Distributed Certified Information Access for Mobile Devices

Clemente Galdi
Universita' di Napoli "Federico II" - Italy

Joint with
Aniello Del Sorbo and Giuseppe Persiano
Universita' di Salerno - Italy

Motivating example

- Given a database answer to a query, can we "trust" the received information are the ones "actually contained" in the DB?
- Currently trust in database replies is ensured in different ways
  - Depends on the application scenario.
  - Sometimes it is obvious.
  - Sometimes it is guaranteed by the "trust" on the DB owner.
  - Sometimes it is due to some "third-party"

"Obvious" and "Owner-based" trust

- If the information are not sensitive/valuable, there is no need to give wrong answers.
  - E.g.: An online phonebook service.
  - Frequent failure of such services will decrease users' trust and "kill" the service.
- If the DB-owner has an "incentive" to give correct answers, the DB will always reply correctly
  - E.g., A traffic control service operated ("owned") by the local police.
  - Wrong answers to queries will create traffic jams (and policemen will have much more work to do).

Third-party-based trust

- Trust in ensured by some third-party.
  - E.g., Credit card billing. Everybody may gain money from transaction
    - The Card holder may try not to pay some expenses
    - A seller may try to gain more money by duplicating/modifying transactions
- Correctness (and Trust) is ensured by the known protocols
  - Such protocols cannot be implemented "easily" in a mobile environment

Motivating Example

Consider a service that, given a position, allows searching for "closest" shop of a given type:

- E.g, "Greek Restaurant" close to "current position"
- In general, and especially in a mobile environment, the user may not know/trust the service provider.
- The DB owner may be willing to reply by sending some "wrong" information
  - E.g., Only "Italian restaurants" that provide free food to the service owner.
- The user obtains information that do not match the actual content of the database.

Certified Information Access

- In a CIA service, each reply consists of
  - The content of the database
  - A proof that the answer is consistent with the content of the database
- The proof has to be verified against some public information generated <u>before</u> the query was issued.
- The DB cannot give wrong answer!
  - Unless he can generate a verifiable proof.

Certified Information Access

- Trivial (insecure and useless) implementation: Publish the DB.
  - Privacy of the information is lost.
  - The user may verify the correct answer by checking the public copy of the DB.
  - Communication complexity is linear in the DB size.

Certified Information Access

- Prerequisites:
  - DB privacy should be guaranteed
  - Communication complexity should be as low as possible
  - Users should be guaranteed of answer correctness
  - (After an initialization phase) operations should not be computational intensive

Certified Information Access

- Parties:
  - Certified DB Owner: Controls the DB. Publishes a "secure snapshot" of its content before starting answering queries
  - User: Issues queries and verifies the answers
  - PubInformationStorage: Generates "public parameters" and publicly stores DB snapshot

CIA via Commitments

- A commitment for a message m is a pair (com, dec)
  - com corresponds to a safe containing m.
  - To open the commitment com, it is enough to send m and dec.
  - The receiver verifies that com is consistent with m and dec.
  - Given com, it is infeasible to:
    - "Change" the message m. (User guarantee)
    - Compute information on the value of m (DB privacy)
- Build a tree using binary representation of keys:
  - Leaves are commitments of DB entries
  - Internal nodes are commitments of concatenation of their children
- Problem: Exponential size.
  - Need to assign a special symbol to non-existing entries.
  - The tree must be complete.

Mercurial Commitments

- MC are variants of classical commitments.
- HardCommit(m,W)->(com, dec):
  - Given the public parameters W, creates a commitment to the string m
  - Correspond to "classical" commitments.
- SoftCommit(W)->(scom, sdec):
  - Does NOT take any message as input.
  - Creates a commitment scom that can be associated to any string
- Hard and Soft commitments are indistinguishable
  - Given com (or scom), it is impossible to say whether it is a hard or a soft commitment.

Mercurial Commitments

- Tease(m,tcom,W)->tdec:
  - computes the teasing ("proof") that tcom is a MC for m.
  - If tcom is a hard commitment, teasing is possible only m is the "original" message used for creating tcom.
  - If tcom is a soft commitment, teasing is possible for every message m.
- VerifyOpen(m, dec, com, W): Verifies that m and dec are <u>consistent</u> with com
  - Only hard commitment can be opened
- VerifyTease(m, tdec, tcom, W): Verifies that m and the teasing tdec are consistent with the commitment tcom.

MC-Implementation

- Public parameters: $W=(g,h,p)$
  - g,h generators of Z_p^*
- HardCommit $(m,W)=(com,dec)$:
  - $com=(g^m(h^r)^s, h^r)$
  - $dec=(s,r)$
- SoftCommit $(W)=(scom,sdec)$:
  - $scom=(g^s, g^r)$
  - $sdec=(s,r)$

Indistinguishable: Both hard and soft commitments are pairs of random elements in Z_p^*

MC-Implementation

- VerifyOpen(m, dec=(S, R), com=(C_0,C_1), W) returns true iff
  - $C_0 = g^m C_1^S$
  - $C_1 = h^R$
  - Note that VerifyOpen fails for soft commitments.
- Tease for a Hard commitment
  - Tease(m, com, dec=(s,r),W)=s
- Tease for a Soft commitment
  - Tease(m, scom, sdec=(s,r),W) =(s-m)/r mod p-1
- VerifyTease(m, t, (C_0,C_1),W) returns true iff
  - $C_0 = g^m C_1^T$

CIA via MC

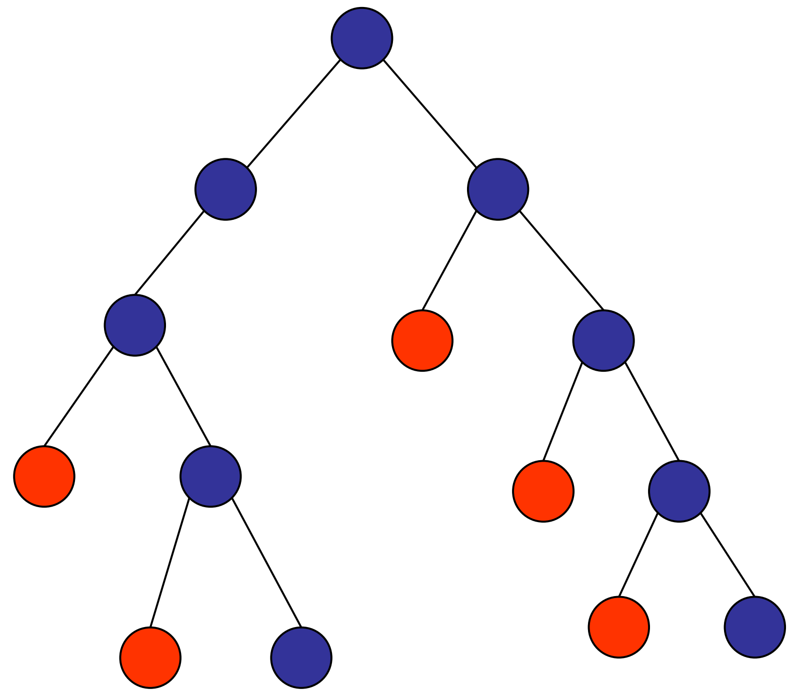
- Each information in the tree can be seen as (key, value).
- The binary representation of key identifies a path in a binary tree.
- The tree is constructed starting from the leaves containing HC(key,value)
- Internal nodes contain hard commitments of the content of their children.
  - Only if at least one child "exists"
- "Missing" leaves/internal nodes contain soft commitments.
- No need of building a complete tree!

CIA via MC

- The root is a hard commitment.
  - The DB cannot cheat!

Hard Commitments

Soft Commitments



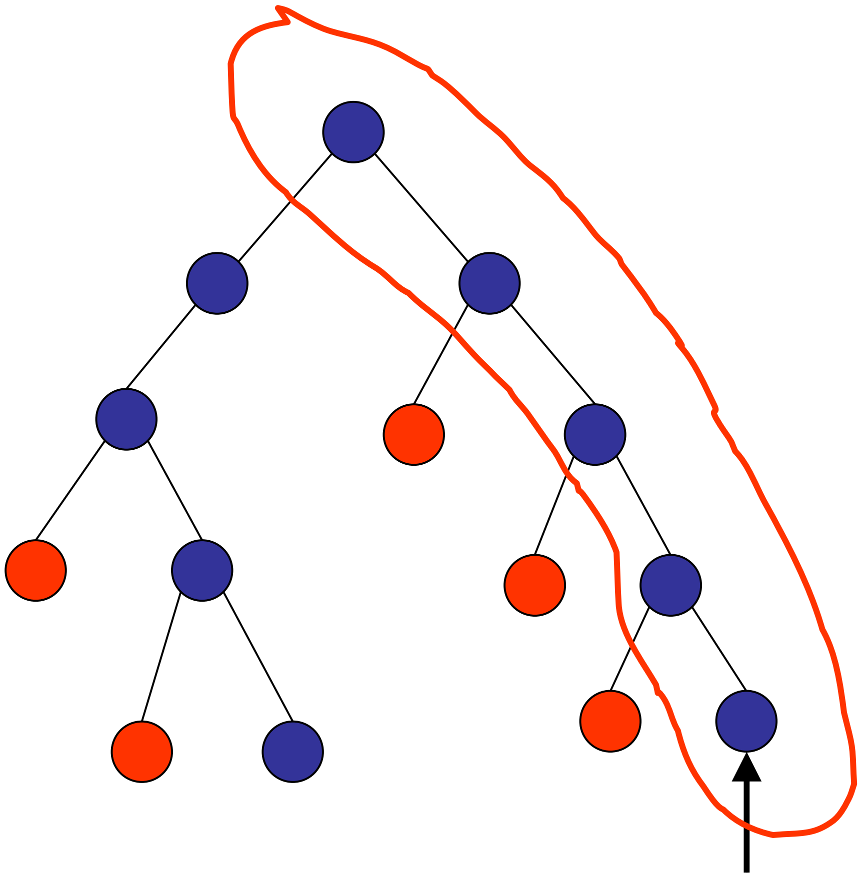
CIA via MC

Querying an element that belongs to the DB:

The DB replies with opening of the hard commitments on the path

The User uses "VerifyOpen"

HC cannot be changed.



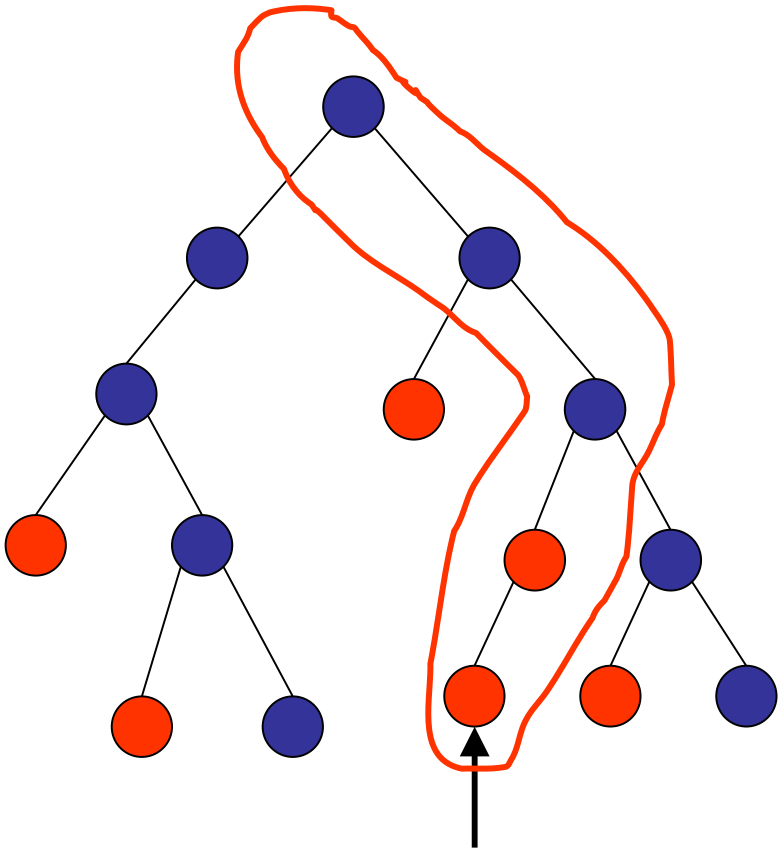
CIA via MC

Querying an element that does NOT belong the DB:

The DB builds (and stores) the "missing" path using soft commitments.

The DB replies with teasing of the nodes on the path

The User uses "VerifyTease"



CIA via MC

- Querying an existing element:
  - The proof consists of opening of Hard commitments.
  - The DB cannot cheat the user.
- Querying a non-existing element:
  - The path contains hard commitments follows by some soft commitments.
  - In particular the first soft commitment in the list
    - Has been created before the query was issued.
    - Its parent in the tree consists of a hard commitment.
  - The user expects to see teasing of commitments.

Distributing CIA

- Problem: modular exponentiation is computationally intensive.
- We need to compute:
  - com=$(g^m (h^r)^s, h^r)$
    - m must be kept secret
    - r and s need to be secret
    - In $(h^r)^s$, both the base (h^r) and the exponent need to be secret.
      - An observer recognize the pair as being a hard commitment.
  - scom=(g^s, g^r)
    - s and r need t be secret
  - Similar arguments hold for the verification.

Distributing CIA

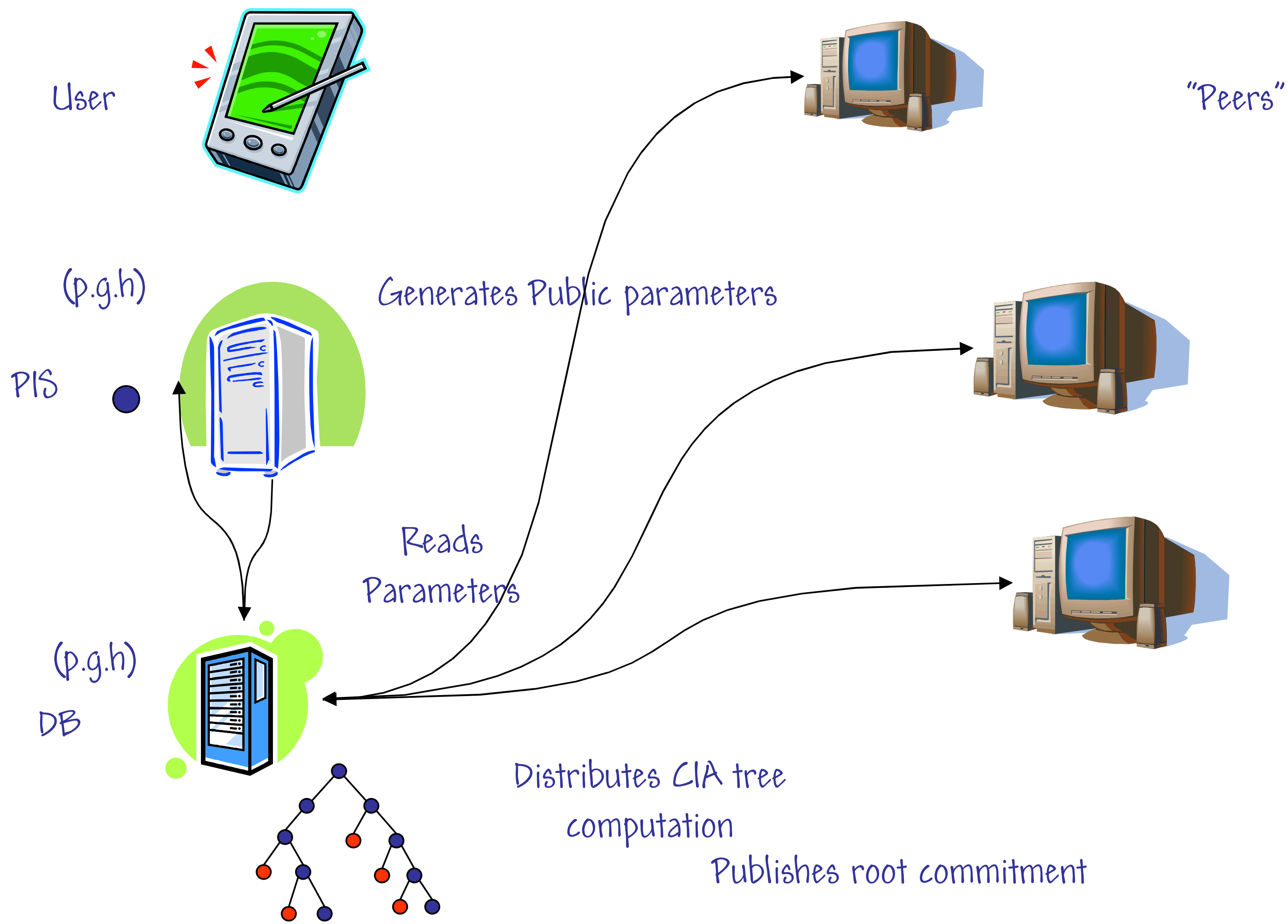
- Secure__Exp(b,e,p,k) distributes the computation of (b^e mod p) securely among k players
  - e has to be private
    - $e=e_1+\ldots+e_k$ mod p-1
    - Require exponentiation of $c_i=(b,e_i)$
    - Compute $c=c_1\ldots c_k$
- Secure__Base__Exp(b,e,p,k) distributes the computation of (b^e mod p) securely among k^2 players
  - Both b and e has to be private.
    - Share both the exponent and the base.
- All exponentiations of the form g^r and h^r can be pre-computed

Distributing CIA

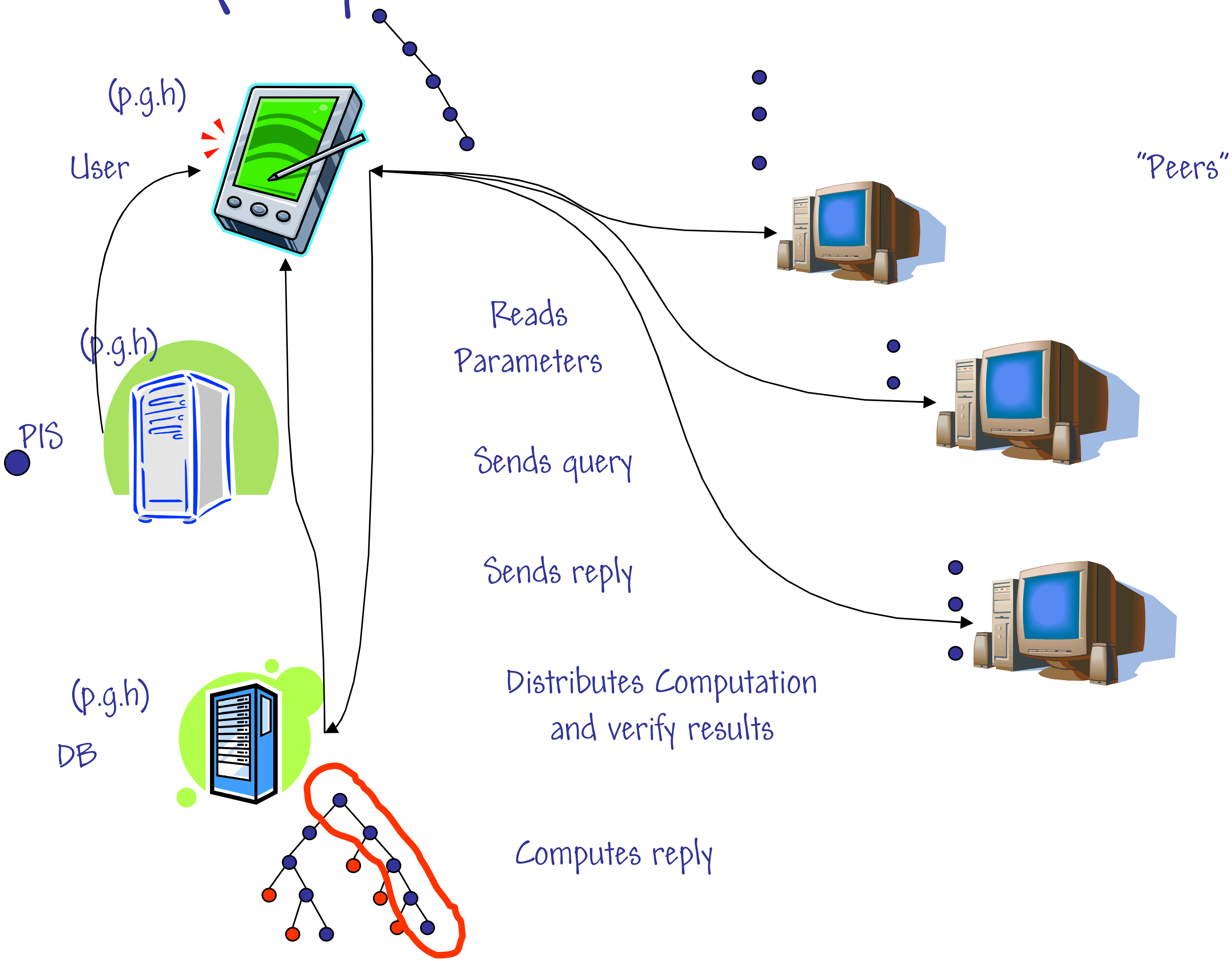
- Given the above primitives, it is possible to distribute all CIA operations.
- DB and User just need to compute modular summations and multiplications.

```
Procedure Commit(m,W,k)
   pick r,s in Z_{p-1}
   w=Secure_Exp(g,m,p,k)
   y_1=Secure_Exp(h,r,p,k)
   y_0=Secure_Base_Exp(y_1,s,p,k)
   return com=(wy_0, y_1), dec=(s,r)
```

Tree construction



User Query


(p.g.h)
User
"Peers"
Reads
Parameters
(p.g.h)
PIS
Sends query
Sends reply
(p.g.h)
DB
Distributes Computation
and verify results
Computes reply

Conclusions

- We have introduced the concept of Certified Information Access.
  - Such primitive can be implemented by using Mercurial Commitments
- We have shown a distributed architecture that can be used to implement it.
- Currently working on distributed implementation of verification with pre-computation.
- Open Problems:
  - Is it possible to implement CIA by using other primitives ?
  - Design of "efficient" dynamic MC schemes. (Current MC are "efficient" only for static DB).