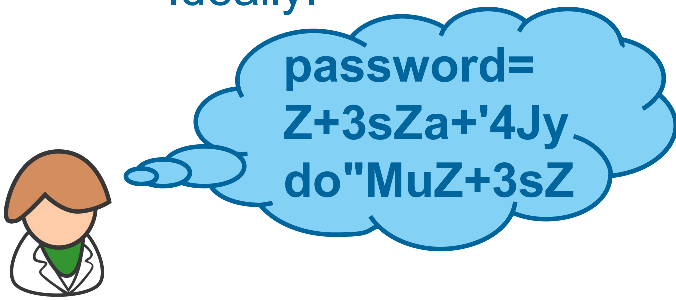
Two-Server Password-Authenticated Secret Sharing UC-Secure Against Transient Corruptions

Jan Camenisch, Robert R. Enderlein, Gregory Neven
IBM Research – Zurich & ETH Zurich

Our Goal: Protect Your Data

- Protect user data = provide access to authenticated users.
- How to authenticate users? Usually: with passwords.
- Most users choose easy-to-remember, insecure passwords.
  - Low entropy: 16 character passwords have only approx. 30 bits of entropy [NIST].
  - Password databases compromised = attacker can recover passwords (even if hashed and salted).
    - A rig of 25 GPUs can test 350 billion passwords/second.
    - 60% of LinkedIn passwords cracked within 24 hours (2012).

Ideally:



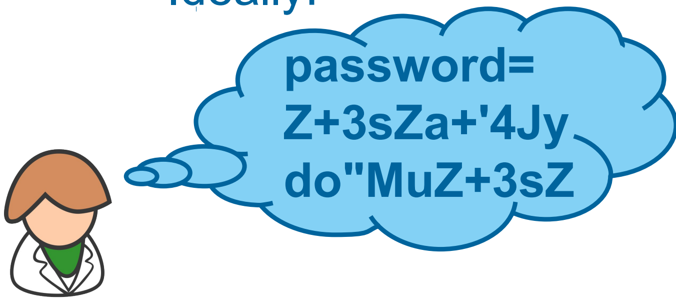
In reality:

[NIST]: NIST Special Publication 800-63-1 (2011).

Our Goal: Protect Your Data

- Protect user data = provide access to authenticated users.
- How to authenticate users? Usually: with passwords.
- Most users choose easy-to-remember, insecure passwords.
  - Low entropy: 16 character passwords have only approx. 30 bits of entropy [NIST].
  - Password databases compromised = attacker can recover passwords (even if hashed and salted).
    - A rig of 25 GPUs can test 350 billion passwords/second.
    - 60% of LinkedIn passwords cracked within 24 hours (2012).

Ideally:



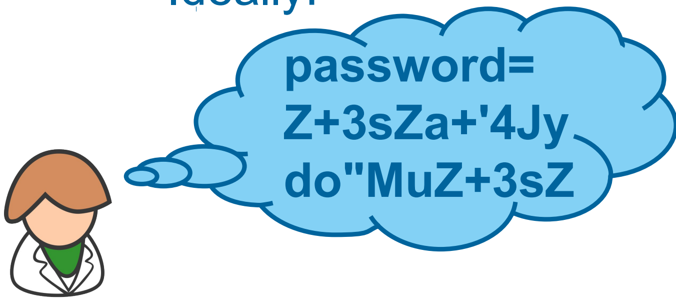
In reality:

[NIST]: NIST Special Publication 800-63-1 (2011).

Our Goal: Protect Your Data

- Protect user data = provide access to authenticated users.
- How to authenticate users? Usually: with passwords.
- Most users choose easy-to-remember, insecure passwords.
  - Low entropy: 16 character passwords have only approx. 30 bits of entropy [NIST].
  - Password databases compromised = attacker can recover passwords (even if hashed and salted).
    - A rig of 25 GPUs can test 350 billion passwords/second.
    - 60% of LinkedIn passwords cracked within 24 hours (2012).

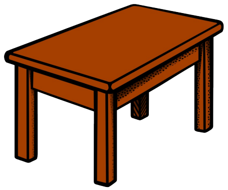
Ideally:



In reality:

[NIST]: NIST Special Publication 800-63-1 (2011).

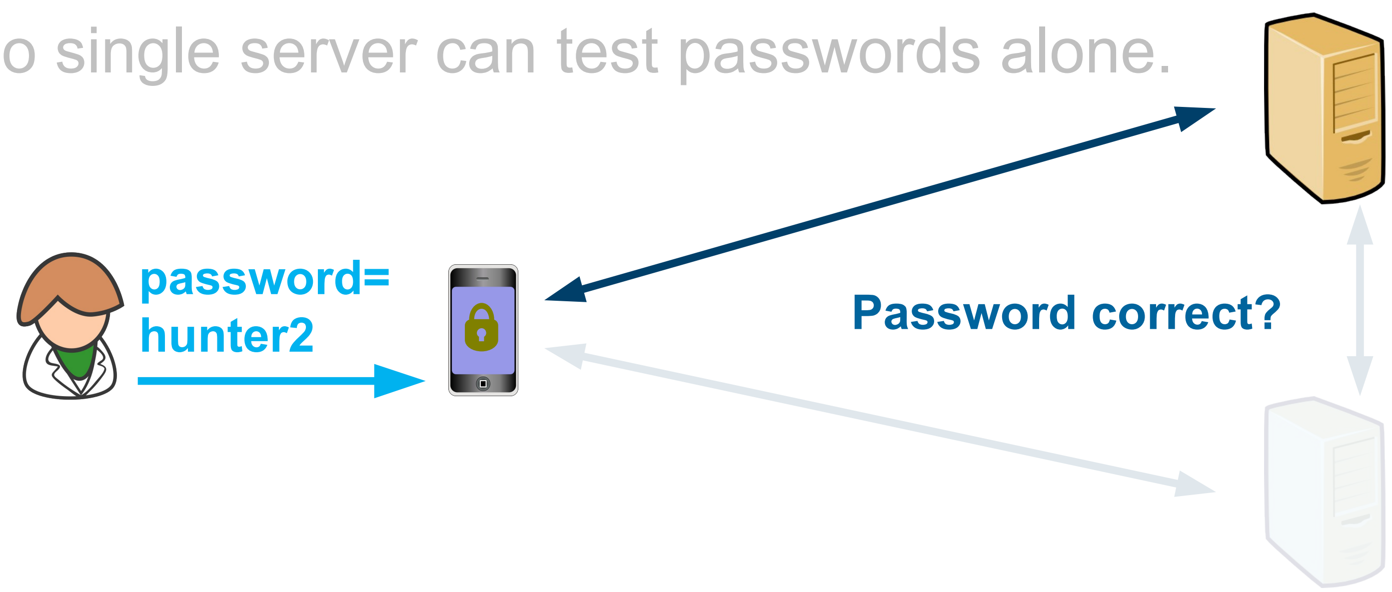
Table of contents



Our Goal: Protect Your Data

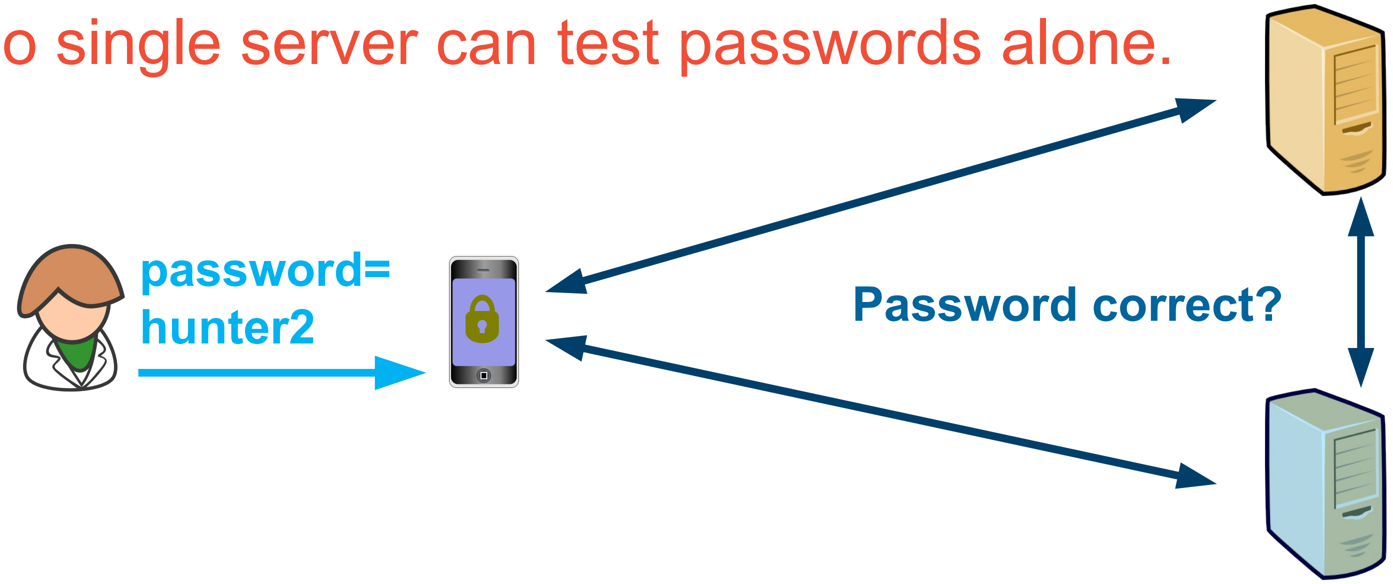
Are Passwords Inherently Insecure?

- No! We are using them incorrectly.
- Single-server solutions inherently vulnerable to offline-guessing attack if compromised.
- Instead use two server solution where no single server can test passwords alone.



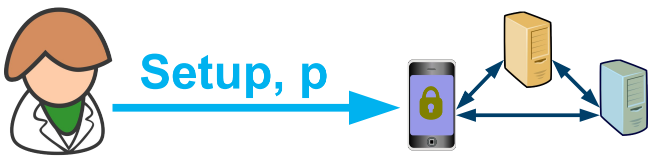
Are Passwords Inherently Insecure?

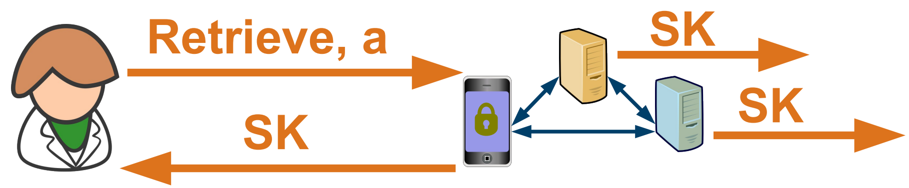
- No! We are using them incorrectly.
- Single-server solutions inherently vulnerable to offline-guessing attack if compromised.
- Instead use two server solution where no single server can test passwords alone.



Two-Server Password-Authenticated Protocols

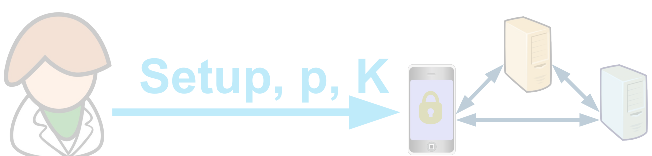
- Threshold Password-Authenticated Key Exchange (T-PAKE):

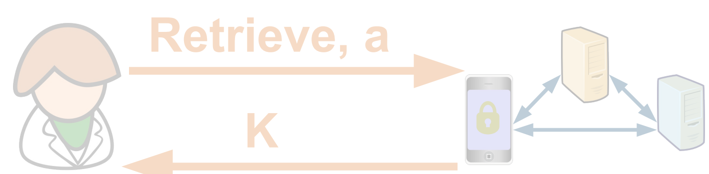




  – If password attempt is correct, share a random session key.

- Password-Authenticated Secret Sharing (PASS):

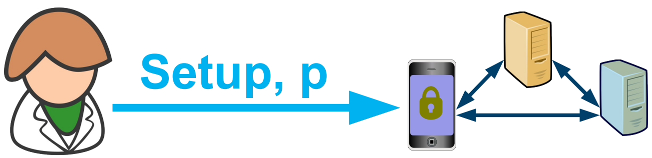


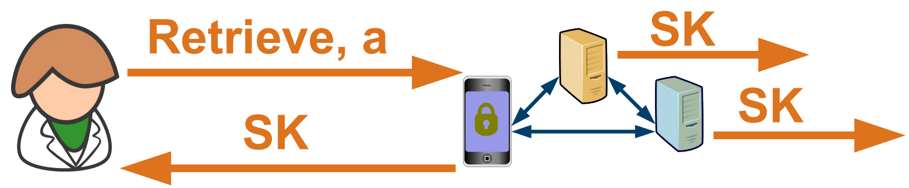


  – User also submits a strong secret K at setup.
  – If password correct, retrieves that K.
  – After the protocol user has a strong cryptographic key, which can be used to protect the rest of his data.

Two-Server Password-Authenticated Protocols

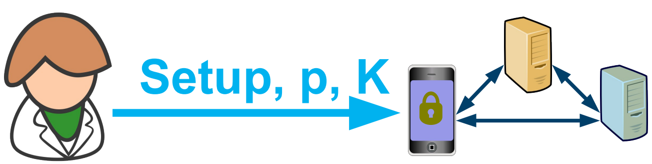
- Threshold Password-Authenticated Key Exchange (T-PAKE):

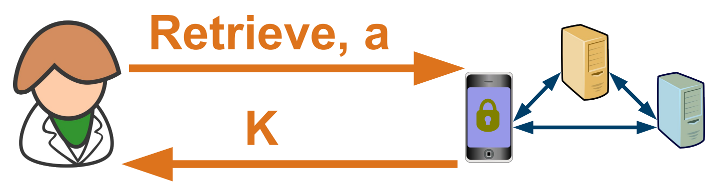




  – If password attempt is correct, share a random session key.

- Password-Authenticated Secret Sharing (PASS):

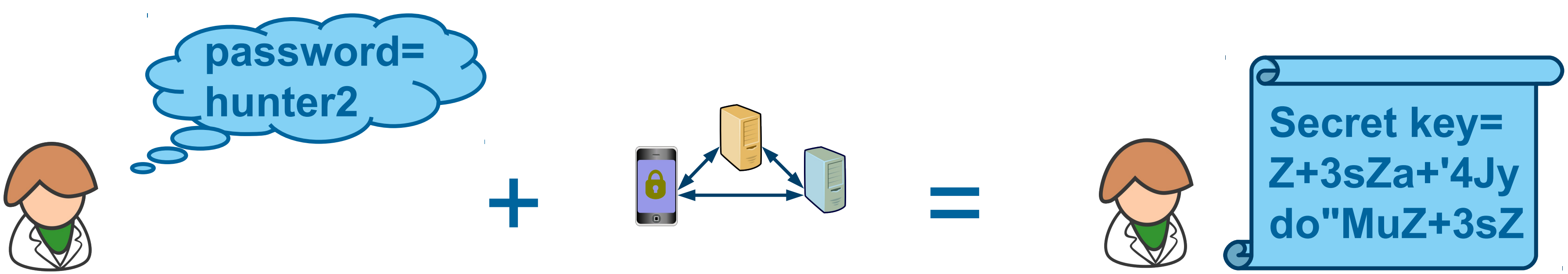




  – User also submits a strong secret K at setup.
  – If password correct, retrieves that K.
  – After the protocol user has a strong cryptographic key, which can be used to protect the rest of his data.

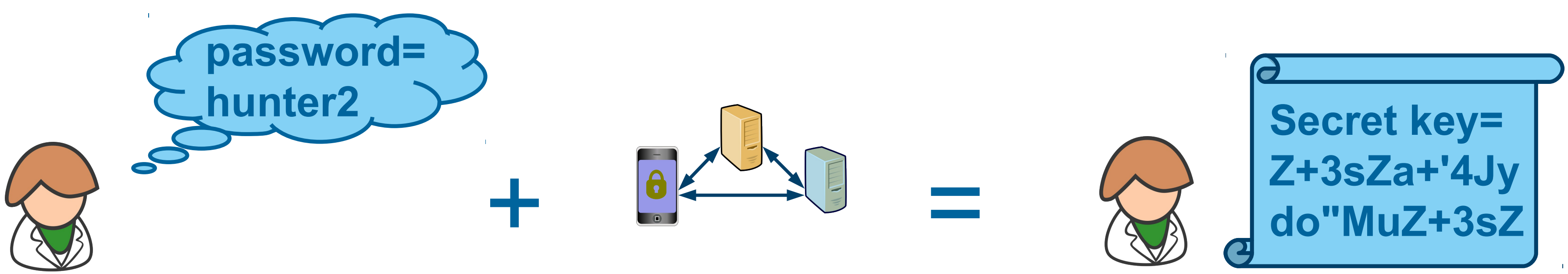
Design Goals
for 2-Server Password-Authenticated Secret Sharing

- User remembers weak password, user name, server names.
- User deposits and later reconstruct a strong secret K.
  (K can then be used to encrypt further data.)
- One server compromised:
  - Cannot perform an offline attack on the password.
    (Can only do individual on-line attempts with other server.)
- Servers can recover from being compromised.



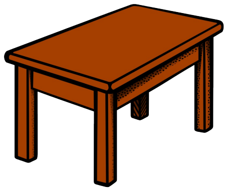
Design Goals
for 2-Server Password-Authenticated Secret Sharing

- User remembers weak password, user name, server names.
- User deposits and later reconstruct a strong secret K.
  (K can then be used to encrypt further data.)
- One server compromised:
  - Cannot perform an offline attack on the password.
    (Can only do individual on-line attempts with other server.)
- Servers can recover from being compromised.



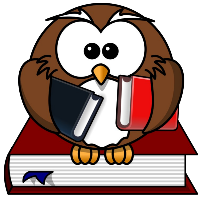
IBM

Table of contents



Our Goal: Protect Your Data

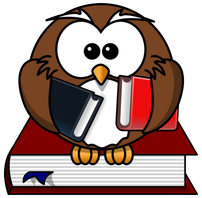
Related Work



- UC-Secure PASS, static corruptions, (ROM):
  - Camenisch et al. (2012), Camenisch et al. (2014).
- Other non-UC secure PASS protocols:
  - Bagherzandi et al. (2011), Jarecki et al. (2014).
- Non-UC secure 2-server T-PAKE: Katz et al. (2005 & 2012).
- Non-UC secure 1-server PAKE protocols:
  - Ford-Kaliski (2000), Jablon (2001), Brainard et al. (2003), MacKenzie et al. (2002), Di Raimondo-Gennaro (2003), Szydlo-Kaliski (2005).

- Our paper: UC-Secure, transient corruptions, standard model.

Related Work



- UC-Secure PASS, static corruptions, (ROM):
  - Camenisch et al. (2012), Camenisch et al. (2014).
- Other non-UC secure PASS protocols:
  - Bagherzandi et al. (2011), Jarecki et al. (2014).
- Non-UC secure 2-server T-PAKE: Katz et al. (2005 & 2012).
- Non-UC secure 1-server PAKE protocols:
  - Ford-Kaliski (2000), Jablon (2001), Brainard et al. (2003), MacKenzie et al. (2002), Di Raimondo-Gennaro (2003), Szydlo-Kaliski (2005).

- Our paper: UC-Secure, transient corruptions, standard model.

What to do when a server is hacked?

- Previous solutions secure only against malicious servers (i.e., against static corruptions).
  - Technically, no security guarantees in case of adaptive hacking:
    Static security + guessing who will get corrupted is not good enough.
- Our solution is secure also if servers are hacked (UC-secure against dynamic corruptions).
  - Servers can also recover from corruption (i.e., security against transient corruptions).

What to do when a server is hacked?

- Previous solutions secure only against malicious servers (i.e., against static corruptions).
  - Technically, no security guarantees in case of adaptive hacking:
    Static security + guessing who will get corrupted is not good enough.
- Our solution is secure also if servers are hacked (UC-secure against dynamic corruptions).
  - Servers can also recover from corruption (i.e., security against transient corruptions).

Why UC?

- UC Definition for 2-PASS:
  - Passwords can be chosen according to arbitrary distributions.
  - The adversary sees all authentications (also ones with typos), not just correct ones.
  - The non-negligible success probability of adversary guessing the password is handled correctly.
  - Our protocol composes nicely with itself and other protocols.

- Property-based definition:
  - Passwords must be chosen independently according to uniform distribution.
  - The adversary sees only successful authentications.
  - Sucess probability = negl() + Pr[guess_password]
  - Does not compose.

Why UC?

- UC Definition for 2-PASS:
  - Passwords can be chosen according to arbitrary distributions.
  - The adversary sees all authentications (also ones with typos), not just correct ones.
  - The non-negligible success probability of adversary guessing the password is handled correctly.
  - Our protocol composes nicely with itself and other protocols.

- Property-based definition:
  - Passwords must be chosen independently according to uniform distribution.
  - The adversary sees only successful authentications.
  - Sucess probability = negl() + Pr[guess_password]
  - Does not compose.

Why UC?

- UC Definition for 2-PASS:
  - Passwords can be chosen according to arbitrary distributions.
  - The adversary sees all authentications (also ones with typos), not just correct ones.
  - The non-negligible success probability of adversary guessing the password is handled correctly.
  - Our protocol composes nicely with itself and other protocols.

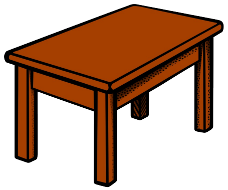
- Property-based definition:
  - Passwords must be chosen independently according to uniform distribution.
  - The adversary sees only successful authentications.
  - Sucess probability = negl() + Pr[guess_password]
  - Does not compose.

Why UC?

- UC Definition for 2-PASS:
  - Passwords can be chosen according to arbitrary distributions.
  - The adversary sees all authentications (also ones with typos), not just correct ones.
  - The non-negligible success probability of adversary guessing the password is handled correctly.
  - Our protocol composes nicely with itself and other protocols.

- Property-based definition:
  - Passwords must be chosen independently according to uniform distribution.
  - The adversary sees only successful authentications.
  - Sucess probability = negl() + Pr[guess_password]
  - Does not compose.

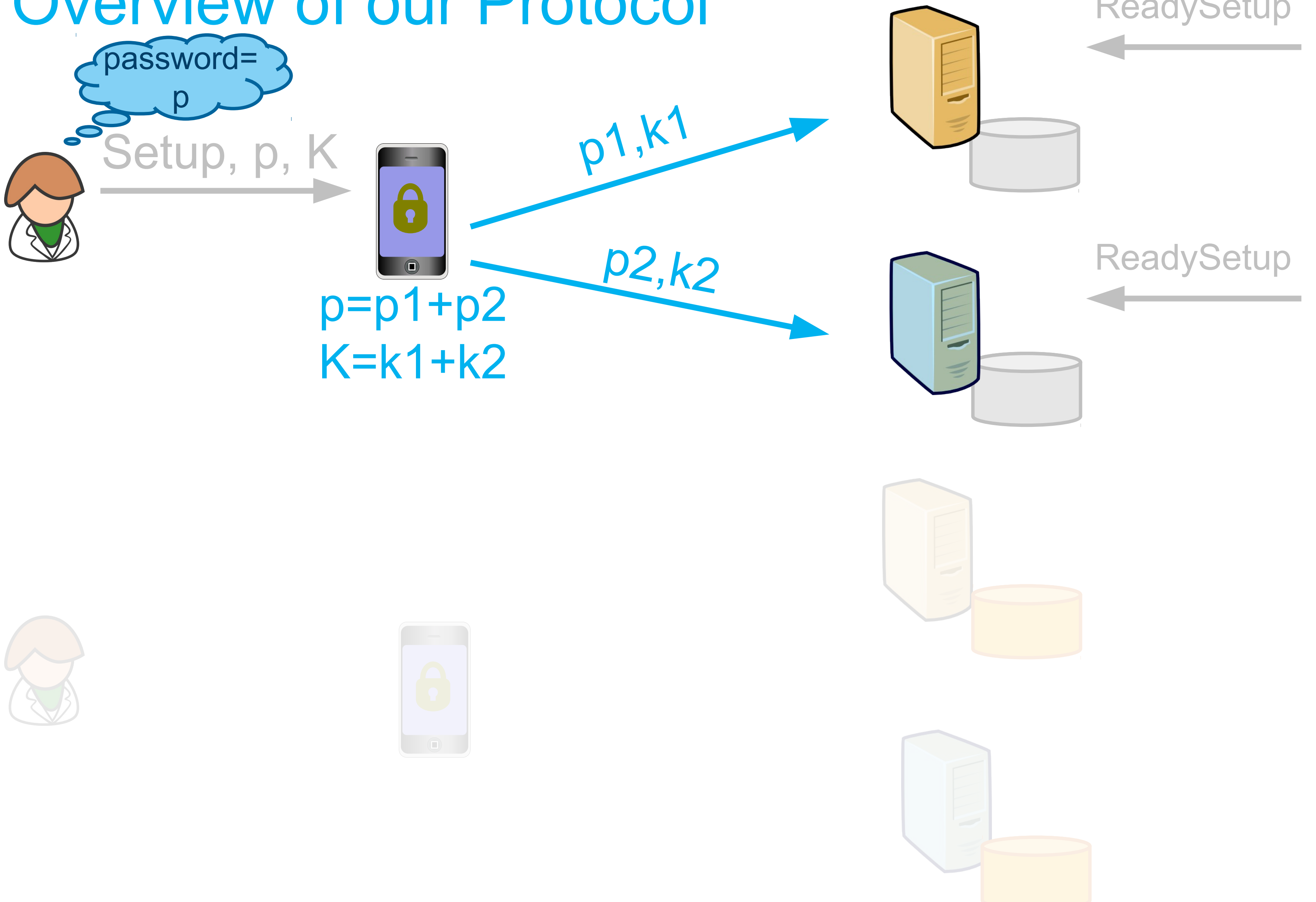
Table of contents



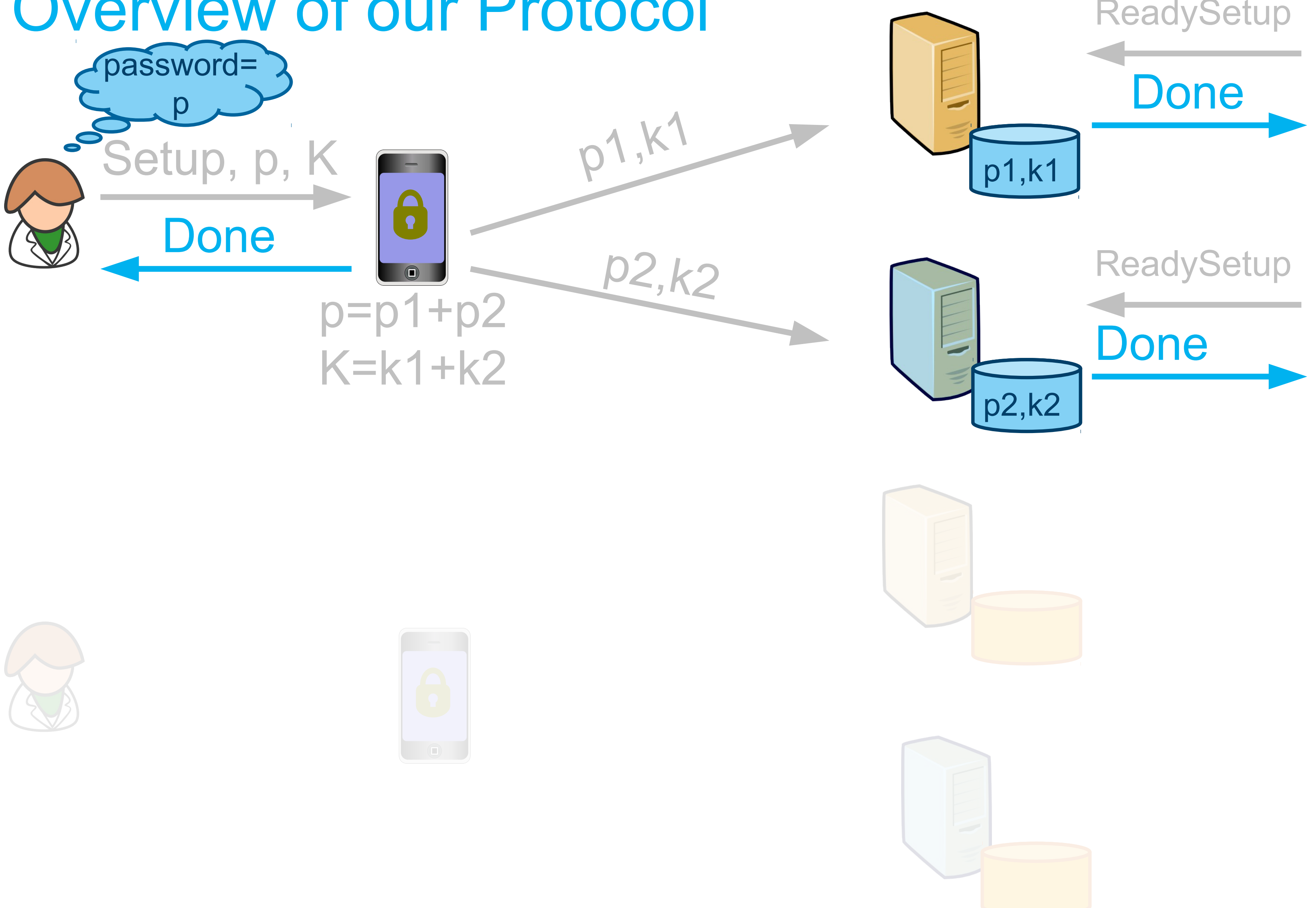
Our Goal: Protect Your Data

Overview of our Protocol

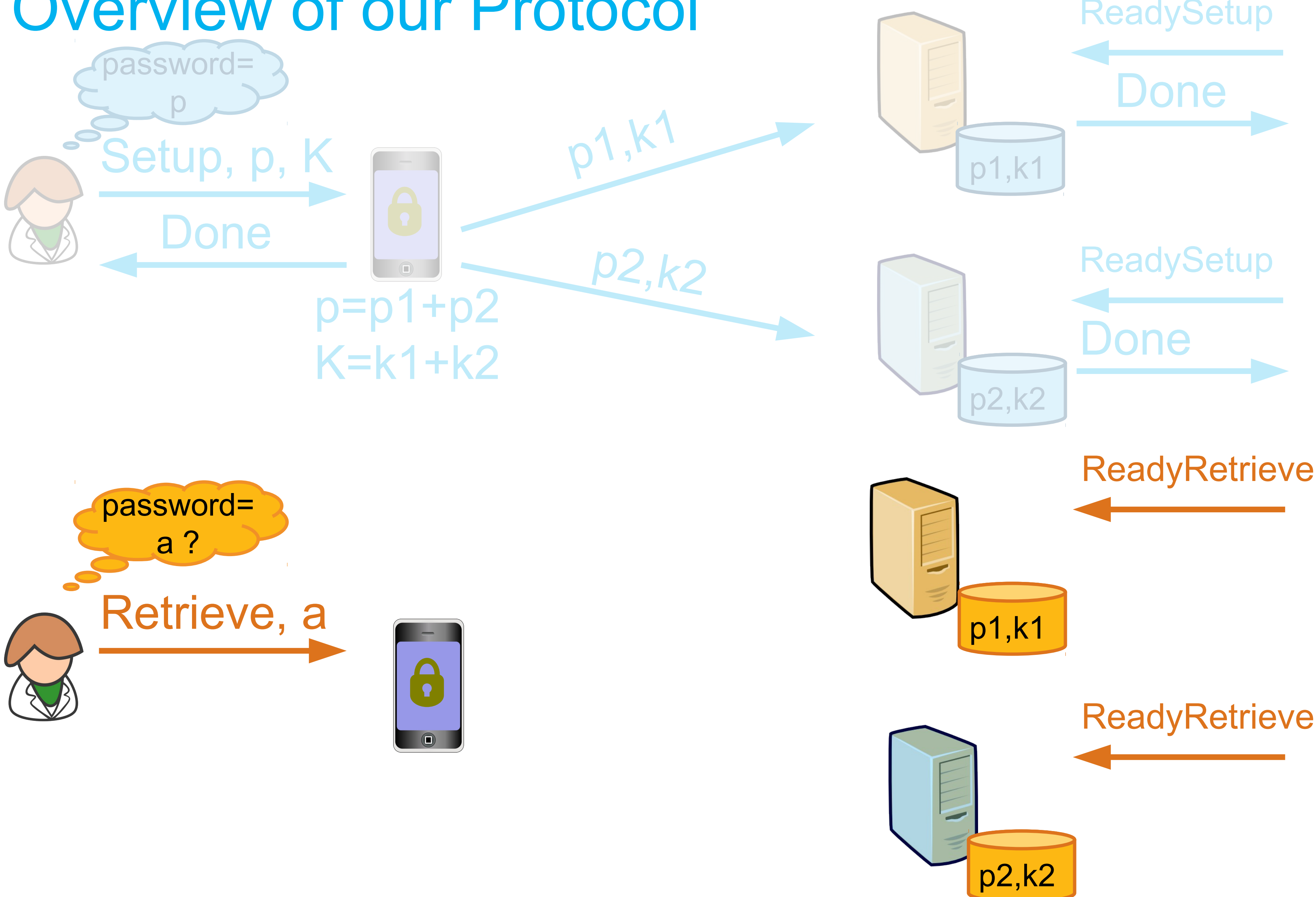
Overview of our Protocol



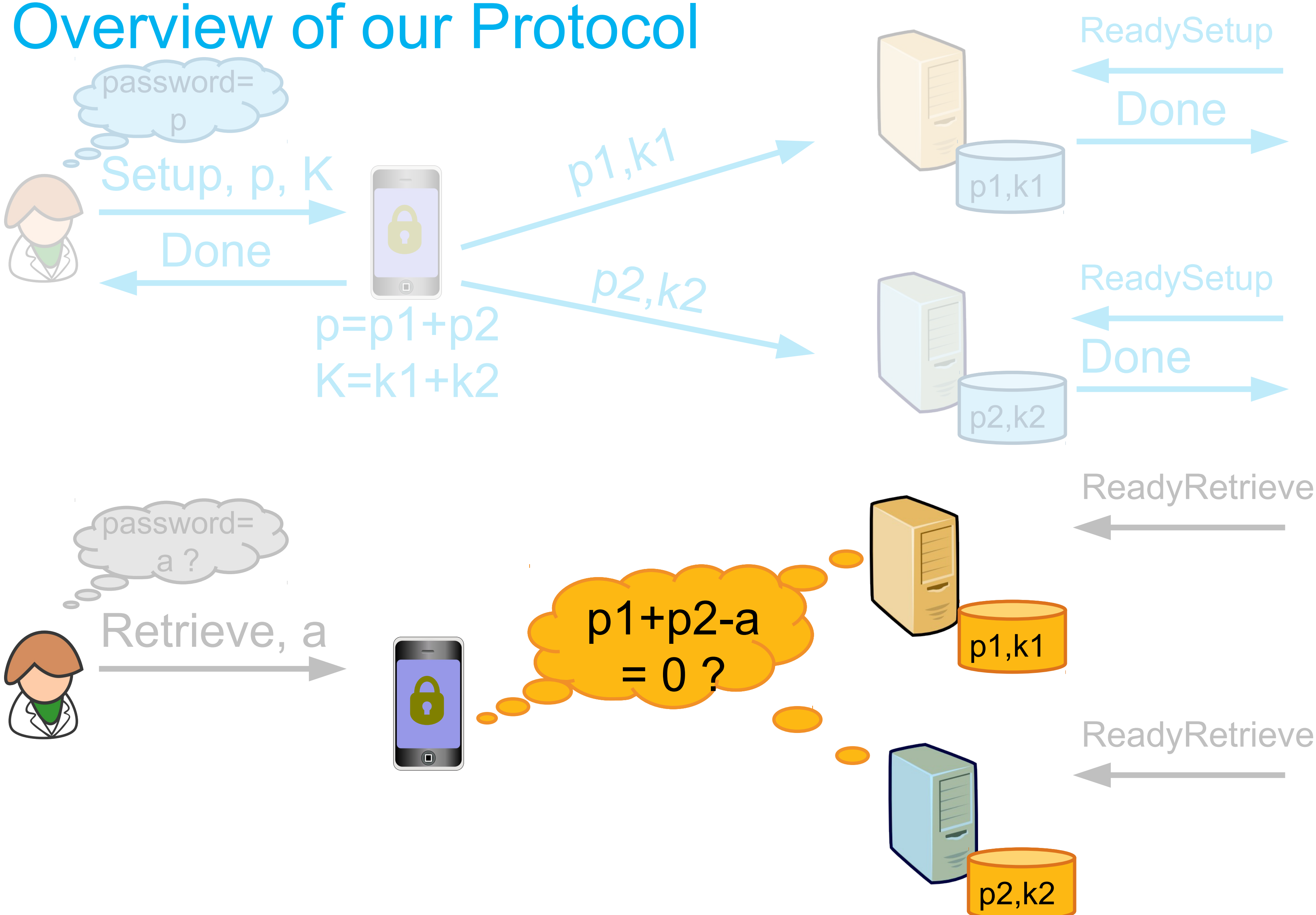
Overview of our Protocol



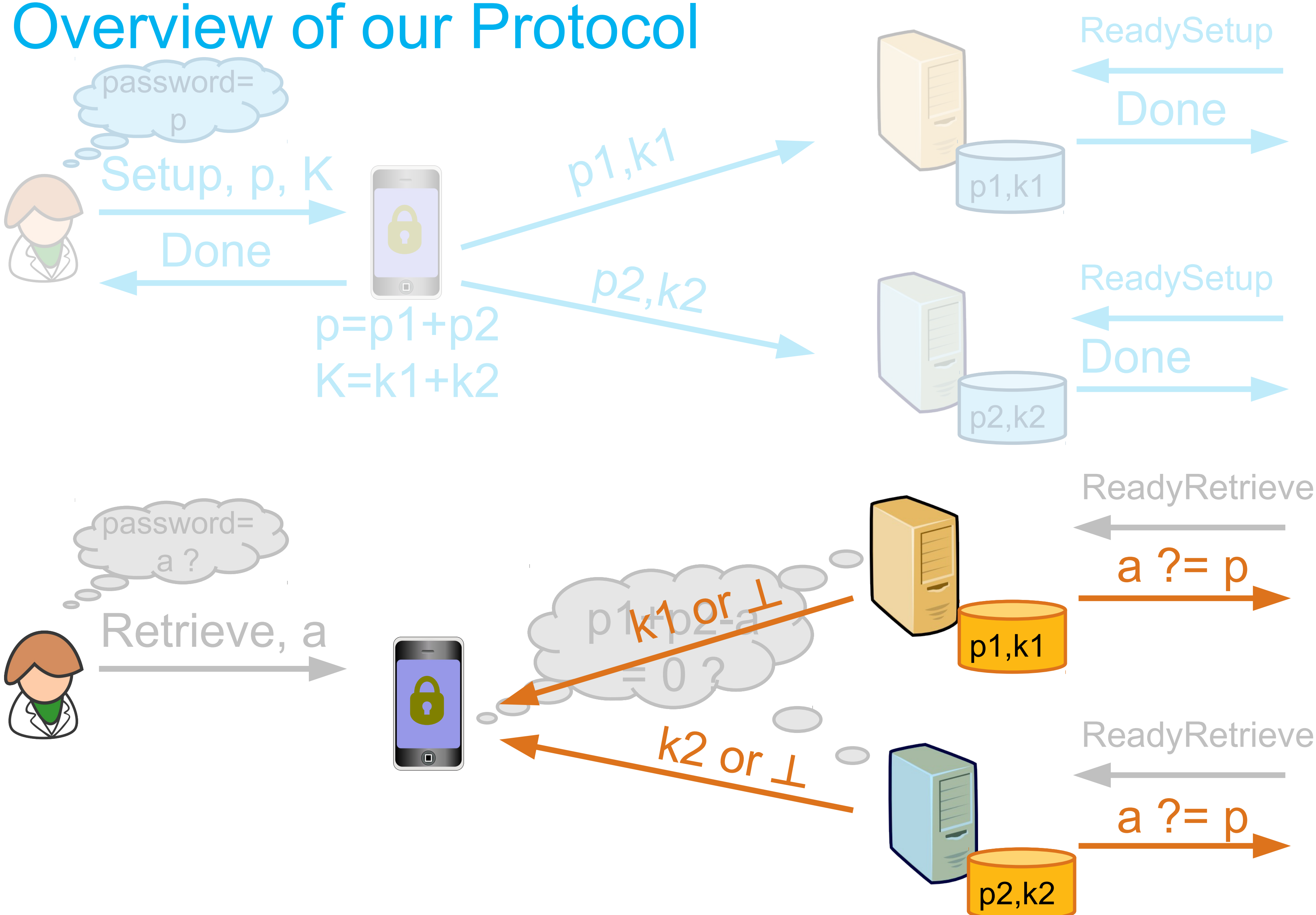
Overview of our Protocol



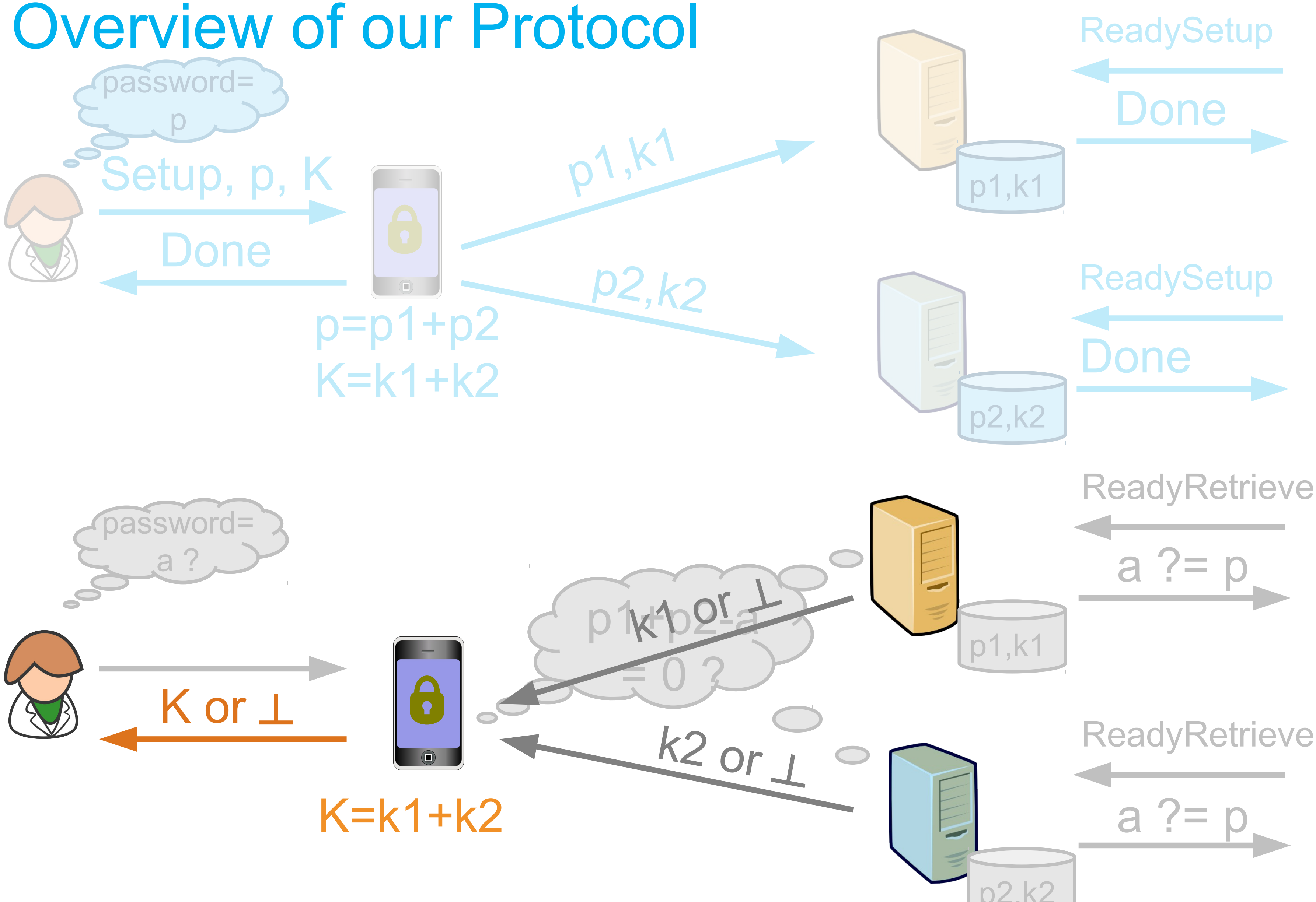
Overview of our Protocol



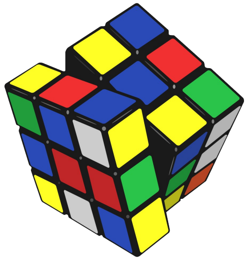
Overview of our Protocol



Overview of our Protocol



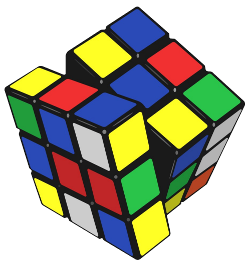
Difficulties with Security Against Dynamic Corruptions



- Selective decommitment problem:
  parties must never be committed to their input.
  - A party cannot send a ciphertext containing their input to another party: unsimulatable when recipient is then corrupted.
- We must work around this limitation,
  e.g., by using non-committing encryption based on one-time pads and secure erasures [BH91].
- Further modifications so that ZK proofs are still possible.

[BH91]: Beaver, Haber. *Cryptographic Protocols Provably Secure Against Dynamic Adversaries*. EUROCRYPT 1991.

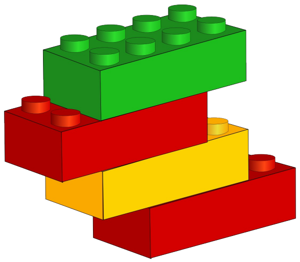
Difficulties with Security Against Dynamic Corruptions



- Selective decommitment problem:
  parties must never be committed to their input.
  - A party cannot send a ciphertext containing their input to another party: unsimulatable when recipient is then corrupted.
- We must work around this limitation,
  e.g., by using non-committing encryption based on one-time pads and secure erasures [BH91].
- Further modifications so that ZK proofs are still possible.

[BH91]: Beaver, Haber. *Cryptographic Protocols Provably Secure Against Dynamic Adversaries*. EUROCRYPT 1991.

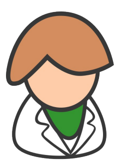
Building Blocks of Protocol



- Functionalities:
  - –One-sided authenticated channels (user ↔ servers), where only the servers are authenticated.
  - –Authenticated channels (server 1 ↔ server 2).
  - –(Local) common reference strings (CRS).
- Cryptographic schemes:
  - –Zero-knowledge proofs.
  - –Perfectly-hiding commitments (of special form).
  - –Non-committing encryption based on OTP and erasures.

High-level Idea of Protocol: Setup

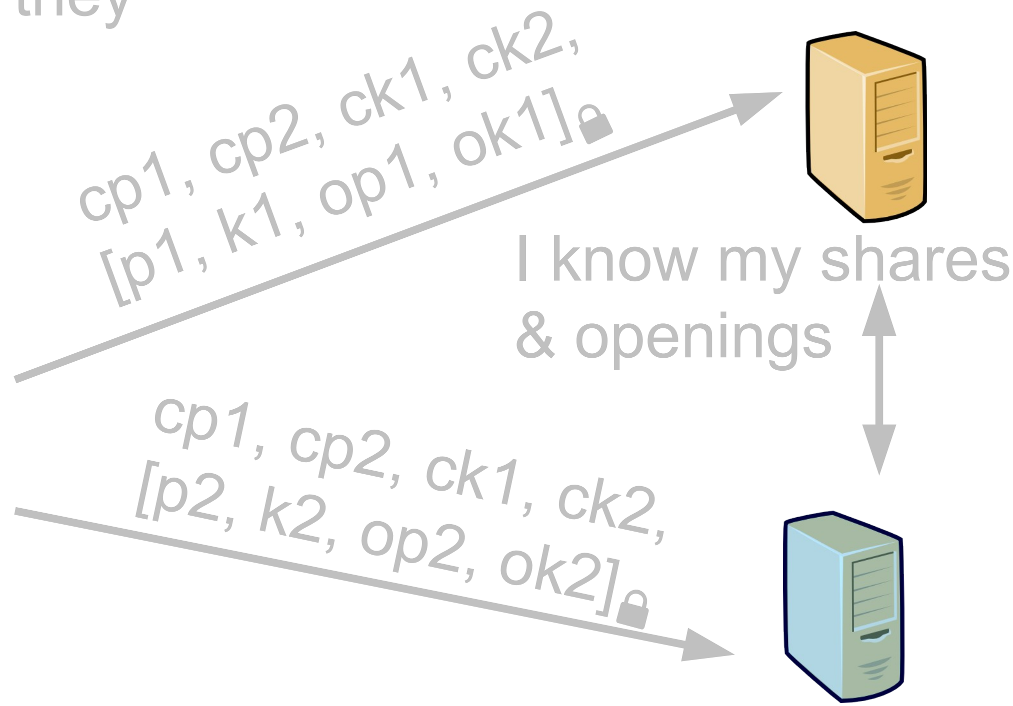
- User splits (p, K) into additive shares.
- Commits to shares. Sends all commitments to servers.
- Sends encrypted shares and openings to respective server.
- Servers prove to each other they know their shares.



(p, K)

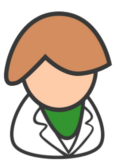
(p1+p2, k1+k2)

cp1 = Com(p1, op1)
cp2 = Com(p2, op2)
ck1 = Com(k1, ok1)
ck2 = Com(k2, ok2)



High-level Idea of Protocol: Setup

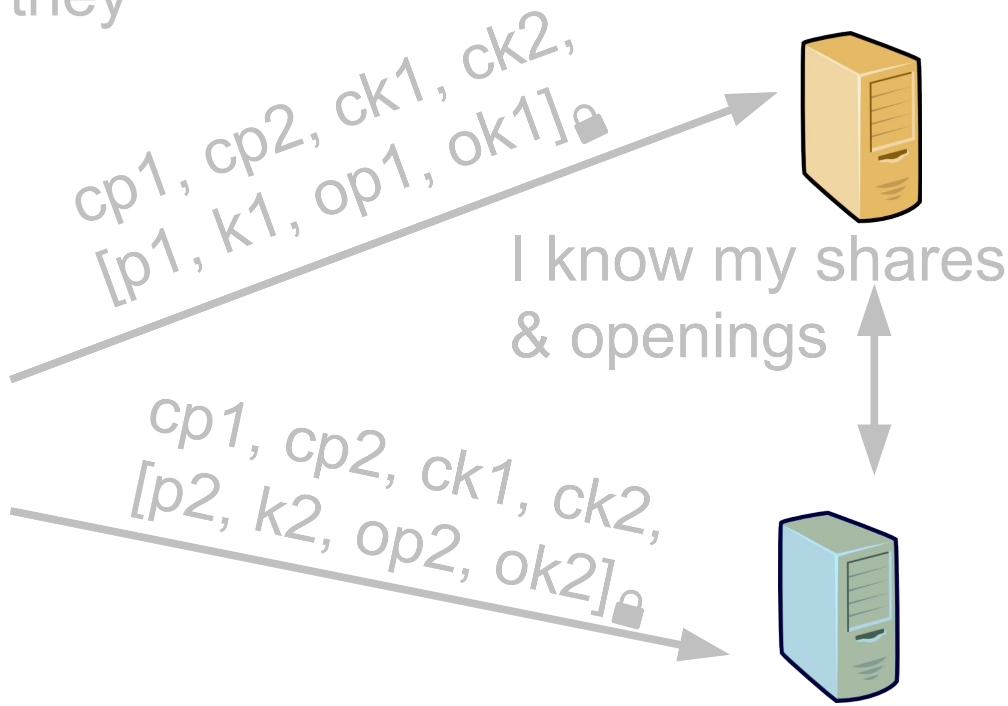
- User splits (p, K) into additive shares.
- Commits to shares. Sends all commitments to servers.
- Sends encrypted shares and openings to respective server.
- Servers prove to each other they know their shares.



(p, K)

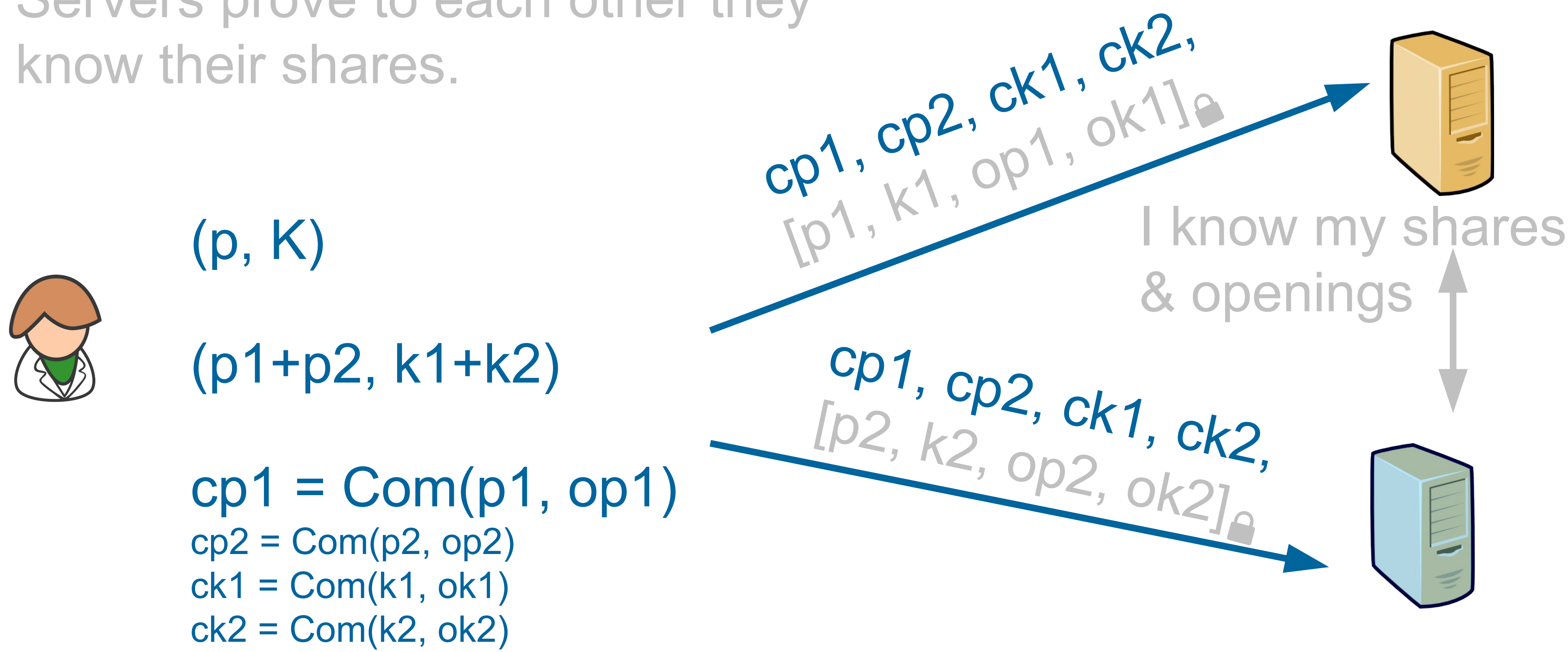
(p1+p2, k1+k2)

cp1 = Com(p1, op1)
cp2 = Com(p2, op2)
ck1 = Com(k1, ok1)
ck2 = Com(k2, ok2)



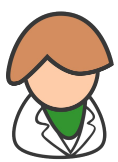
High-level Idea of Protocol: Setup

- User splits (p, K) into additive shares.
- Commits to shares. Sends all commitments to servers.
- Sends encrypted shares and openings to respective server.
- Servers prove to each other they know their shares.



High-level Idea of Protocol: Setup

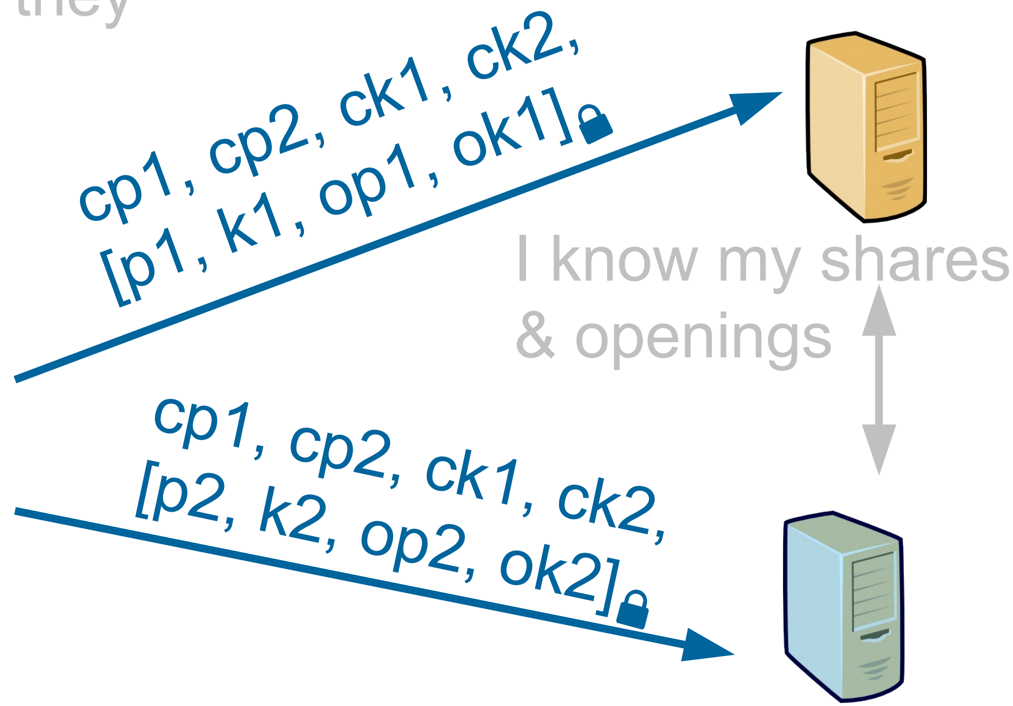
- User splits (p, K) into additive shares.
- Commits to shares. Sends all commitments to servers.
- Sends encrypted shares and openings to respective server.
- Servers prove to each other they know their shares.



(p, K)

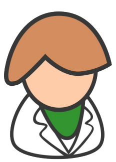
(p1+p2, k1+k2)

cp1 = Com(p1, op1)
cp2 = Com(p2, op2)
ck1 = Com(k1, ok1)
ck2 = Com(k2, ok2)



High-level Idea of Protocol: Setup

- User splits (p, K) into additive shares.
- Commits to shares. Sends all commitments to servers.
- Sends encrypted shares and openings to respective server.
- Servers prove to each other they know their shares.



(p, K)

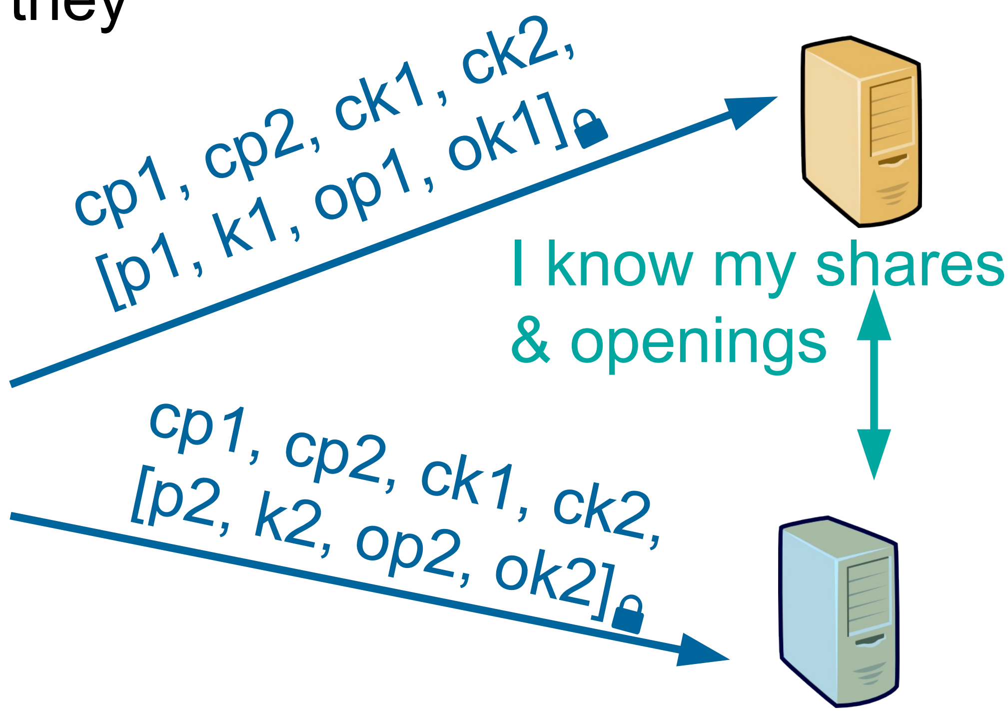
(p1+p2, k1+k2)

cp1 = Com(p1, op1)

cp2 = Com(p2, op2)

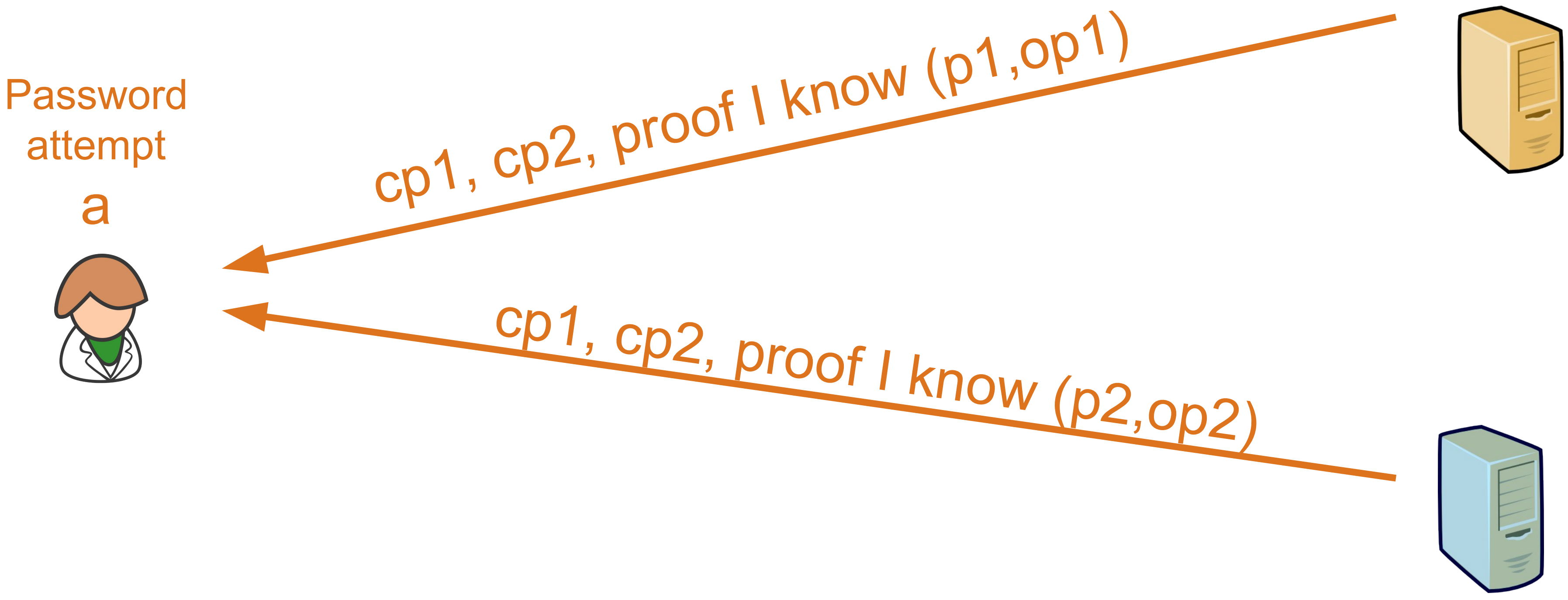
ck1 = Com(k1, ok1)

ck2 = Com(k2, ok2)



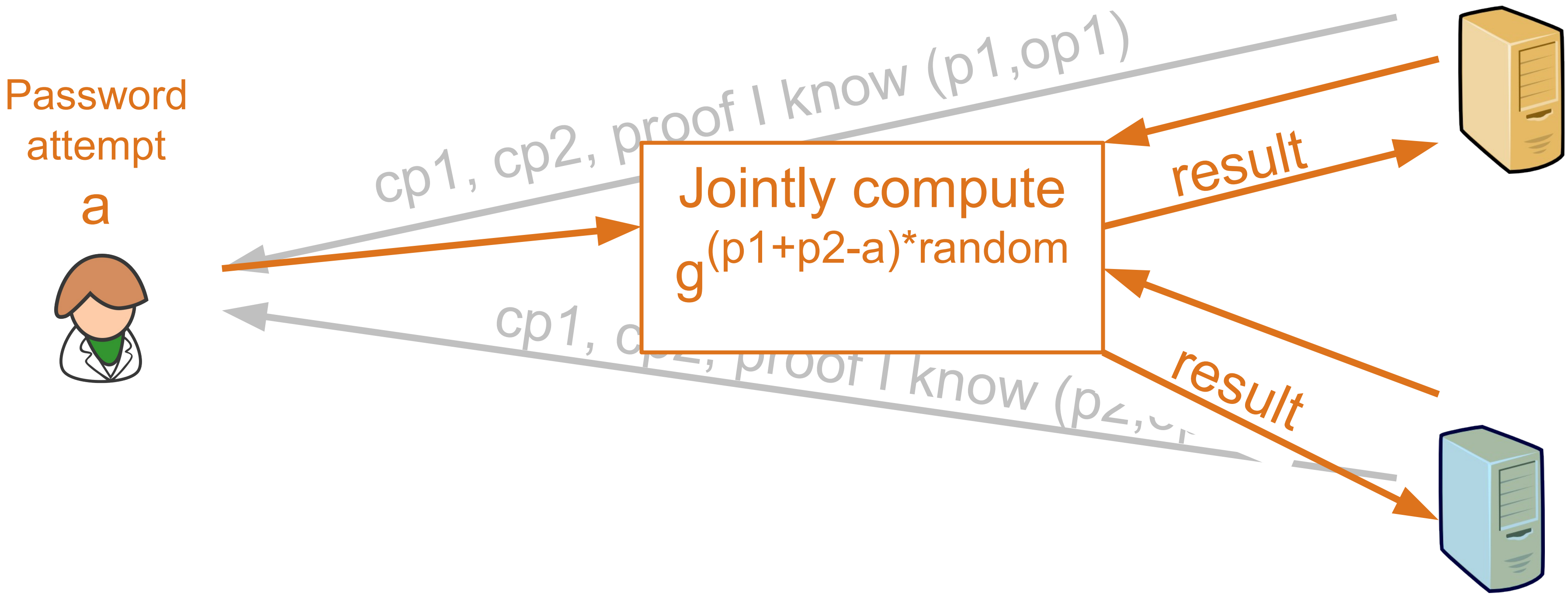
High-level Idea of Protocol: Retrieve

- Servers send commitments to user & prove they know shares.
- Servers jointly compute $g^{\delta * random}$ with help of user. $\delta = p1 + p2 - a$.
- If result=g^0: server send their shares of K & openings to user.



High-level Idea of Protocol: Retrieve

- Servers send commitments to user & prove they know shares.
- Servers jointly compute $g^{\delta * random}$ with help of user. δ=p1+p2-a.
- If result=g^0: server send their shares of K & openings to user.



High-level Idea of Protocol: Retrieve

- Servers send commitments to user & prove they know shares.
- Servers jointly compute $g^{\delta * random}$ with help of user. δ=p1+p2-a.
- If result=g^0: server send their shares of K & openings to user.

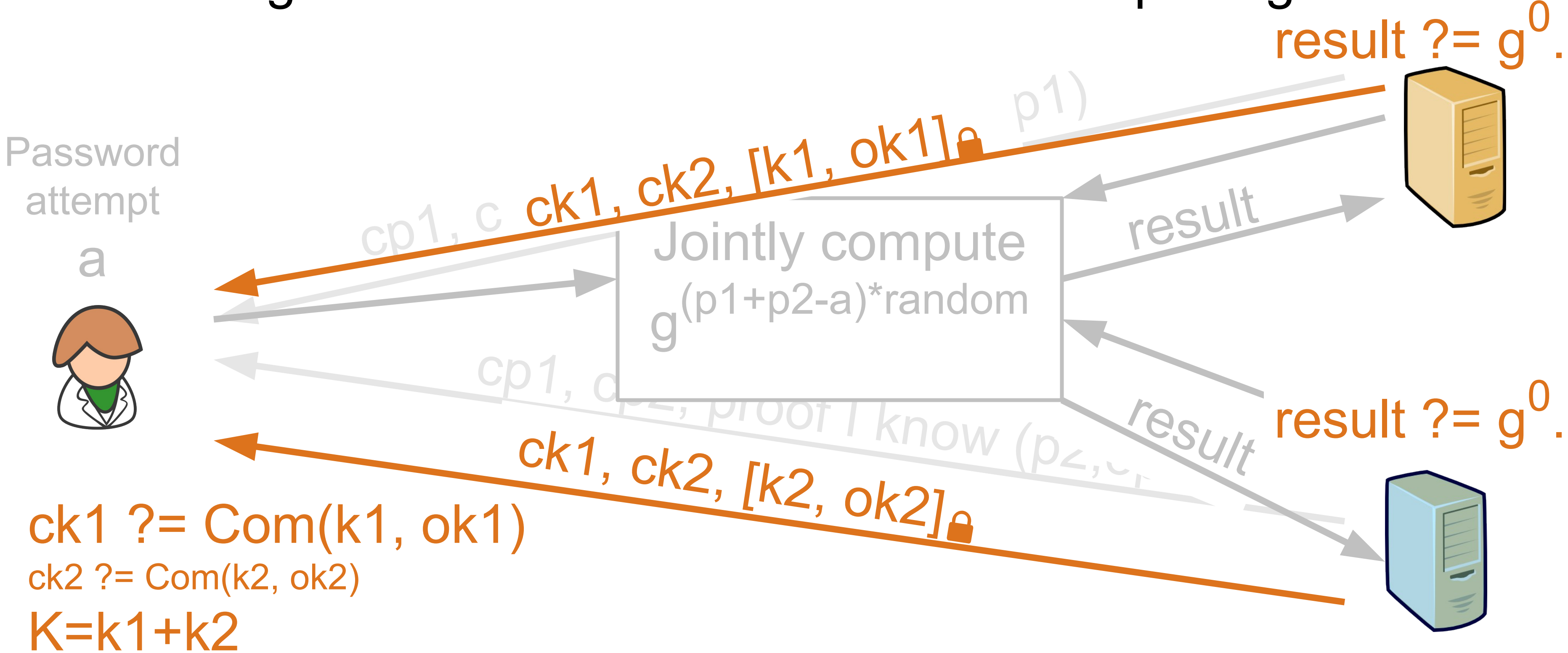
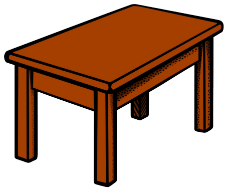


Table of contents

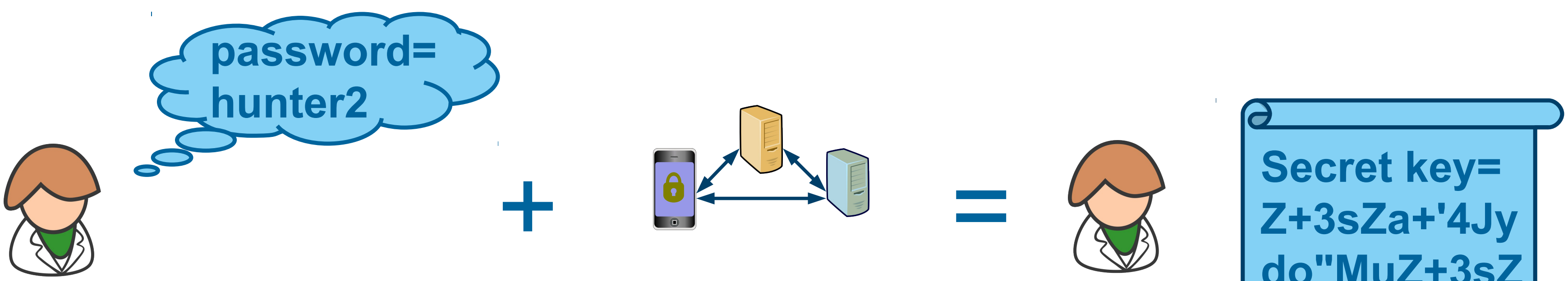


Our Goal: Protect Your Data

Conclusion

- First efficient 2-PASS that is UC-secure against dynamic corruptions.
  - Password protected from offline attack when ≥ 1 server honest.
  - Secret K protected when ≥ 1 server honest.

- Servers can recover from corruption.
- Efficient construction in standard model (w/ erasures). (A few hundred exponentiations ; ≤ 0.2 seconds total.)

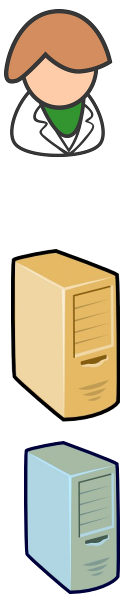


Our Goal: Protect Your Data

Backup slides: Protocol Detail

First Idea: Compute $g^{\delta\text{*random}}=g^{(p1+p2-a)\text{*random}}$

- Basic idea: use homomorphic properties of ElGamal.
- Let Elg[plaintext, rand, shkey]=$(h^{rand}, g^{plaintext}h^{rand*shkey})$.
  where $\log_g(h)$*shkey is the El-Gamal secret key.



From a, cp1, cp2 extract Elg[δ, -1, -op1-op2].

Exponentiate by random r0: → Elg[δ*r0, -r0, (-op1-op2)*r0].

Remove op1 & exp by r1: → Elg[δ*r0*r1, -r0*r1, -op2*r0*r1].

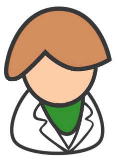
Remove op2 & exp by r2: → Elg[δ*r0*r1*r2, -r0*r1*r2, 0] =

$(h^{-r0*r1*r2}, g^{-\delta*r0*r1*r2})$.

- Parties additionally use zero-knowledge proofs throughout.

Final Idea: Compute $g^{\delta*random}=g^{(p1+p2-a)*random}$

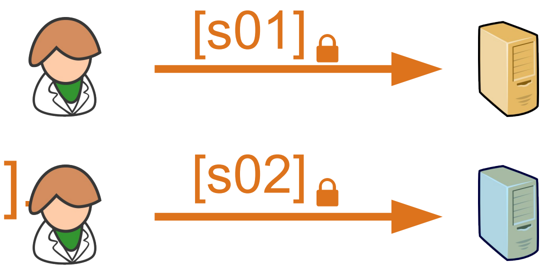
- Doesn't work: we need non-committing ciphertexts for dynamic corruption.
- Idea: add shared keys s01, s02, s12 (& send in a non-committing way).



From a, cp1, cp2 extract Elg[δ, -1, -op1-op2].

Add s01, s02 & exponentiate by r0:

→ Elg[δ*r0, -r0, (-op1-op2+s01+s02)*r0].



Add s12 & remove op1, s01 & exponentiate by r1:

→ Elg[δ*r0*r1, -r0*r1, (-op2+s02+s12)*r0*r1].

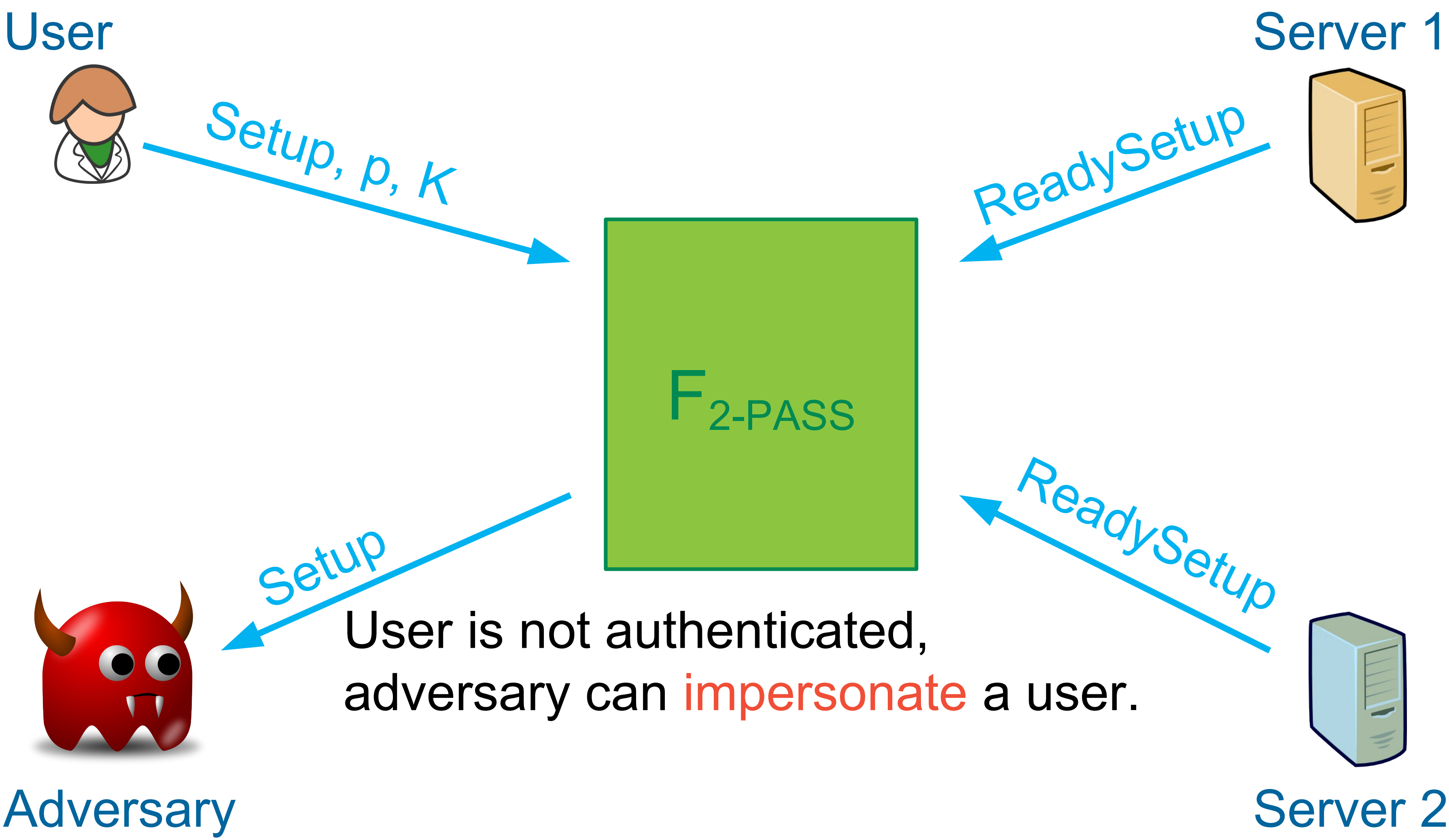
Remove op2, s02, s12 & exponentiate by r2:

→ Elg[δ*r0*r1*r2, -r0*r1*r2, 0] =(..., $g^{-\delta*r0*r1*r2}$).

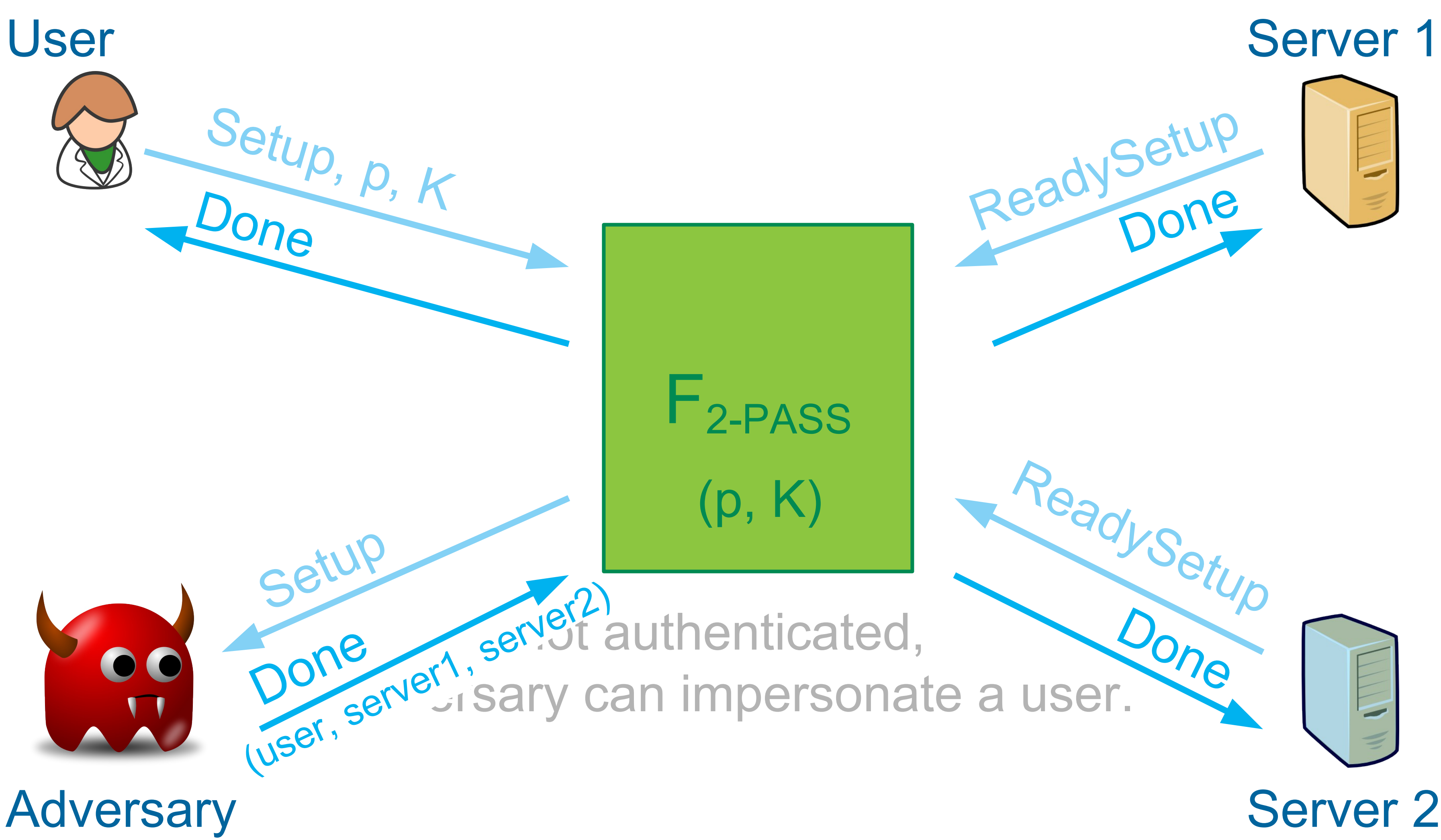
- Parties additionally use zero-knowledge proofs throughout, and use perfect-hiding commitments to keep track of s01, s02, s12.

Backup slides: Ideal Functionality

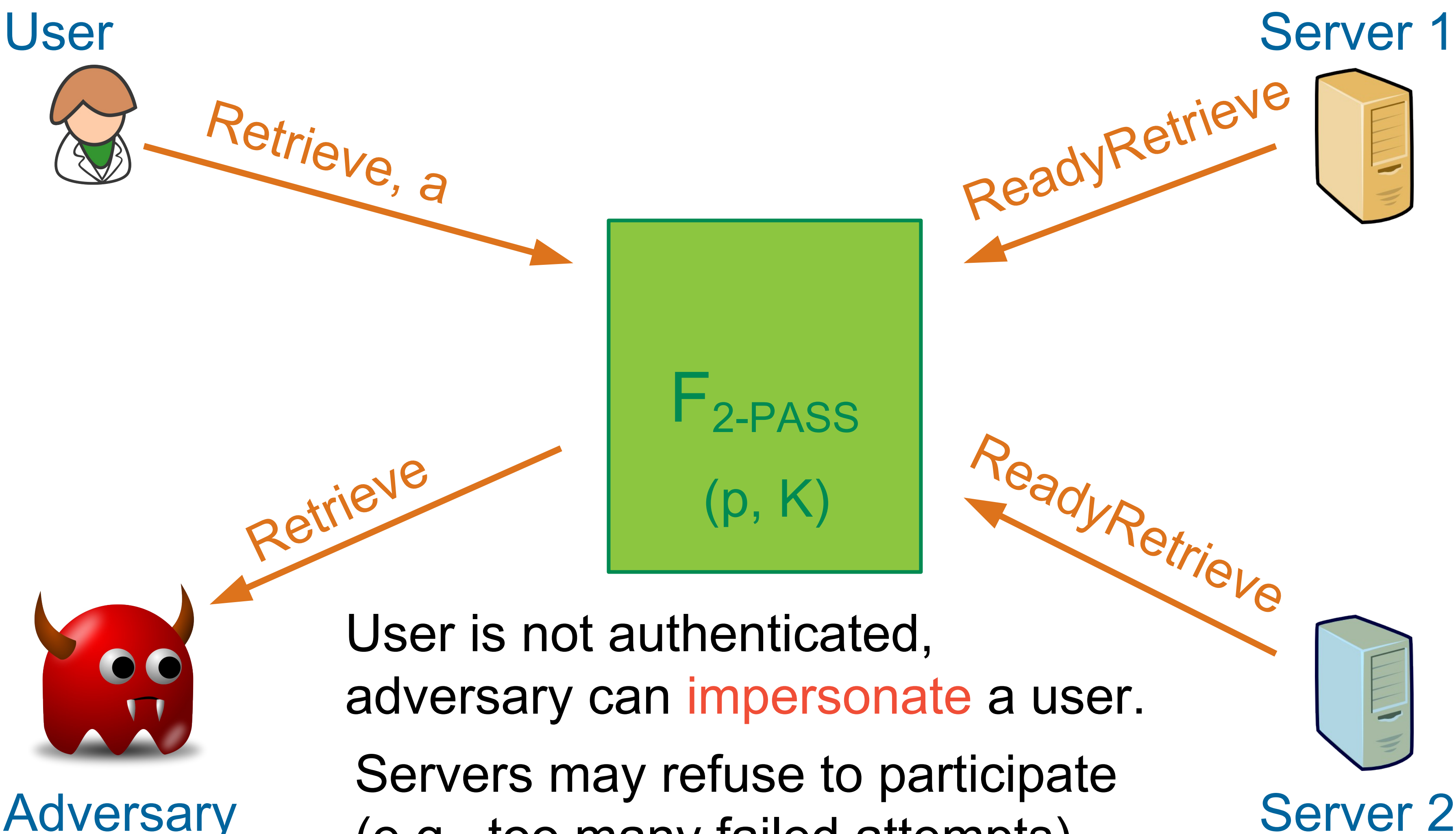
Ideal Functionality $F_{2\text{-PASS}}$: Setup



Ideal Functionality $F_{2\text{-}PASS}$: Setup



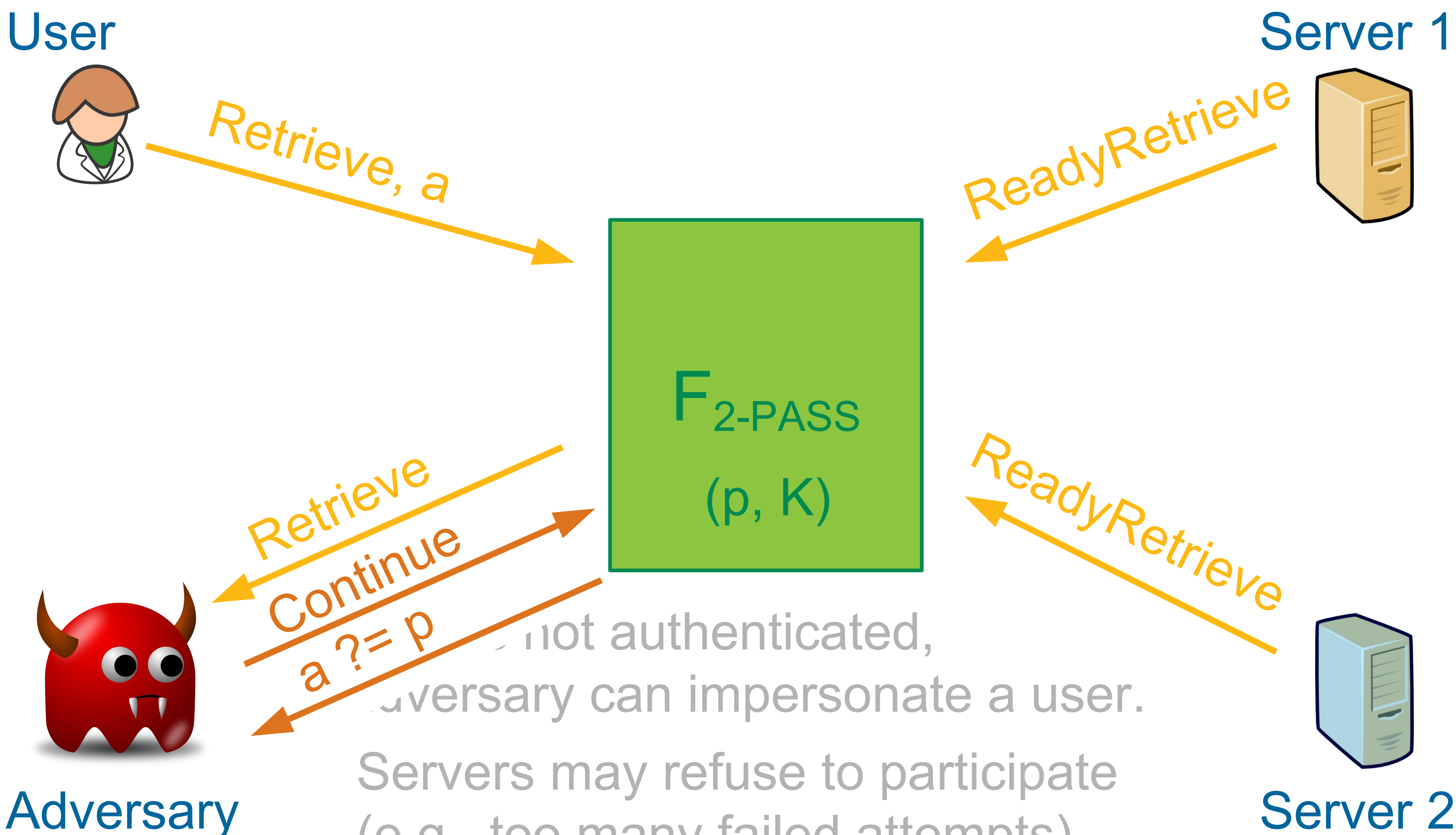
Ideal Functionality $F_{2\text{-PASS}}$: Retrieve



User is not authenticated, adversary can impersonate a user.

Servers may refuse to participate (e.g., too many failed attempts).

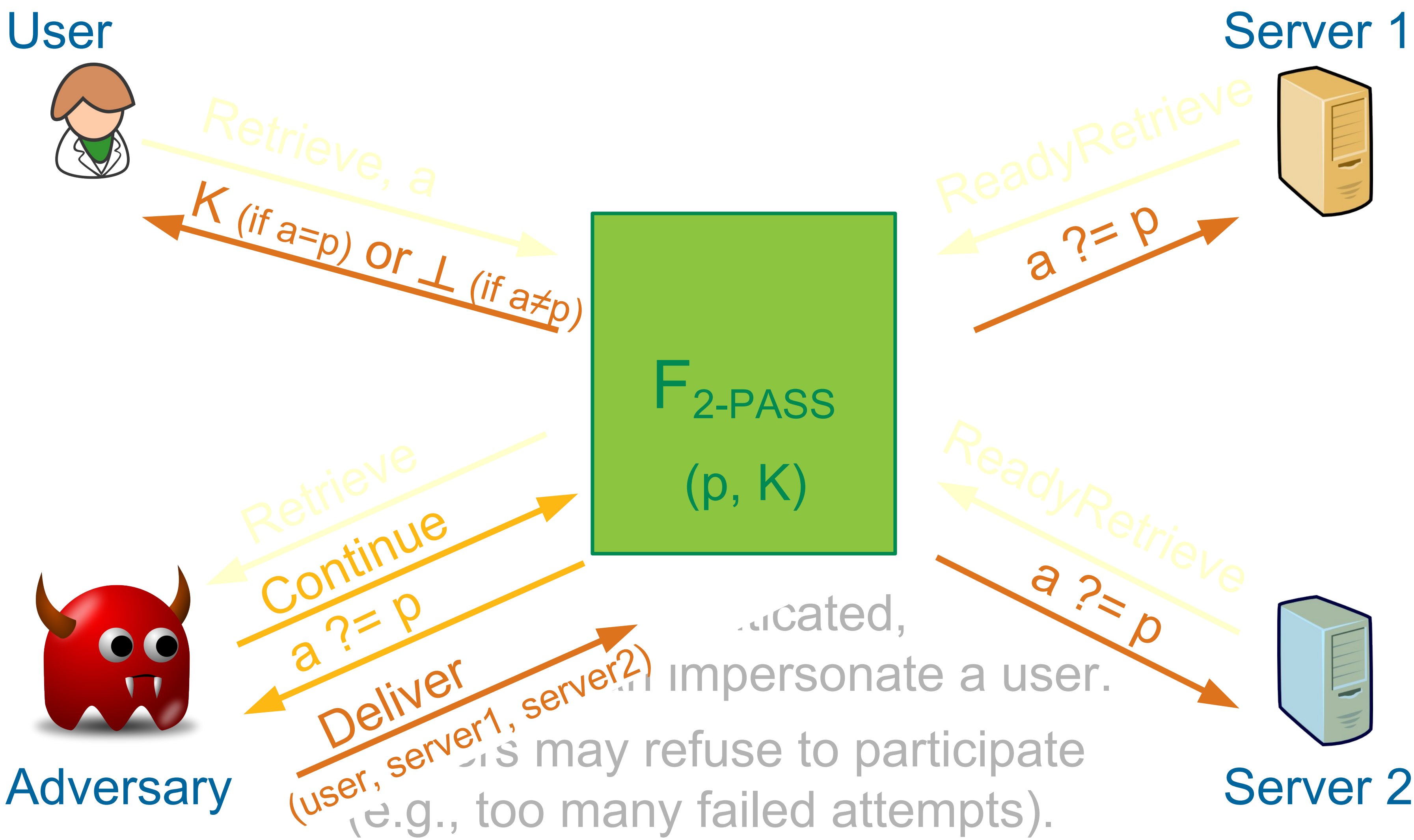
Ideal Functionality $F_{2\text{-PASS}}$: Retrieve



not authenticated,
versary can impersonate a user.

Servers may refuse to participate (e.g., too many failed attempts).

Ideal Functionality $F_{2\text{-PASS}}$: Retrieve



$F_{2\text{-PASS}}$: Modelling corruption

- Modelling corruption is necessary to be realistic.
- Corruption of user (per query):
  - –Adversary controls input & output.
  - –Adversary sees previous inputs for that query.
- Corruption of one server:
  - –Adversary controls input & output.
- Corruption of both servers:
  - –Adversary also learns (p, K) from $F_{2\text{-PASS}}$.
  - –Adversary can set (p, K) in $F_{2\text{-PASS}}$ for every query or permanently.

$F_{2\text{-PASS}}$: Recovery from Corruption

- Models that server detects it was hacked and takes remedial action (e.g., recovers from backup).
- Adversary may leave a corrupted server.
  - Both servers then run a Refresh protocol.
    - This aborts all currently running queries.
  - Afterwards, server is then not corrupted anymore (adversary doesn't control input & output).

2-PASS Ideal Functionality.

- Servers can refuse to service Retrieve queries (to defend against on-line brute force attacks).
- Servers and adversary learn if p = a (password attempt).
- If only one server compromised:
  - Adversary doesn't learn anything about the p, K, & a.
  - Cannot cause user to get wrong K.
- Two servers compromised: adversary gets (p, K), but not password attempts. (Also if user contacts wrong servers.)