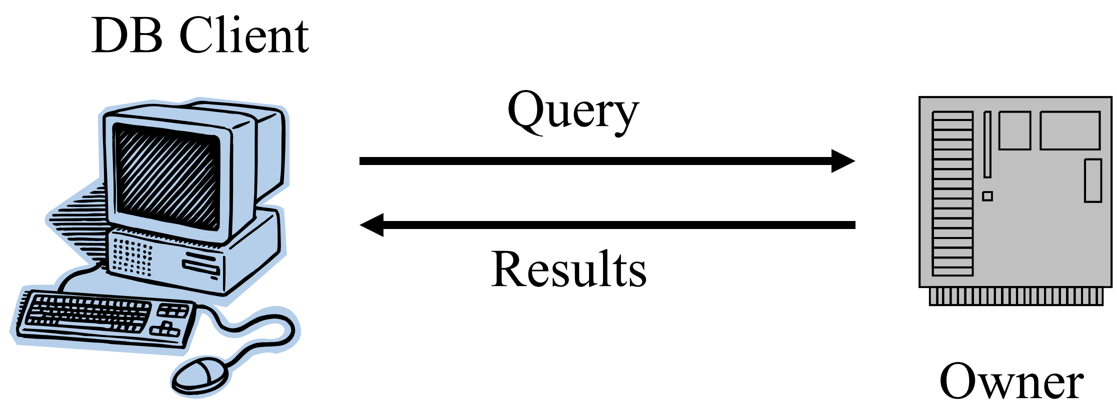
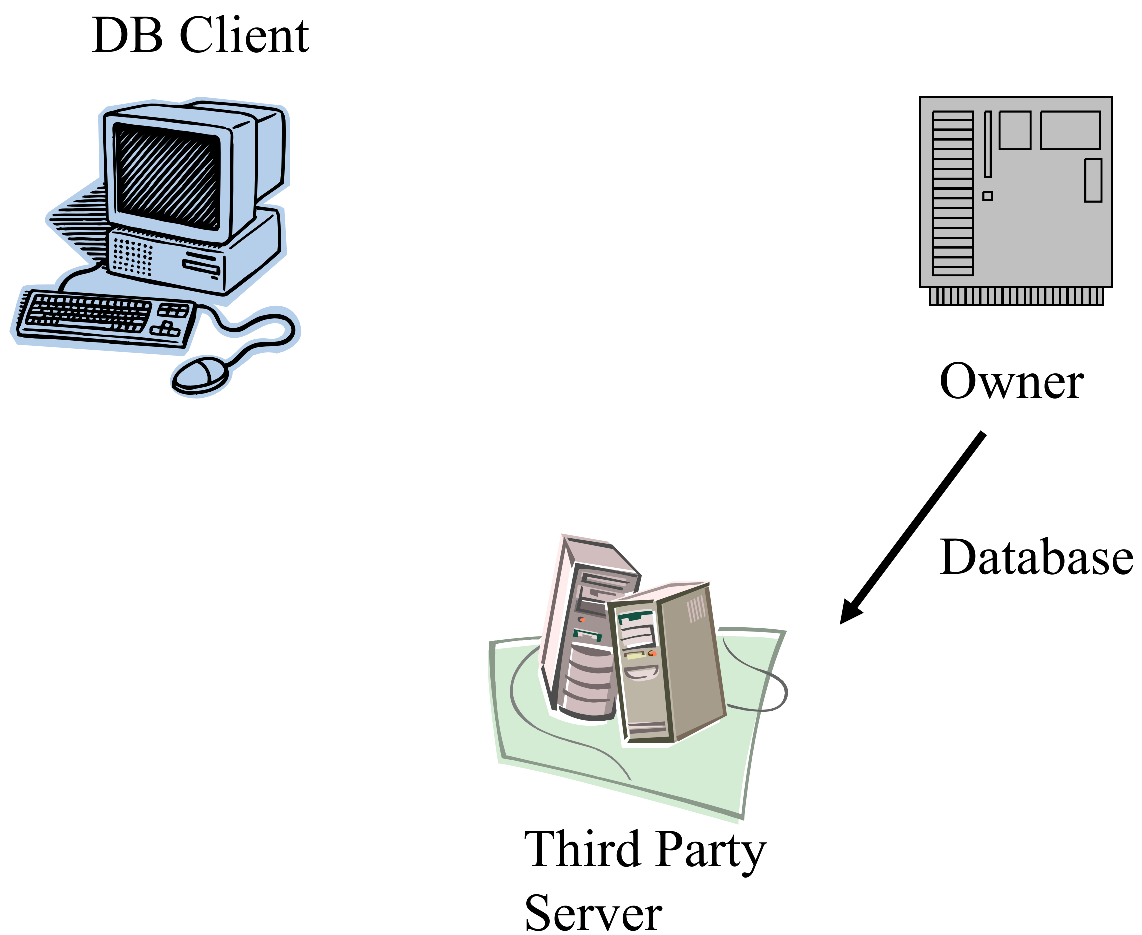
Security in Outsourced Databases (Query Answer Assurance)

Traditional Client-Server Arch.

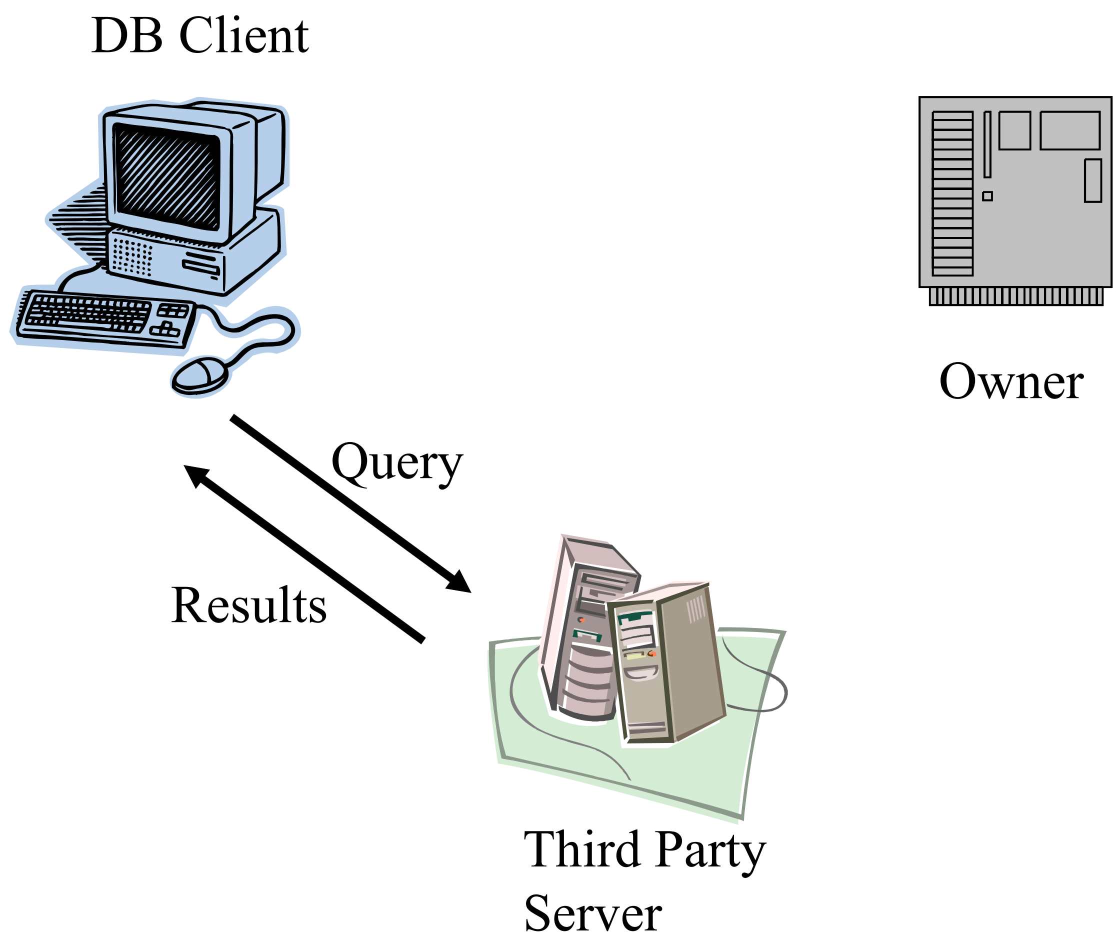


- Client queries are satisfied by a trusted server
- Secure the server
- Secure the communication channel, e.g. use SSL

Data Publishing (Database-as-a-Service)



Data Publishing

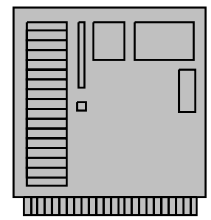


Data Publishing

- Pushes business logic and data processing from corporate data centers to third party servers at the “edge” of the network
  - Distribution of (part of) the database to edge servers
  - Edge servers perform query processing
- **Why?**
  - Most organizations need DBMSs
  - DBMSs extremely complex to deploy, setup, maintain
  - Require skilled DBAs (at very high cost!)
- Advantages
  - Cuts down network latency and produces faster responses
  - Cheaper way to achieve scalability
  - Lowers dependency on corporate data center (removes single point of failure)
  - Reduced cost to client
    - Get what you need, pay for what you use and not for: hardware, software infrastructure or personnel to deploy, maintain, upgrade…
  - Reduced overall cost
    - cost amortization across users
  - Better service
    - leveraging experts

The Challenge

DB Client



Owner

Query

Results

Third Party
Server
Untrusted!

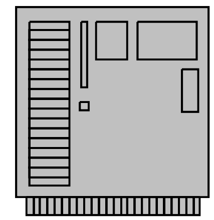
The Truth?
The Whole Truth?
Nothing But The Truth?

The Challenge

DB Client

Sel * FROM Emp
WHERE Sal < 5000

Owner



ID	Name	Sal	Dept
5	A	2000	1
2	C	3500	2
1	D	8010	1
4	B	2200	3
3	E	7000	2

Server

The Challenge

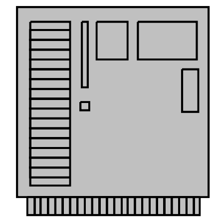
DB Client

Sel * FROM Emp
WHERE Sal < 5000

Result =

5	A	2000	1
2	C	3500	2
4	B	2200	3

Owner

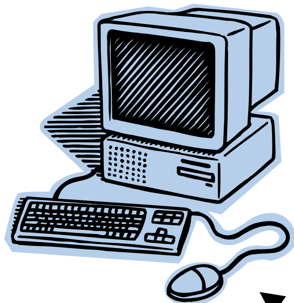


ID	**Name**	**Sal**	**Dept**
5	A	2000	1
2	C	3500	2
1	D	8010	1
4	B	2200	3
3	E	7000	2

Server

Security Concerns

DB Client



5	A	2000	1
2	C	3500	2
4	B	2200	3

Result =

Query

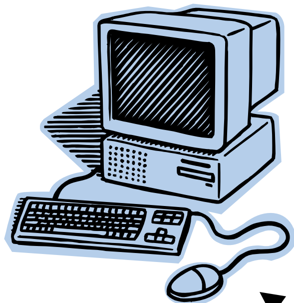
Result’

Server

ID	Name	Sal	Dept
5	A	2000	1
2	C	3500	2
1	D	8010	1
4	B	2200	3
3	E	7000	2

Security Concerns

DB Client



Result =

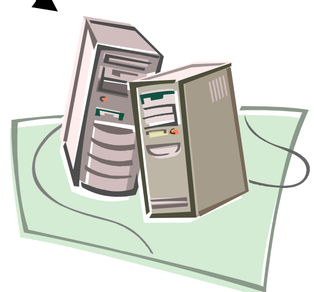
5	A	2000	1
2	C	3500	2
4	B	2200	3

Query

Result’

5	A	2000	1
2	C	3500	2
4	B	2200	3

Server is trustworthy!

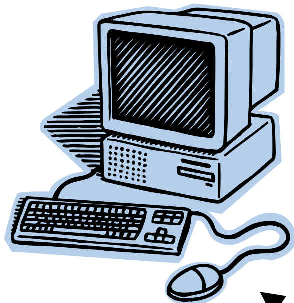


Server

ID	**Name**	**Sal**	**Dept**
5	A	2000	1
2	C	3500	2
1	D	8010	1
4	B	2200	3
3	E	7000	2

Security Concerns

DB Client



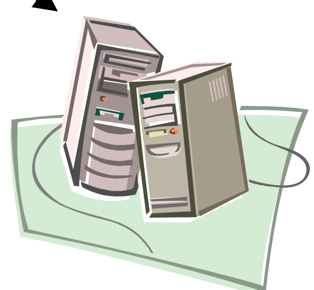
Result =

5	A	2000	1
2	C	3500	2
4	B	2200	3

Query

Result’

5	A	**3500**	1
2	**D**	3500	2
4	B	2200	**1**



Server

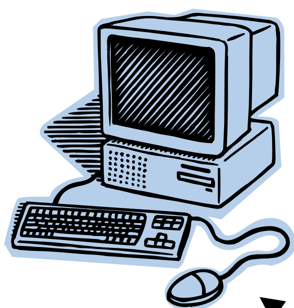
ID	**Name**	**Sal**	**Dept**
5	A	2000	1
2	C	3500	2
1	D	8010	1
4	B	2200	3
3	E	7000	2

Server is malicious!

Records are tampered

Security Concerns

DB Client



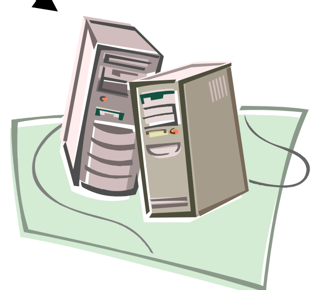
Result =

5	A	2000	1
2	C	3500	2
4	B	2200	3

Query

Result’

5	A	2000	1
2	C	3500	2



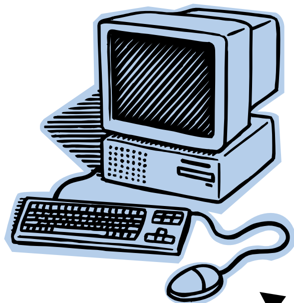
Server

ID	**Name**	**Sal**	**Dept**
5	A	2000	1
2	C	3500	2
1	D	8010	1
4	B	2200	3
3	E	7000	2

Server is malicious!
Answers are dropped
(Incompleteness)

Security Concerns

DB Client



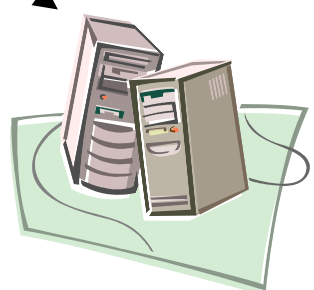
Result =

5	A	2000	1
2	C	3500	2
4	B	2200	3

Query

Result'

5	A	2000	1
2	C	3500	2
4	B	2200	3
1	**D**	**1500**	**2**
6	**E**	**3400**	**1**



Server

ID	**Name**	**Sal**	**Dept**
5	A	2000	1
2	C	3500	2
1	D	8010	1
4	B	2200	3
3	E	7000	2

Server is malicious!

Spurious answers are added

Data Security Challenge:

Design Objectives:

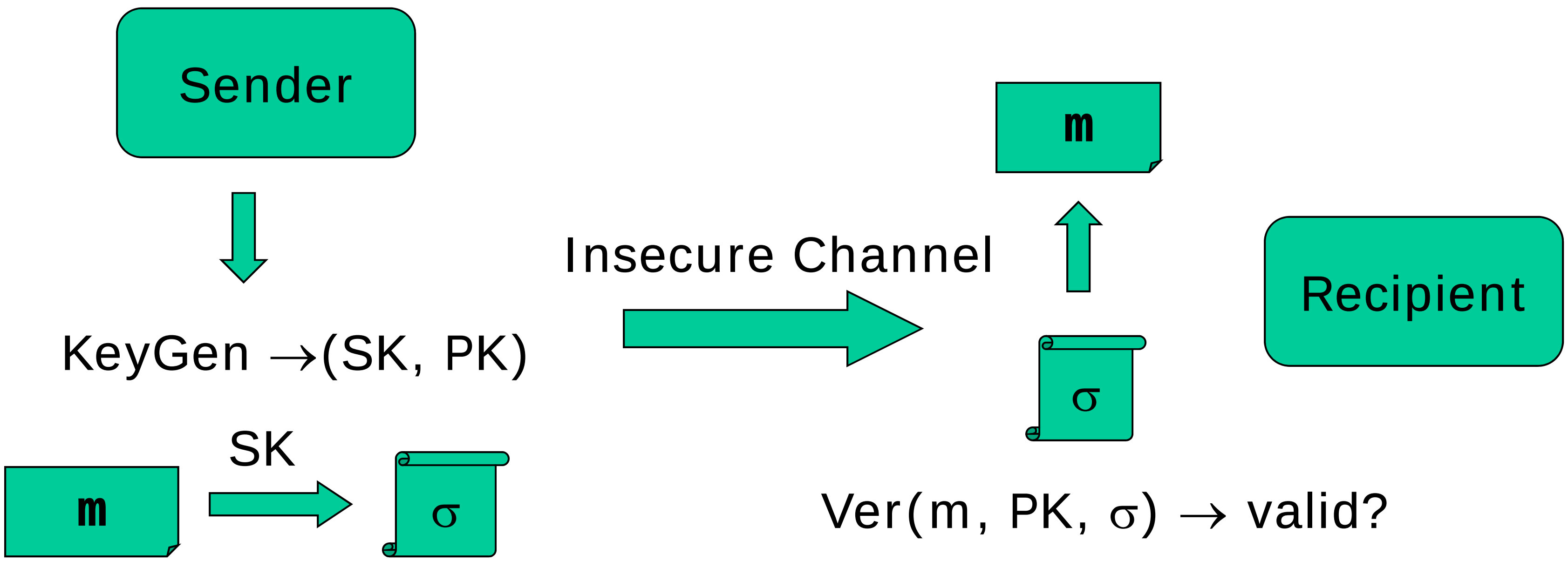
- *Authenticity*: Every entry originated from the owner
- *Completeness*: No result entry is omitted from the answer
- *Precision*: Minimum information leakage
- *Security*: Computationally infeasible to cheat
- *Efficiency*: Polynomial proof

Collision-resistant (one-way) hash functions

- Given x, easy to compute h(x); given h(x), difficult to determine x
- i.e., it is computationally hard to find x_1 and x_2 s.t. $h(x_1)=h(x_2)$
- Computational hard? Based on well established assumptions such as discrete logarithms
- E.g., SHA, MD5

Public key digital signature schemes

Cryptographic tool for authenticating the signed message as well as its origin, e.g., RSA, DSA



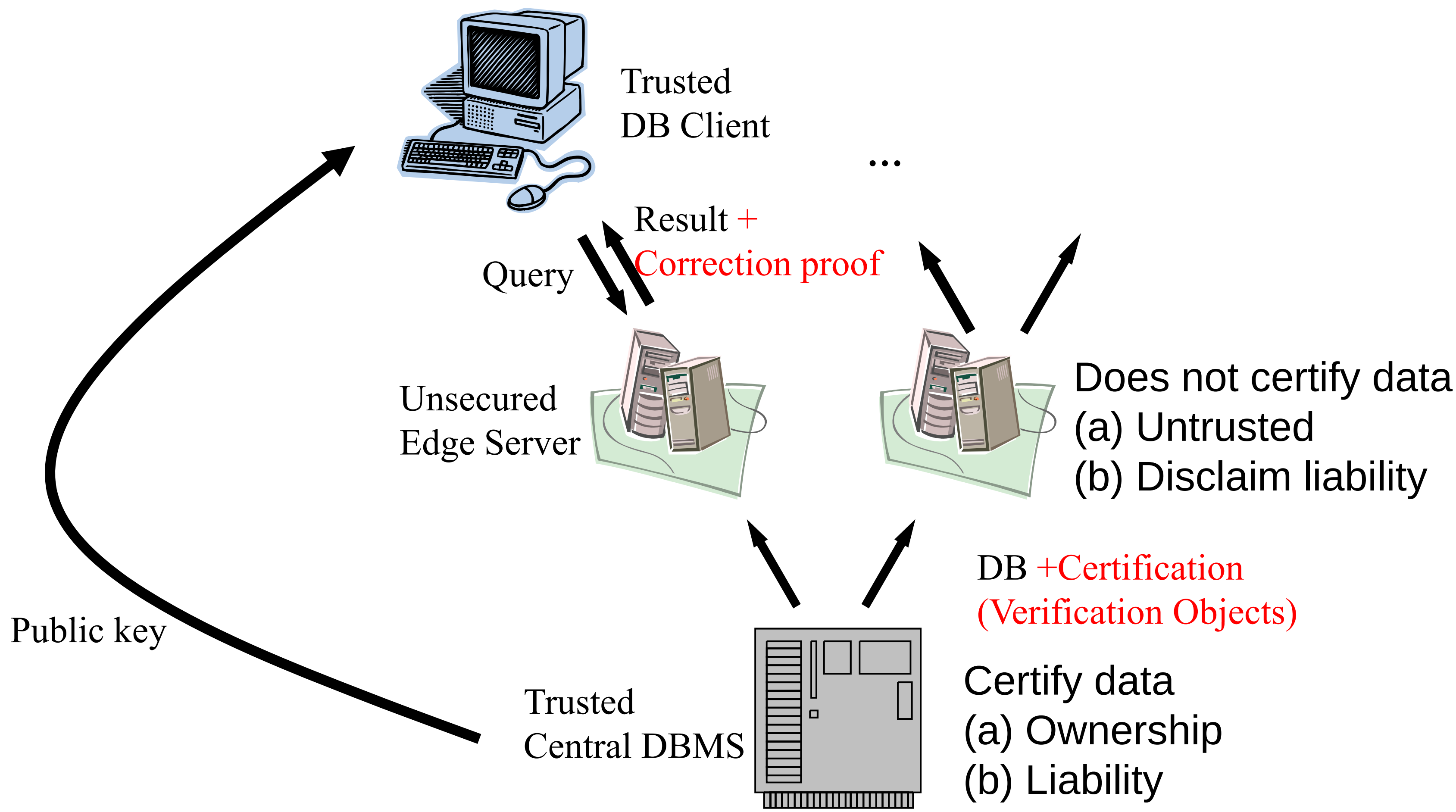
Sign(h(m), SK) → σ

Ver(m, PK, σ) → valid?

By checking:

$h(m) =? Sign^{-1}(PK, \sigma)$

Authentic Publication Scheme



Naïve Scheme

Each attribute has a signed digest
Each tuple has a signed digest

Relation R

D_T	(A_1, D_1)	…	(A_i, D_i)		…

D_T – Signed tuple digest
D_{Ai} – attribute digest

Naïve Scheme

Query: SELECT A_3, A_4, … FROM R

Result tuples

Filtered attributes

D_T	A_3	A_4	…	D_1	D_2	D_5	…

D_T – Signed tuple digest
D_i – attribute digest of A_i

Naïve Scheme (Example)

A1	B1	C1	a1	b1	c1	T1
A2	B2	C2	a2	b2	c2	T2
A3	B3	C3	a3	b3	c3	T3

T = sign(g(h(A)|h(B)|h(C))

g and h are collision-resistant hash functions

ai = h(Ai)

Retrieve whole of first tuple:

Server returns A1, B1, C1, T1; Client can compute h(A1), h(B1) and h(C1), and verify T1 from A1, B1 and C1

Retrieve only attributes A1 and B1 of first tuple:

Server returns A1, B1, c1 and T1; Client has no access to C1, so c1 has to be provided

Issues??

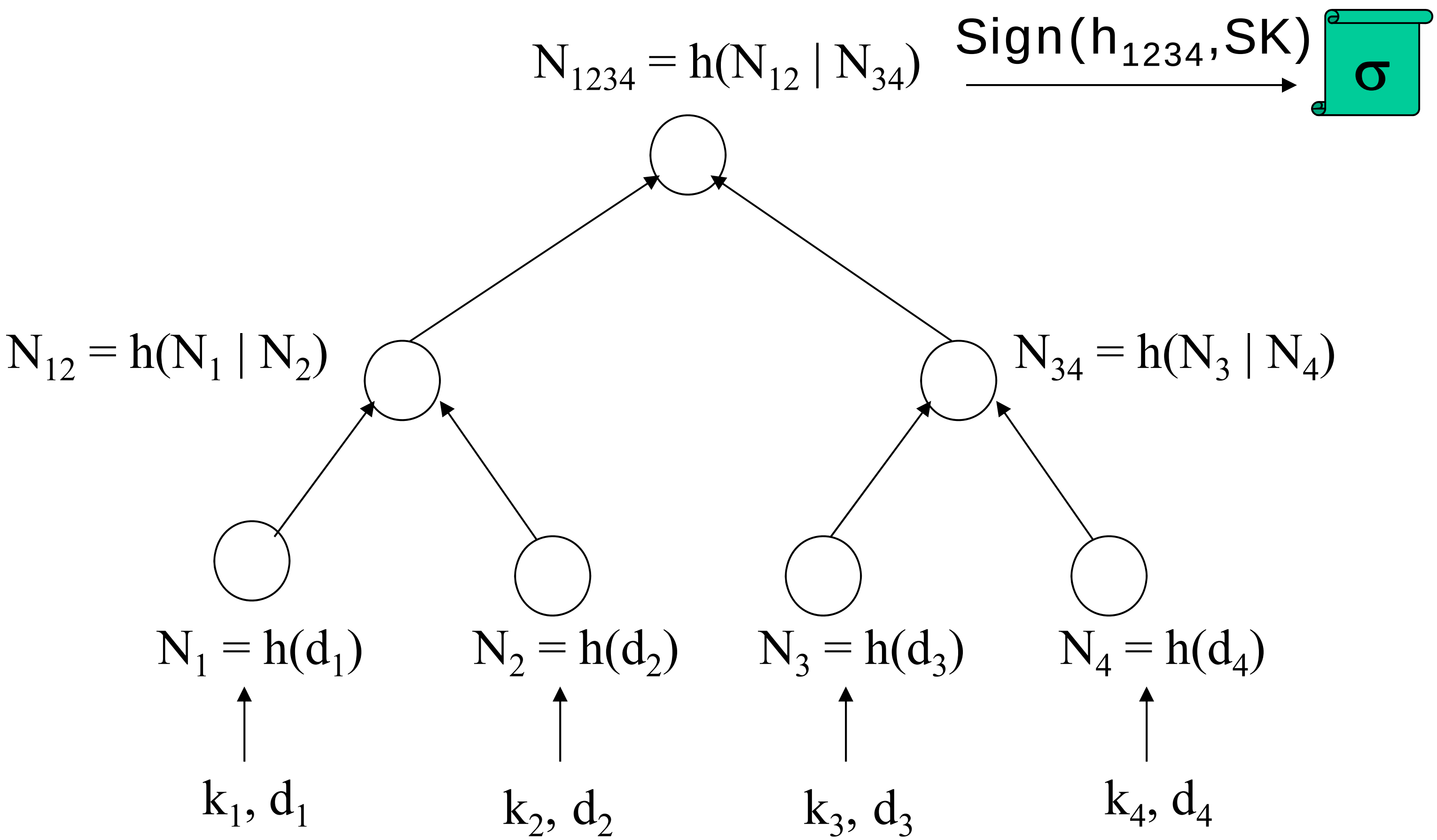
Using Merke Hash Tree (MHT)

- For each tuple t, a tuple hash h(t) is computed

$$h(t) = h(h(t.A1) \mid h(t.A2) \mid \dots \mid h(t.An))$$

- Assume a total order on attribute A of a relation R with |R| tuples (e.g., based on the primary key)
  - MHT(R,A) is a binary tree with |R| leaf nodes and hash values h(i) associated with node i
  - If i is a leaf node, then h(i) = h(ti), ti is the ith tuple in the order
  - If i is an internal node, then h(i) = h(h(l), h(r)) where l and r are the left and right children of node i.
  - The root hash is the digest of all values in the Merkle-hash tree MHT(R,A).

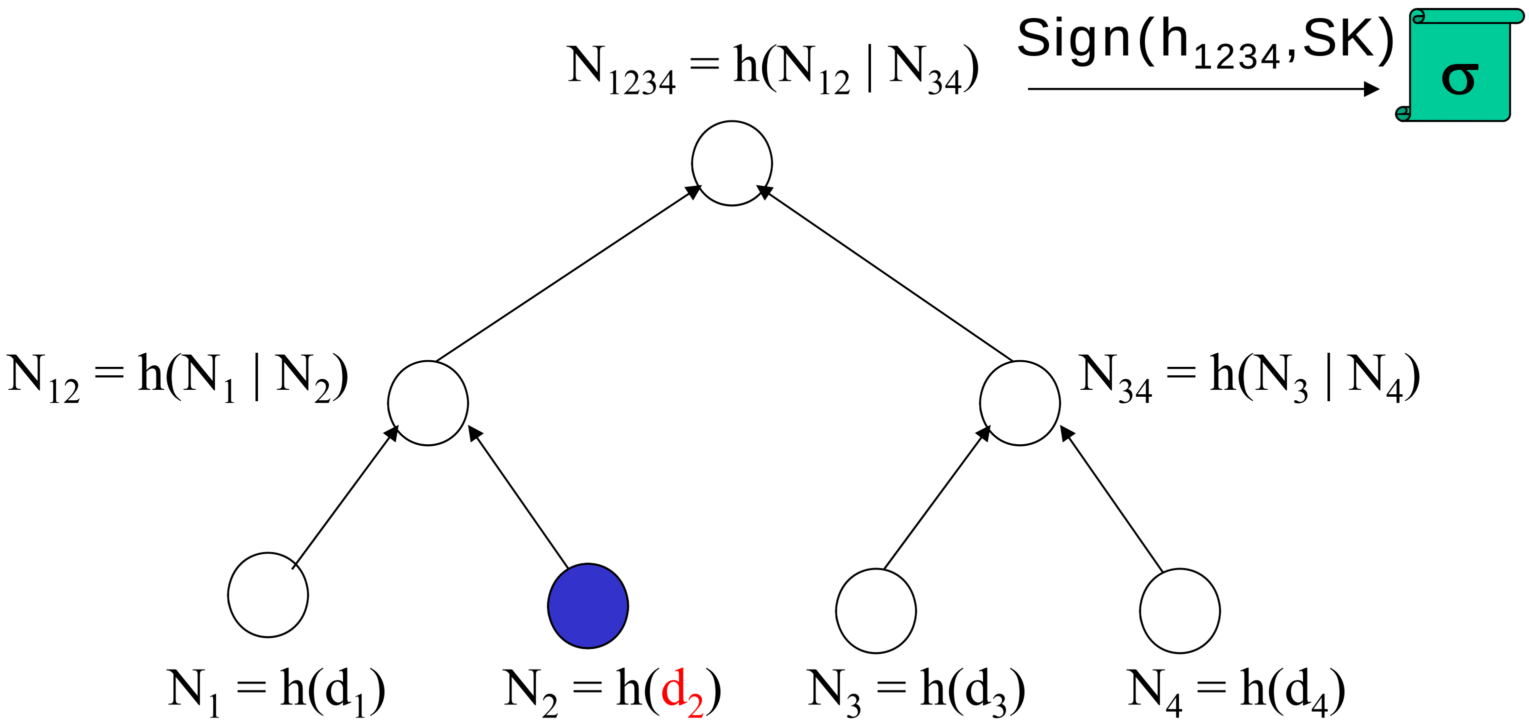
Merkle Hash Tree



Ordering attribute: $k_1 < k_2 < k_3 < k_4$; d_i are tuples

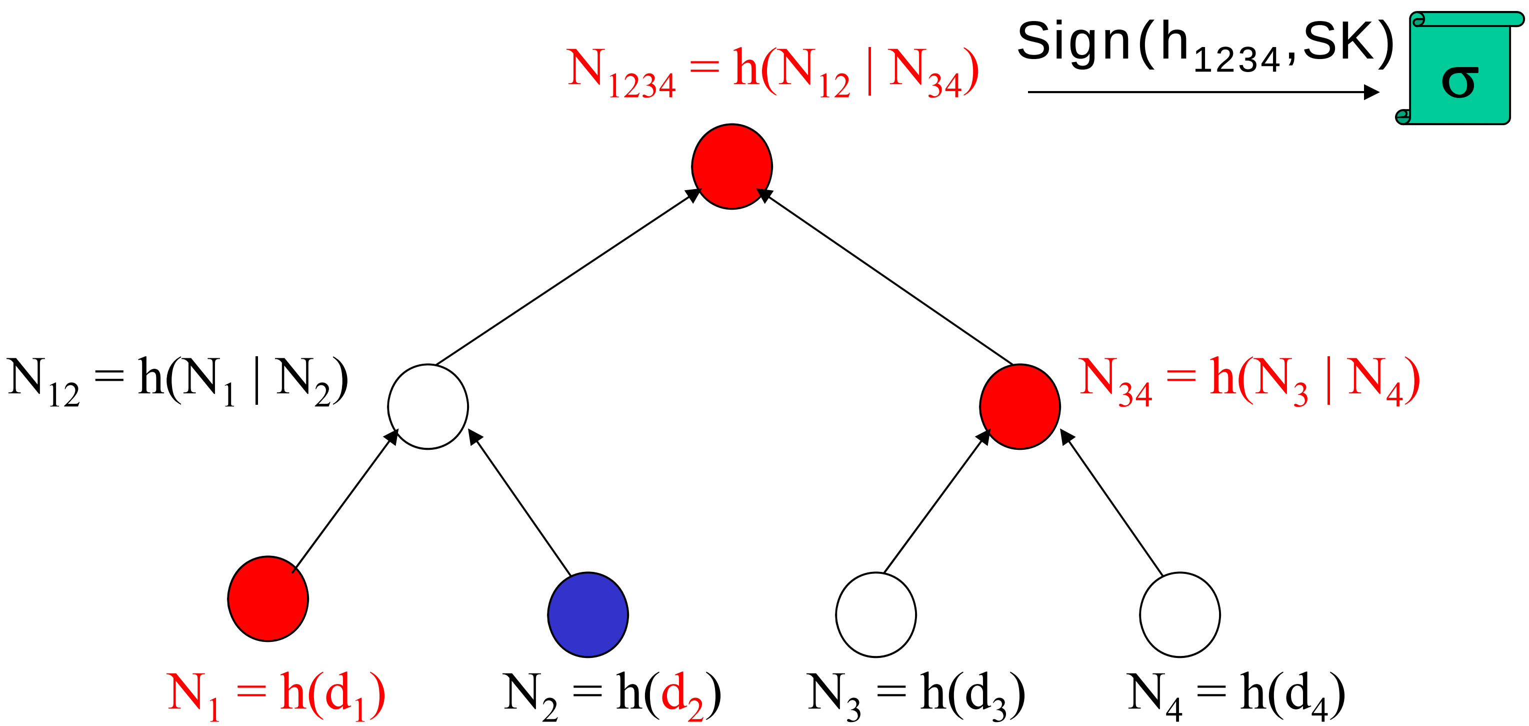
Owner needs to sign root node (N_{1234})

MHT: Point Search



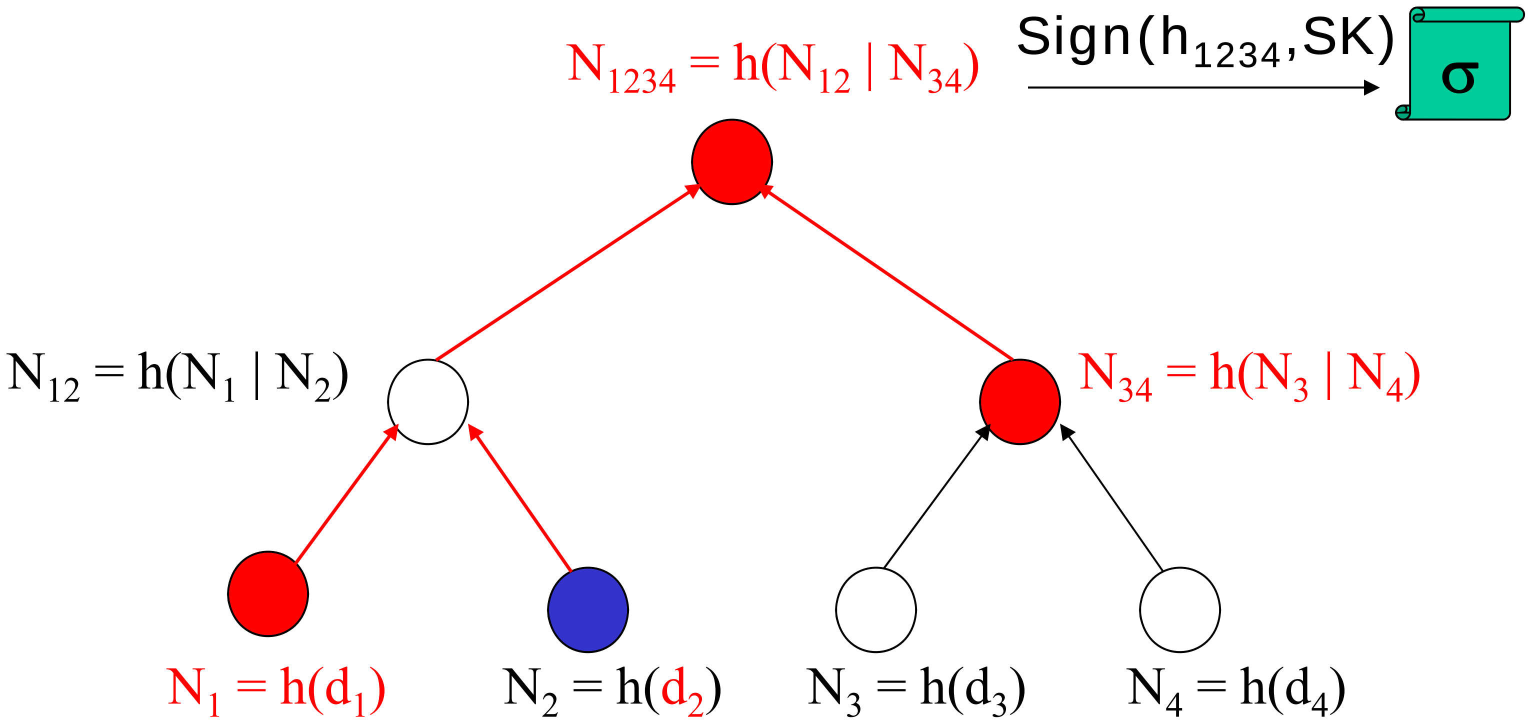
Query: Retrieve tuple d_2

MHT: Point Search



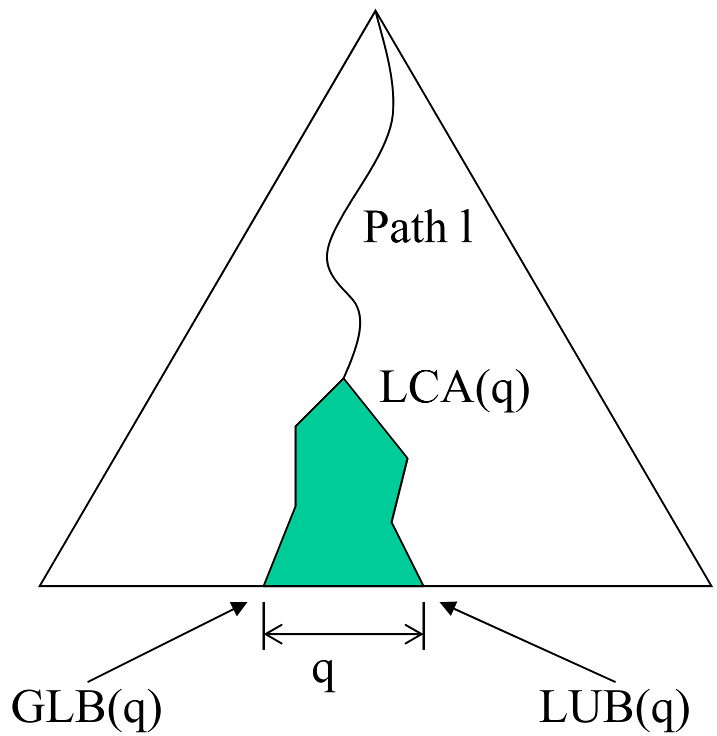
Edge server returns d_2, N_1, N_{34} and signed N_{1234}
Client computes $N_{1234} = h(h(h(d_2)|N_1), N_{34})$ and verify that the signed value is correct

MHT: Point Search

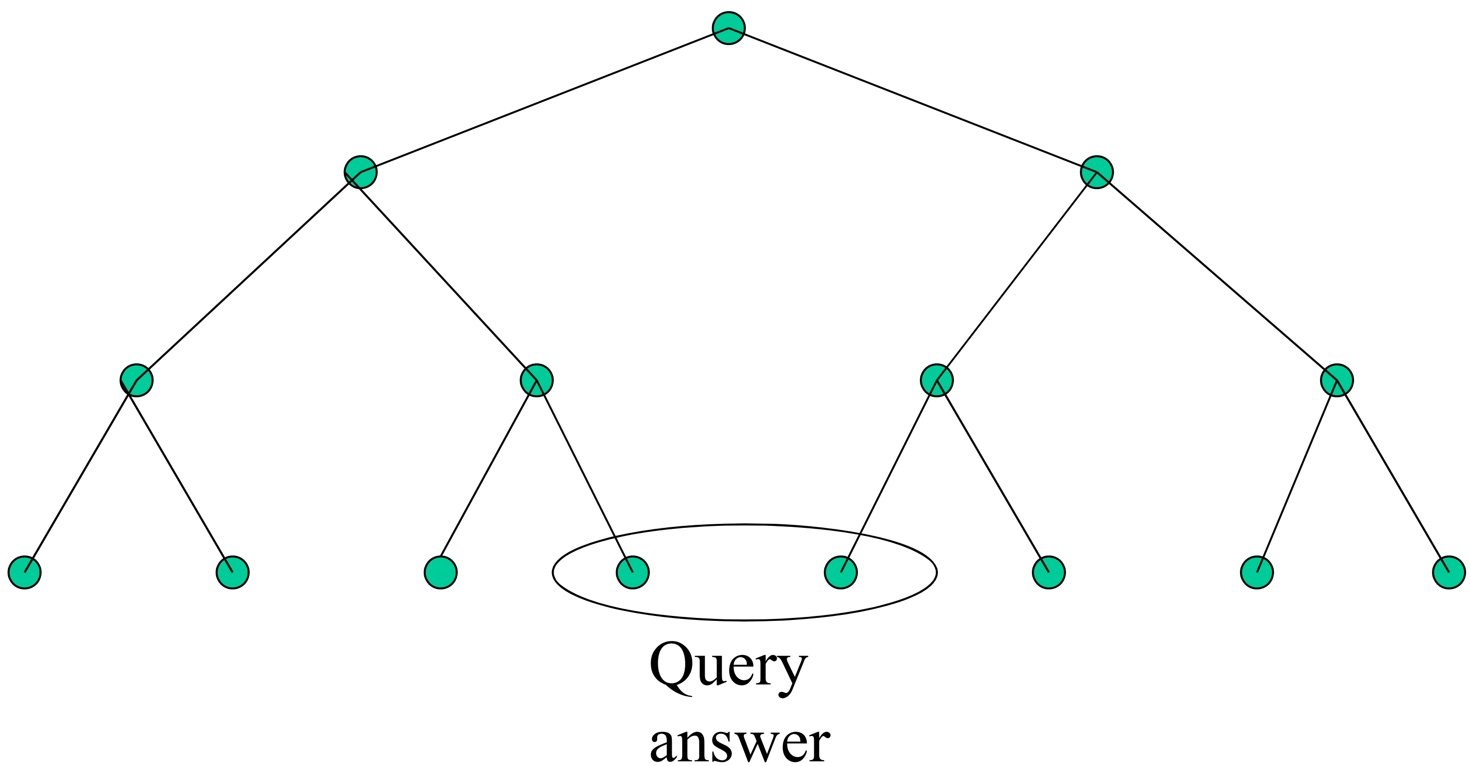


Edge server returns d_2, N_1, N_{34} and signed N_{1234} (and the structure)
Client computes $N_{1234} = h(h(h(d_2)|N_1), N_{34})$ and verify that the signed value is correct

Range Queries

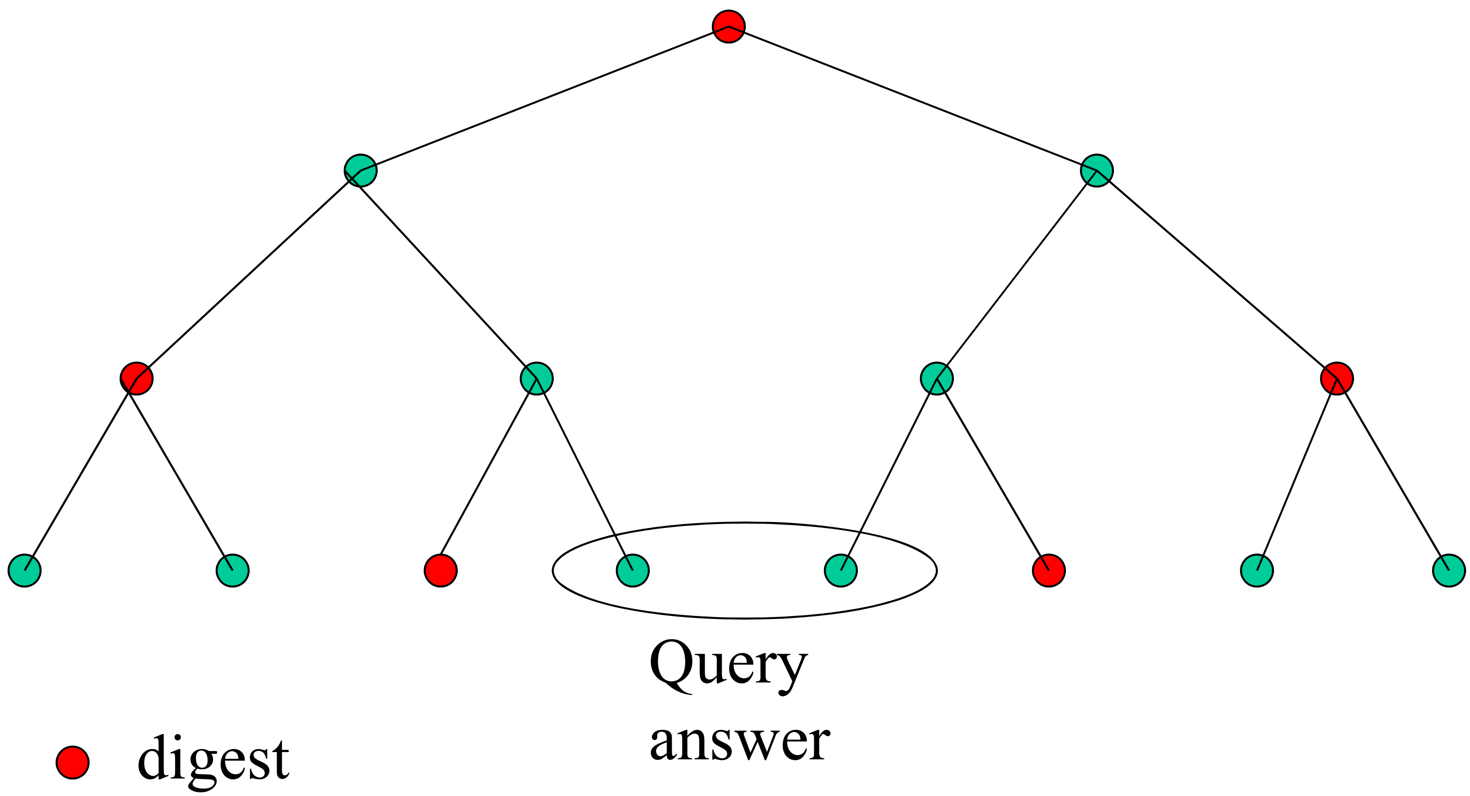


Example: Range queries



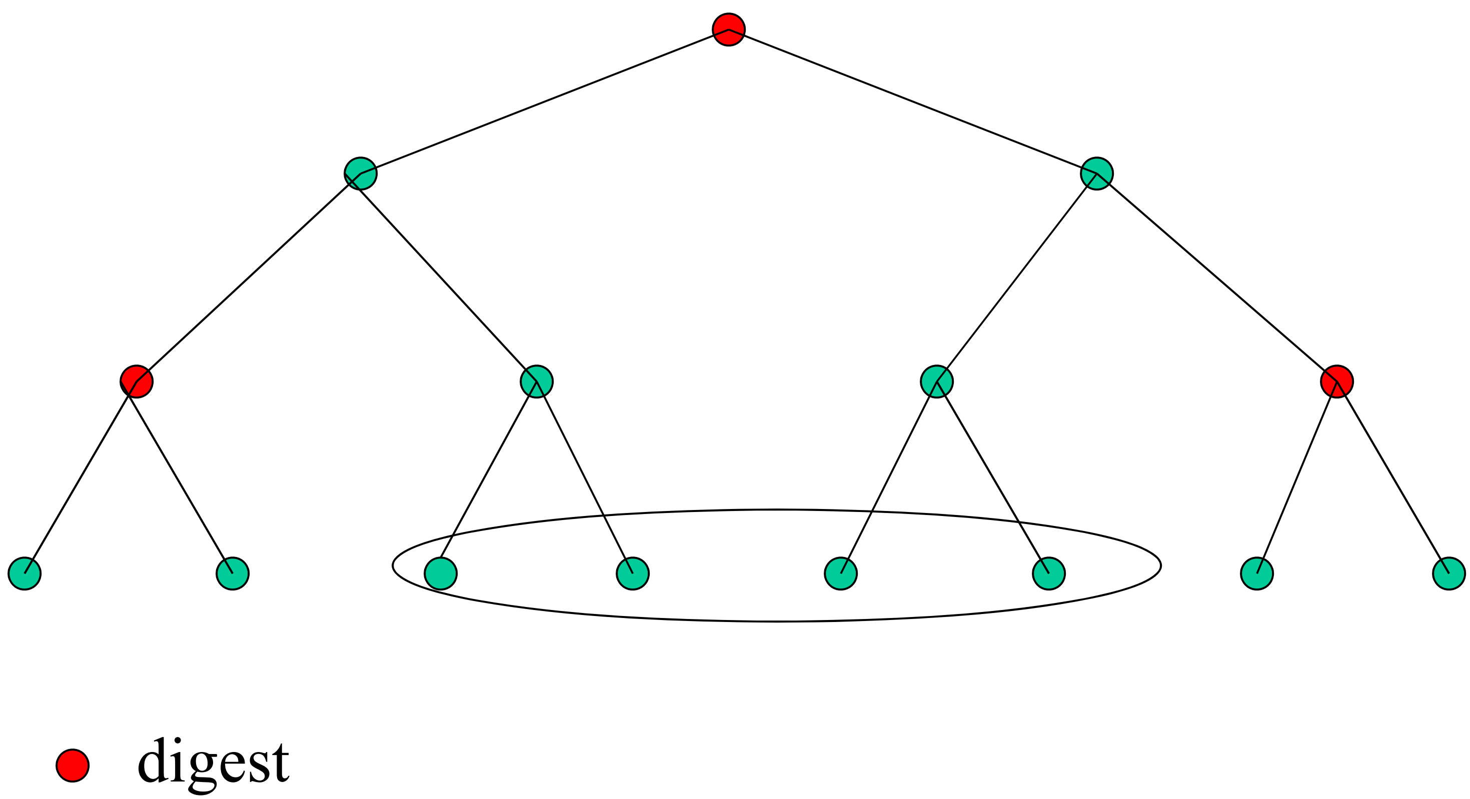
What are returned?

Example: Range queries



What are returned?

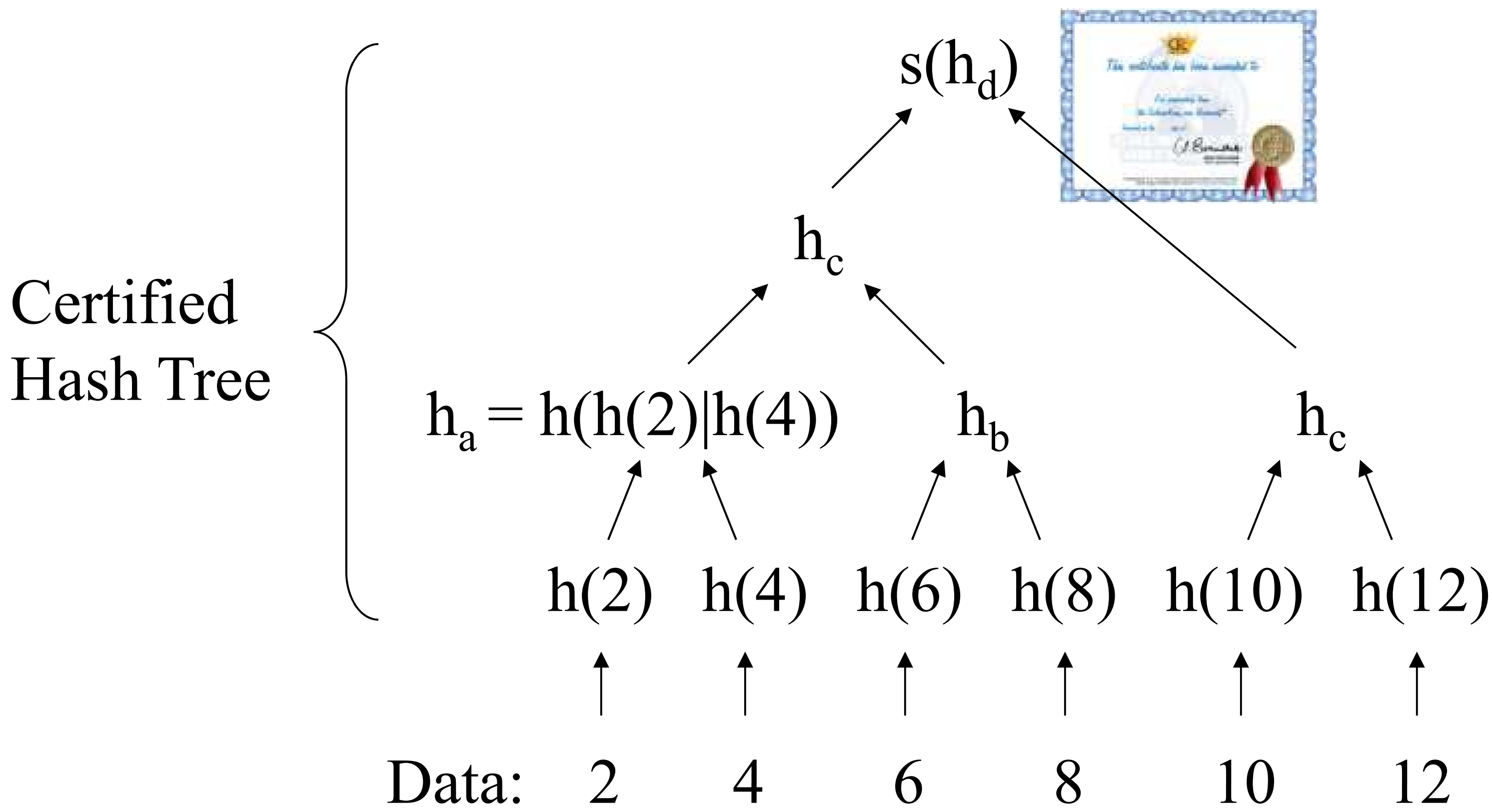
Example: Range queries



digest

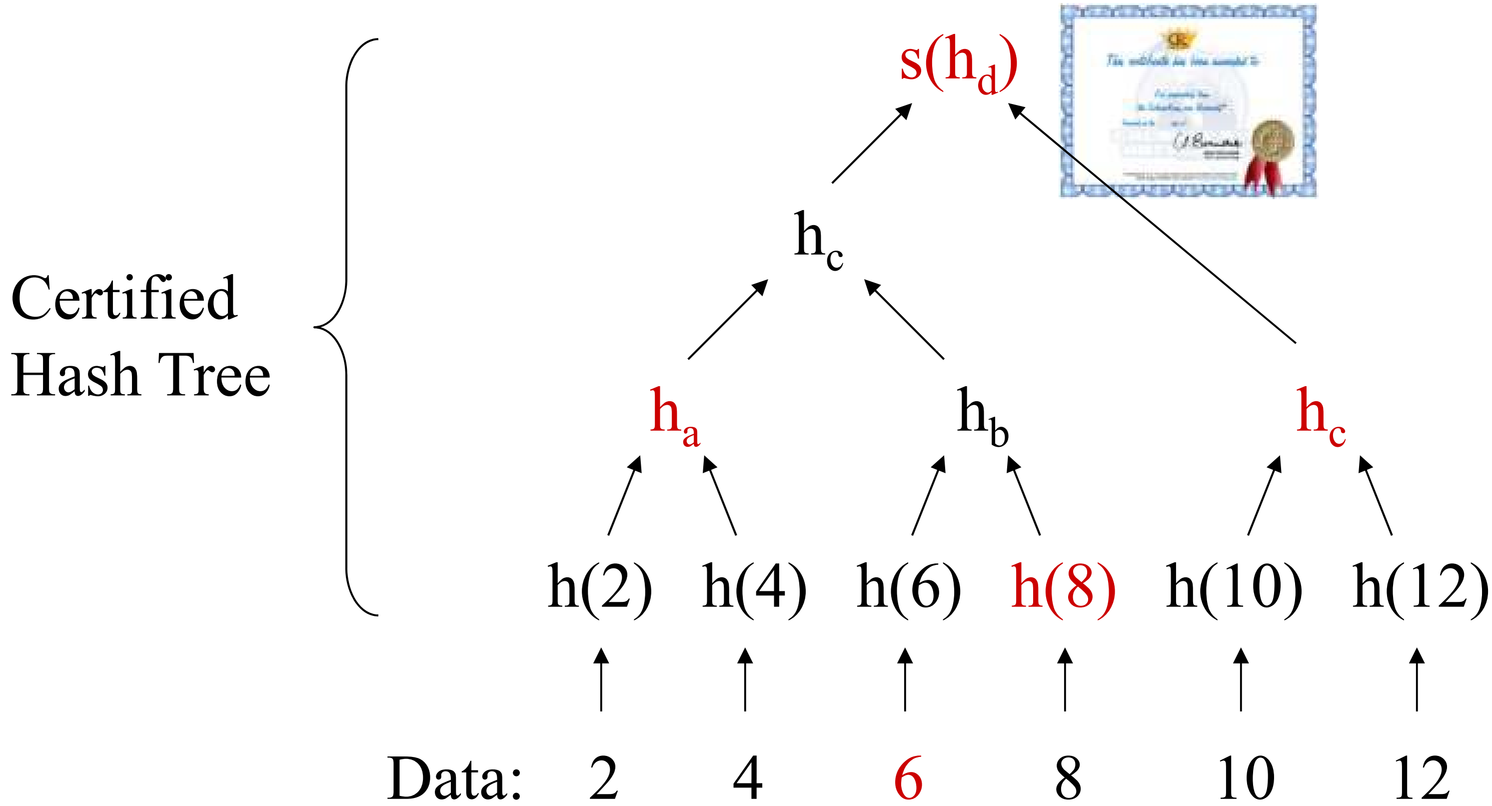
What are returned?

Proving Authenticity is Easy



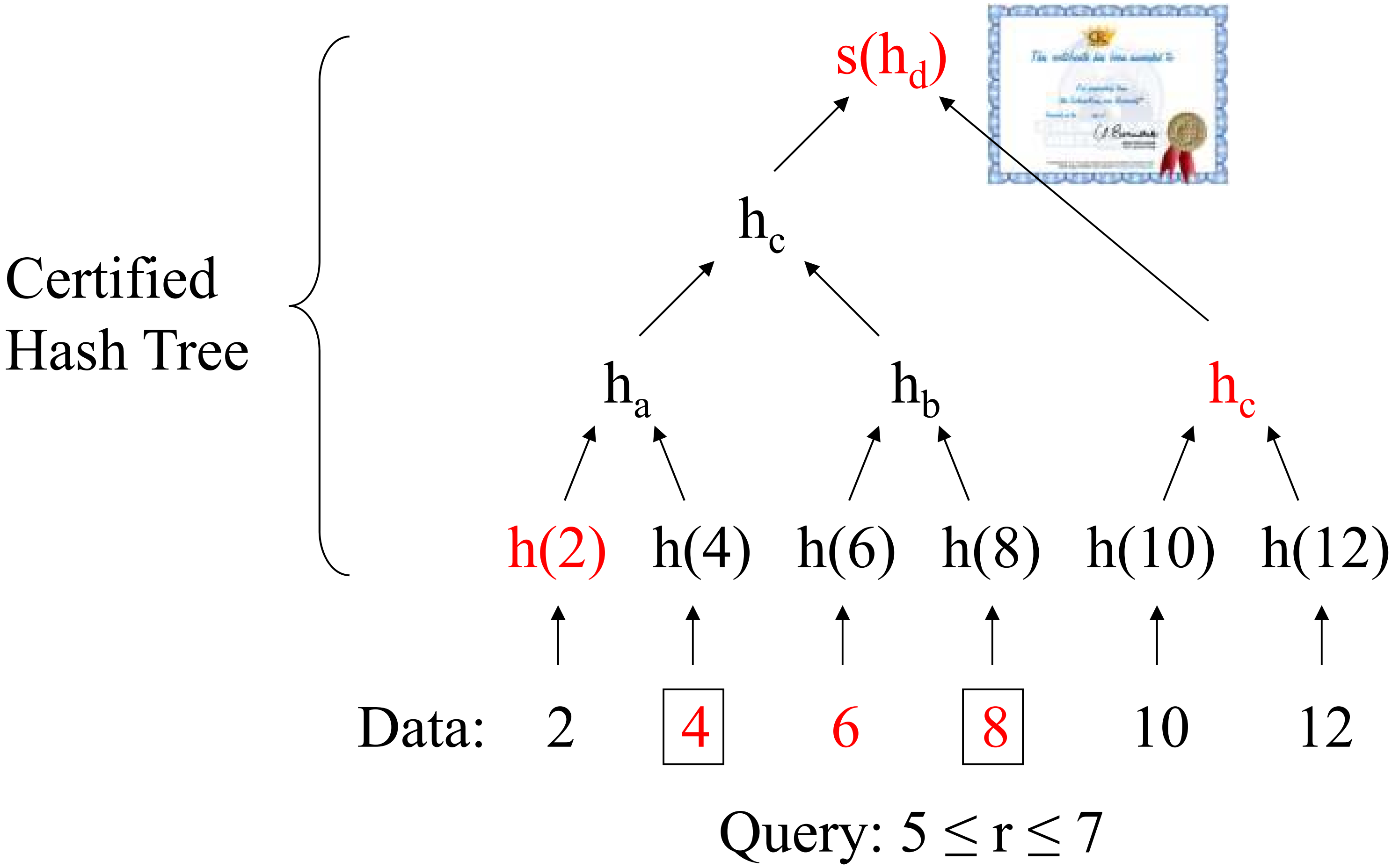
Query: $5 \leq r \leq 7$

Proving Authenticity is Easy

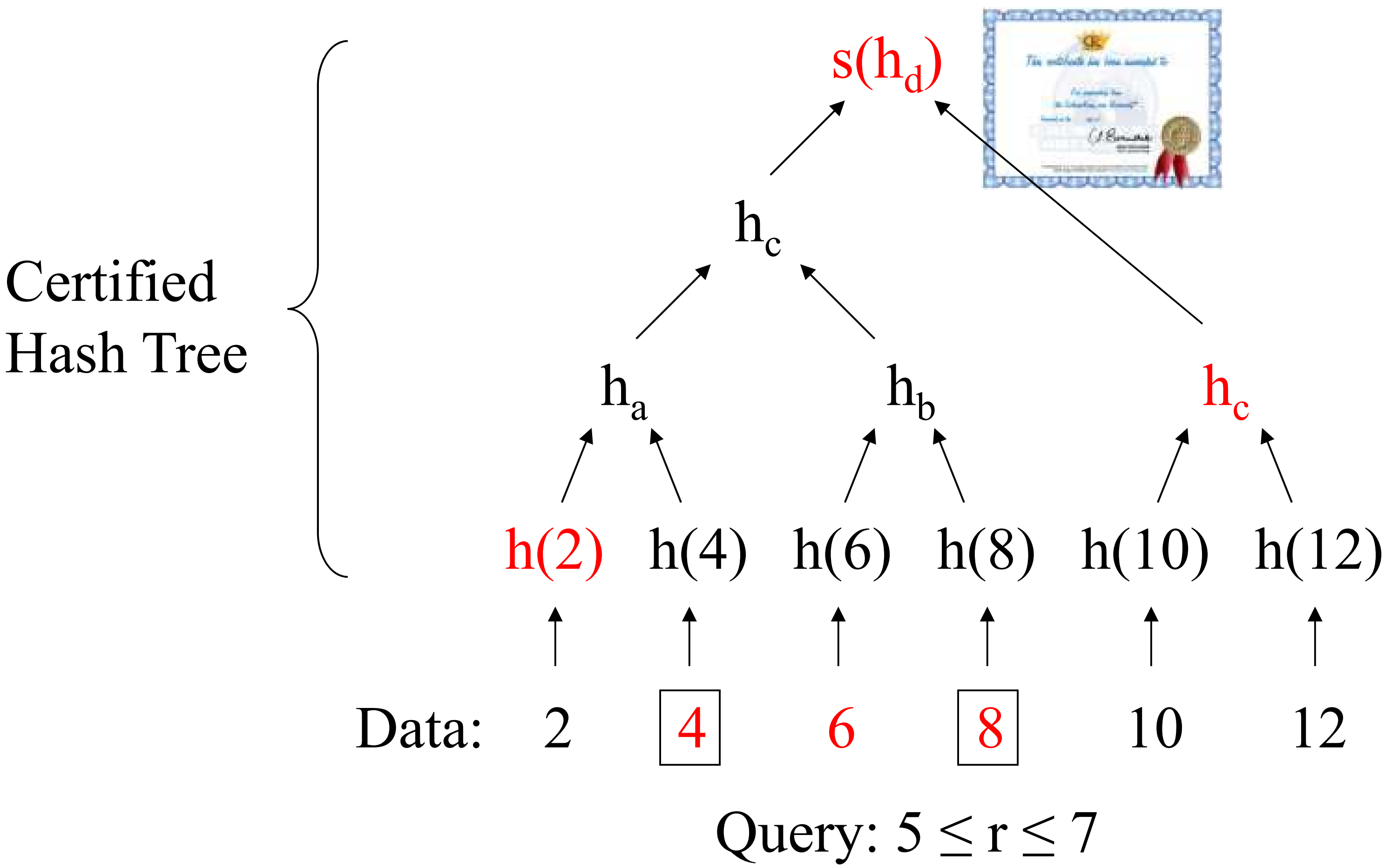


Query: $5 \leq r \leq 7$

Proving Completeness is Easy But …



Precision may be compromised!



- Compromise precision: Disclose left and right neighbors
- May violate access control policy

Example

- Access control: U can only see records with salary < 8000
- Results are records 2, 3, and 5.
- If system does not return record 1, U will not know that the answer is complete since it is possible that there is a record with Sal > 7000 but < 8000 that is not returned.
- If system returns record 1, then it violates the access control policy!
- Need an authentication mechanism that verifies completeness without compromising access control rules

ID	Name	Sal	Dept
5	**A**	**2000**	**1**
2	**C**	**3500**	**2**
1	D	8010	1
4	B	2200	3
3	**E**	**7000**	**2**

What's the problem?

- A Merkle hash tree is needed for every sort-order on a table
- VO (Verification Object – the data used for verification) needs to contain links all the way to the root,
  - VO grows linearly to query result and logarithmic to base table size
- Projections may have to be performed by clients
- No provision for dynamic updates on the database
- Weak in terms of access control
  - Attributes that are supposed to be filtered out must also be returned for verification

A signature-chain-based scheme: Let's start simple …

- Consider a sorted list of distinct integers, $R = \{r_1 \ldots, r_{i-1}, r_i, r_{i+1}, \ldots r_n\}$
- Retrieve record whose value is greater than or equal to α
  - $\alpha \leq r$ (i.e., $\sigma_{\alpha \leq r}(R)$)
- Result $Q = \{r_a, r_{a+1}, \ldots r_b\}$, i.e., $r_{a-1} < \alpha \leq r_a < r_{a+1} < \ldots r_{b} = r_n$
- Result is complete iff:
  - Contiguity: Each pair of successive entries r_i, r_{i+1} in Q also appears in R (based on Signature Chain)
  - Terminal: Last element of Q is also last element of R, i.e., $r_b = r_n$ (based on Signature Chain)
  - Origin: r_a is the first element in R that satisfies the query condition, i.e.,$r_{a-1} < \alpha \leq r_a$ (based on Private Boundary Proof)

Signature Chain

- For each data value, there is an associated signature
  - Computed from its own value, and that of its left and right neighbors
  - $sig(r_i) = s(h(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1})))$

$$\cdots \; r_{i-1} \; r_i \; r_{i+1} \; r_{i+2} \; \cdots$$

- Owner stores the (r_i, $sig(r_i)$) pair in the server
- During querying, server returns (answer, signature) pairs and more …(verification objects) …

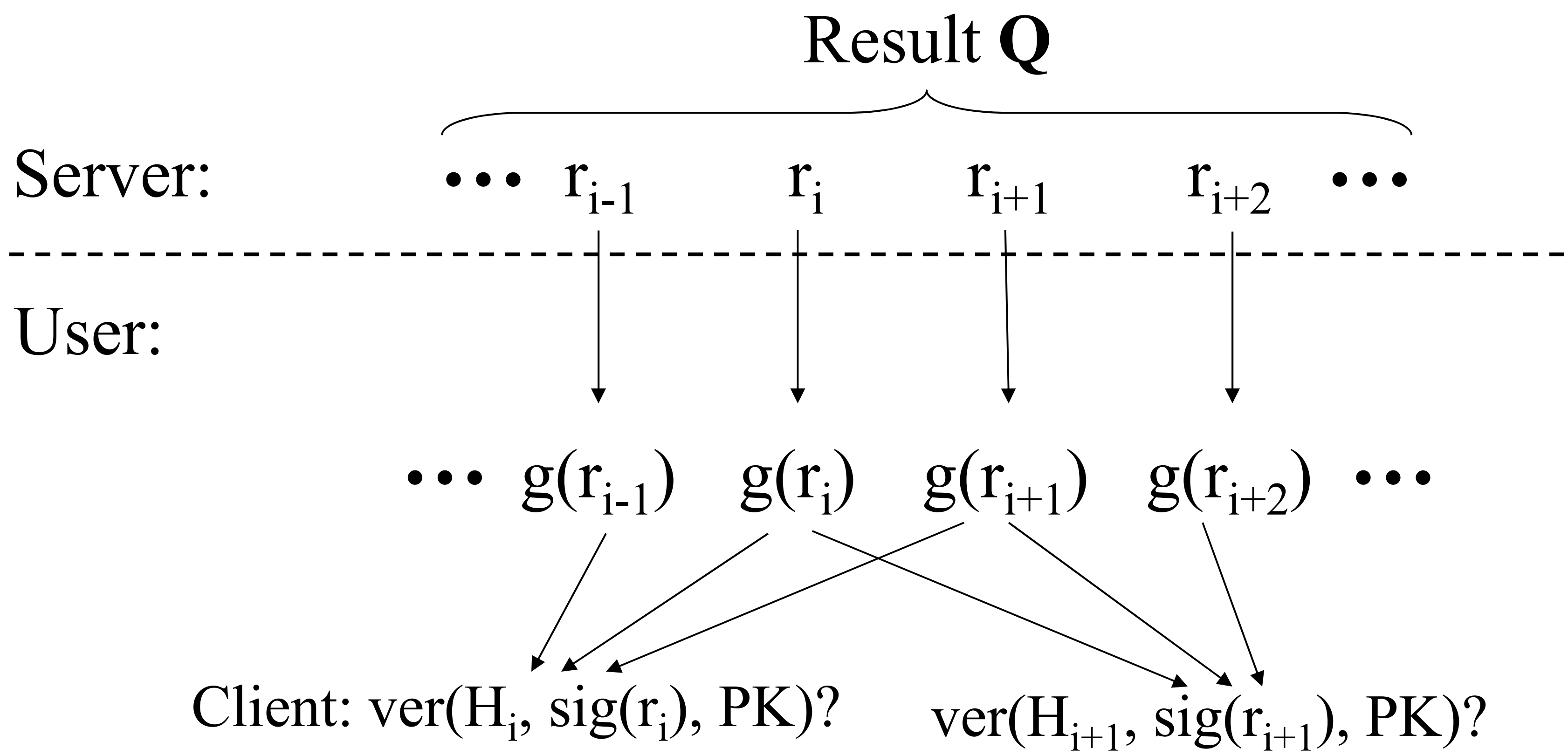
$h^i(r) = h^{i-1}(h(r))$ $\quad$ $h^0(r) = h(r)$ $\quad$ $g(r) = h^{U-r-1}(r)$

U = max value outside of domain (known to all users)

s is a signature function using owner's private key

Signature Chain Ensures Contiguity

Server returns (r_i , $sig(r_i)$)-pairs

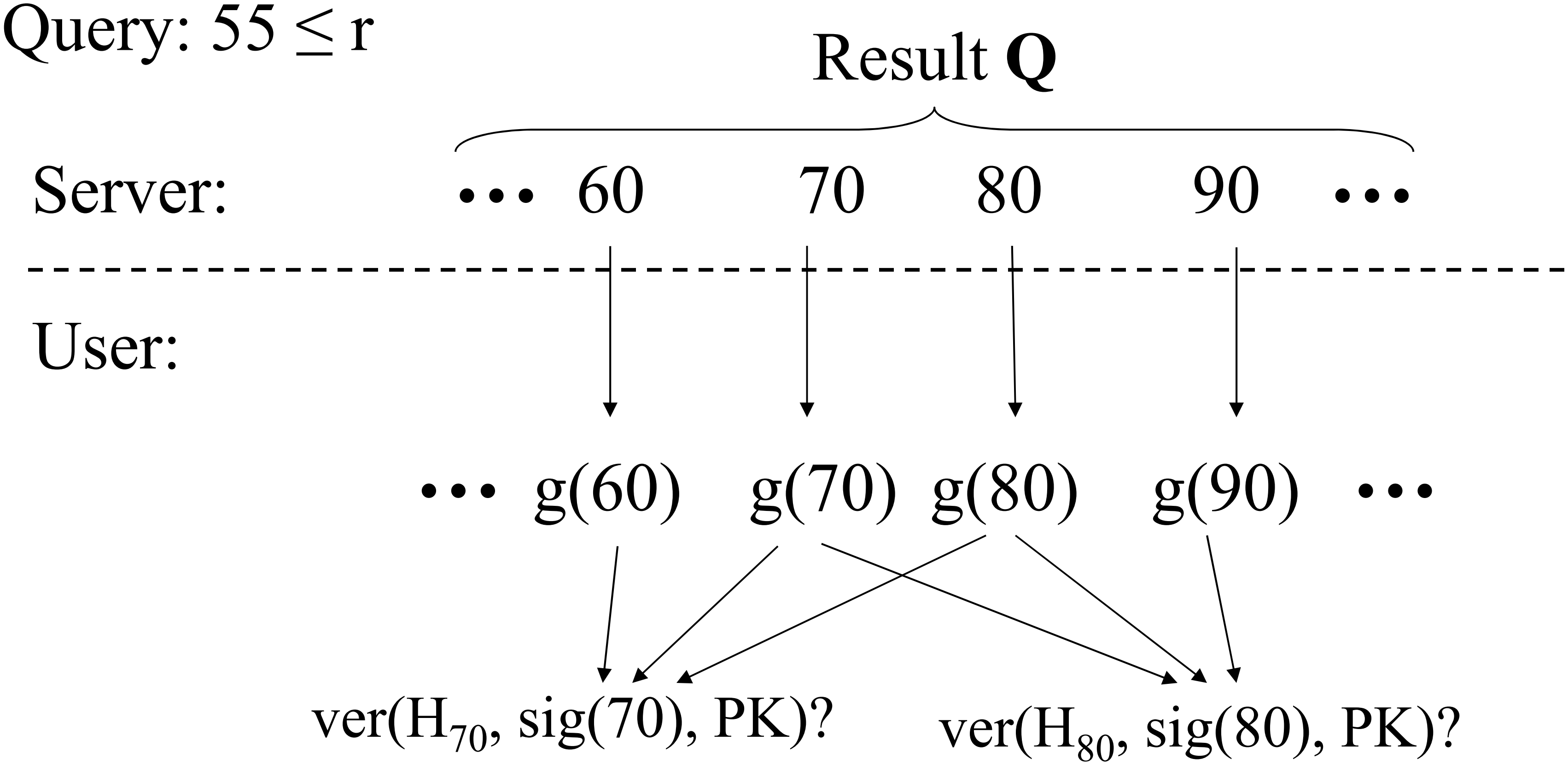


$H_i = h(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1}))$

Signature chain: $sig(r_i) = s(h(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1})))$

Signature Chain Ensures Contiguity

Query: $55 \leq r$



Signature Chain Ensures Contiguity

Query: $55 \leq r$

Result **Q**

Server: ••• 60 75 80 90 •••

User:

••• g(60) g(75) g(80) g(90) •••

ver(H_{75}, sig(70), PK)?

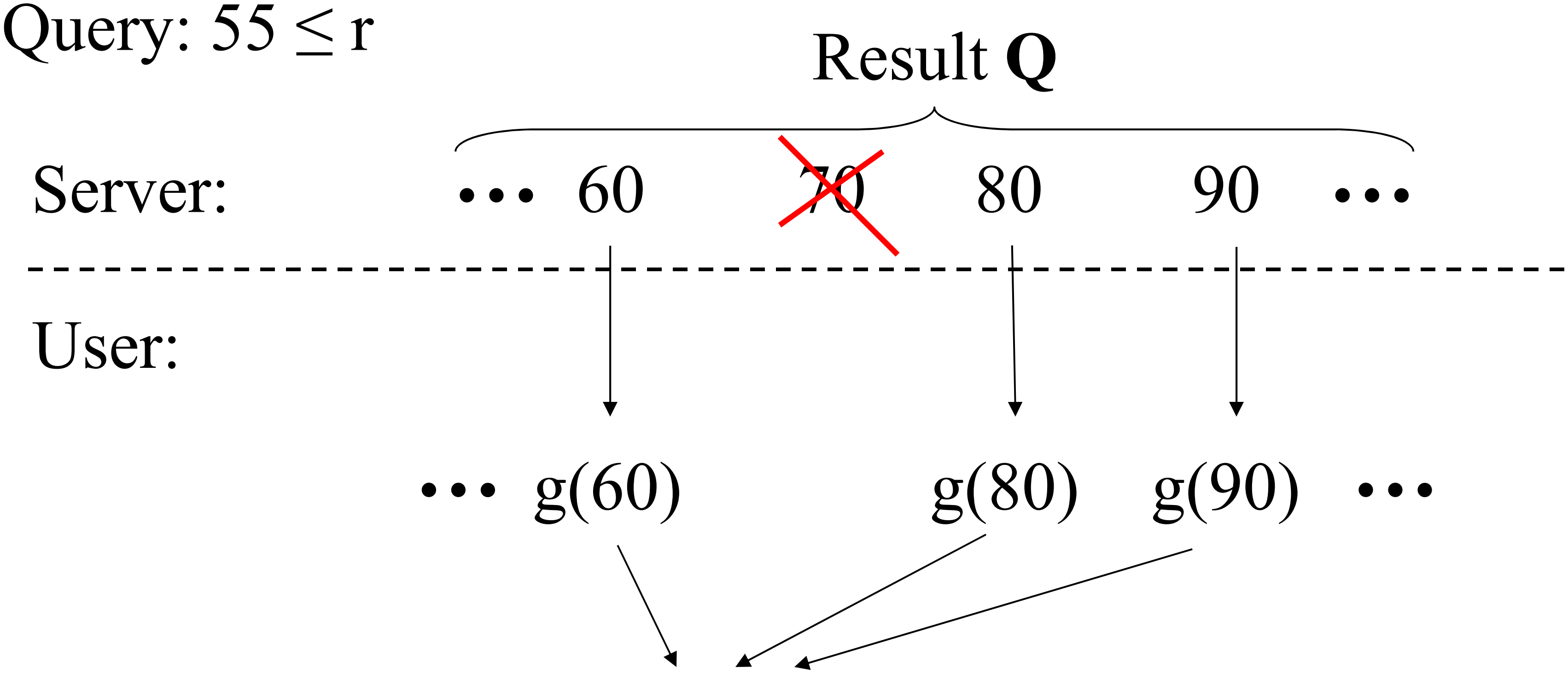
INCORRECT!

ver($H_{80\sim75}$, sig(80), PK)?

Data has been tampered!

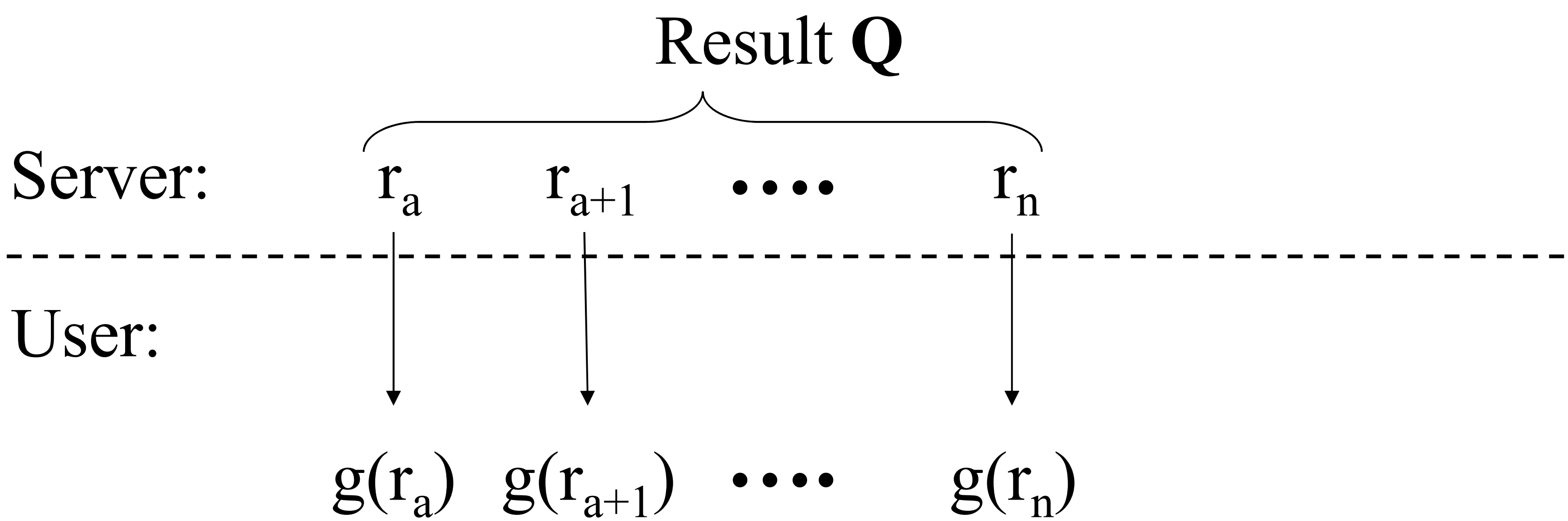
Signature Chain Ensures Contiguity

Query: $55 \leq r$



ver(H_{80}, sig(80), PK)?
$H_{60\sim80}$ will be computed (without 70) -
will not match sig(80). INCORRECT!!!!

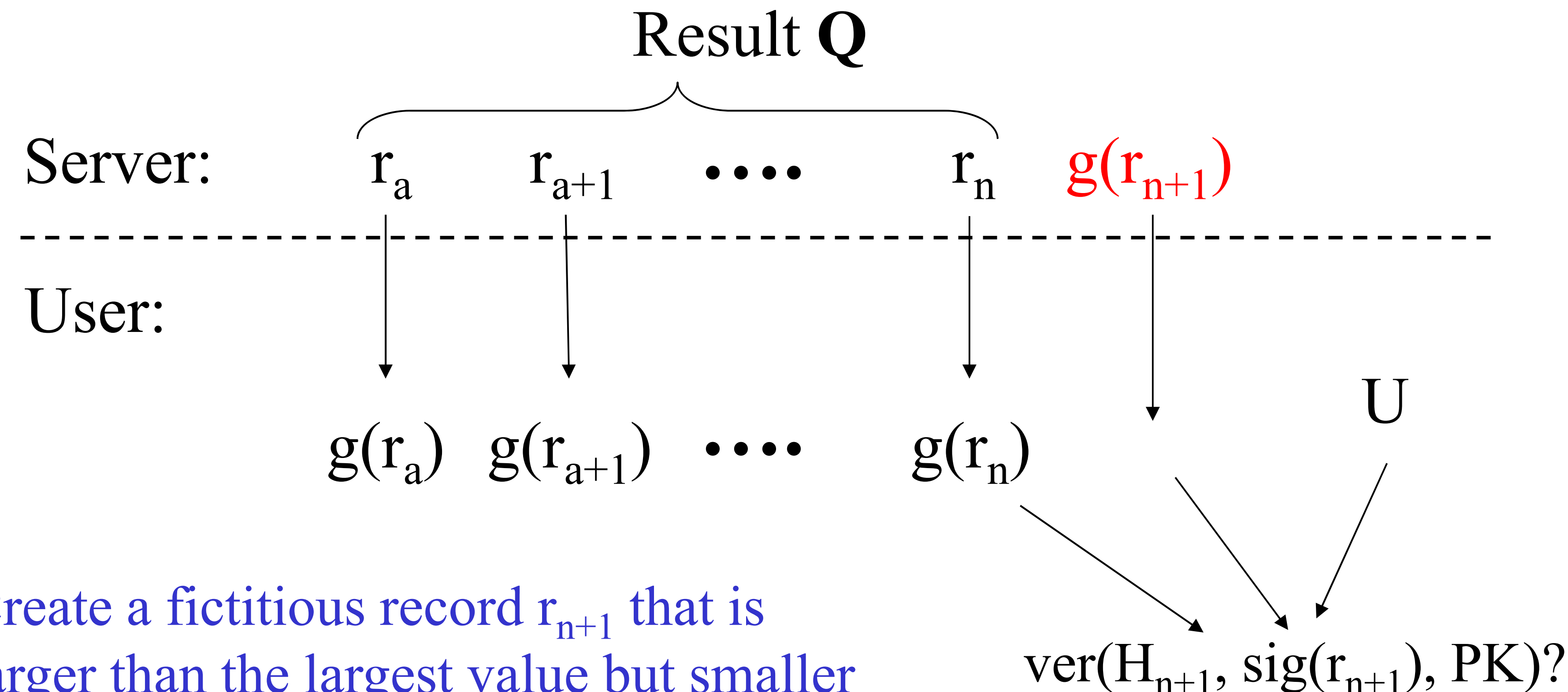
How To Ensure r_n Is The Last Record?



Create a fictitious record r_{n+1} that is larger than the largest value but smaller than U

- $sig(r_{n+1}) = s(h(g(r_n)|g(r_{n+1})|h(U)))$

Signature Chain Ensures Terminal



Create a fictitious record r_{n+1} that is larger than the largest value but smaller than U

- $sig(r_{n+1}) = s(h(g(r_n)|g(r_{n+1})|h(U)))$
- server returns $g(r_{n+1})$ instead of r_{n+1}

How to prove Origin (without revealing the boundary point)??

40 50 60 70 80 ….

How to prove Origin??

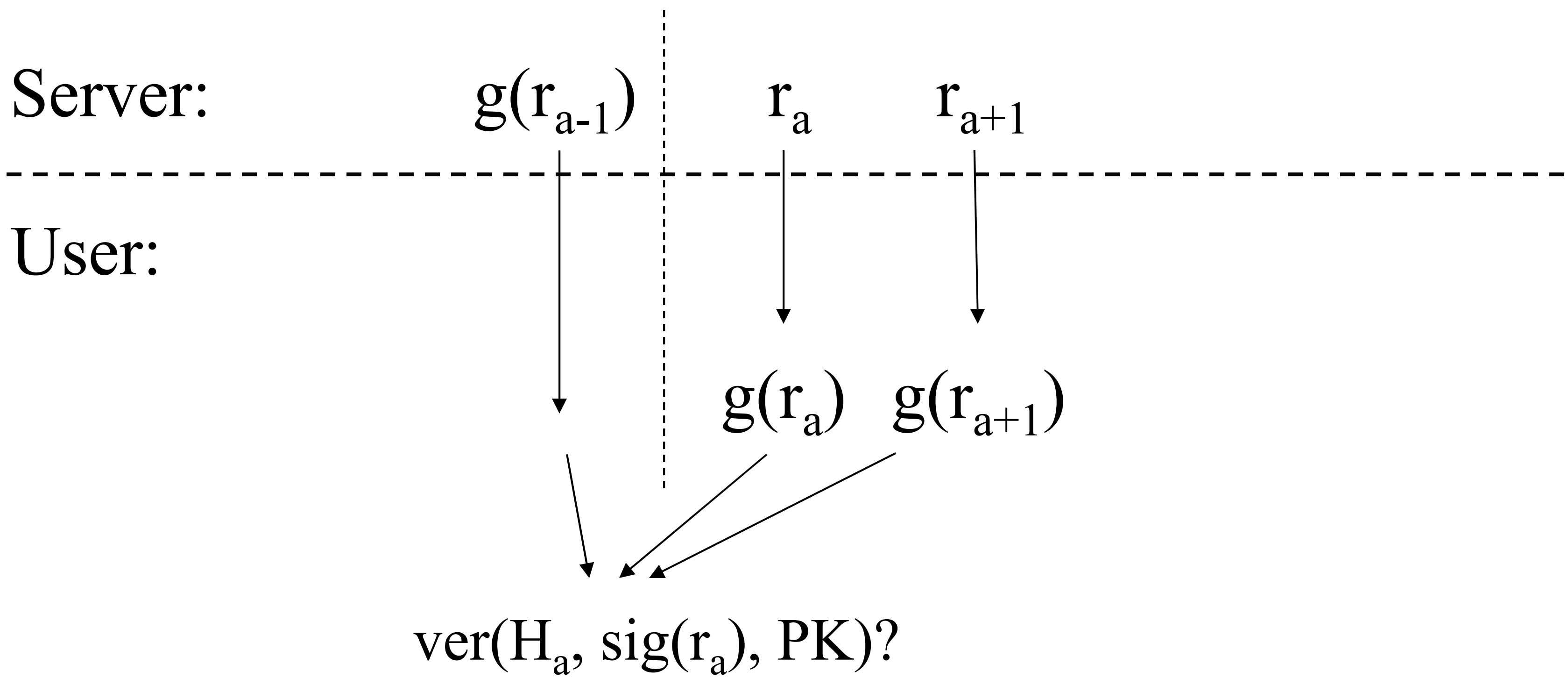
40 50 60 70 80 ….

↑

Query: $55 \leq r$

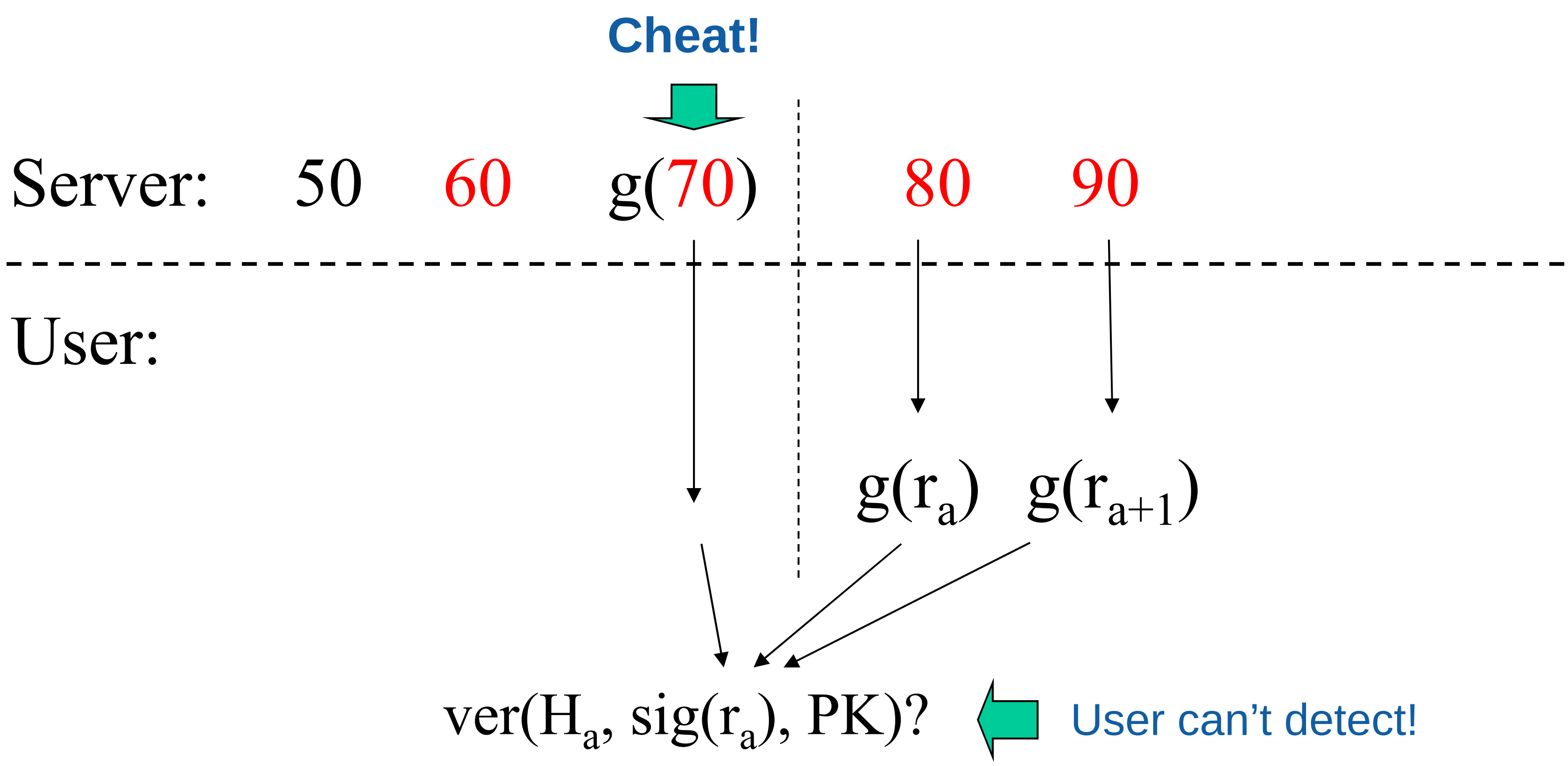
A naïve solution is to return 50. By proving that 50 is chained to 60, we know that no answer has been dropped. But, this reveals the value of 50.

How about this …



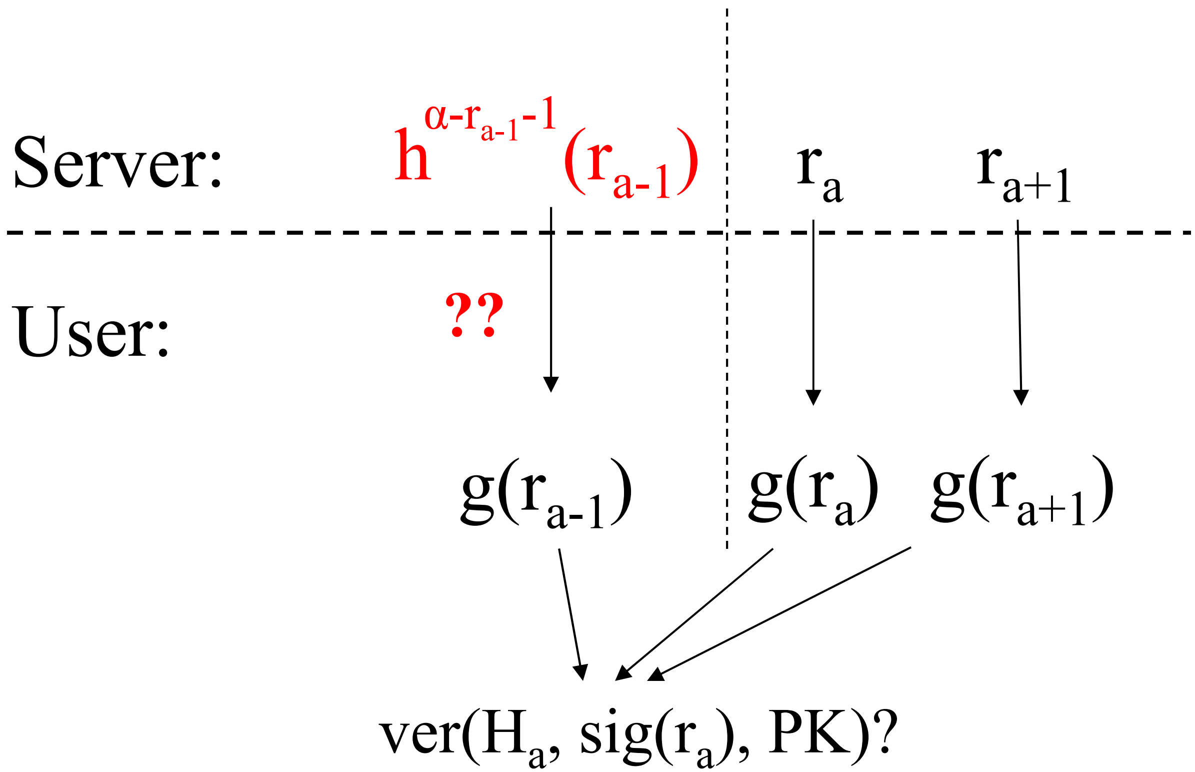
Query: $\alpha \leq r$

The basic idea fails …



Query: $55 \leq r$

Private Boundary Proof Ensures Origin

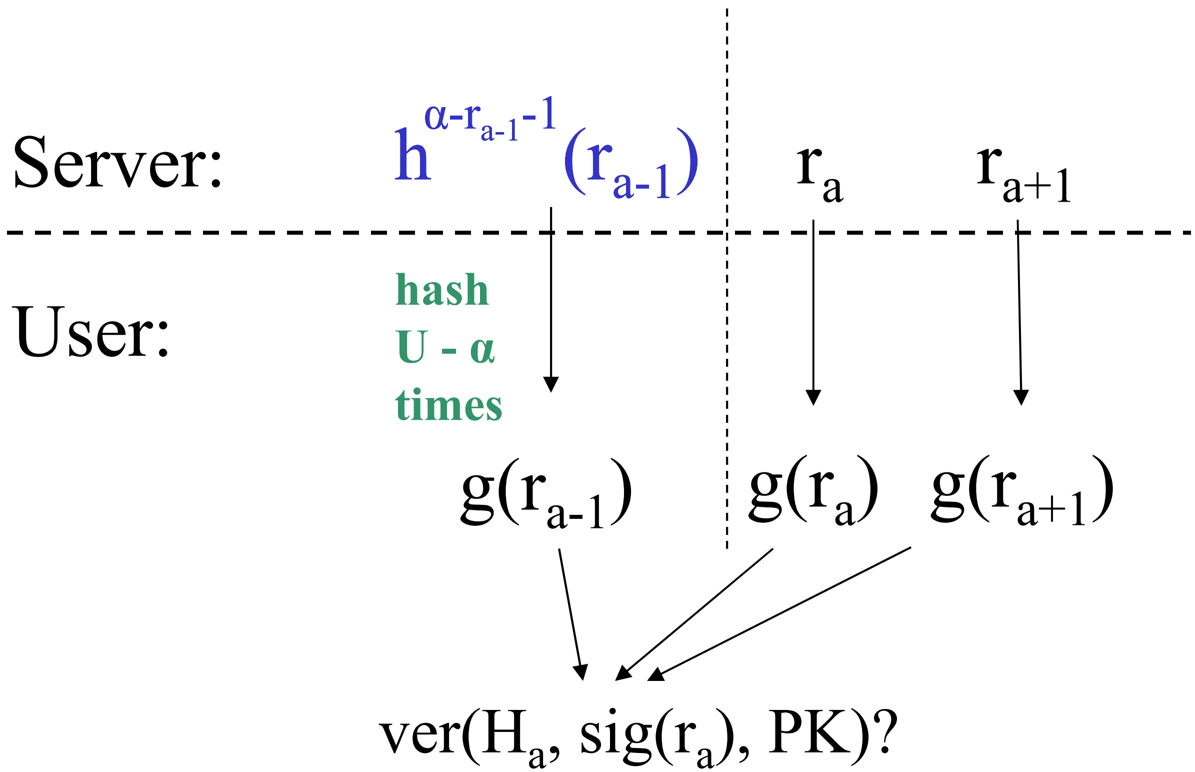


$h^i(r) = h^{i-1}(h(r))$

$g(r) = h^{U-r-1}(r)$

Query: $\alpha \leq r$

Private Boundary Proof Ensures Origin



A **collaborative scheme** to compute the hash value

$$h^i(r) = h^{i-1}(h(r))$$

$$g(r) = h^{U-r-1}(r)$$

$$= \mathbf{h}^{\mathbf{U}-\alpha}(\mathbf{h}^{\alpha-\mathbf{r}-1}(\mathbf{r}))$$

Query: $\alpha \leq r$

Back to our example

Cheat!

Server: $h^{\alpha-70-1}$ (70) 80 90

Require that the inverse of h^i for $i < 0$ be undefined

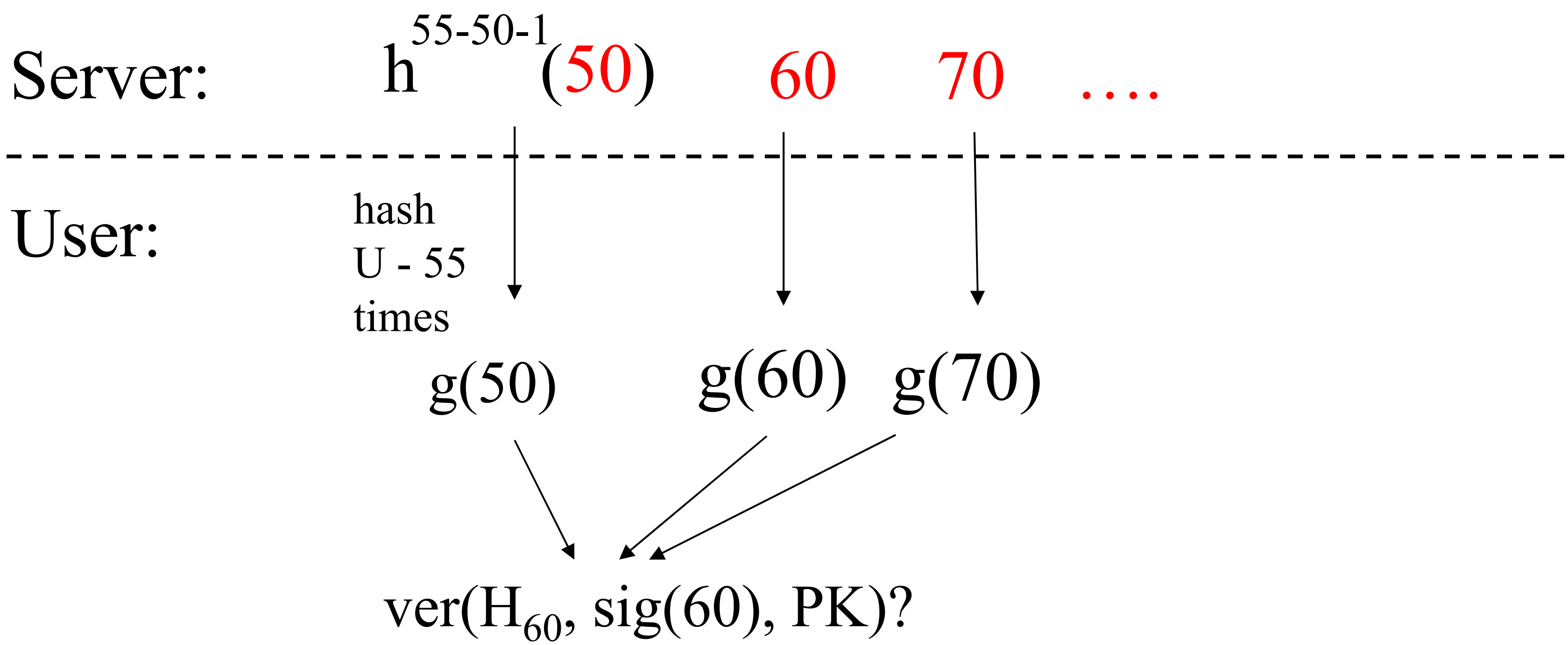
User: hash U - 55 times

Wrong! Undefined! $g(r_a)$ $g(r_{a+1})$

ver(H_a, sig(r_a), PK)? User detects cheating

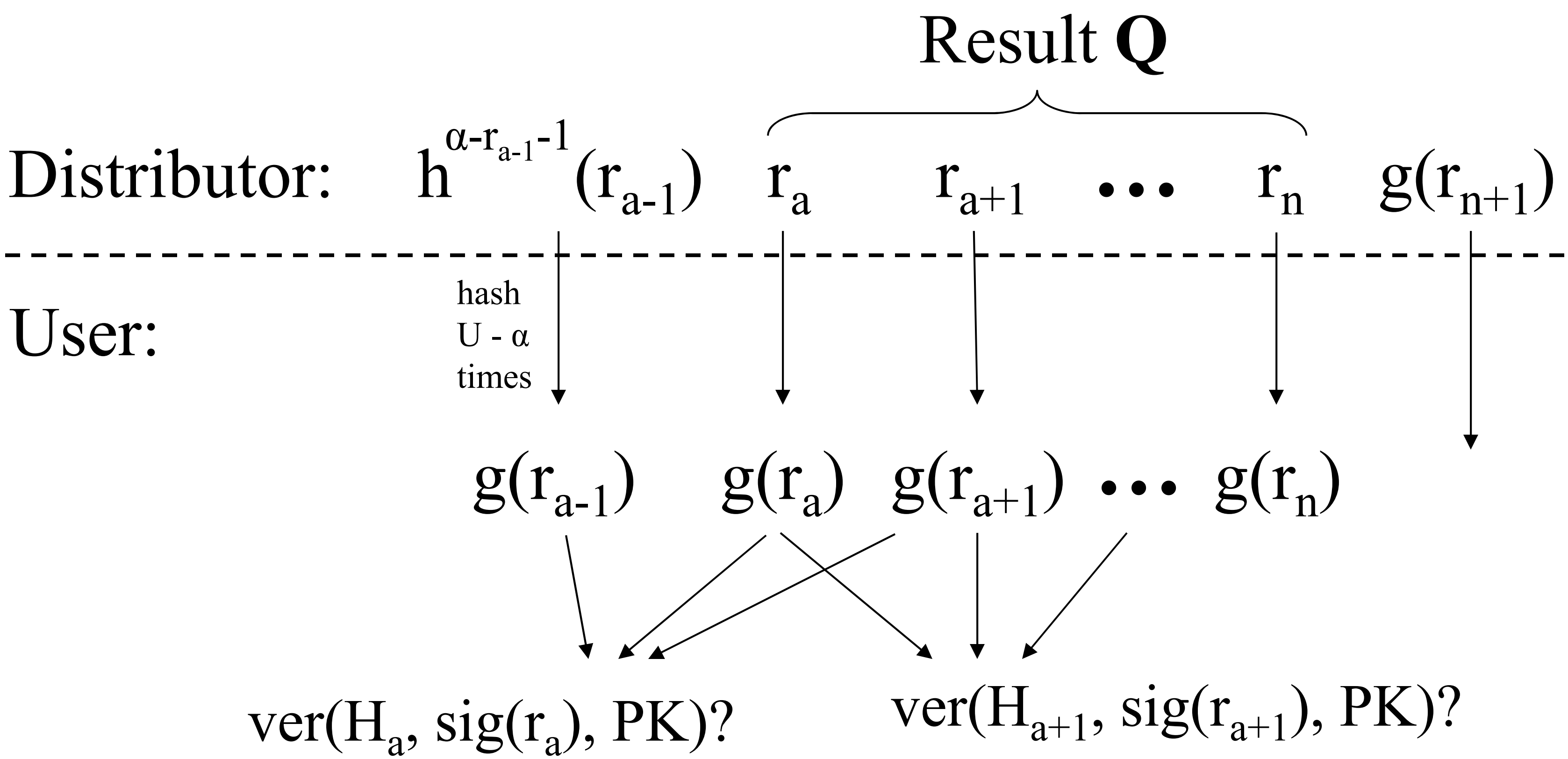
Query: $55 \leq r$

Back to our example



Query: $55 \leq r$

Putting the Pieces Together



Query: $\alpha \leq r$

Other cases

- $\alpha \leq r$
- $\beta \geq r$ (Result = $\{r_a, r_{a+1}, \ldots r_b\}$, i.e., $r_a, \ldots r_b \leq \beta < r_{b+1}$
  - Need to verify that $r_{b+1} > \beta$
  - Define $g(r) = \mathbf{h}^{r-L-1}(\mathbf{r}) = \mathbf{h}^{\beta-L}(\mathbf{h}^{r-\beta-1}(\mathbf{r}))$ where L is a value outside of the minimum value of the domain
- So, we have $\alpha \leq r \leq \beta$
- $r = \alpha \equiv \alpha \leq r \leq \alpha$
- $\alpha < r < \beta \equiv \alpha+1 \leq r \leq \beta-1$
- $\alpha \neq r \equiv (L < r < \alpha) \cup (\alpha < r < R)$

NULL Answers??

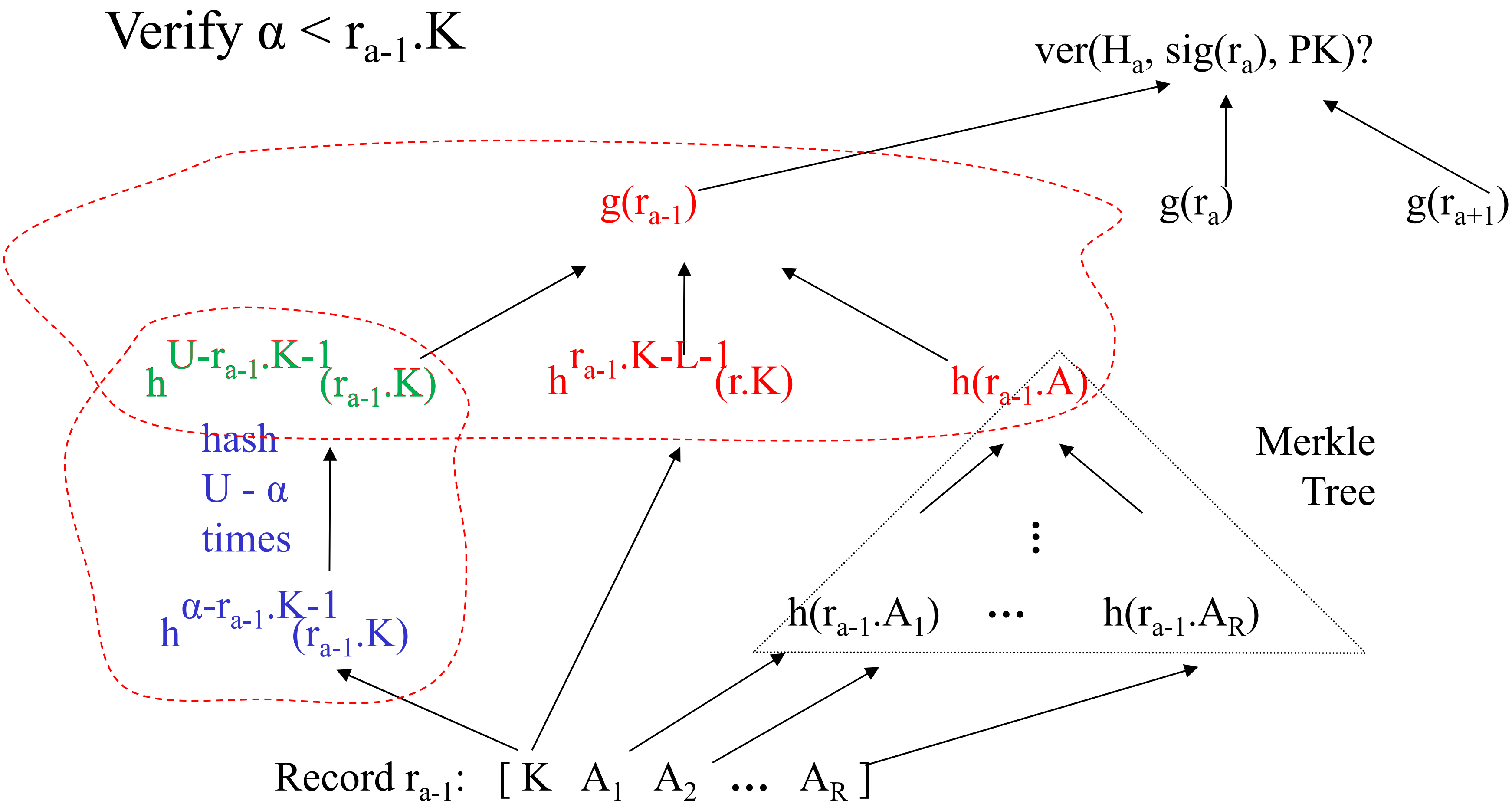
- Consider Q: $\alpha \leq r$.
- Q = Ø because $r_n < \alpha$.
  - Server returns $\mathbf{h}^{\alpha - r_n - 1}\mathbf{(r)}$, $\mathbf{g(r_{n+1})}$, $\mathbf{sig(r_{n+1})}$
  - User computes $\mathbf{h}^{U - \alpha}(\mathbf{h}^{\alpha - r_n - 1}\mathbf{(r)})$ and verifies ver(H_{n+1}, sig(r_{n+1}), PK)?

- How about $r_i < \alpha \leq \beta < r_{i+1}$?

One More Vulnerability

- User can discover r_{a-1} through brute force enumeration of numbers below r_a
- Solution:
  - Record [K, A_1, .., A_m], K = ordering attribute
  - $g(r_i.K \mid r_i.A_1 \mid \ldots \mid r_i.A_m)$
  - Brute-force attack is no longer feasible

Completeness Verification for Range Queries



Query: $\alpha \le K \le \beta$. Result: $\{ r_a, r_{a+1}, \ldots, r_b \}$

Other queries

- SP Query
  - Based on MHT(r.A)
  - Ordering attribute has to be returned (even if it is not part of the target attributes). Why?
  - For attributes that are filtered out, digests may need to be returned
- SPJ Query
  - R.Ai = S.Aj (Ai is foreign-key in R, Aj is primary key in S)
    - Referential integrity constraint mandates that every instance of R.Ai must have a matching entry in S.Aj
    - So, only need to deal with selection conditions on R.Ai or S.Aj
    - Create a signature chain for R.Ai

What else?

- What about data freshness?
- More efficient scheme
- Ad-hoc joins
- Aggregates
- Multi-dimensional data
- Computation
- Complete (complex) queries

Summary

- Malicious service provider may cheat
- Users need assurance on their query answers
- Merkle hash tree offers a good solution but …
- Signature chain guarantee completeness without violating access control policy