Memento: How to Reconstruct your Secrets from a Single Password in a Hostile Environment

Jan Camenisch, Anja Lehmann, Anna Lysyanskaya, Gregory Neven
IBM Research Zurich & Brown University

Motivation

How to secure personal data

without assuming trusted user storage ?

Motivation

How to secure personal data

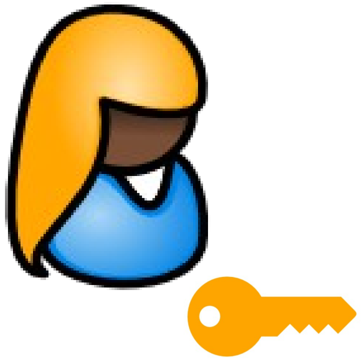
without assuming trusted user storage ?

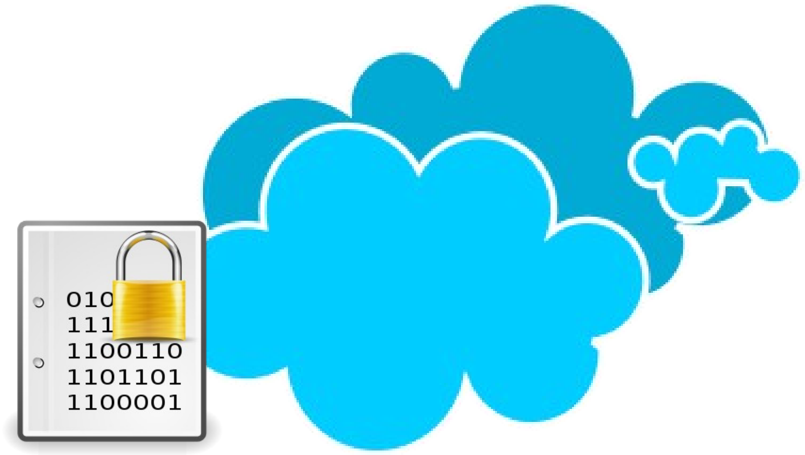
upload encrypted data to the cloud

Motivation

How to secure personal data

without assuming trusted user storage ?

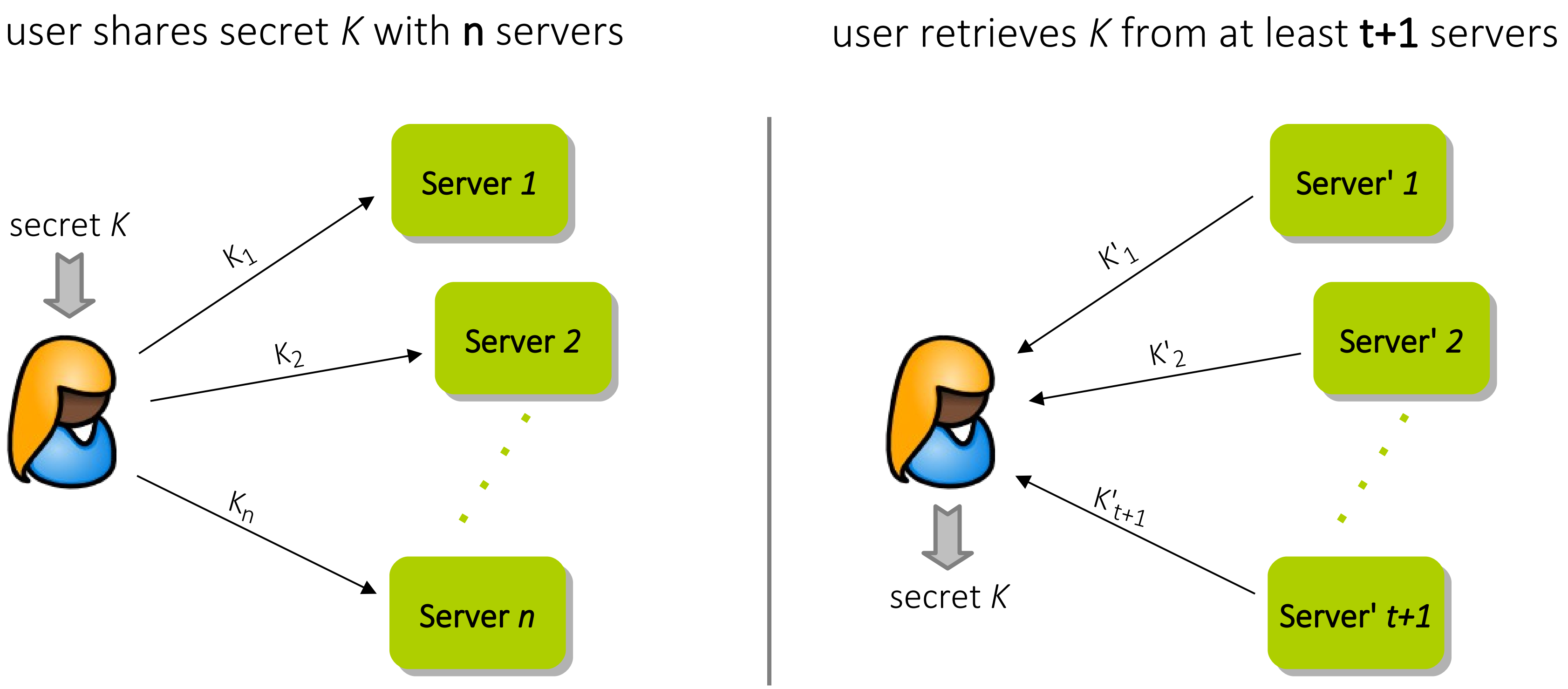




upload encrypted data to the cloud

How to secure the decryption key?

Introduction – Secret Sharing

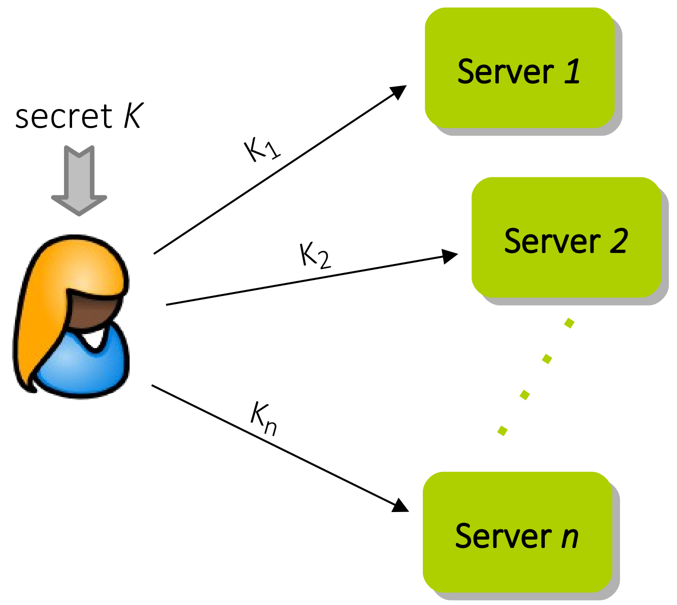


t+1 shares needed to reconstruct *K*

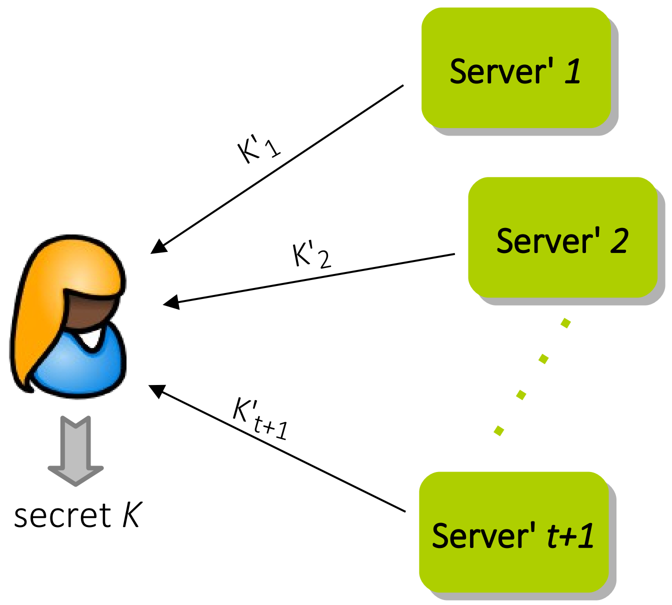
if at most t servers are corrupt → they don't learn anything about *K*

Introduction – Secret Sharing

user shares secret *K* with **n** servers



user retrieves *K* from at least **t+1** servers



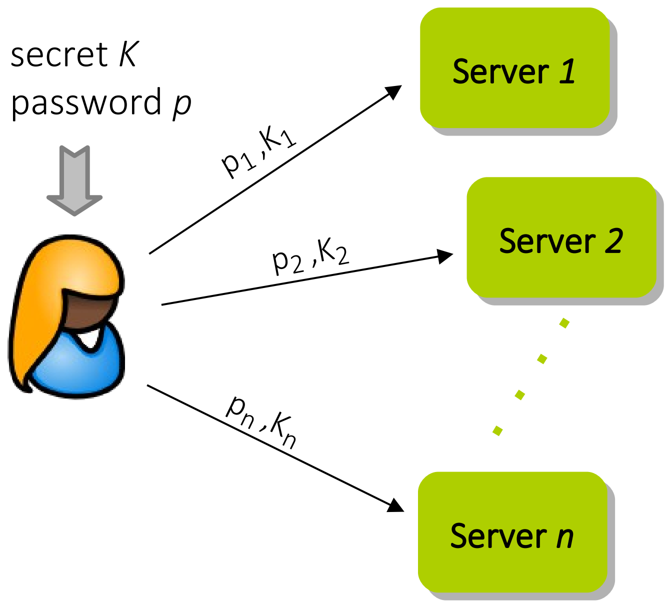
How to ensure it's the legitimate user?

t+1 shares needed to reconstruct *K*

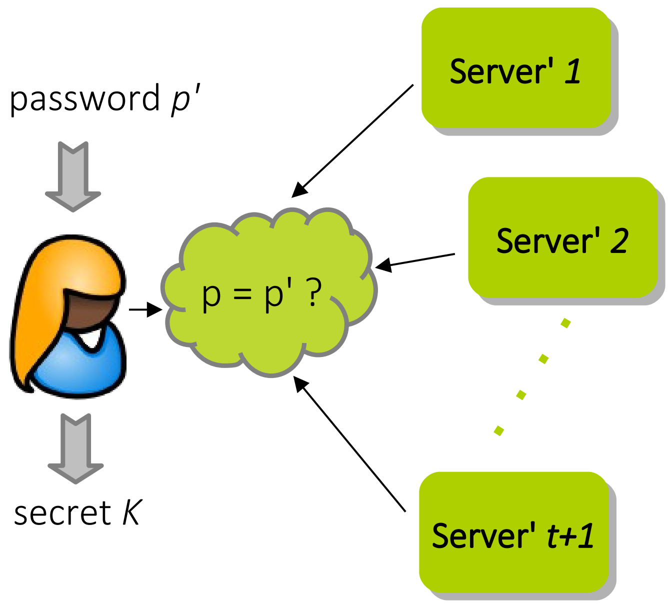
if at most t servers are corrupt → they don't learn anything about *K*

Introduction – Password-Authenticated Secret Sharing

user shares secret *K* with **n** servers protected by password *p*

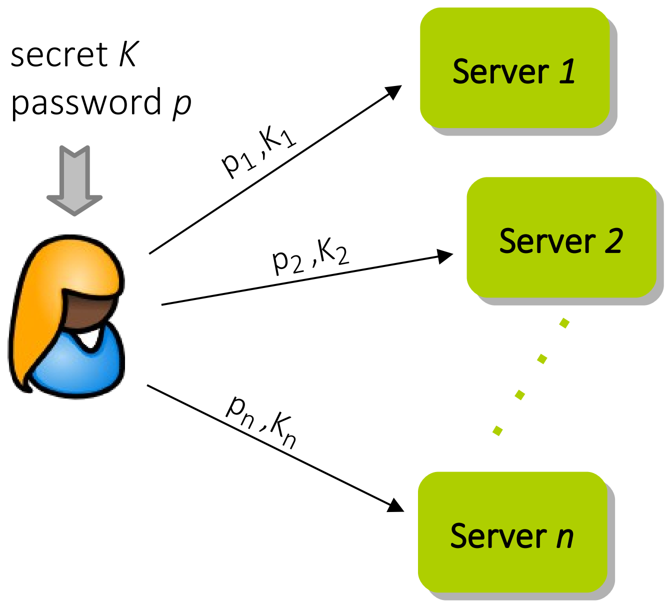


user retrieves *K* from at least **t+1** servers using password *p'*

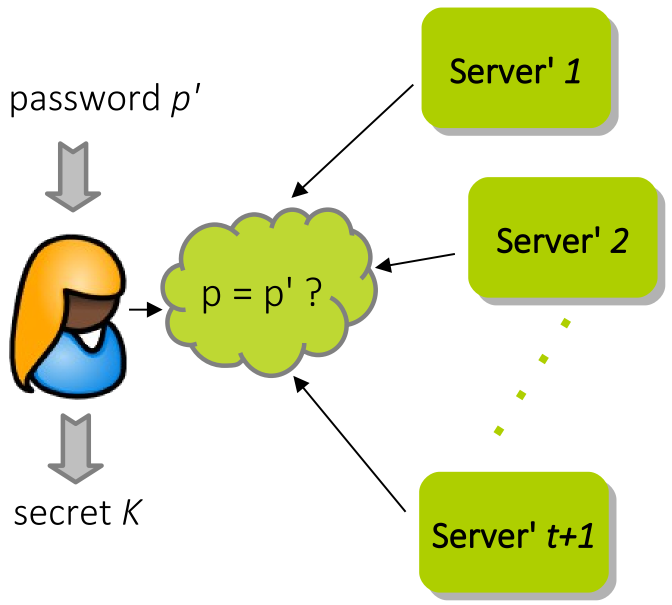


Introduction – Password-Authenticated Secret Sharing

user shares secret *K* with **n** servers protected by password *p*



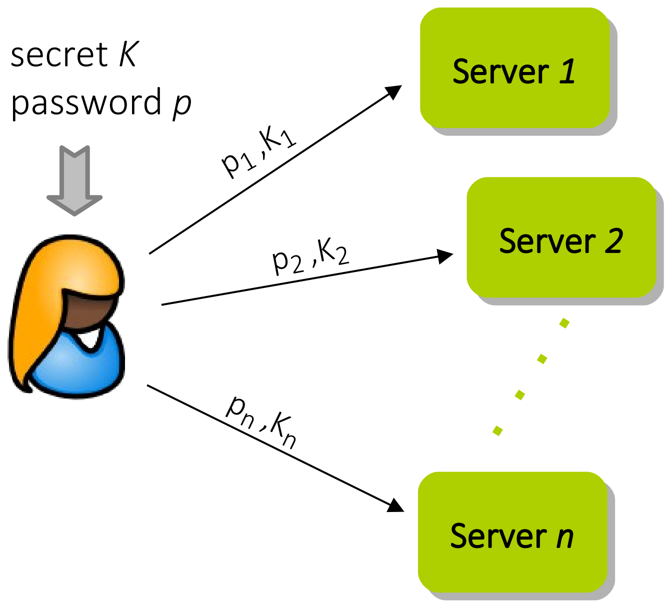
user retrieves *K* from at least **t+1** servers using password *p'*



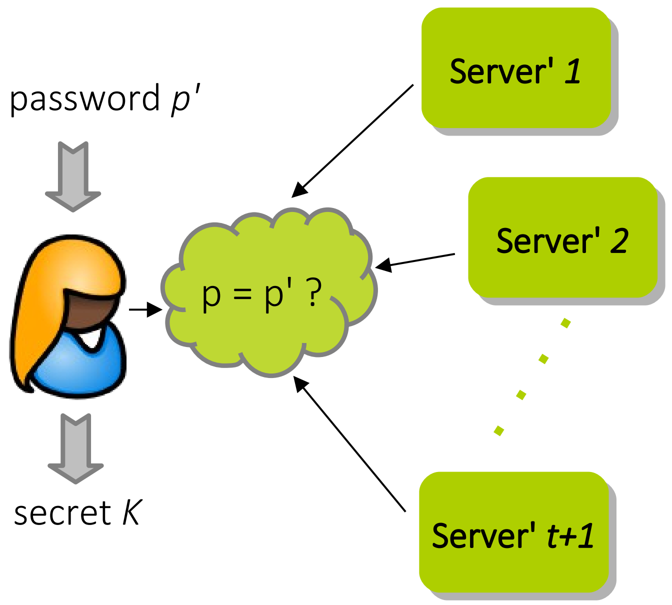
Aren't passwords a really, really bad idea ?

Introduction – Password-Authenticated Secret Sharing

user shares secret *K* with **n** servers protected by password *p*



user retrieves *K* from at least **t+1** servers using password *p'*

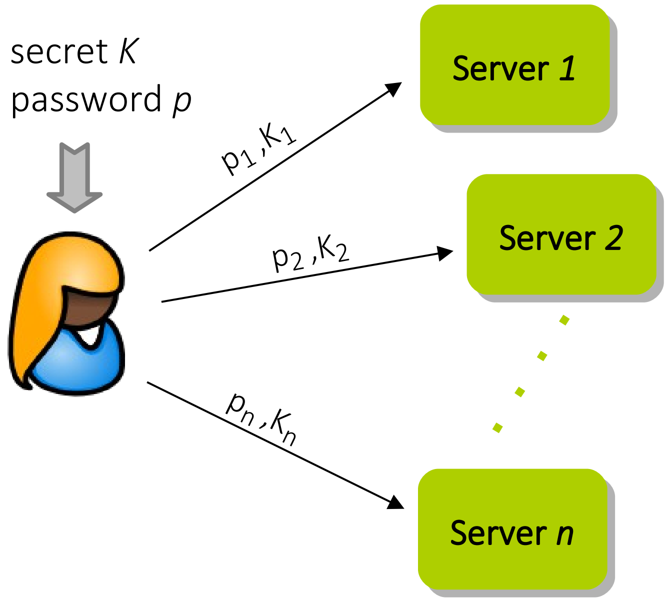


Aren't passwords a really, really bad idea ?

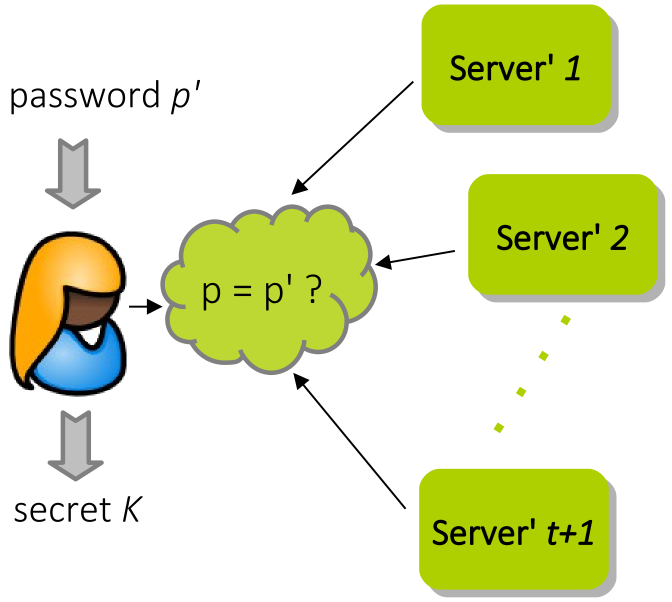
No, not if offline attacks can be prevented!

Introduction – Password-Authenticated Secret Sharing

user shares secret *K* with **n** servers protected by password *p*



user retrieves *K* from at least **t+1** servers using password *p'*



Aren't passwords a really, really bad idea ?

No, not if offline attacks can be prevented!

t+1 shares needed to reconstruct *K* and to verify whether *p = p'*

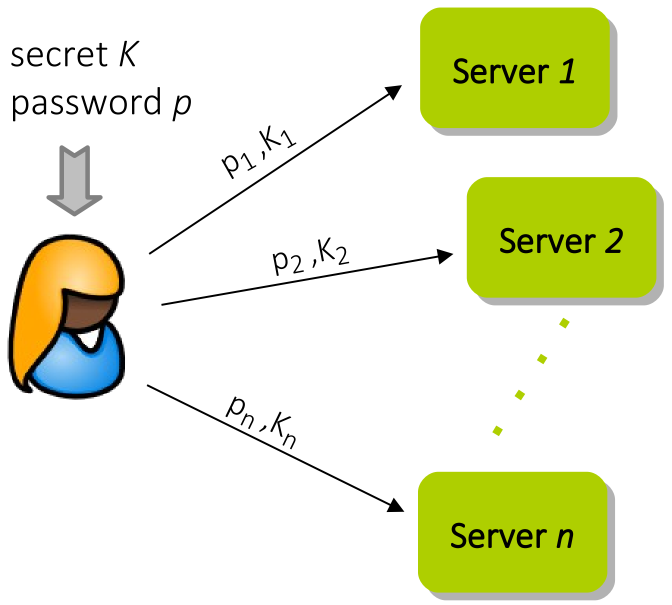
if at most t servers are corrupt → they don't learn anything about *K* or can offline attack *p*

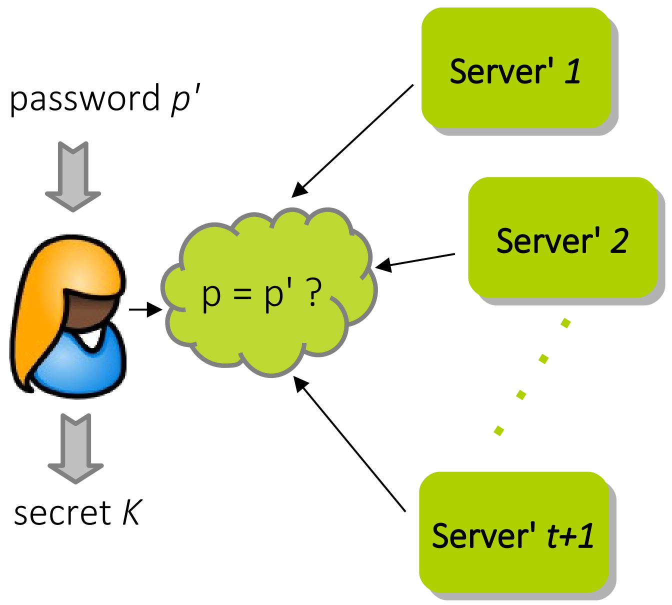
honest server throttle verification after too many (failed) attempts

Introduction – Password-Authenticated Secret Sharing

user shares secret *K* with **n** servers protected by password *p*

user retrieves *K* from at least **t+1** servers using password *p'*





Existing Solutions:

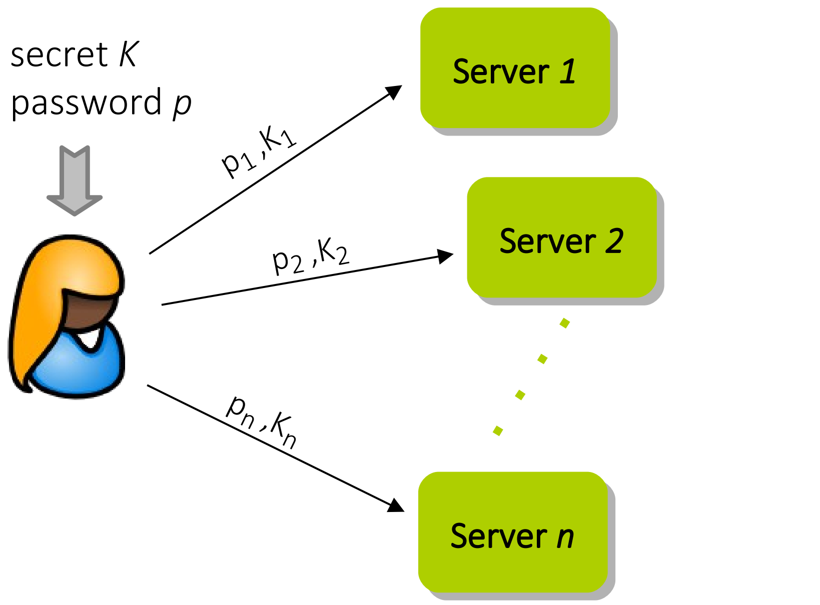
[BJSL'11] Bagherzandi, Jarecki, Saxena, Lu. *Password-protected secret sharing.* CCS 2011

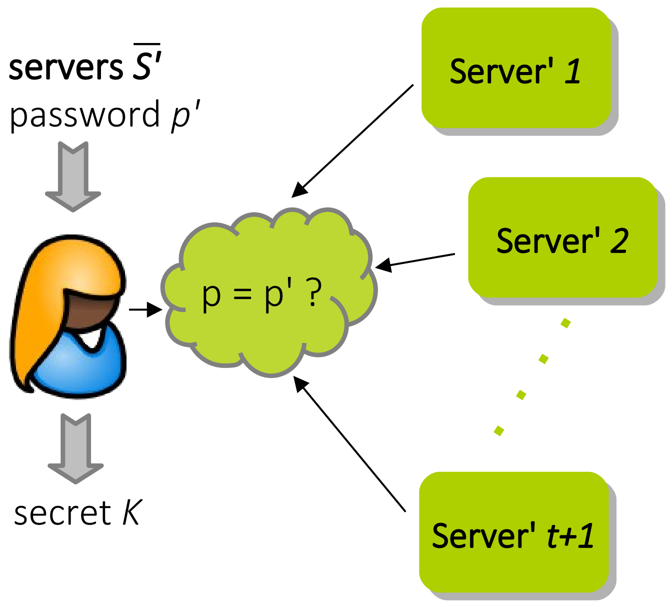
[CLN'12] Camenisch, Lysyanskaya, Neven. *Practical yet universally composable secure two-server password-authenticated secret sharing.* CCS 2012

Introduction – Password-Authenticated Secret Sharing

user shares secret *K* with **n** servers protected by password *p*

user retrieves *K* from at least **t+1** servers using password *p'*





Existing Solutions:

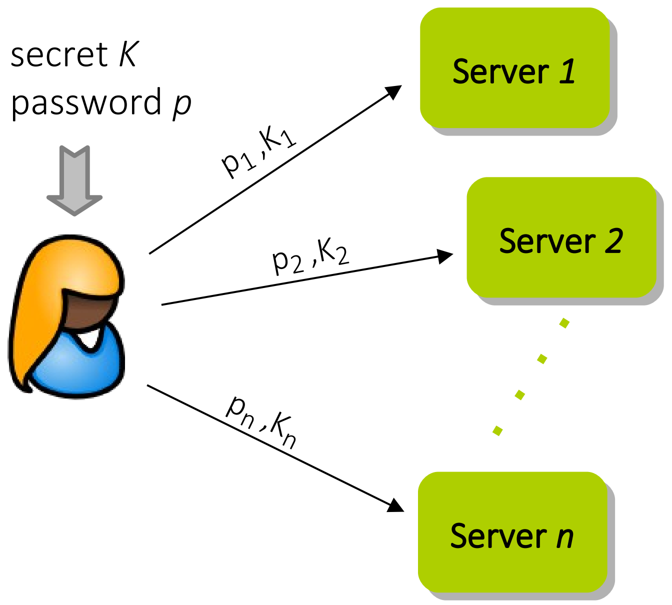
[BJSL'11] Bagherzandi, Jarecki, Saxena, Lu. *Password-protected secret sharing.* CCS 2011

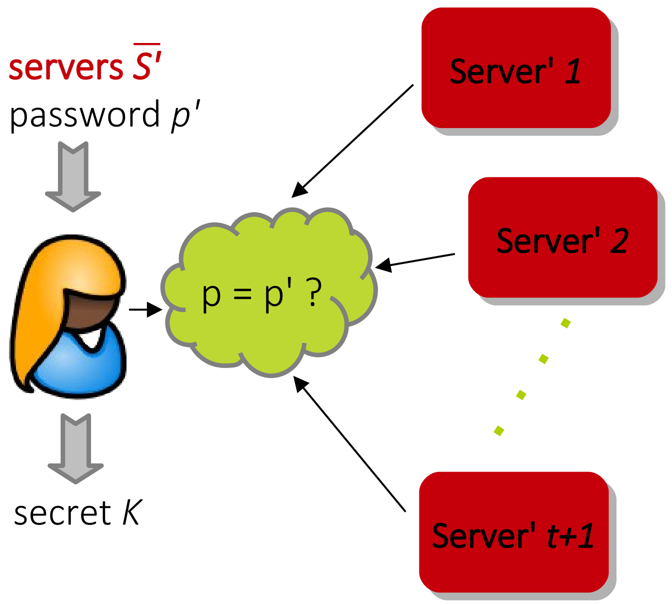
[CLN'12] Camenisch, Lysyanskaya, Neven. *Practical yet universally composable secure two-server password-authenticated secret sharing.* CCS 2012

Introduction – Password-Authenticated Secret Sharing

user shares secret *K* with **n** servers protected by password *p*

user retrieves *K* from at least **t+1** servers using password *p'*



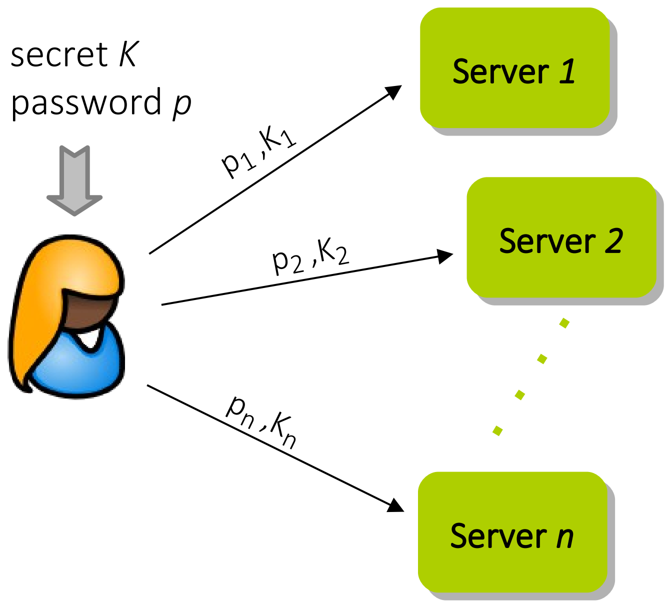


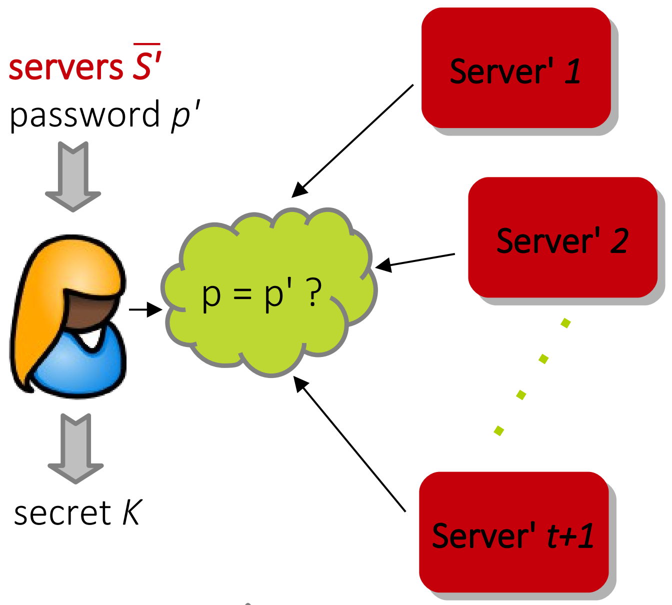
if user gets tricked into retrieval with t+1 corrupt servers
→ password p' is leaked

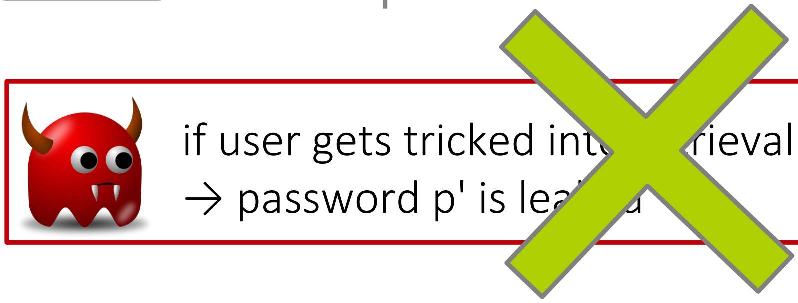
Introduction – Password-Authenticated Secret Sharing

user shares secret *K* with **n** servers protected by password *p*

user retrieves *K* from at least **t+1** servers using password *p'*







if user gets tricked int[illegible]rieval with t+1 corrupt servers
→ password p' is lea[illegible]d

Our Work

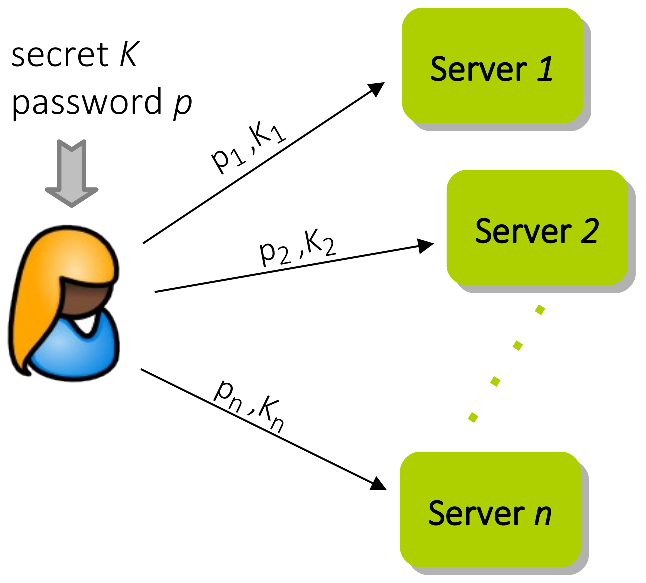
TPASS without trusted user storage

- threshold (t-out-of-n) password-authenticated secret sharing (TPASS)
  - user only needs to remember username & password
  - no requirement of trusted user storage
  - retrieval with all bad servers does not leak password

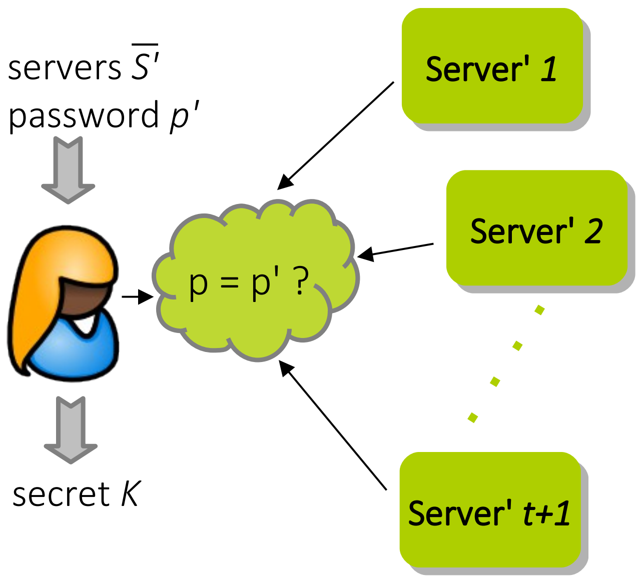
- UC definition for (t,n)-TPASS without trusted user storage
  - UC seems more natural than a property-based definition
    environment chooses passwords & password attempts
    → no assumptions on distributions, typos covered
  - composition with other protocols (e.g., to use K to decrypt data)

Our Work – TPASS without trusted user storage

user shares secret *K* with **n** servers protected by password *p*



user retrieves *K* from **t+1** servers using password *p'*



- if at most t servers are corrupt, Adv does not learn anything about p,K

- Adv learns if p=p' only if *all* honest server in $\overline{S'}$ cooperate (throttling)
- if all t+1 servers are corrupt, Adv only gets a single guess against p'
- Adv cannot set up user with wrong K* (unless t+1 servers are corrupt & guessed p')

Our Protocol (High-Level-Idea)

- similar idea as in [BJSL'11]
  - & removing need of trusted user storage
  - & making the protocol UC secure

- main building block:
  - (t; n)-threshold homomorphic encryption scheme

    $\mathrm{TKGen}(1^k) \rightarrow pk, sk_1, \ldots, sk_n$

    $\mathrm{TEnc}(pk, m) \rightarrow C$

    $\mathrm{PDec}(sk_i, C) \rightarrow d_i$

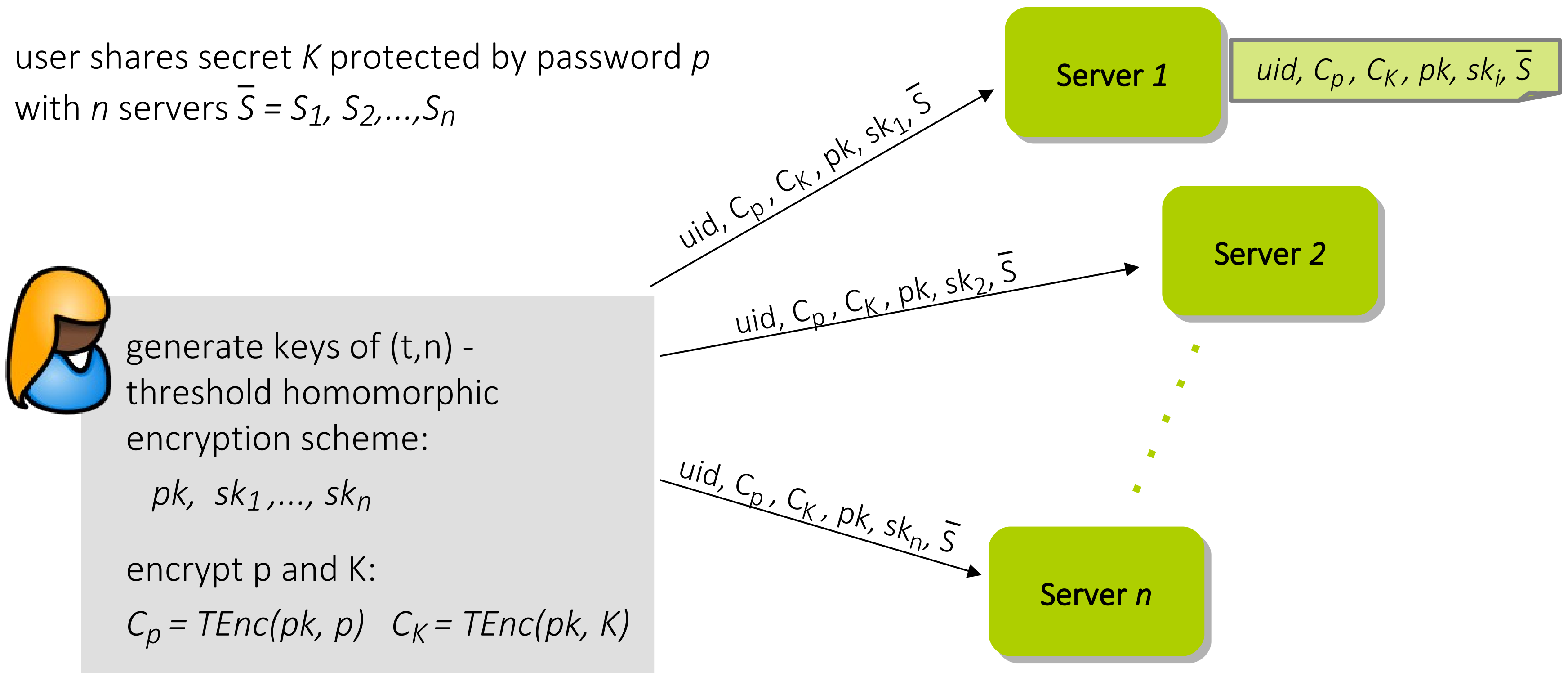
    $\mathrm{TDec}(C, d_1, \ldots, d_{t+1}) \rightarrow m$

    homomorphism: $C_1 = \mathrm{TEnc}(pk, m_1)$ and $C_2 = \mathrm{TEnc}(pk, m_2)$

    $\rightarrow C_1 \odot C_2 = \mathrm{TEnc}(pk, m_1 \cdot m_2)$

Our Protocol (High-Level-Idea) | **Setup**

user shares secret K protected by password p
with n servers $\bar{S} = S_1, S_2, \ldots, S_n$


Server 1
$uid, C_p, C_K, pk, sk_i, \bar{S}$
$uid, C_p, C_K, pk, sk_1, \bar{S}$
Server 2
$uid, C_p, C_K, pk, sk_2, \bar{S}$
Server n
$uid, C_p, C_K, pk, sk_n, \bar{S}$

generate keys of (t,n) - threshold homomorphic encryption scheme:

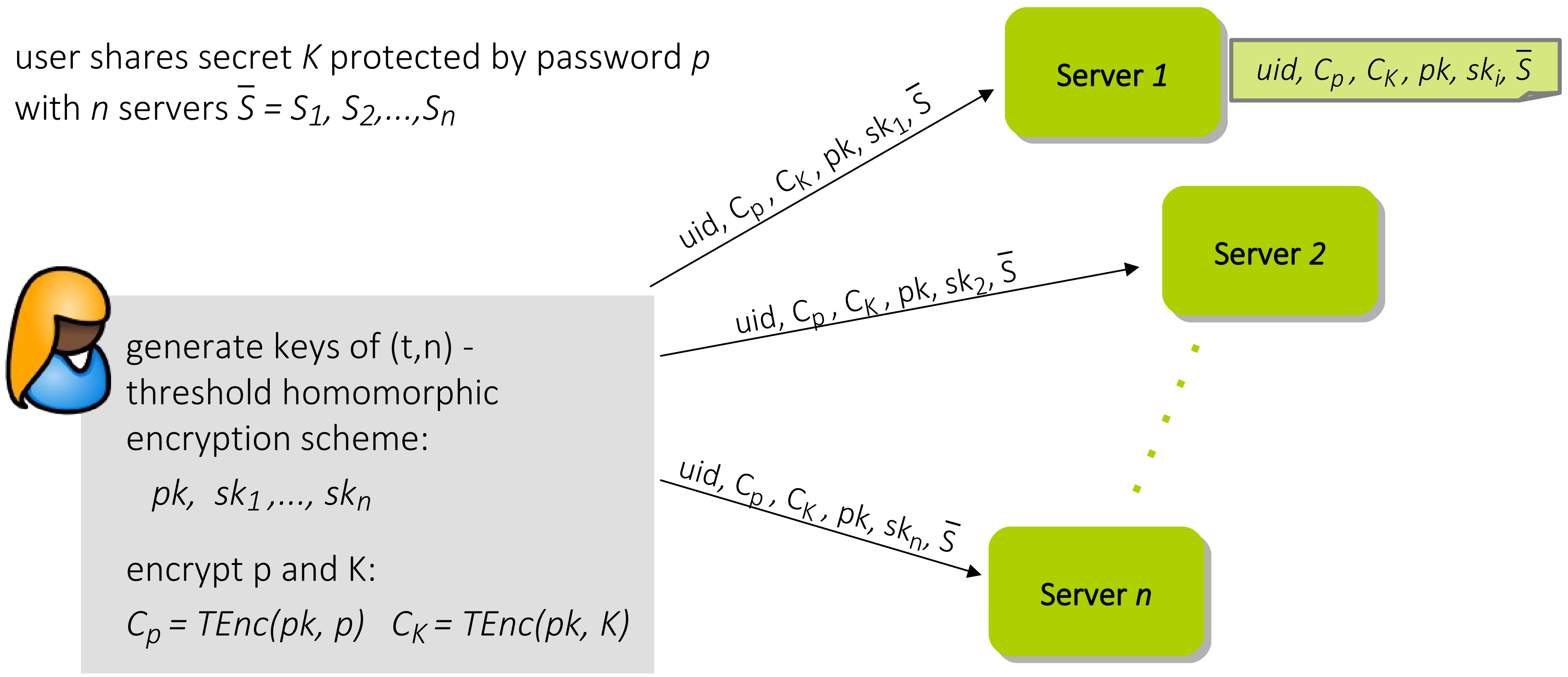
$pk, \; sk_1, \ldots, sk_n$

encrypt p and K:

$C_p = TEnc(pk, p) \quad C_K = TEnc(pk, K)$

Our Protocol (High-Level-Idea) | **Setup**

user shares secret K protected by password p
with n servers $\bar{S} = S_1, S_2, \ldots, S_n$



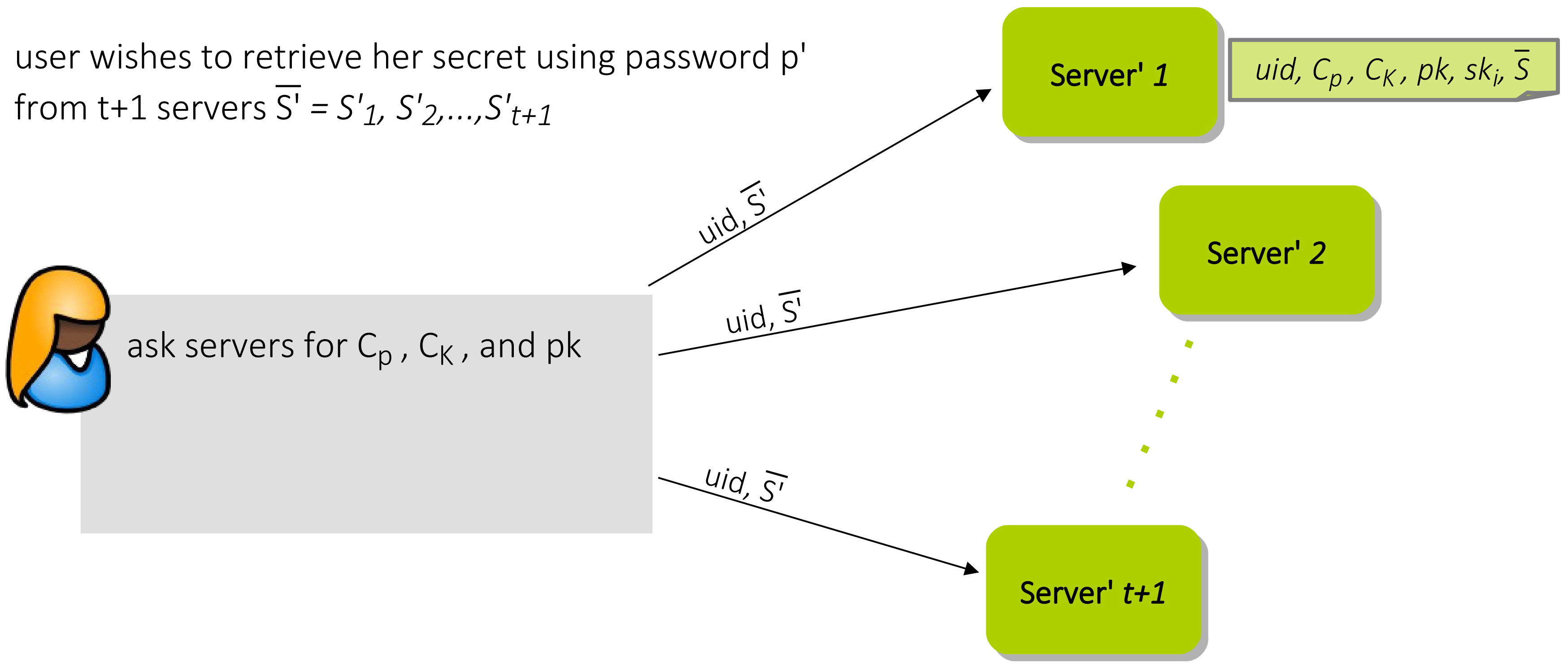
generate keys of (t,n) - threshold homomorphic encryption scheme:
$pk,\ sk_1, \ldots, sk_n$
encrypt p and K:
$C_p = TEnc(pk, p) \quad C_K = TEnc(pk, K)$
$uid, C_p, C_K, pk, sk_1, \bar{S}$
$uid, C_p, C_K, pk, sk_2, \bar{S}$
$uid, C_p, C_K, pk, sk_n, \bar{S}$
Server *1*
Server *2*
Server *n*
$uid, C_p, C_K, pk, sk_i, \bar{S}$

if ≤ **t** servers are corrupt:

p & K secure by semantic security of threshold encryption scheme

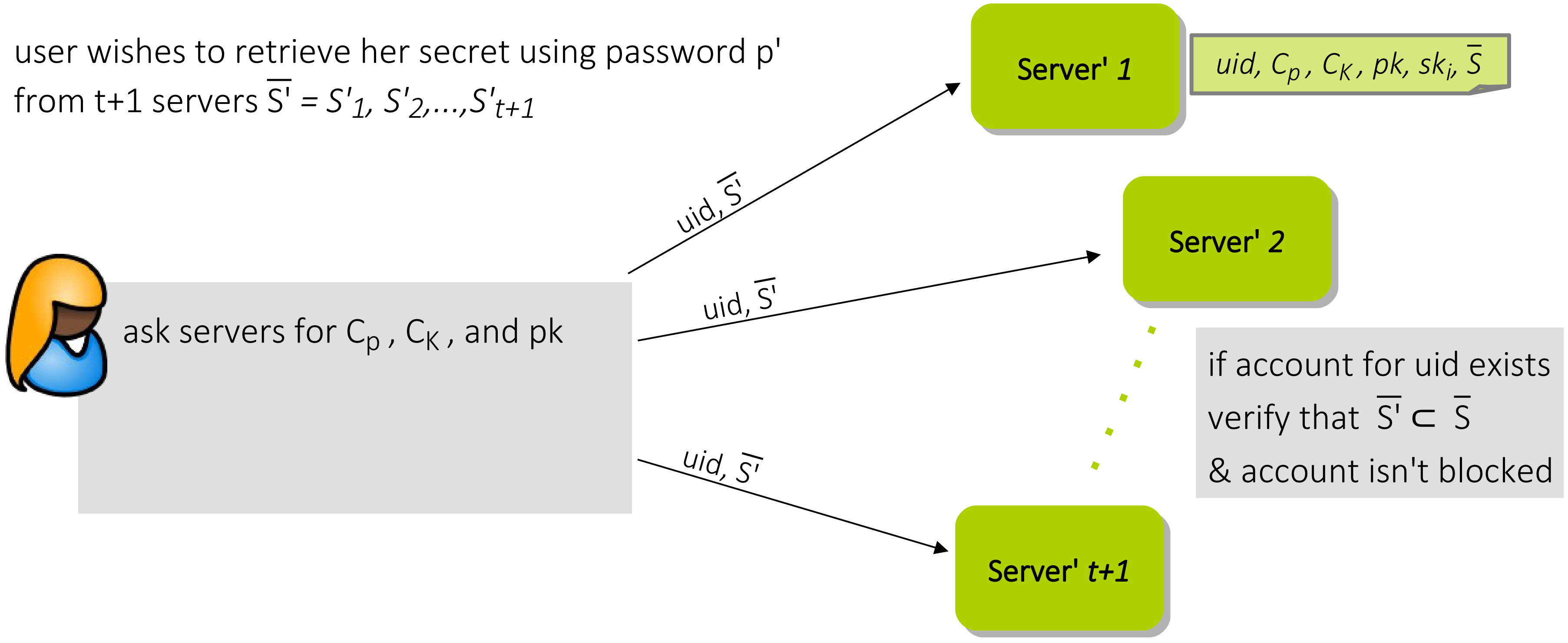
Our Protocol (High-Level-Idea) | **Retrieval**

user wishes to retrieve her secret using password p'
from t+1 servers $\overline{S'} = S'_1, S'_2,\ldots,S'_{t+1}$



Our Protocol (High-Level-Idea) | **Retrieval**

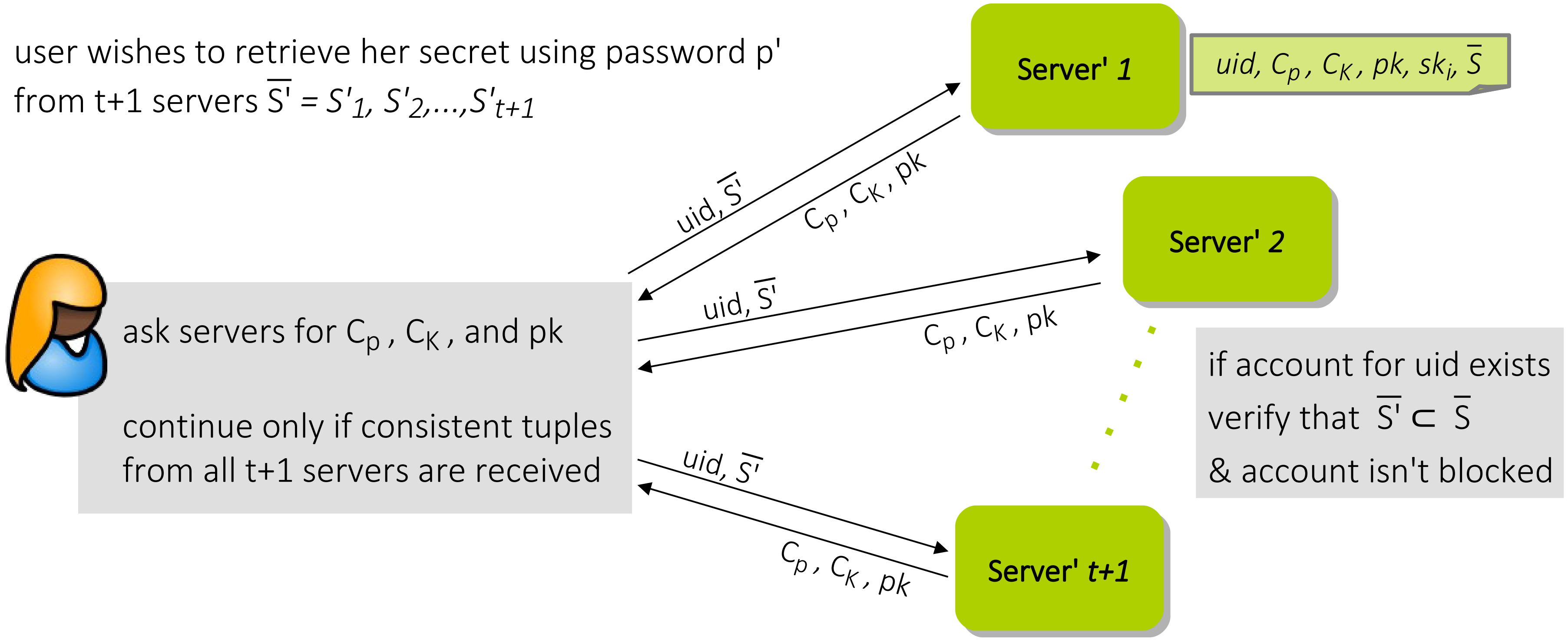
user wishes to retrieve her secret using password p' from t+1 servers $\overline{S'} = S'_1, S'_2,\ldots,S'_{t+1}$



if account for uid exists
verify that $\overline{S'} \subset \overline{S}$
& account isn't blocked

Our Protocol (High-Level-Idea) | **Retrieval**

user wishes to retrieve her secret using password p'
from t+1 servers $\overline{S'} = S'_1, S'_2,\ldots,S'_{t+1}$



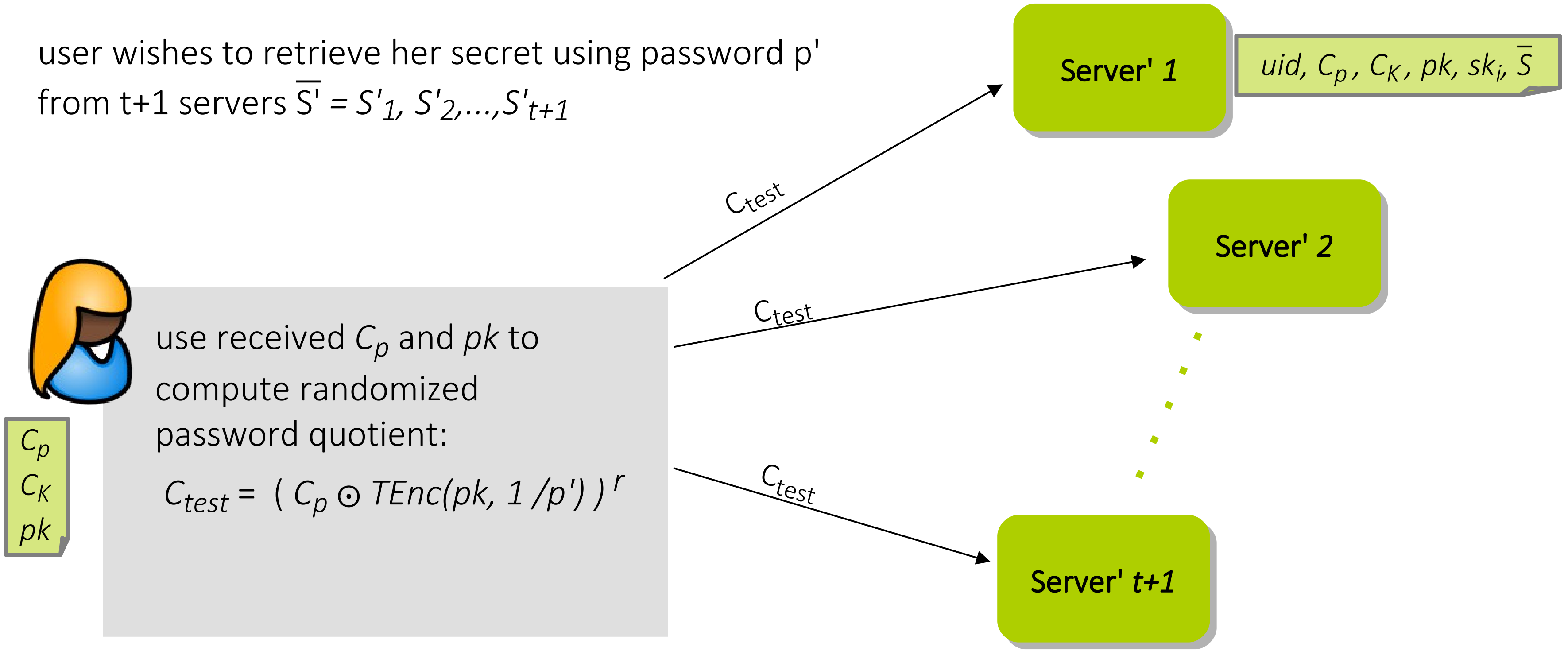
ask servers for C_p , C_K , and pk

continue only if consistent tuples from all t+1 servers are received

if account for uid exists
verify that $\overline{S'} \subset \overline{S}$
& account isn't blocked

Our Protocol (High-Level-Idea) | **Retrieval**

user wishes to retrieve her secret using password p' from t+1 servers $\overline{S'} = S'_1, S'_2, \ldots, S'_{t+1}$

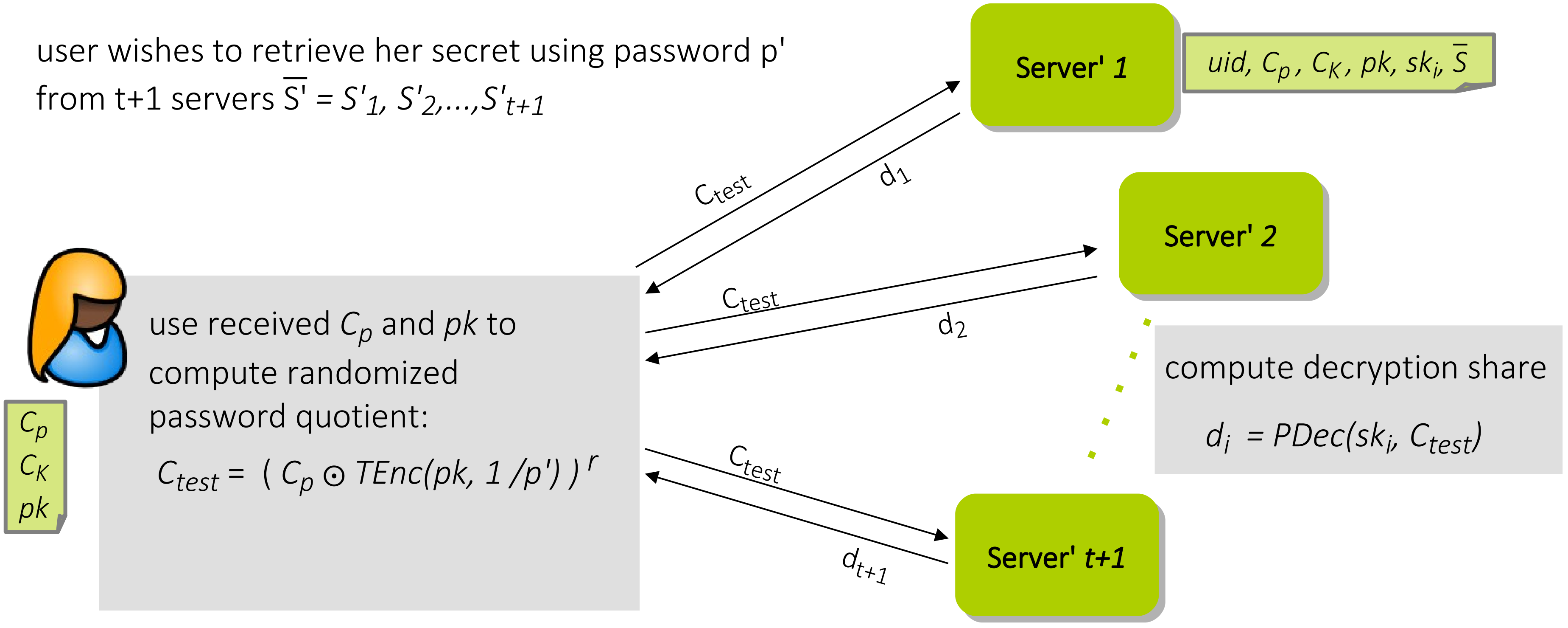


use received C_p and pk to compute randomized password quotient:

$$C_{test} = (C_p \odot TEnc(pk, 1/p'))^r$$

Our Protocol (High-Level-Idea) | **Retrieval**

user wishes to retrieve her secret using password p' from t+1 servers $\overline{S'} = S'_1, S'_2, \ldots, S'_{t+1}$



use received C_p and pk to compute randomized password quotient:

$$C_{test} = \left(C_p \odot TEnc(pk, 1/p') \right)^r$$

compute decryption share

$$d_i = PDec(sk_i, C_{test})$$

Our Protocol (High-Level-Idea) | **Retrieval**

user wishes to retrieve her secret using password p'
from t+1 servers $\overline{S'} = S'_1, S'_2, \ldots, S'_{t+1}$

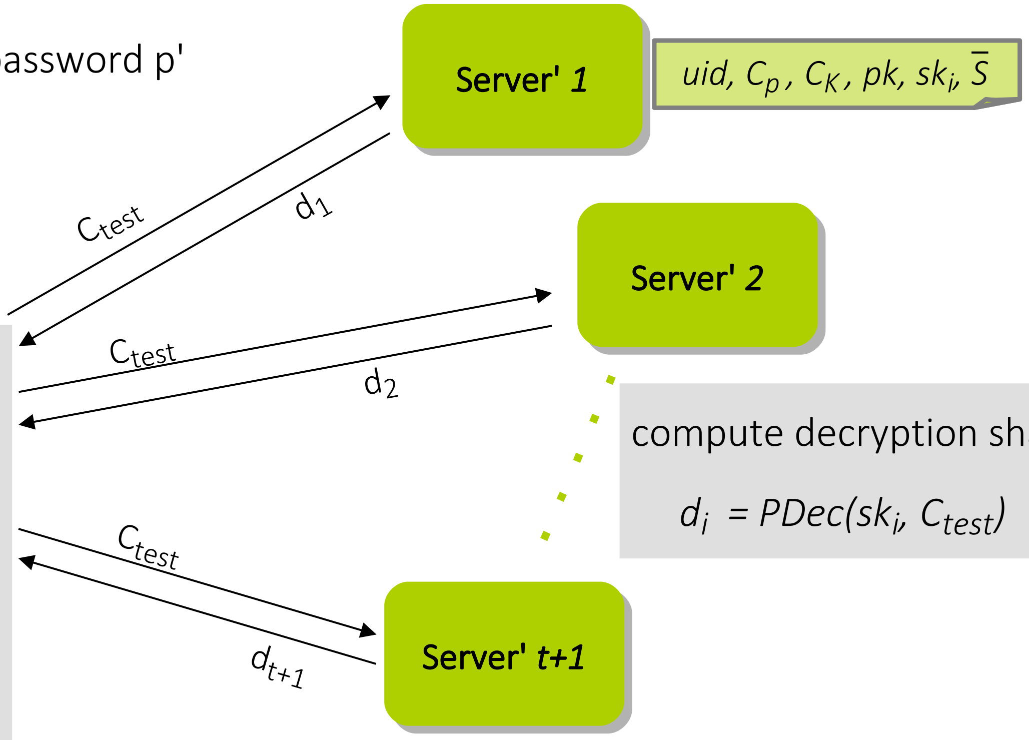
C_p
C_K
pk

use received C_p and pk to compute randomized password quotient:

$C_{test} = (C_p \odot TEnc(pk, 1/p'))^r$

continue only if

$1 = TDec(C_{test}, d_1, \ldots, d_{t+1})$



compute decryption share

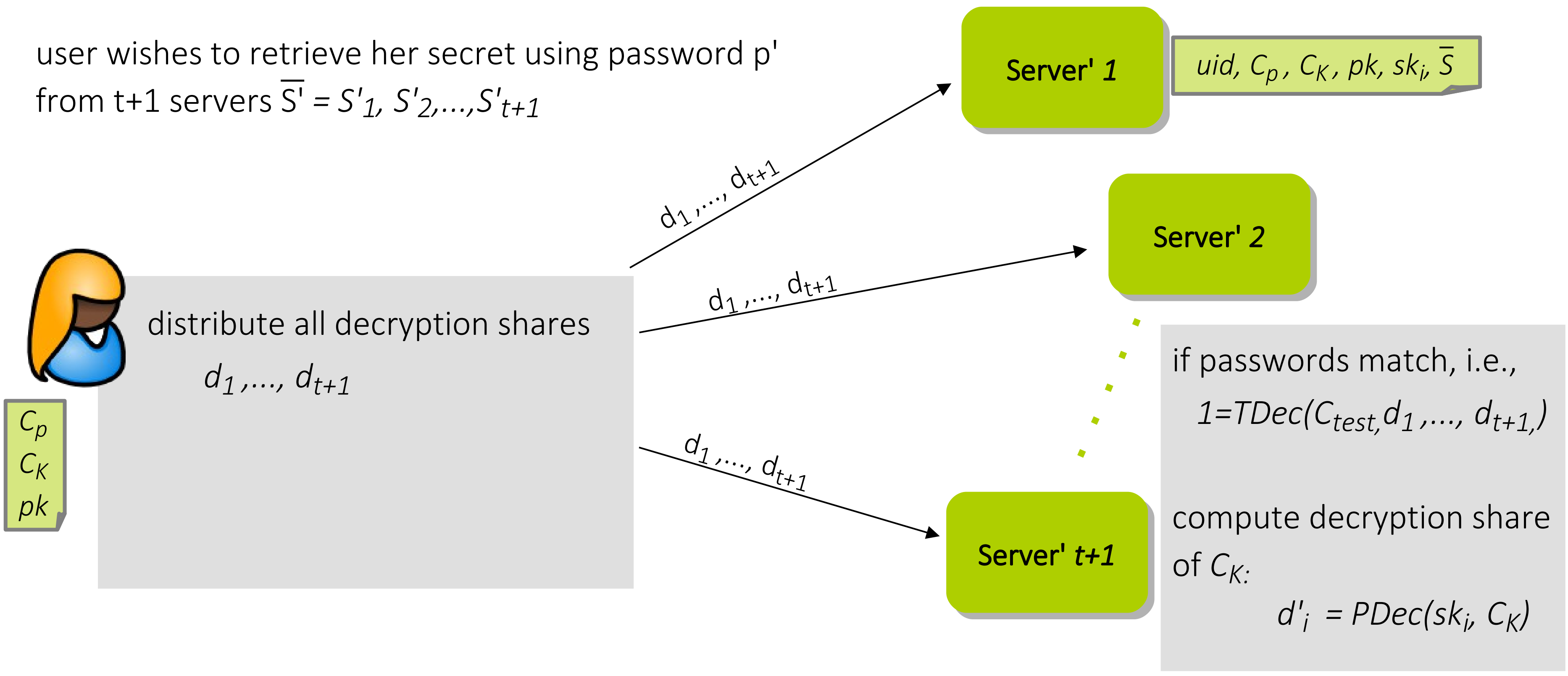
$d_i = PDec(sk_i, C_{test})$

if p' = p → C_{test} is an encryption of "1"
if p' ≠ p → C_{test} is an encryption of a random value

even when all t+1 servers are corrupt and provided a wrong C_{p*}, $pk*$
they can only learn whether p' = p* ?

Our Protocol (High-Level-Idea) | **Retrieval**

user wishes to retrieve her secret using password p'
from t+1 servers $\overline{S'} = S'_1, S'_2, \ldots, S'_{t+1}$



distribute all decryption shares

$d_1, \ldots, d_{t+1}$

if passwords match, i.e.,

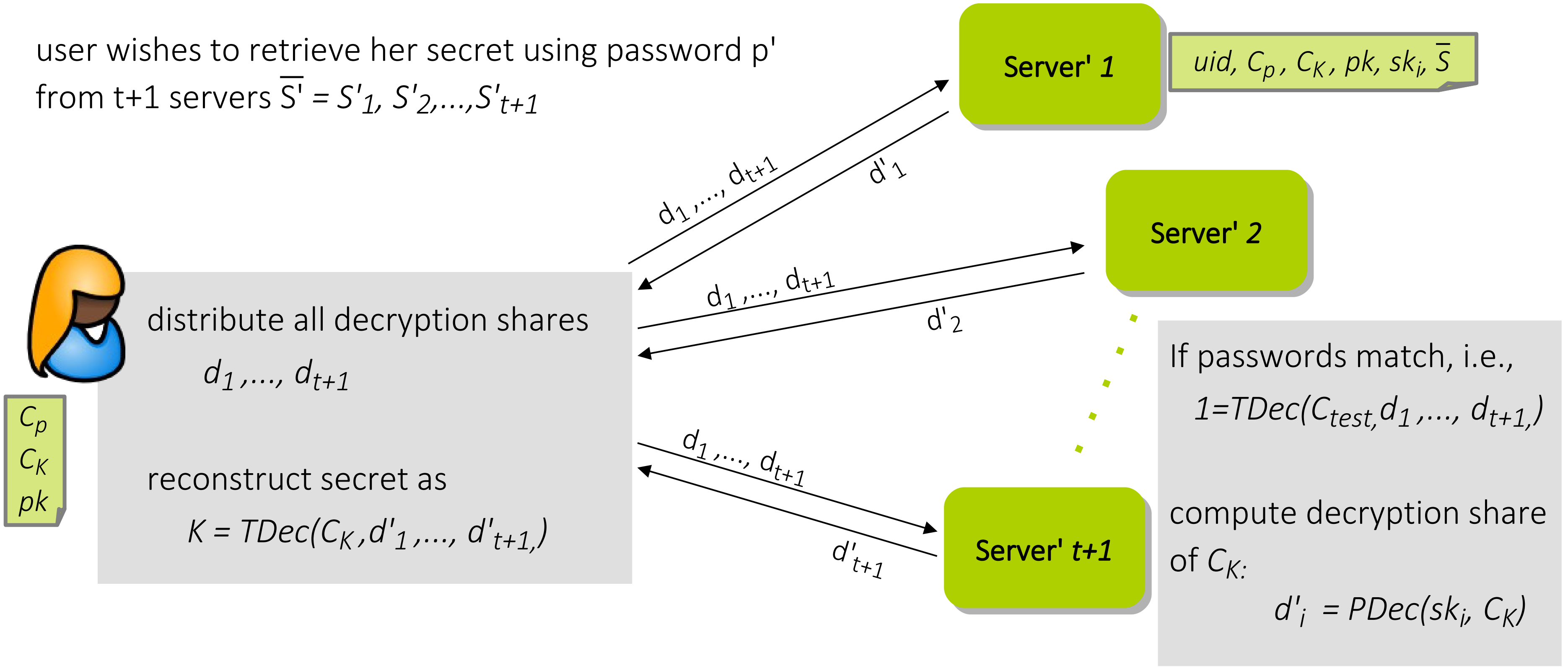
$1 = TDec(C_{test}, d_1, \ldots, d_{t+1})$

compute decryption share of C_K:

$d'_i = PDec(sk_i, C_K)$

Our Protocol (High-Level-Idea) | **Retrieval**

user wishes to retrieve her secret using password p'
from t+1 servers $\overline{S'} = S'_1, S'_2, \dots, S'_{t+1}$



distribute all decryption shares
$d_1, \dots, d_{t+1}$

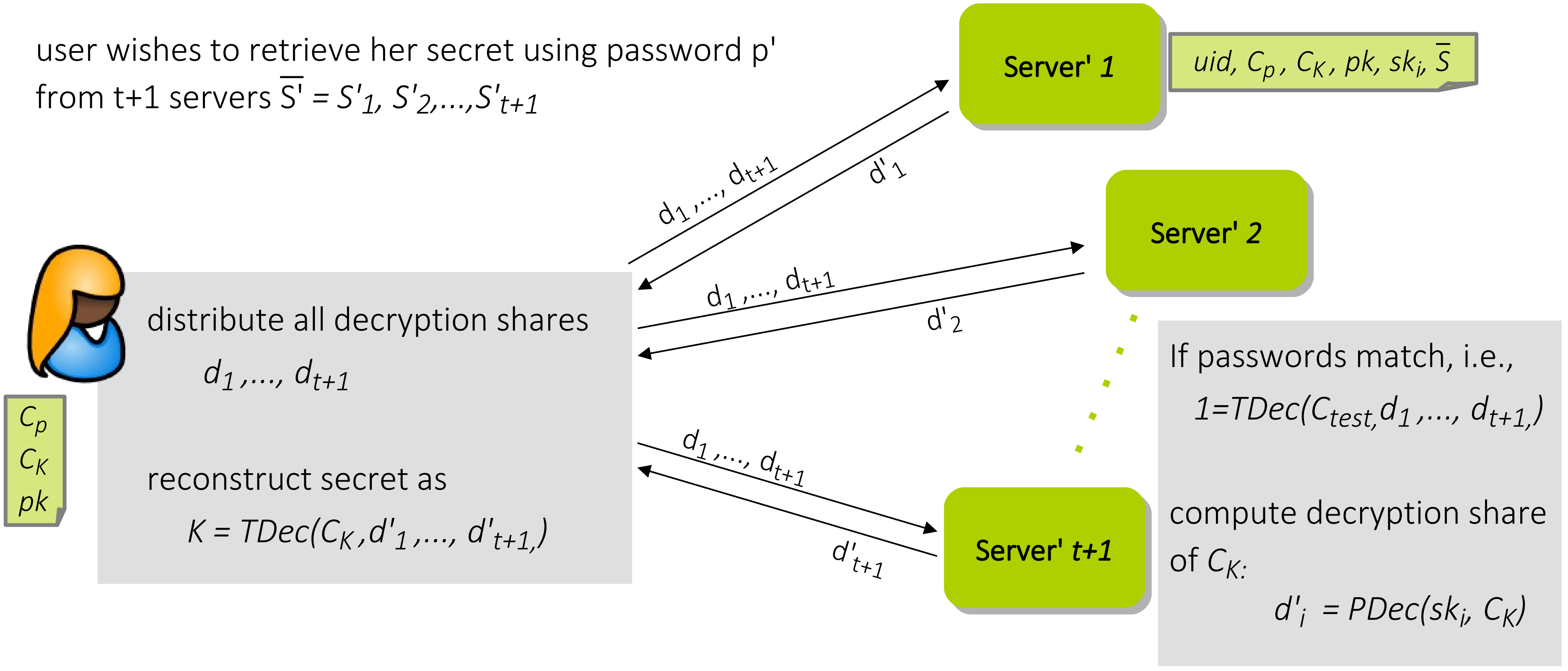
reconstruct secret as
$K = TDec(C_K, d'_1, \dots, d'_{t+1})$

If passwords match, i.e.,
$1 = TDec(C_{test}, d_1, \dots, d_{t+1})$

compute decryption share of C_K:
$d'_i = PDec(sk_i, C_K)$

Our Protocol (High-Level-Idea) | **Retrieval**

user wishes to retrieve her secret using password p' from t+1 servers $\overline{S'} = S'_1, S'_2, \ldots, S'_{t+1}$



distribute all decryption shares

$d_1, \ldots, d_{t+1}$

reconstruct secret as

$K = TDec(C_K, d'_1, \ldots, d'_{t+1})$

If passwords match, i.e.,

$1 = TDec(C_{test}, d_1, \ldots, d_{t+1})$

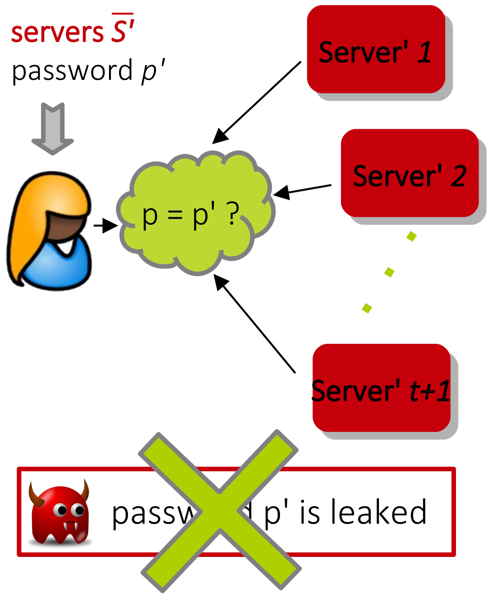
compute decryption share of C_K:

$d'_i = PDec(sk_i, C_K)$

decryption shares for *K* are encrypted under fresh key of the user & come with a proof of correctness

Security & Summary

- our protocol securely realizes $\mathcal{F}_{\text{TPASS}}$ based on
  - (t; n)-semantically secure threshold homomorphic cryptosystem
  - CPA-secure encryption & CCA2-secure labeled encryption schemes
  - existentially unforgeable signature scheme
  - simulation-sound zero-knowledge proof system
  - in $\mathcal{F}_{\text{CRS}}$ and $\mathcal{F}_{\text{CA}}$ hybrid model



Summary:

- threshold (t-out-of-n) password-authenticated secret sharing (TPASS)
  - store & reconstruct strong secret K (and thereby any encrypted data)
  - user only needs to remember username and password
  - if at most t servers are corrupt: Adv does not learn anything about K and p
  - retrieval with all bad servers does not leak password p'
  - UC secure (nice composability guarantees)

Thank you!

Instantiation & Efficiency

- instantiation based on ElGamal encryption scheme & Schnorr signatures

  → TPASS secure under DDH assumption in random oracle model

- efficiency
  - computation

	setup	retrieval
user	5n + 15	14t + 24
server	n + 18	7t + 28

# of exponentiations

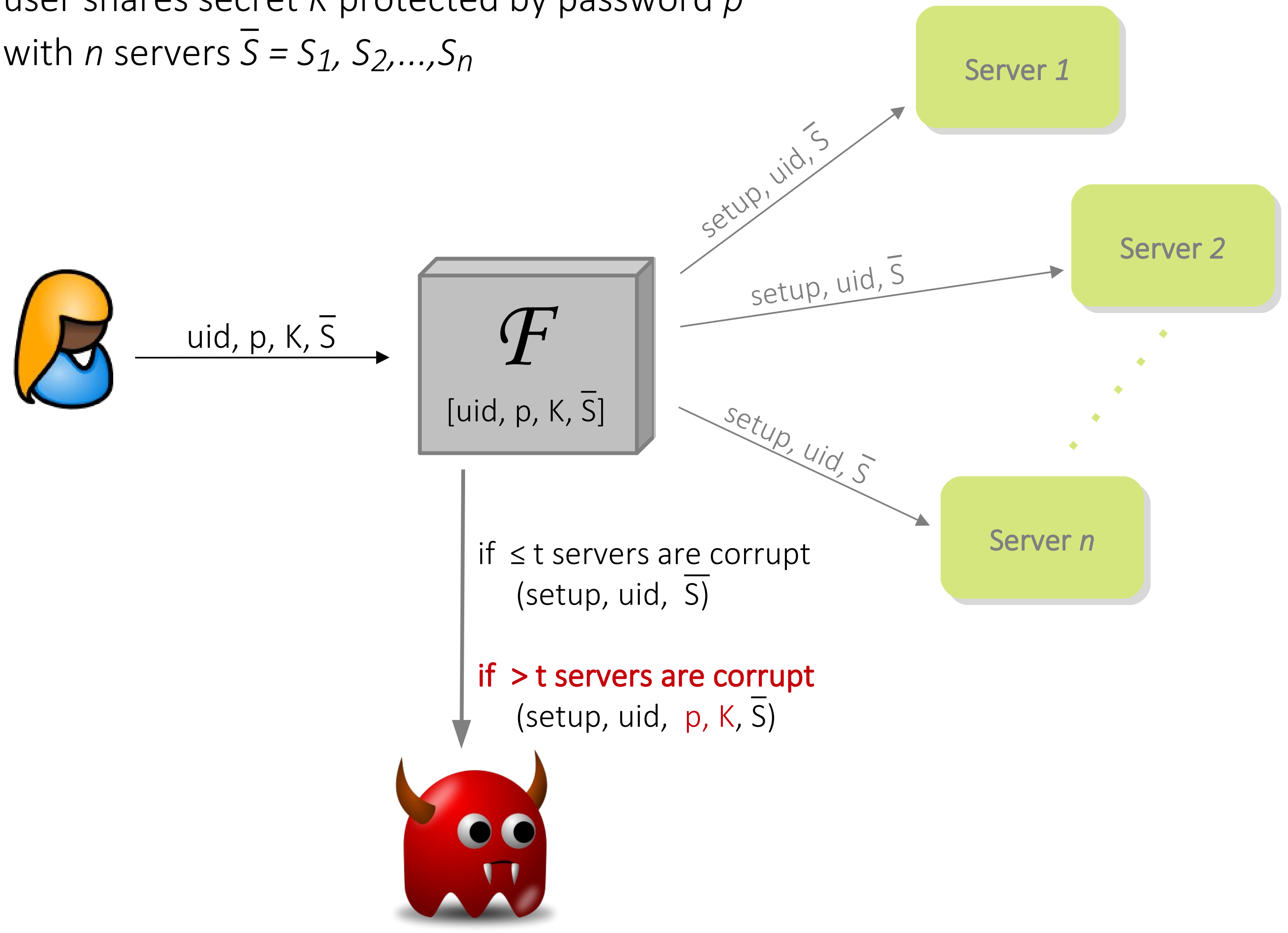
  - communication

	setup	retrieval
rounds	4	10
# group elements	n(2.5n + 18.5)	(t + 1)(36.5 + 2.5n + 10.5(t + 1))

hash value ≈ half a group element

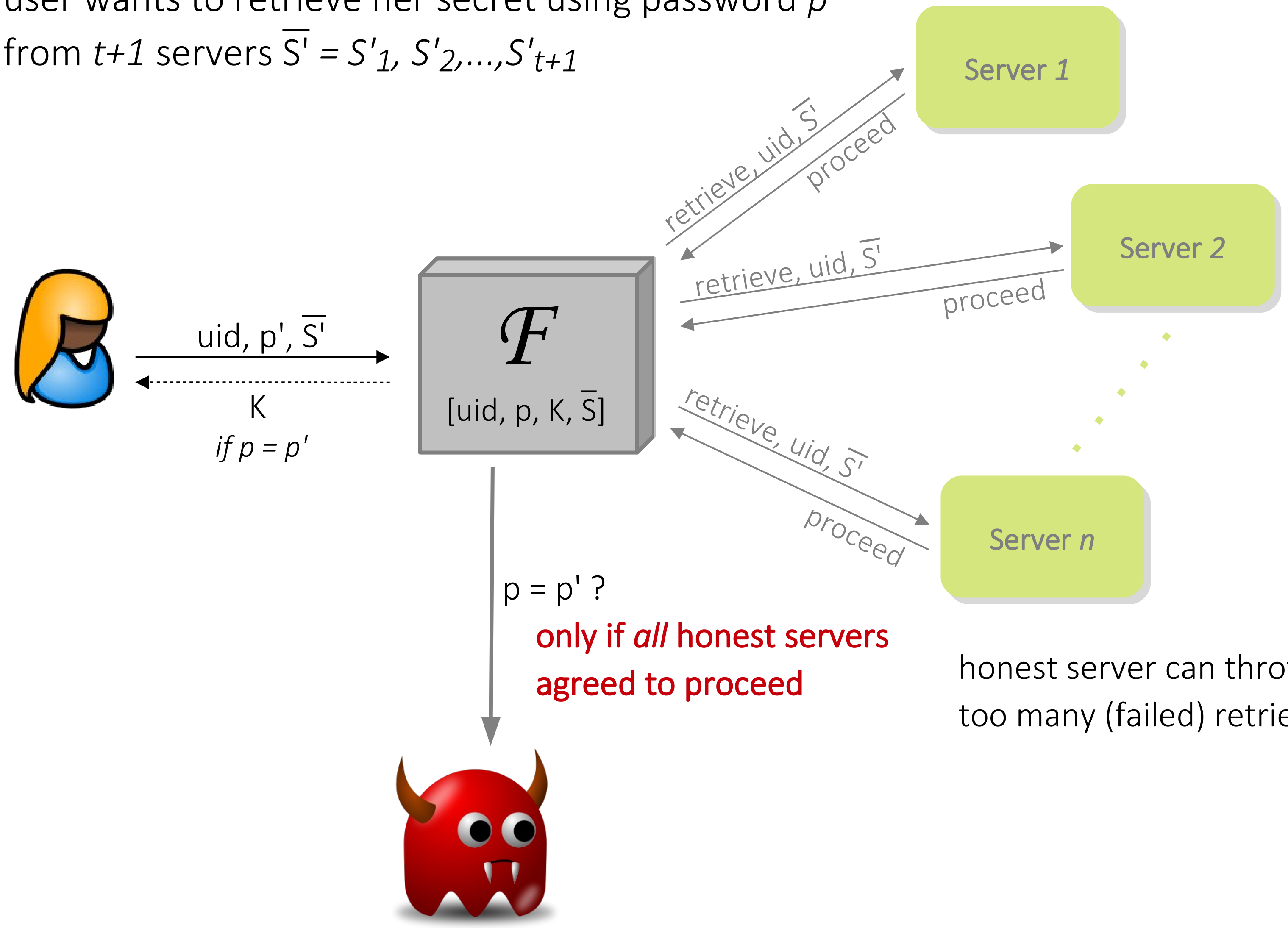
Our Model – Setup

user shares secret *K* protected by password *p*
with *n* servers $\bar{S} = S_1, S_2, \ldots, S_n$



Our Model – Retrieval (at least one server is honest)

user wants to retrieve her secret using password *p'*
from *t+1* servers $\overline{S'} = S'_1, S'_2, \ldots, S'_{t+1}$



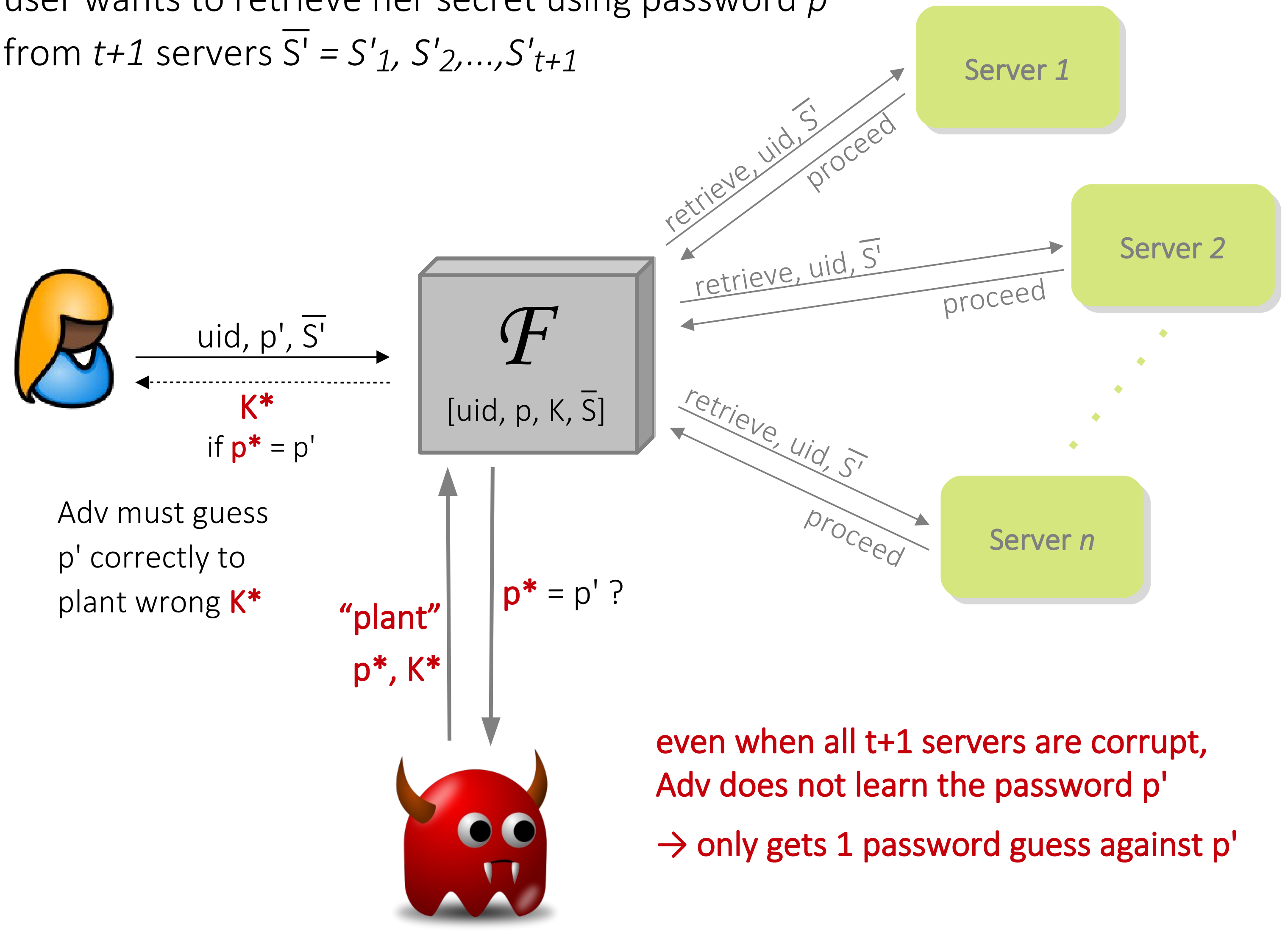
Server *1*
Server *2*
Server *n*
retrieve, uid, $\overline{S'}$
proceed
retrieve, uid, $\overline{S'}$
proceed
retrieve, uid, $\overline{S'}$
proceed
uid, p', $\overline{S'}$
K
if p = p'
$\mathcal{F}$
[uid, p, K, $\overline{S}$]
p = p' ?

only if *all* honest servers agreed to proceed

honest server can throttle account after too many (failed) retrieval attempts

Our Model – Retrieval (all servers are corrupt)

user wants to retrieve her secret using password p'
from $t+1$ servers $\overline{S'} = S'_1, S'_2, \ldots, S'_{t+1}$



uid, p', $\overline{S'}$
K*
if p* = p'
$\mathcal{F}$
[uid, p, K, $\overline{S}$]
retrieve, uid, $\overline{S'}$
proceed
Server 1
retrieve, uid, $\overline{S'}$
proceed
Server 2
retrieve, uid, $\overline{S'}$
proceed
Server n
“plant”
p*, K*
p* = p' ?

Adv must guess p' correctly to plant wrong K*

even when all t+1 servers are corrupt,
Adv does not learn the password p'

→ only gets 1 password guess against p'