Verifiable Set Operations over Outsourced Databases

Ran Canetti
Boston University
& Tel Aviv University

Omer Paneth
Boston University

Dimitris Papadopoulos
Boston University

Nikos Triandopoulos
RSA Laboratories
& Boston University

Outsourced Computation

- Modern Computing
  → asymmetric computational environment

- Powerful Servers

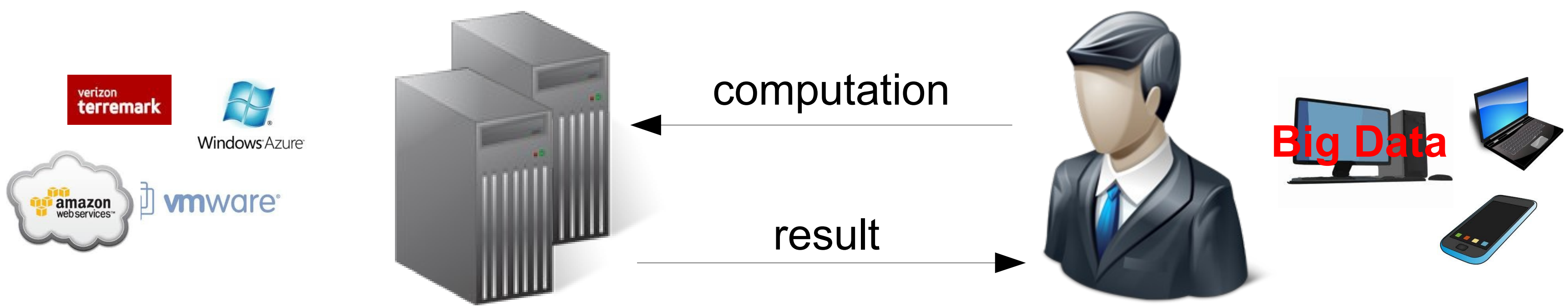
- Multiple types of "weak" devices

Outsourced Computation

- Modern Computing
  → asymmetric computational environment

- Powerful Servers
- Cloud Computing

- Multiple types of "weak" devices

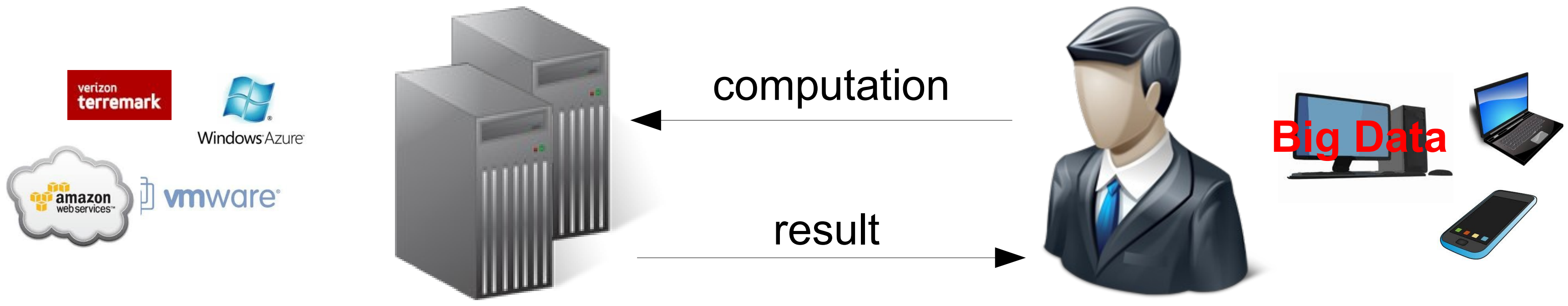


Outsourced Computation

- Modern Computing
  → asymmetric computational environment

- Powerful Servers
- Cloud Computing

- Multiple types of "weak" devices



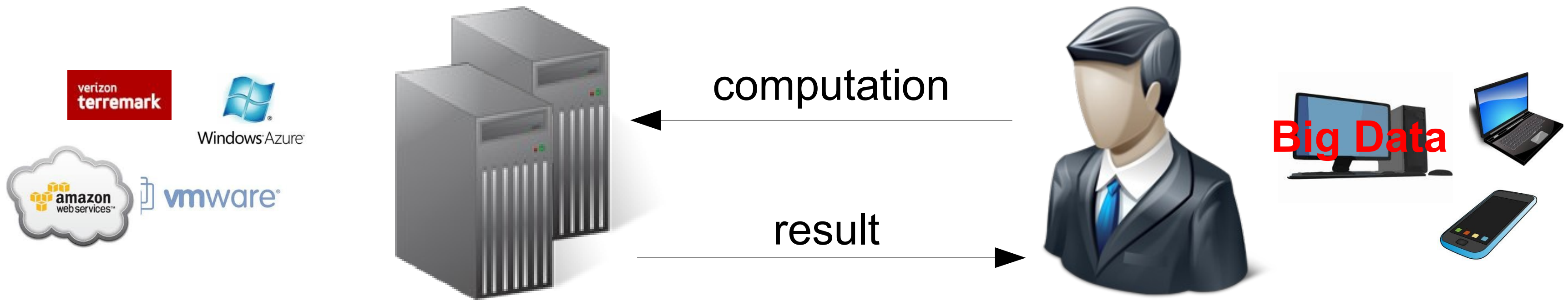
- Integrity-of-computation

Outsourced Computation

- Modern Computing
  → asymmetric computational environment

- Powerful Servers
- Cloud Computing

- Multiple types of "weak" devices

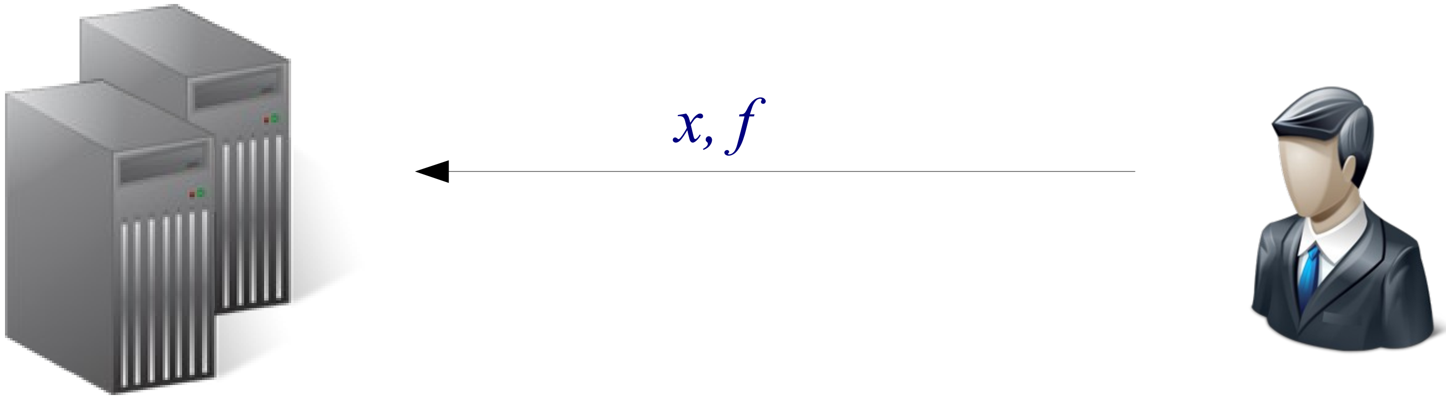


- Integrity-of-computation

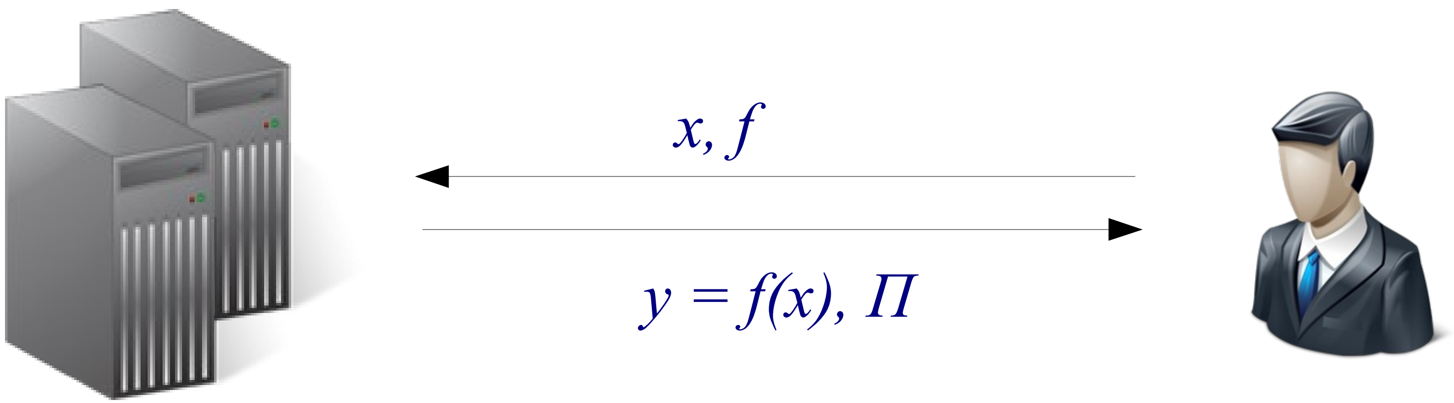
Did you do it correctly?

Verifiable Computation (VC) Protocol

Verifiable Computation (VC) Protocol



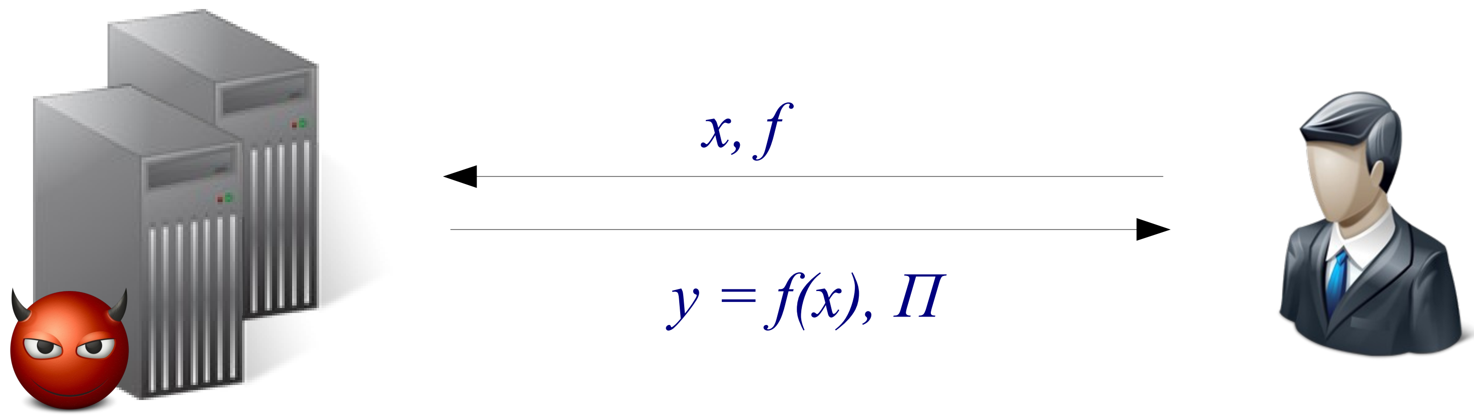
Verifiable Computation (VC) Protocol



$Verify(x,f,y,\Pi) = accept/reject$

Verifiable Computation (VC) Protocol

- Untrusted prover – server can arbitrarily cheat

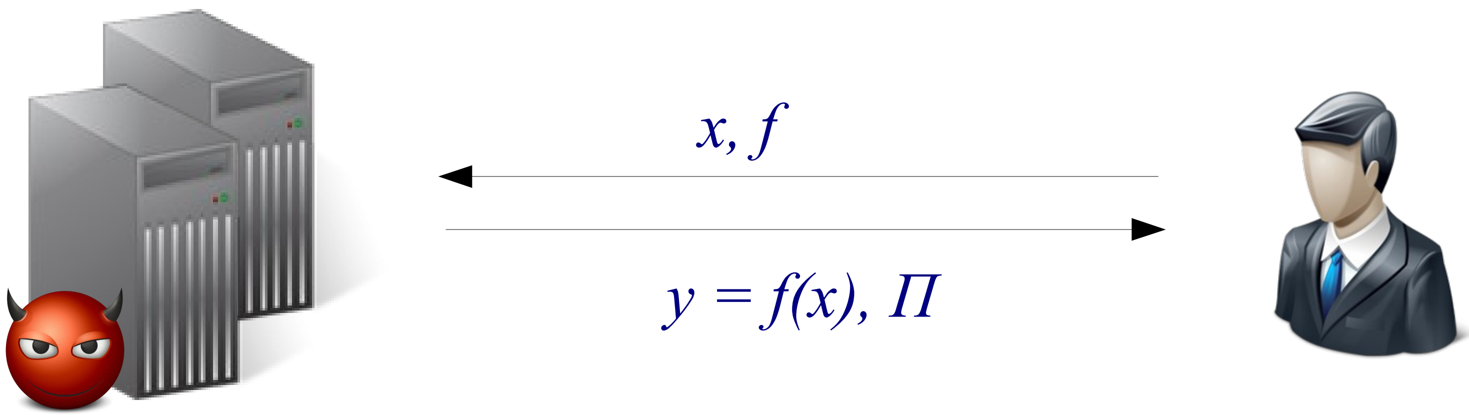


Verify(x,f,y,Π) = accept/reject

Soundness: *Verify* accepts with negligible probability if $y \neq f(x)$

Verifiable Computation (VC) Protocol

- Untrusted prover – server can arbitrarily cheat



$Verify(x,f,y,\Pi) = accept/reject$

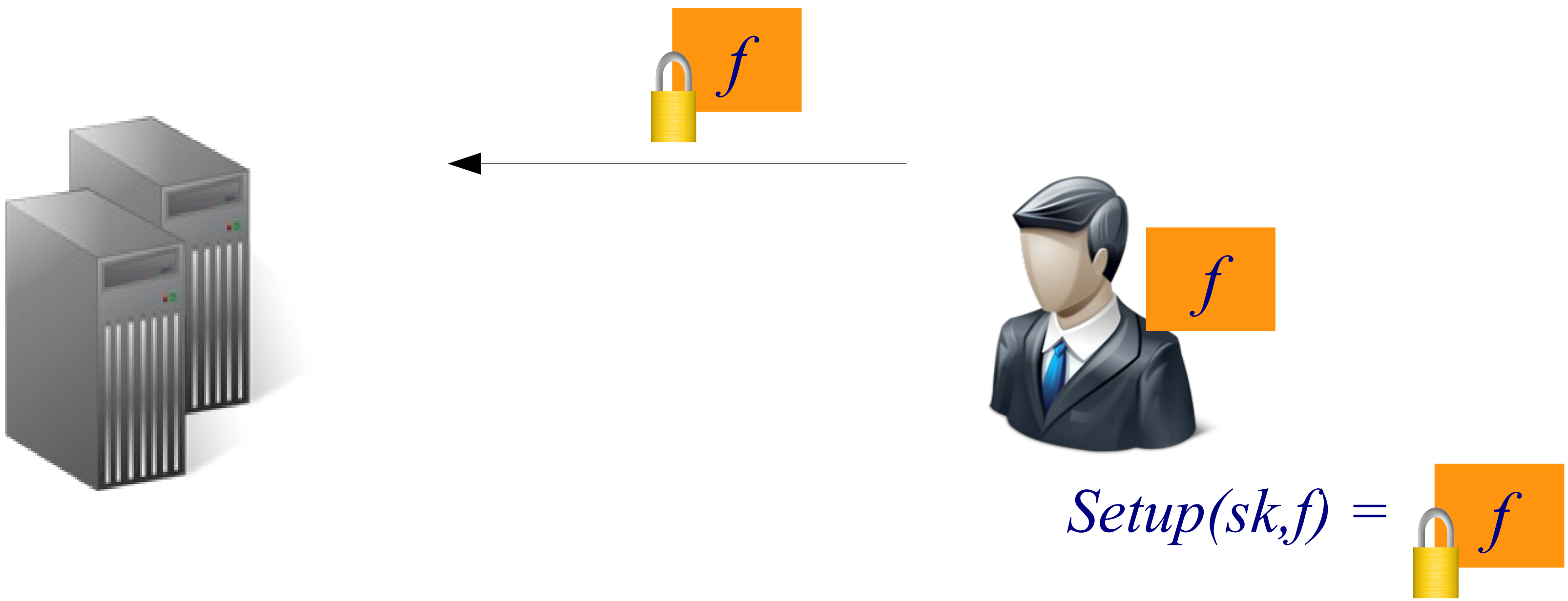
Soundness: *Verify* accepts with negligible probability if $y \neq f(x)$
Efficiency: Verification should be faster than computation

VC with Pre-processing

- Client runs expensive pre-processing for f once

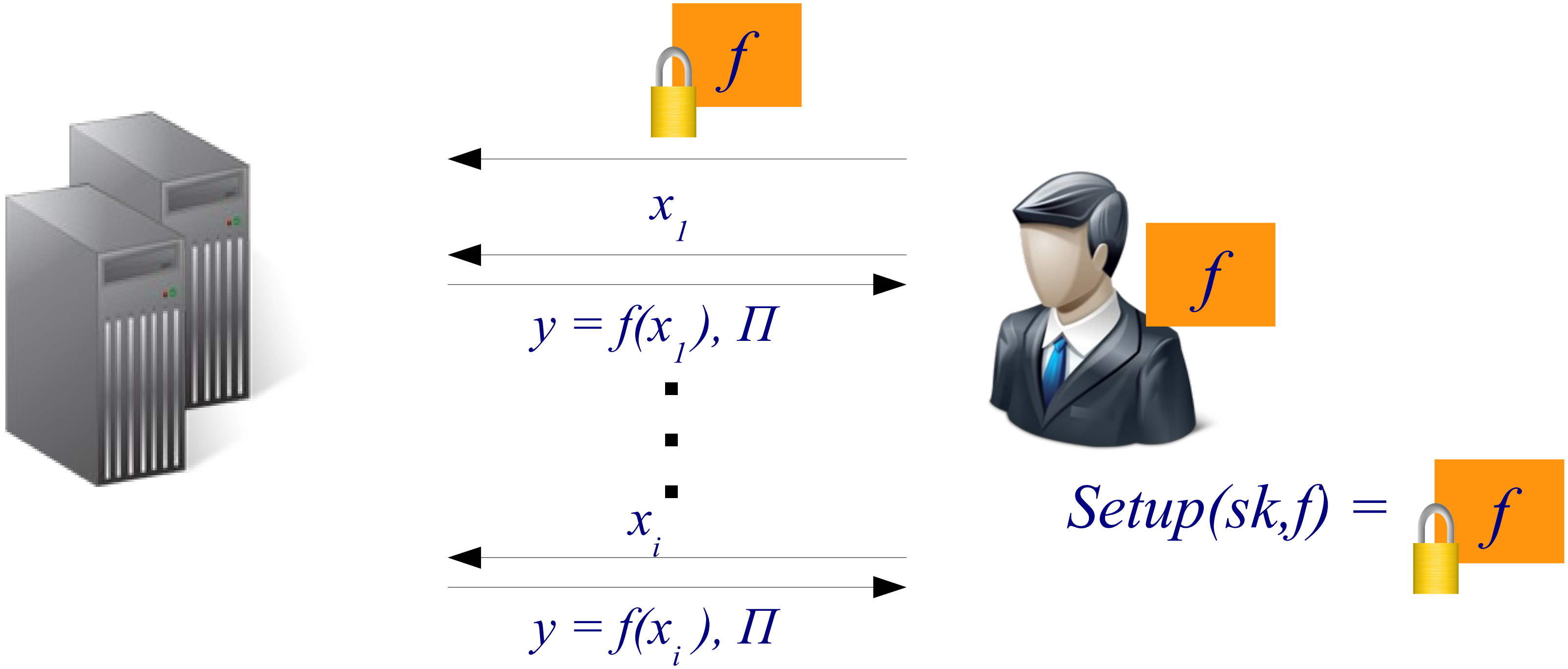
VC with Pre-processing

- Client runs expensive pre-processing for f once



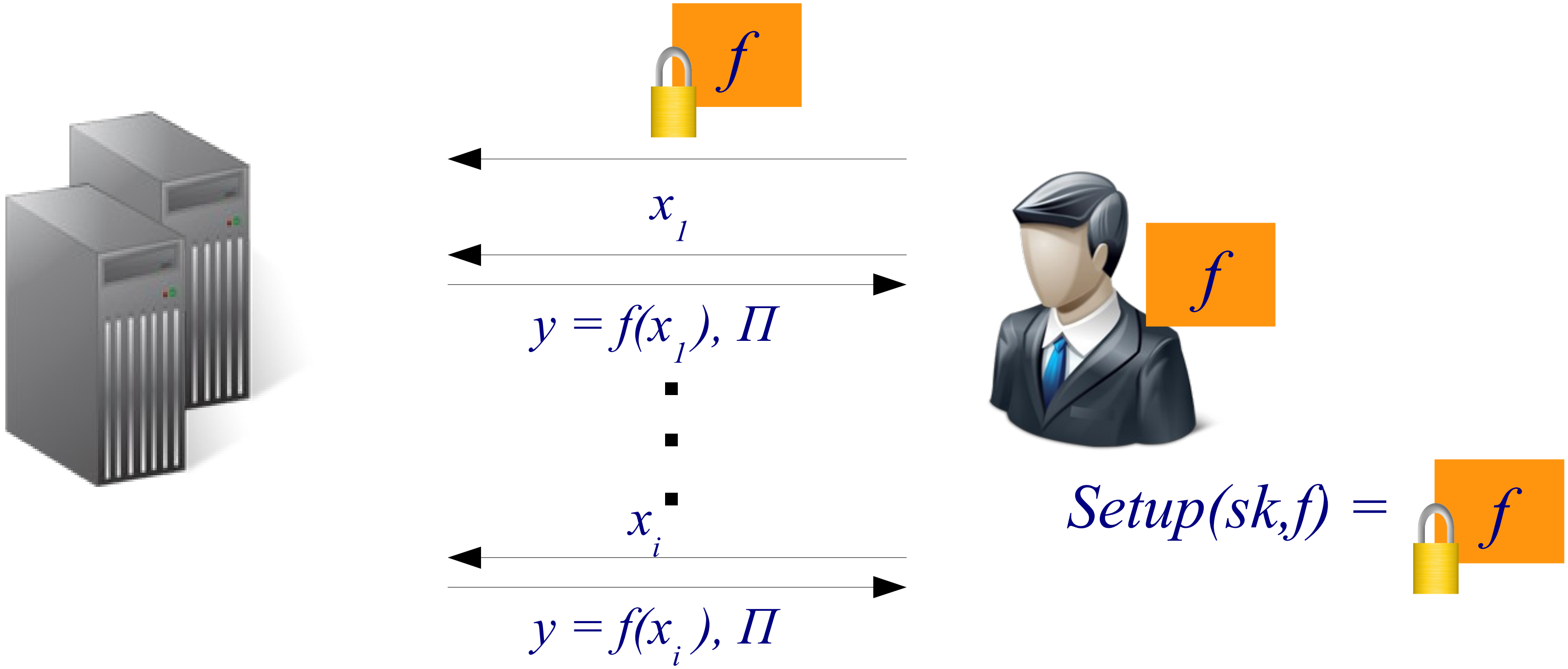
VC with Pre-processing

- Client runs expensive pre-processing for f once
- Amortizes cost over multiple executions



VC with Pre-processing

- Client runs expensive pre-processing for f once
- Amortizes cost over multiple executions



- Pre-processing not inherently necessary
  - [Bitansky,Canetti,Chiesa,Tromer'13]

VC with Outsourced Storage

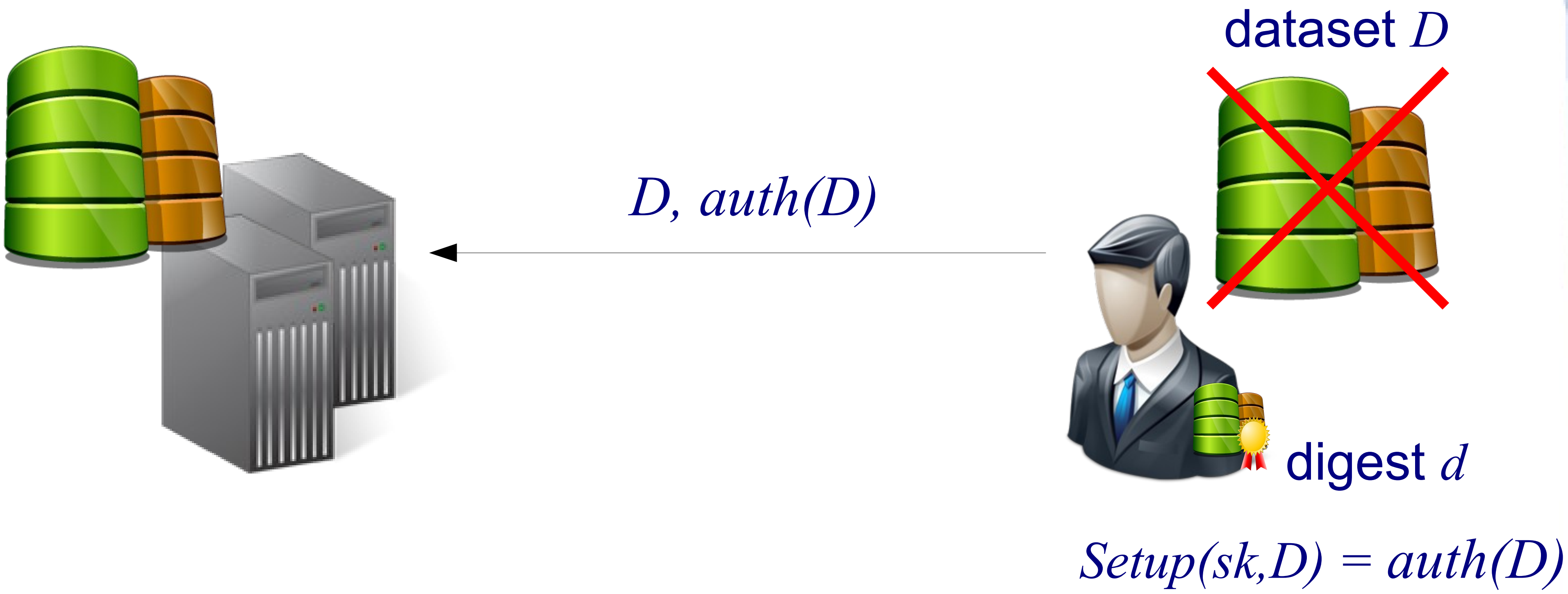
dataset D

VC with Outsourced Storage

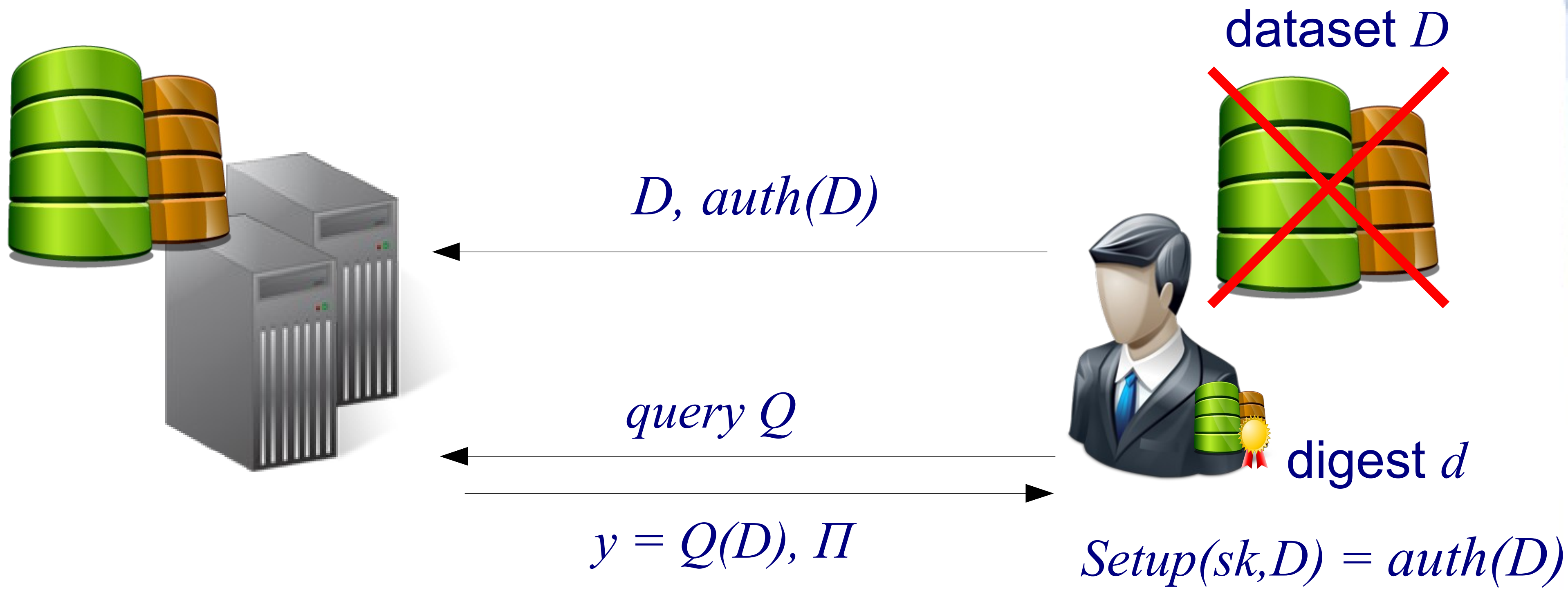
dataset D

$Setup(sk,D) = auth(D)$

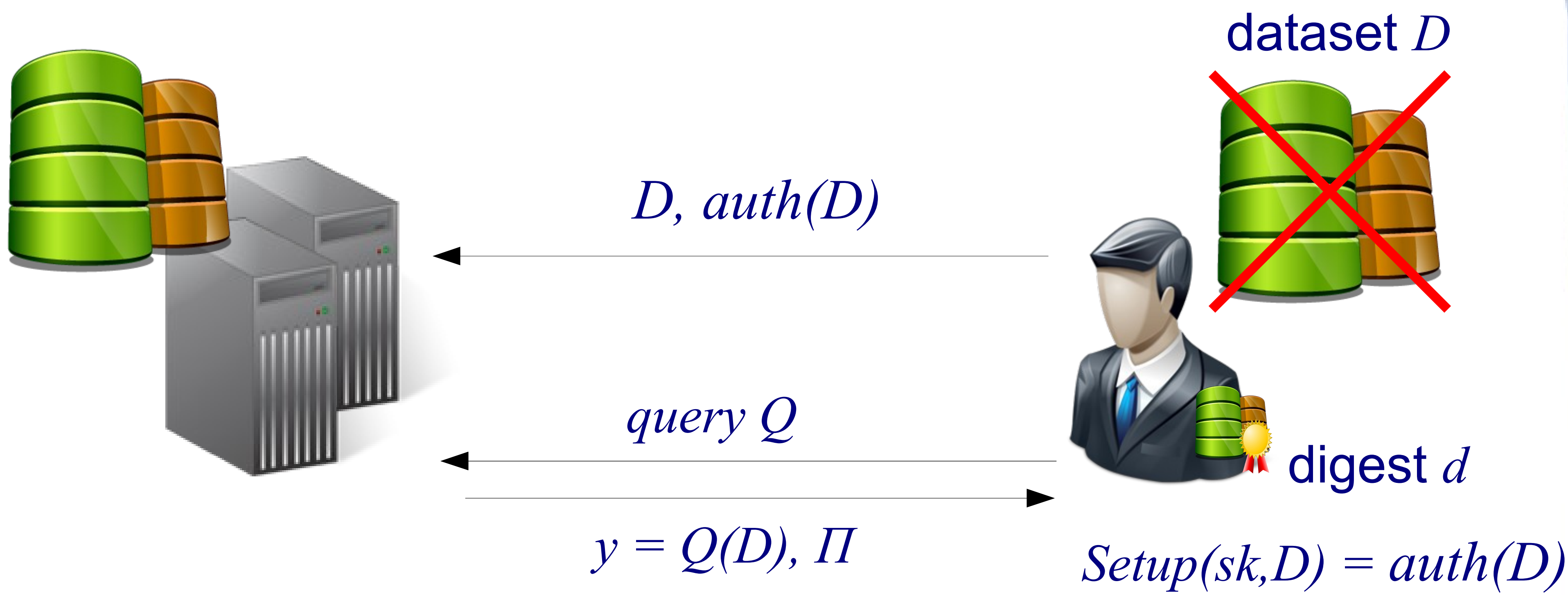
VC with Outsourced Storage



VC with Outsourced Storage

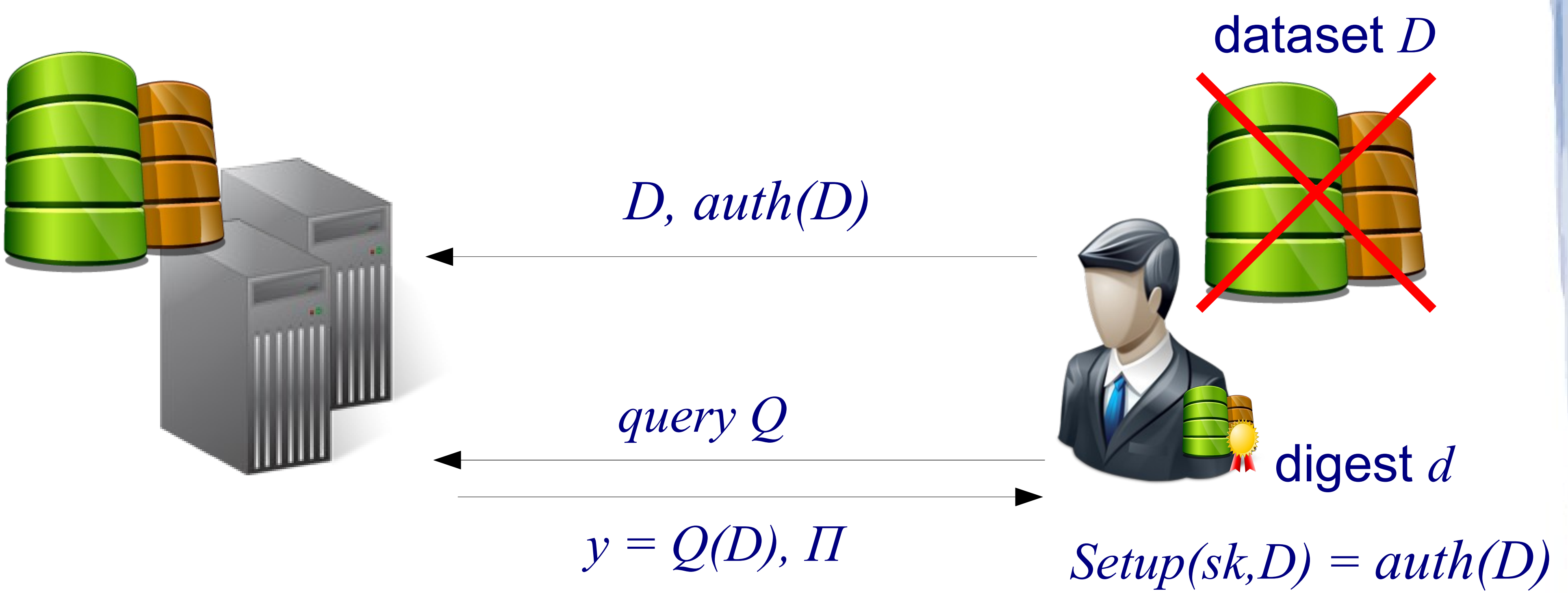


VC with Outsourced Storage

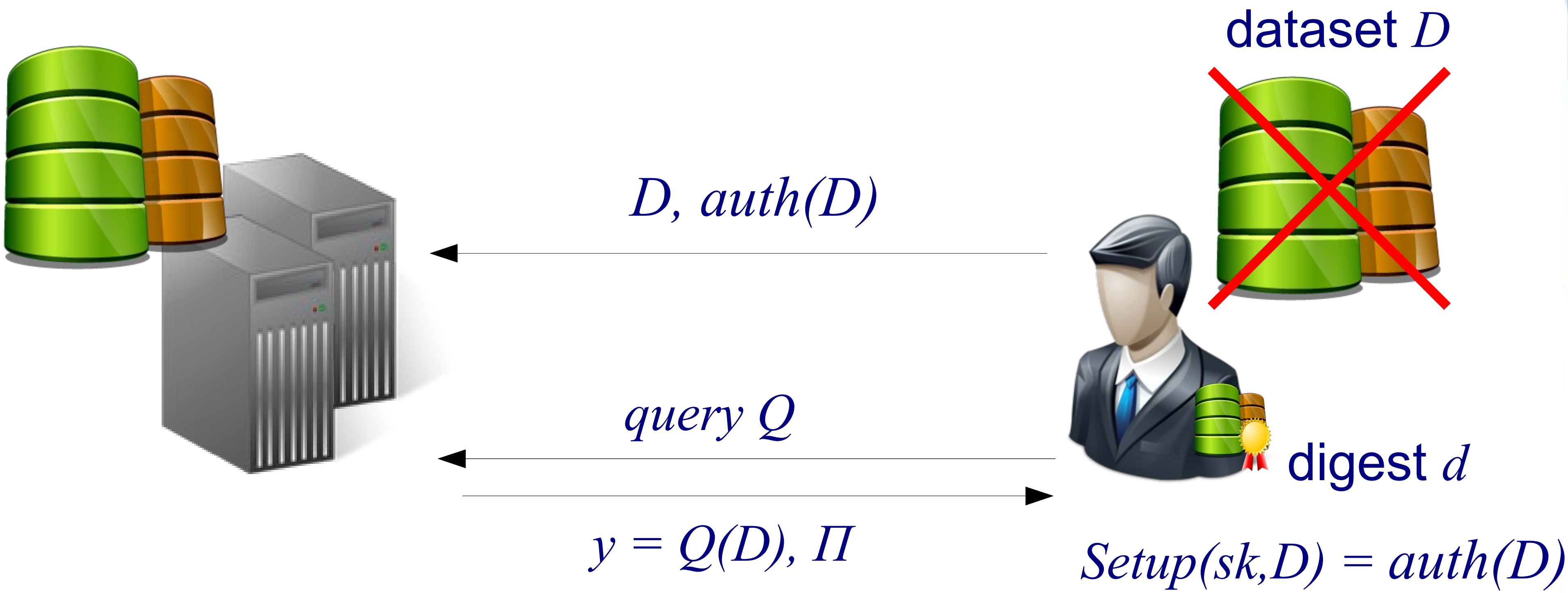


- Studied in existing work
  - *memory delegation* [Chung,Kalai,Liu,Raz'11]
  - *outsourced datasets* [Backes,Fiore,Reischuk'13]
  - *authenticated data structures* [Nissim,Naor'98][Tamassia'03]

VC with Outsourced Storage

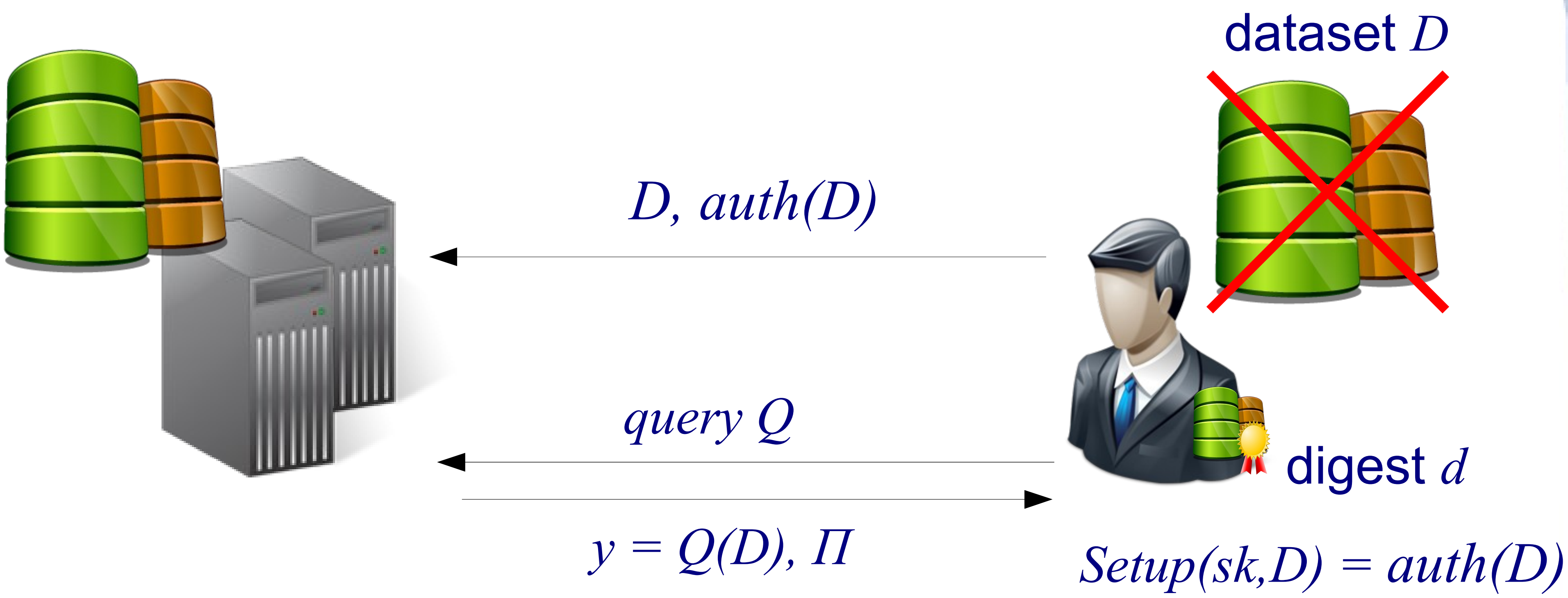


VC with Outsourced Storage



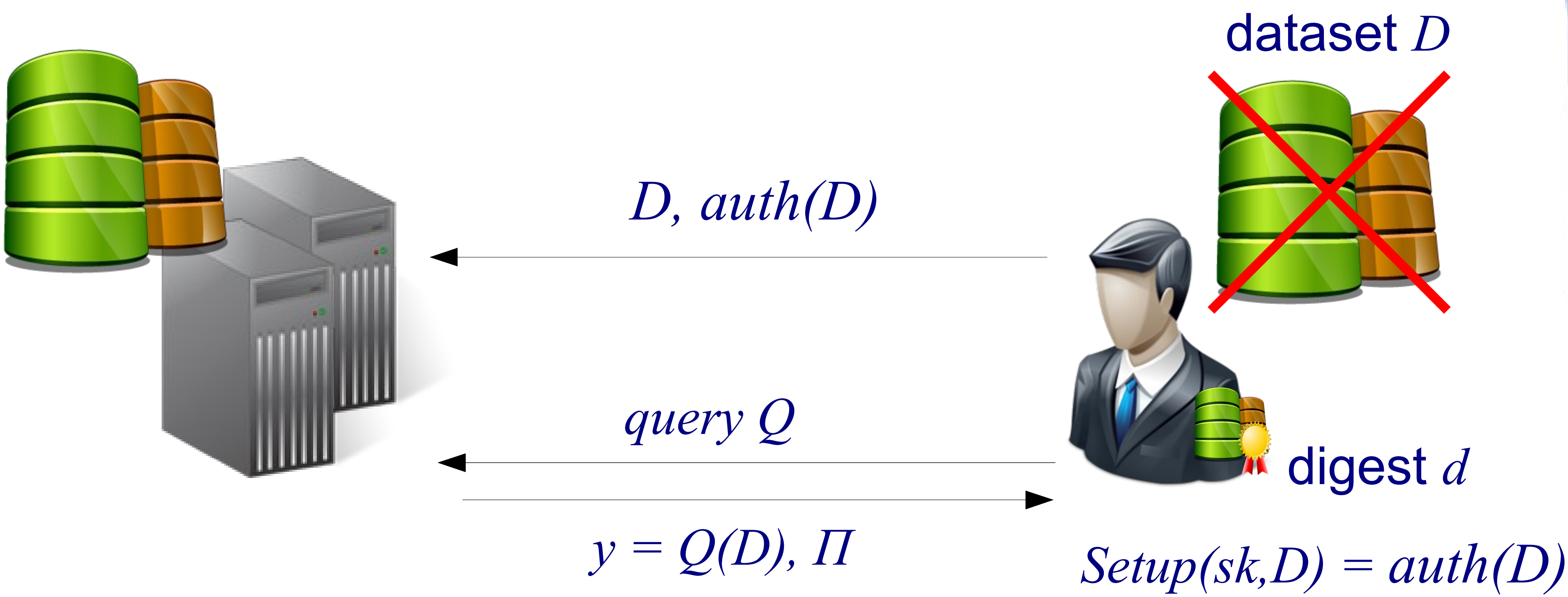
- Dual of the classic model
  - fix function / fix data

VC with Outsourced Storage



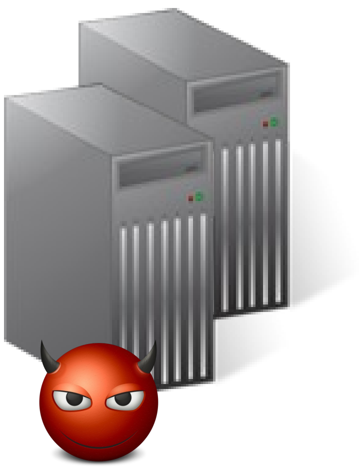
- Dual of the classic model
  - fix function / fix data
- Additional query type: updates in *D*

VC with Outsourced Storage



- Dual of the classic model
  - fix function / fix data
- Additional query type: updates in *D*
  - handle updates *efficiently*

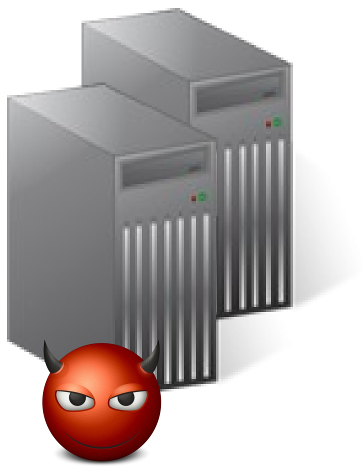
Security Game



$Gen(\$) \rightarrow sk,pk$

Security Game

pk



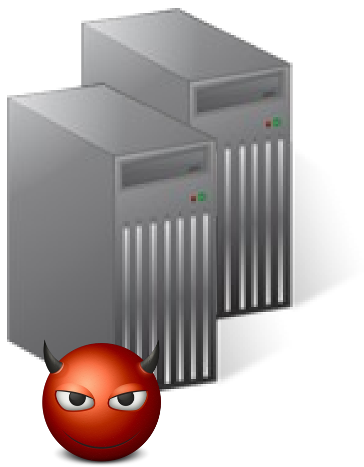
$Gen(\$) \rightarrow sk,pk$

Security Game

pk ←

D_0 →

← $auth(D_0)$



Prove and *Verify* using pk

$Gen(\$) \rightarrow sk,pk$

Provides oracle access to *Setup* and *Update*

Security Game

Prove and *Verify* using *pk*

pk ←

D_0 →

← $auth(D_0)$

$update\ u_1$ →

← $auth(D_0, u_1)$

⋮

$update\ u_t$ →

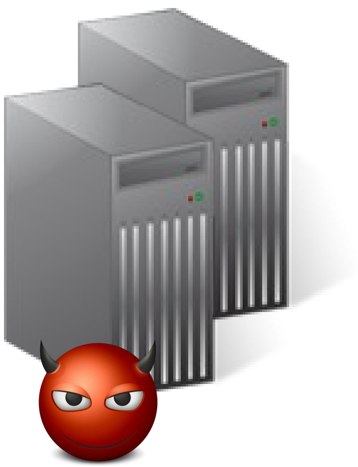
← $auth(D_{t-1}, u_t)$

$Gen(\$) \rightarrow sk,pk$

Provides oracle access to *Setup* and *Update*

Security Game

Finally:



$\{D_i, auth(D_i), d, Q, A^*, \Pi\}$

for $0 \leq i \leq t$

Adv wins if A^* is not the correct answer but *Verify* accepts

Known Solutions
(in this model and others)

- Theoretical Results

  [Micali'00],[Ishai,Kushilevitz,Ostrovsky'08],
  [Goldwasser,Kalai,Rothblum'08],
  [Applebaum,Ishai,Kusilevitz'10],
  [Gennaro,Gentry,Parno'10]
  [Chung,Kalai,Vadhan'10],
  [Canetti,Riva,Rothblum'11],
  [Gennaro,Gentry,Parno,Raykova'13],
  [Bitansky,Canetti,Chiesa,Tromer'13],...

- Implementation Works

  [Cormode,Mitzenmacher,Thaler'12]
  [Setty,Braun,Vu,Blumberg,Parno,Walfish'13],
  [Parno,Gentry,Howell,Raykova'13]
  [Ben-Sasson,Chiesa,Genkin,Tromer,Virza'13]...

State of the art

- ✔ Excellent asymptotic behavior
  - non-interactive
  - general (i.e. for any language in NP)
  - verification cost $O(|\text{input}| + |\text{output}|)$
  - $O(1)$ proof size
  - poly-log overhead for proof computation

State of the art

- ✔ Excellent asymptotic behavior
  - non-interactive
  - general (i.e. for any language in NP)
  - verification cost $O(|\text{input}| + |\text{output}|)$
  - $O(1)$ proof size
  - poly-log overhead for proof computation
- ✗ High concrete overhead
  - server's cost prohibitive for general functions

Examples of Practical Issues

- Delegation in the *circuit-based* model of computation
  - reduce concrete functions to circuit problems
- Prover's overhead should be *query-specific*
  - not determined by “largest” query

Examples of Practical Issues

- Delegation in the *circuit-based* model of computation
  - reduce concrete functions to circuit problems
- Prover's overhead should be *query-specific*
  - not determined by “largest” query

Recent works explore alternative models

- [Goldwasser,Kalai,Popa,Vaikuntanathan,Zeldovich'13]
- [Gentry,Halevi,Raykova,Wichs'14]

In this Work

- Focus on specific class of functions
  - exploit algebraic structure for practical solutions
  - existing works
    - [Benabbas,Gennaro,Vahlis'11],[Backes,Fiore,Reischuk'13], [Papamanthou,Tamassia,Triandopoulos'11] ...

In this Work

- Focus on specific class of functions
  - exploit algebraic structure for practical solutions
  - existing works
    - [Benabbas,Gennaro,Vahlis'11],[Backes,Fiore,Reischuk'13], [Papamanthou,Tamassia,Triandopoulos'11] ...
- Functionality:
  Nested Intersections, Unions and Set Differences

In this Work

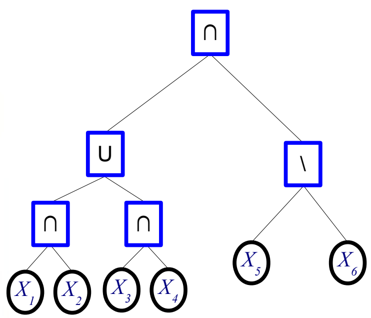
- Focus on specific class of functions
  - exploit algebraic structure for practical solutions
  - existing works
    - [Benabbas,Gennaro,Vahlis'11],[Backes,Fiore,Reischuk'13], [Papamanthou,Tamassia,Triandopoulos'11] ...
- Functionality:
  Nested Intersections, Unions and Set Differences
- Applications
  - A rich class of SQL queries
  - Keyword search
  - Similarity Measurements (e.g. Jaccard distance)
  - Set Membership

Outsourced Sets

- Database D consisting of m sets $X_1,\ldots,X_m$ with elements from Z_p

Outsourced Sets

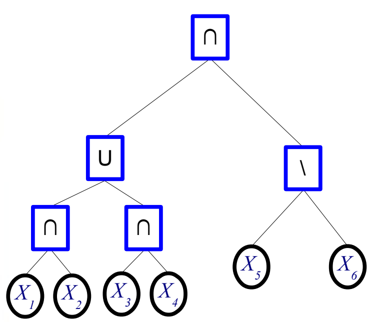
- Database D consisting of m sets $X_1,\ldots,X_m$ with elements from Z_p



- Supports queries expressed as polynomial length formulas of nested intersections, unions, and set differences
- e.g. $((X_2 \cap X_4) \cup (X_8 \cap X_5)) \cap (X_1 \setminus X_9))$

Outsourced Sets

- Database D consisting of m sets $X_1,\ldots,X_m$ with elements from Z_p



- Supports queries expressed as polynomial length formulas of nested intersections, unions, and set differences
- e.g. $((X_2 \cap X_4) \cup (X_8 \cap X_5)) \cap (X_1 \setminus X_9))$
- D changes dynamically under element insertion and deletion

Our Result

- VC with outsourced storage for sets:
  - query-specific proof-construction cost
  - efficient non-interactive updates
  - circuit-independent
  - public verifiability
  - concrete complexity analysis
    - low involved constants

Our Result

- Setup cost:
  - client's pre-processing cost $\rightarrow O(|D|)$
- Given query Q computable in $O(N)$ with answer A:
  - verification time $O(|Q| + |A|)$
  - proof size $O(|Q|)$
  - proof construction $\tilde{O}(N)$
- Update cost:
  - $O(1)$ operations for client and server

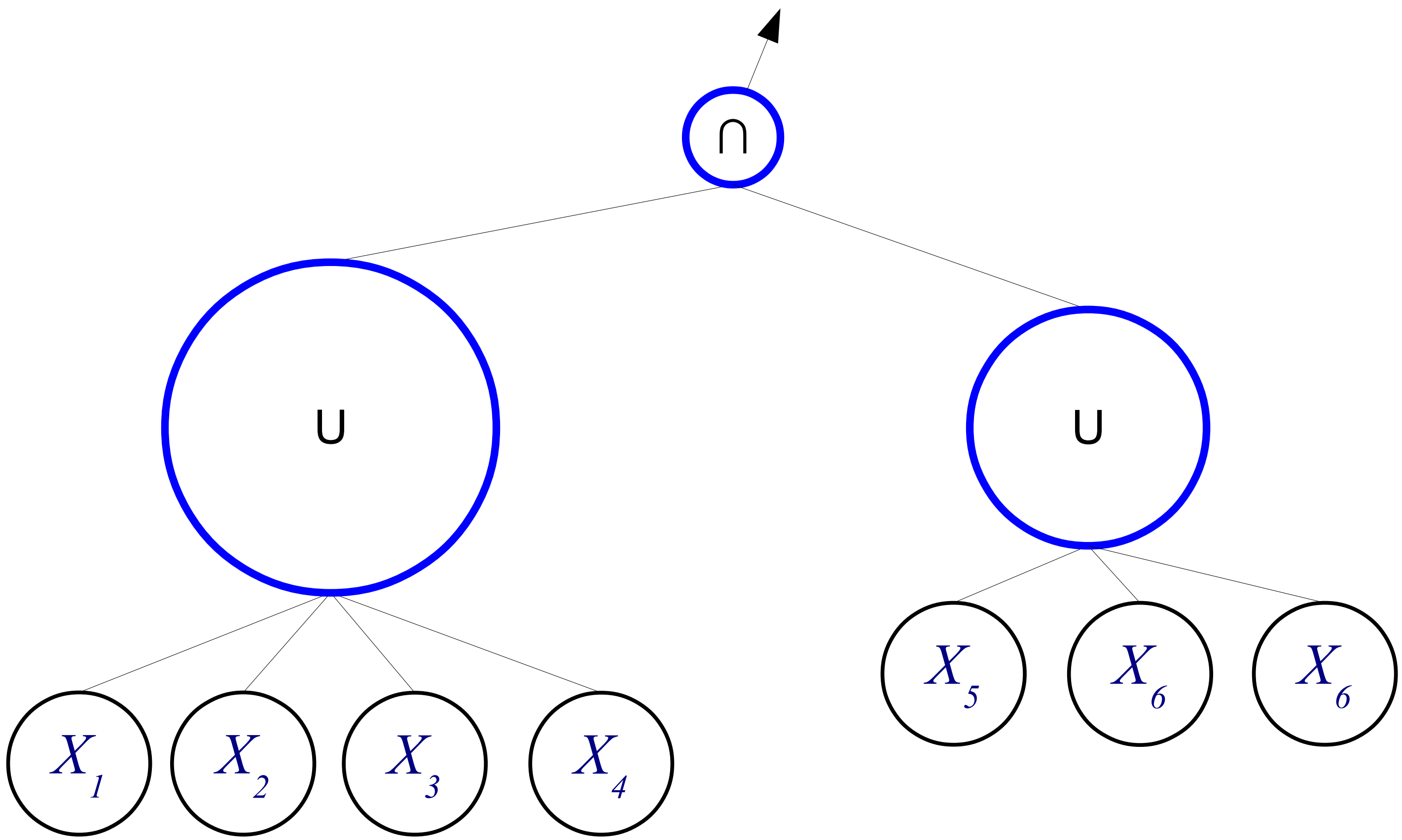
Our Result

- Setup cost:
  - client's pre-processing cost $\rightarrow O(|D|)$
- Given query Q computable in $O(N)$ with answer A:
  - verification time $O(|Q| + |A|)$
  - proof size $O(|Q|)$
  - proof construction $\tilde{O}(N)$

  independent of cardinalities of other sets
- Update cost:
  - $O(1)$ operations for client and server

Large Intermediate Results

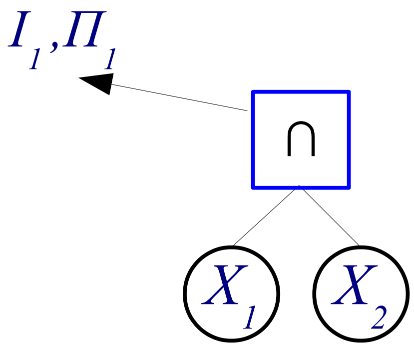
Note
Circle size denotes set cardinality



- Verification cost and proof size should be oblivious to the set cardinalities (except for answer set)

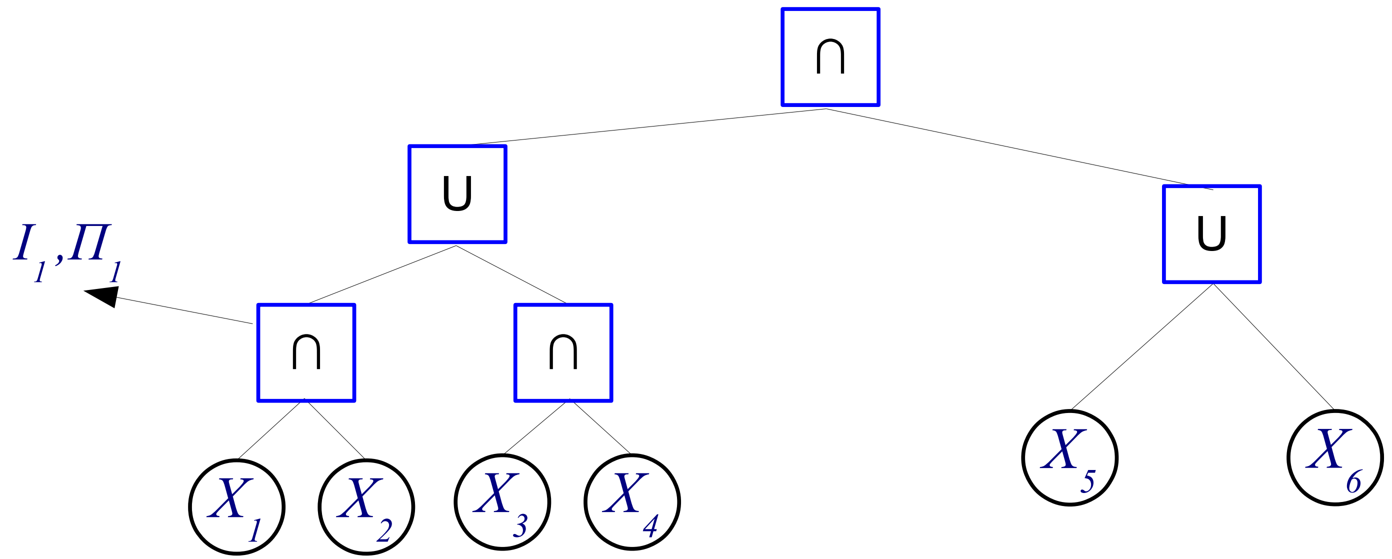
Main Idea (attempt 1)

- [Papamanthou,Tamassia,Triandopoulos'11]
  - construction for a single set operation based on *bilinear accumulators*



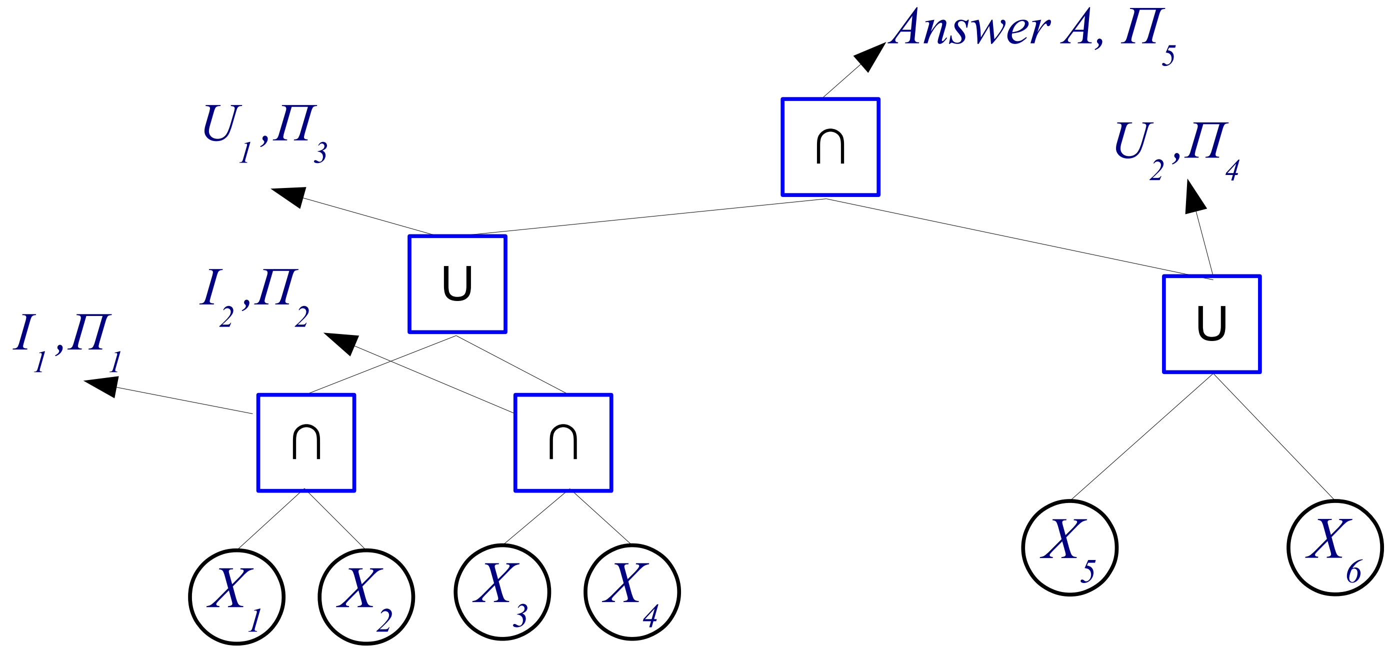
Main Idea (attempt 1)

- [Papamanthou,Tamassia,Triandopoulos'11]
  - construction for a single set operation based on *bilinear accumulators*

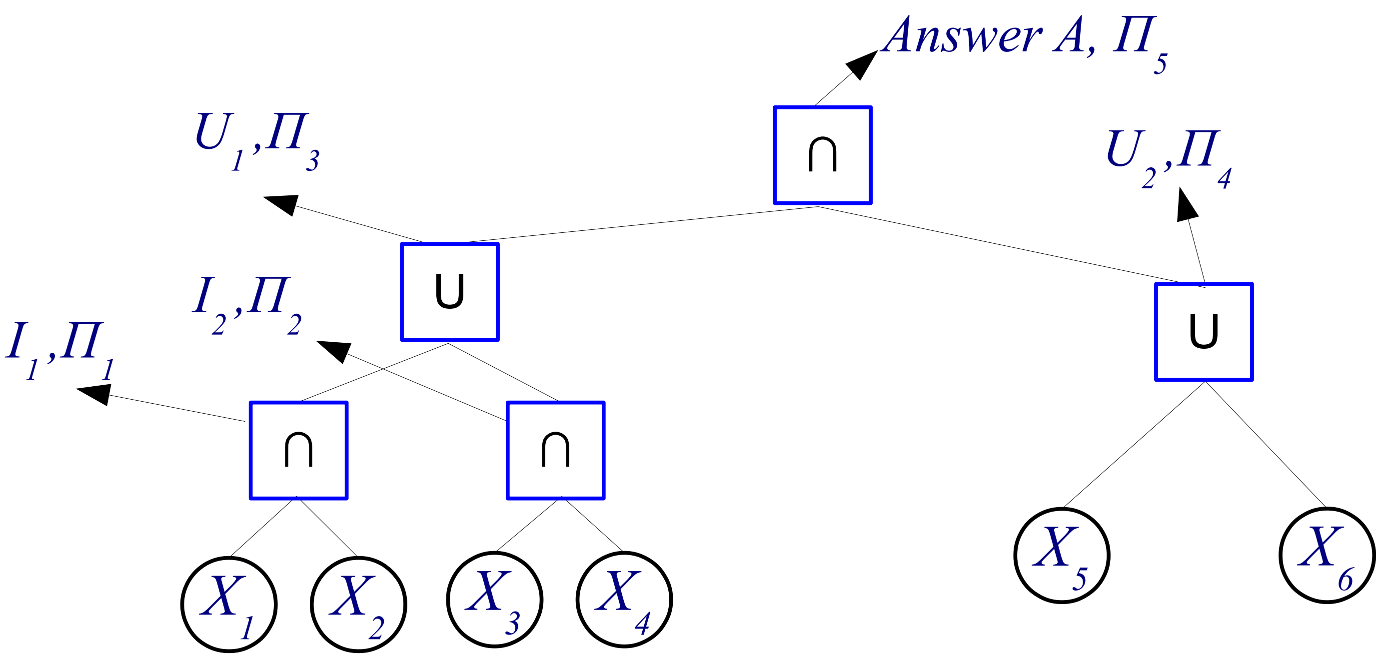


Main Idea (attempt 1)

- [Papamanthou,Tamassia,Triandopoulos'11]
  - construction for a <u>single</u> set operation based on *bilinear accumulators*
- Apply repeatedly per operation?

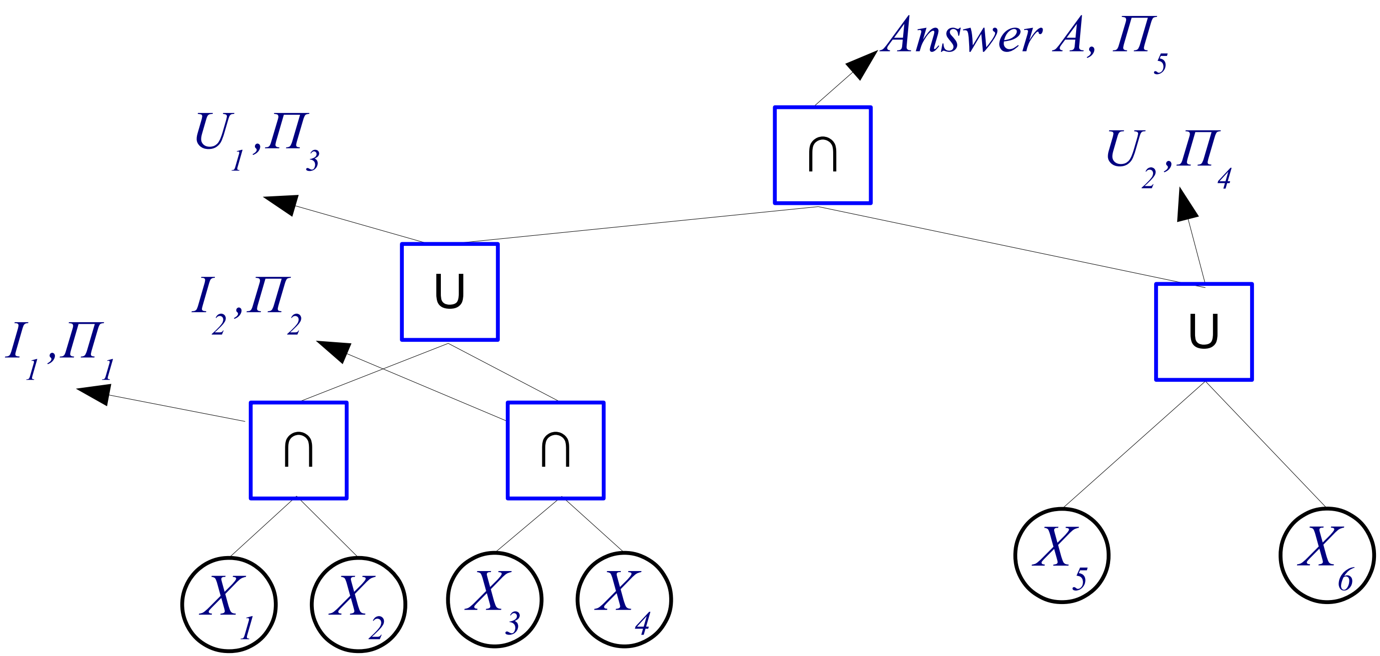


Main Idea (attempt 1)



$$\Pi = \{ (I_1,\Pi_1), (I_2,\Pi_2), (U_1,\Pi_3),(U_2,\Pi_4),(A,\Pi_5) \}$$

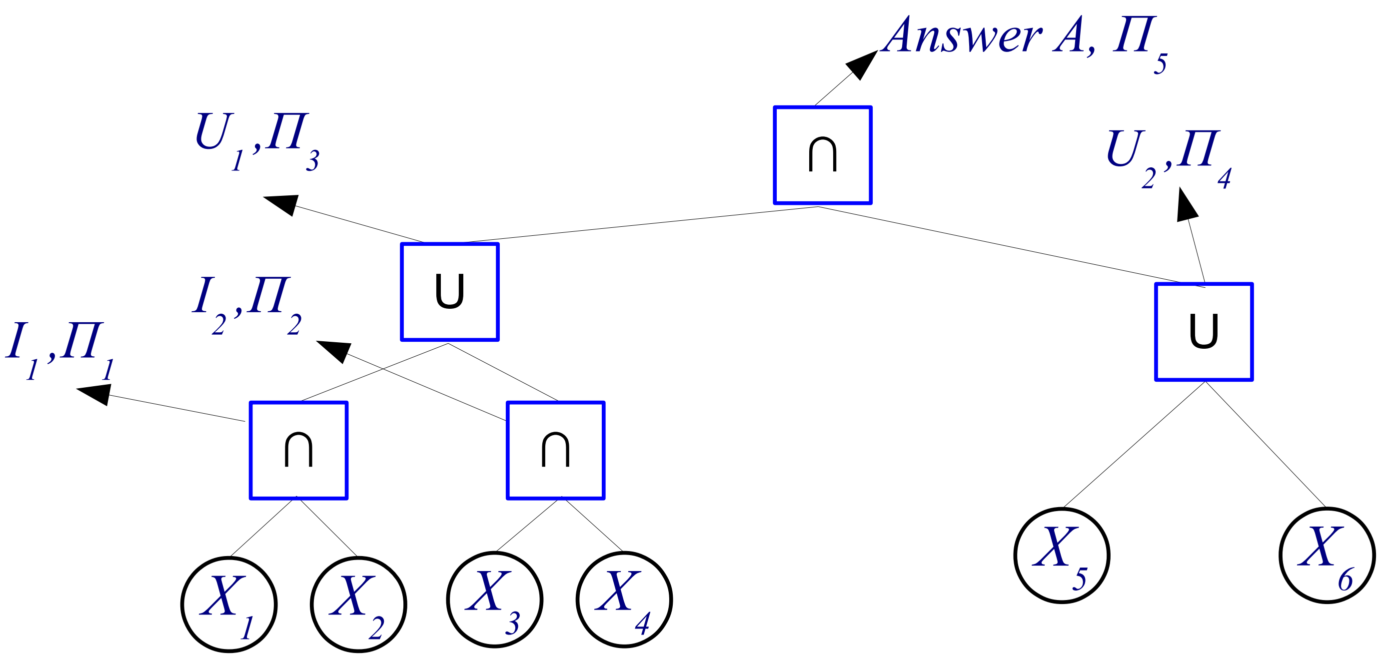
Main Idea (attempt 1)



$$\Pi = \{\ (I_1,\Pi_1),\ (I_2,\Pi_2),\ (U_1,\Pi_3),(U_2,\Pi_4),(A,\Pi_5)\ \}$$

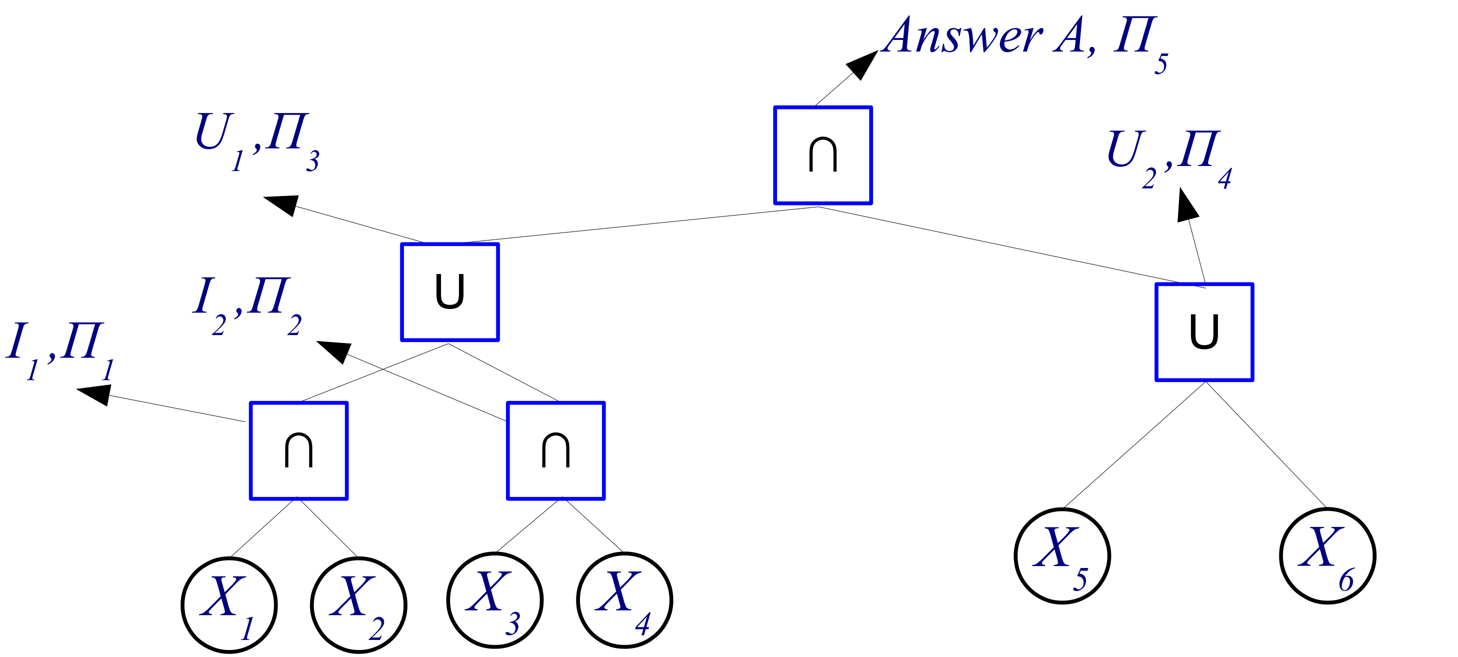
- Not efficient!
- Intermediate sets possibly much larger than answer

Main Idea (attempt 2)



$$\Pi = \{ (I_1,\Pi_1), (I_2,\Pi_2), (U_1,\Pi_3),(U_2,\Pi_4),(A,\Pi_5) \}$$

Main Idea (attempt 2)



$$\Pi = \{ (I_1,\Pi_1), (I_2,\Pi_2), (U_1,\Pi_3),(U_2,\Pi_4),(A,\Pi_5) \}$$

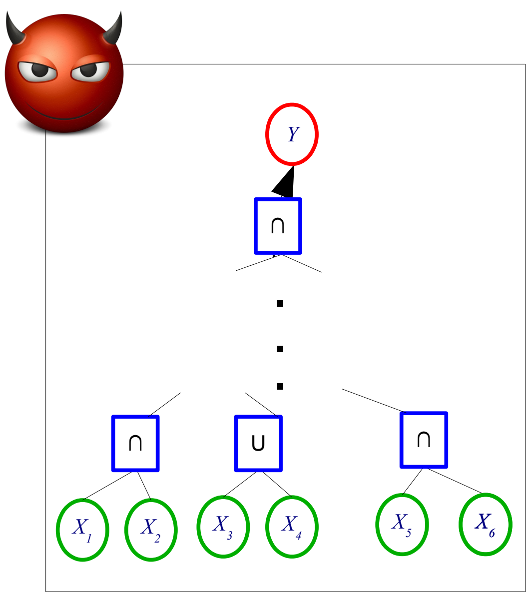
- Remove intermediate sets

Security Proof

- Soundness?
  - construct adversary for a single operation

Security Proof

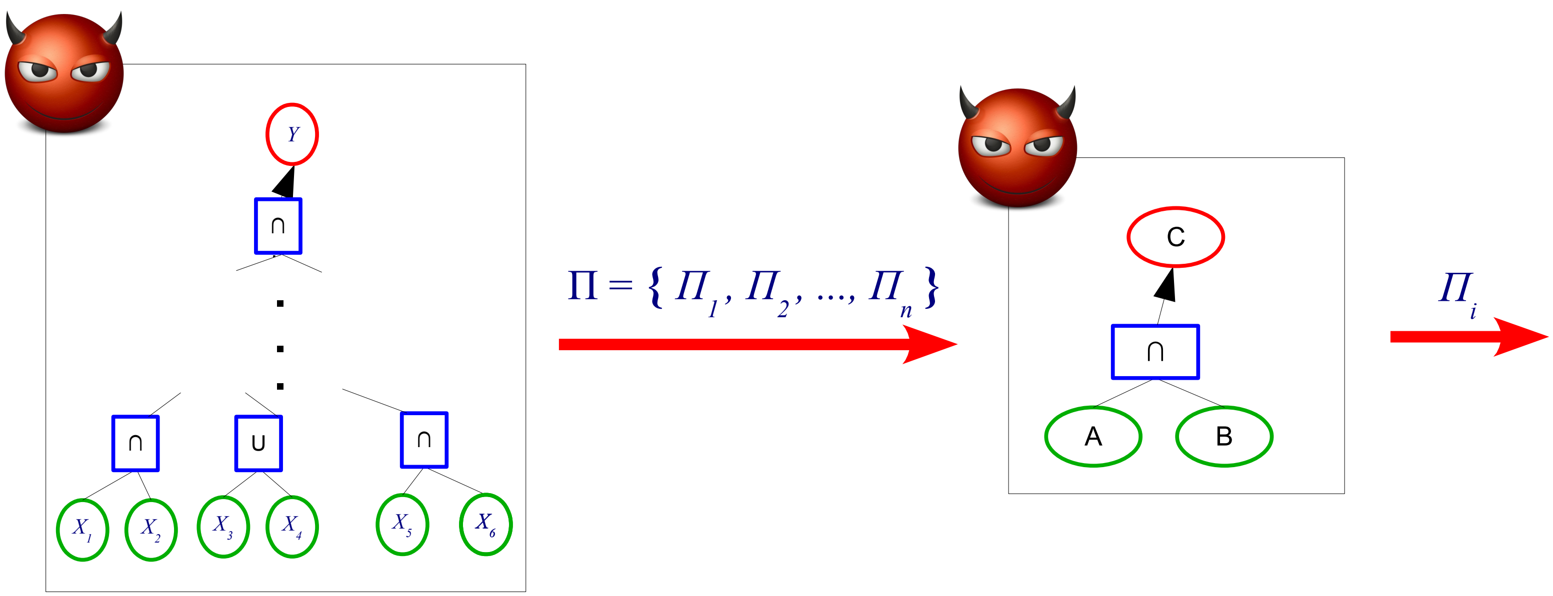
- Soundness?
  - construct adversary for a single operation



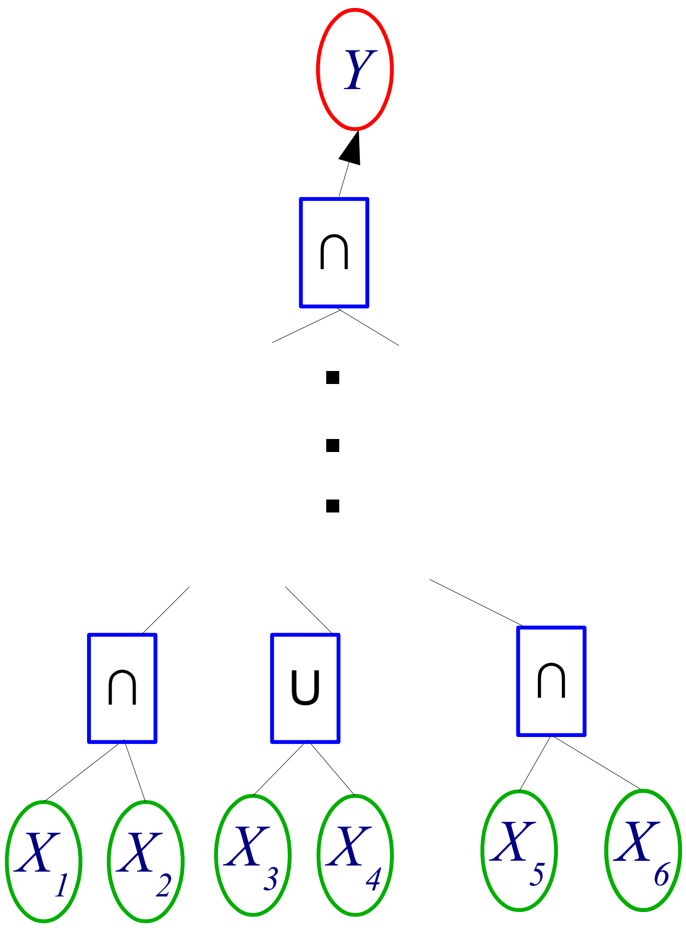
$$\Pi = \{ \Pi_1, \Pi_2, \ldots, \Pi_n \}$$

Security Proof

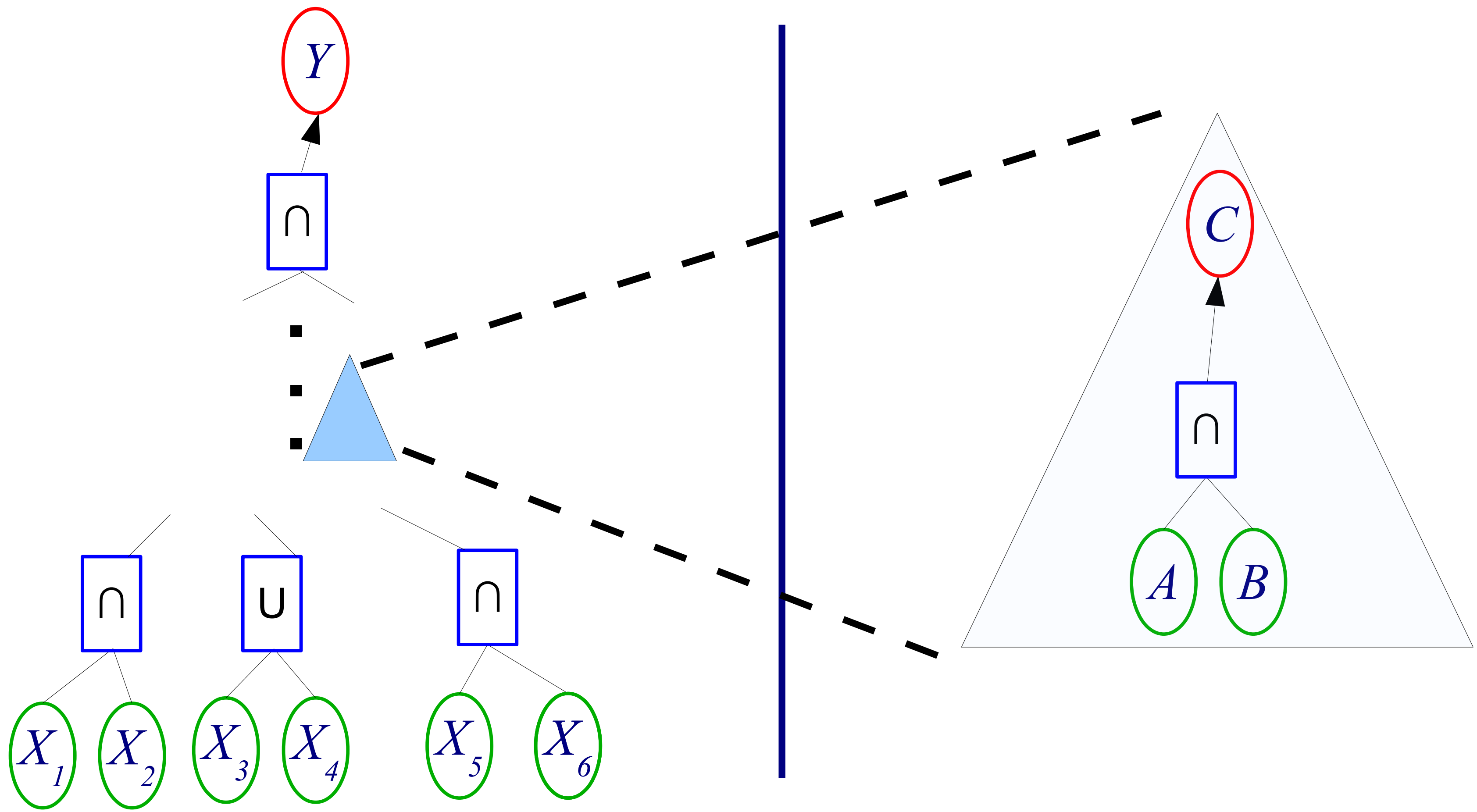
- Soundness?
  - construct adversary for a single operation



Security Proof

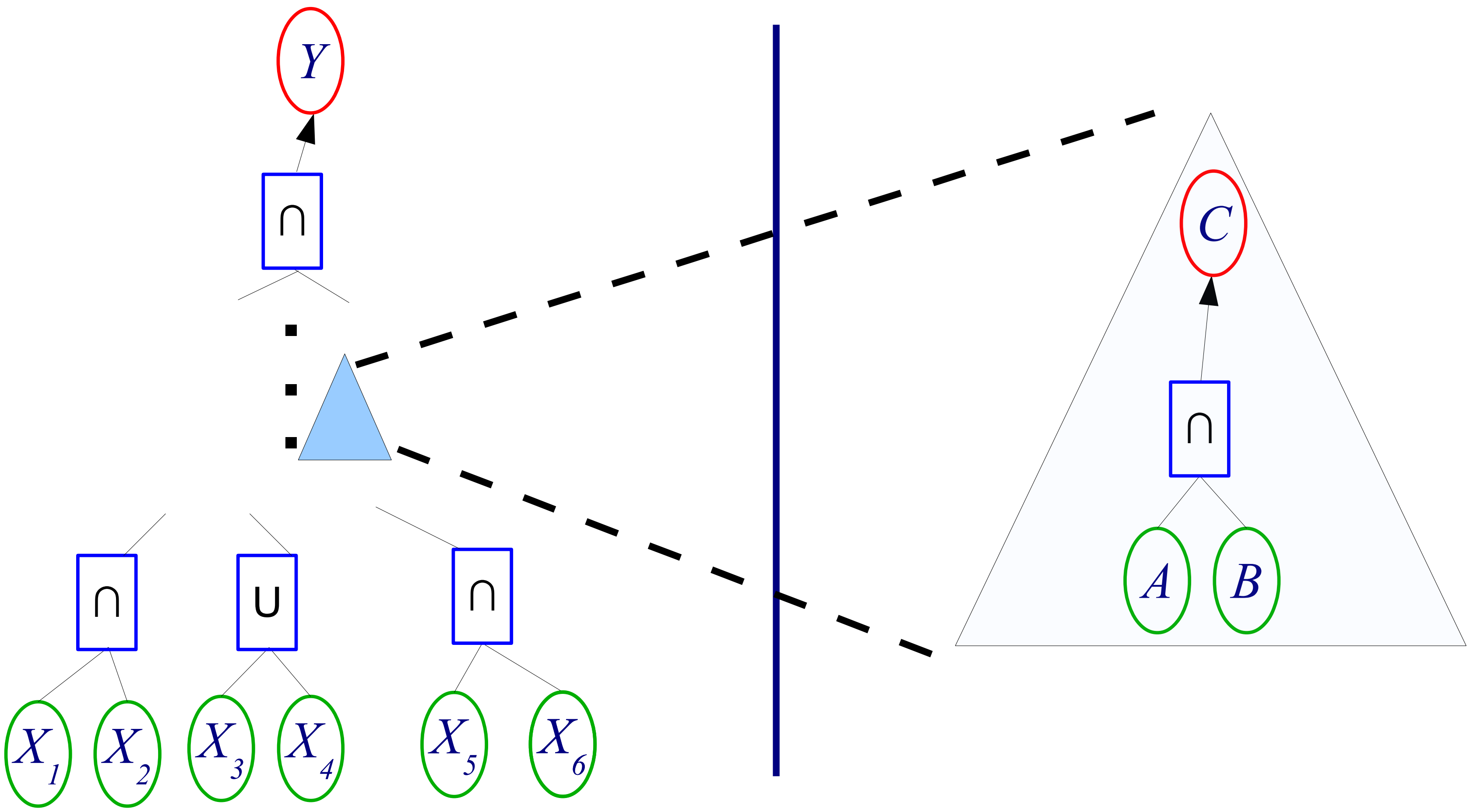


Security Proof



- Exists operation with
  honest input A,B, cheating output C and proof Π_i

Problem



- What is the value of set *C*?
  - even the adversary may not know!

Solution

- Replace proofs Π_i with proofs-of-knowledge

Solution

- Replace proofs Π_i with proofs-of-knowledge
- Proof of Knowledge (PoK)
  - For any convincing (cheating) prover

    ∃ extractor that outputs witness

Solution

- Replace proofs Π_i with proofs-of-knowledge
- Proof of Knowledge (PoK)
  - For any convincing (cheating) prover
    $\exists$ extractor that outputs witness
- Witness $\rightarrow$ cheating sets

PoK for Sets

- Construction based on q-*Knowledge of Exponent* assumption [Groth'10]

PoK for Sets

- Construction based on q-*Knowledge of Exponent* assumption [Groth'10]

- Constant size
  - only two additional group elements on Π_i

PoK for Sets

- Construction based on q-*Knowledge of Exponent* assumption [Groth'10]
- Constant size
  - only two additional group elements on Π_i
- Matches nicely with bilinear accumulators
  - “accumulators with knowledge”

Conclusion

- Verifiable Computation
  - numerous general solutions in literature
  - asymptotically excellent but not practical for general deployment yet

    (continuous improvements though... [SBV+'12],[PGHR'13],[BCGTV'13], etc.)

- Our work: a protocol for specific functions
  - sacrifice generality for practicality

- Follow-up [Kosba, Papadopoulos, Papamanthou, Sayed, Shi, Triandopoulos]
  - constant-size proofs
  - extends the **Q**uadratic **S**pan **P**rogram framework
  - server cost ~30x smaller than [PGHR'13]

Conclusion

- Verifiable Computation
  - numerous general solutions in literature
  - asymptotically excellent but not practical for general deployment yet

    (continuous improvements though... [SBV+'12],[PGHR'13],[BCGTV'13], etc.)

- Our work: a protocol for specific functions
  - sacrifice generality for practicality

- Follow-up [Kosba, Papadopoulos, Papamanthou, Sayed, Shi, Triandopoulos]
  - constant-size proofs
  - extends the **Q**uadratic **S**pan **P**rogram framework
  - server cost ~30x smaller than [PGHR'13]

Thank you!