Password-Based Cryptography: Strong Security from Weak Secrets

Anja Lehmann

IBM Research – Zurich

based on joint work with Jan Camenisch, Anna Lysyanskaya & Gregory Neven

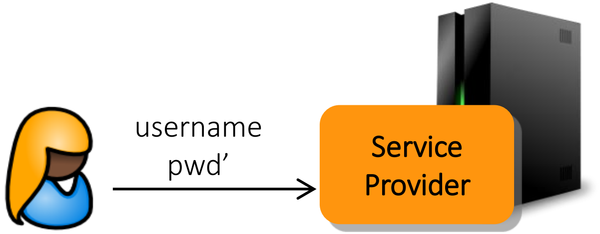
ROADMAP

- Password-Based Authentication
  How to make password checking systems *even* better

- Password-Authenticated Secret Sharing
  How to make cryptography accessible to end users

Password-Based Authentication

- Most prominent form of user authentication – convenient! No key, software, ...



Username	Hash
Alice	wb3822Ujsd4
Bob	b5kMsa8dsbn
Carol	77peCu52Kry

h' = h ?

stores only (salted) password hashes
$h = Hash(pwd)$

Password rules:

upper and lower case letters and numbers at least 16 characters in length

vs. 4-digit PIN for ATM cards

never reuse your password on another site

why the difference?

change your passwords periodically

the ATM will retain the card after 3 failed attempts!

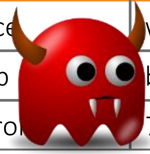
Password-Based Authentication

- If service provider is trusted & throttles after too many failed attempts
  $\rightarrow$ short passwords are sufficient!

- <u>But</u> main threat to password security
  is server compromise

h' = h ?

Username	Hash
Alice	wb3822Ujsd4
Bob	b5kMsa8dsbn
Caro	77peCu52Kry



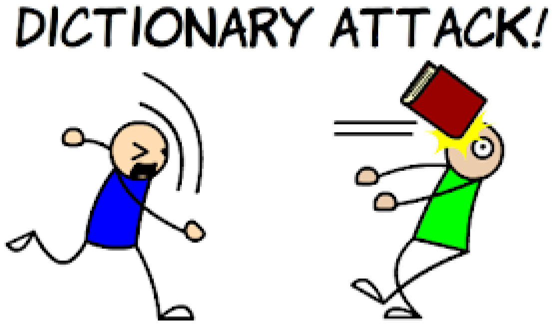
stores only (salted) password hashes
$h = Hash(pwd)$

- The more complicated our passwords are, the more guesses the adversary need

NIST: 16-character passwords have 30 bits of entropy ~ 1 billion possibilities

vs.

$150 GPUs can test ~ 300 billions/second

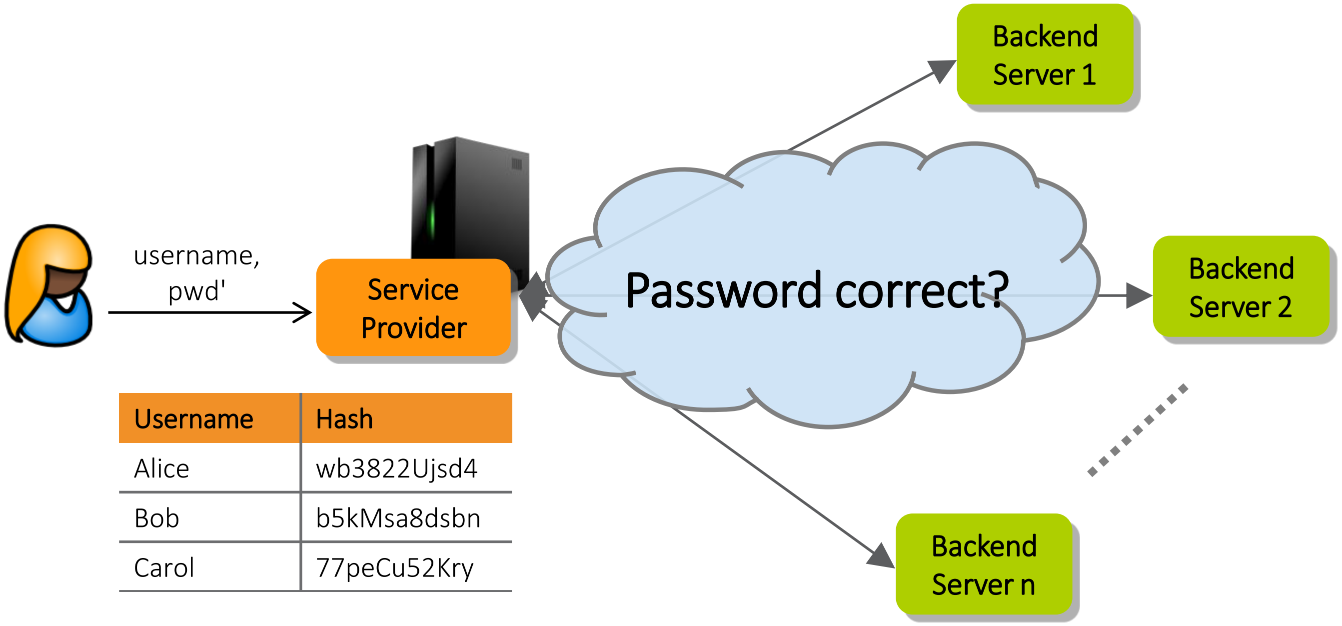


Passwords inherently insecure?

No! We’re just using them incorrectly …

Password-Based Authentication Done Right

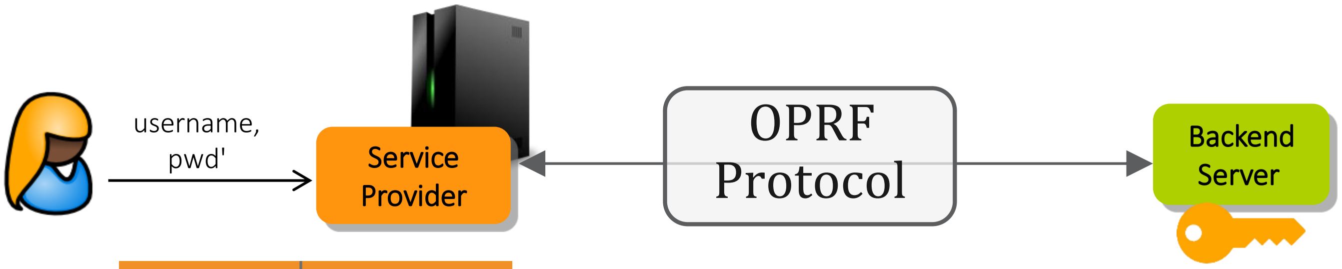
- Offline attacks are inherent in single-server setting
- Solution: split password verification over multiple servers



Username	Hash
Alice	wb3822Ujsd4
Bob	b5kMsa8dsbn
Carol	77peCu52Kry

Pythia: OPRF Service

- Replace $Hash$ by a secure $\mathrm{PRF}($  $, pwd)$
- Store  at remote server & evaluate PRF obliviously

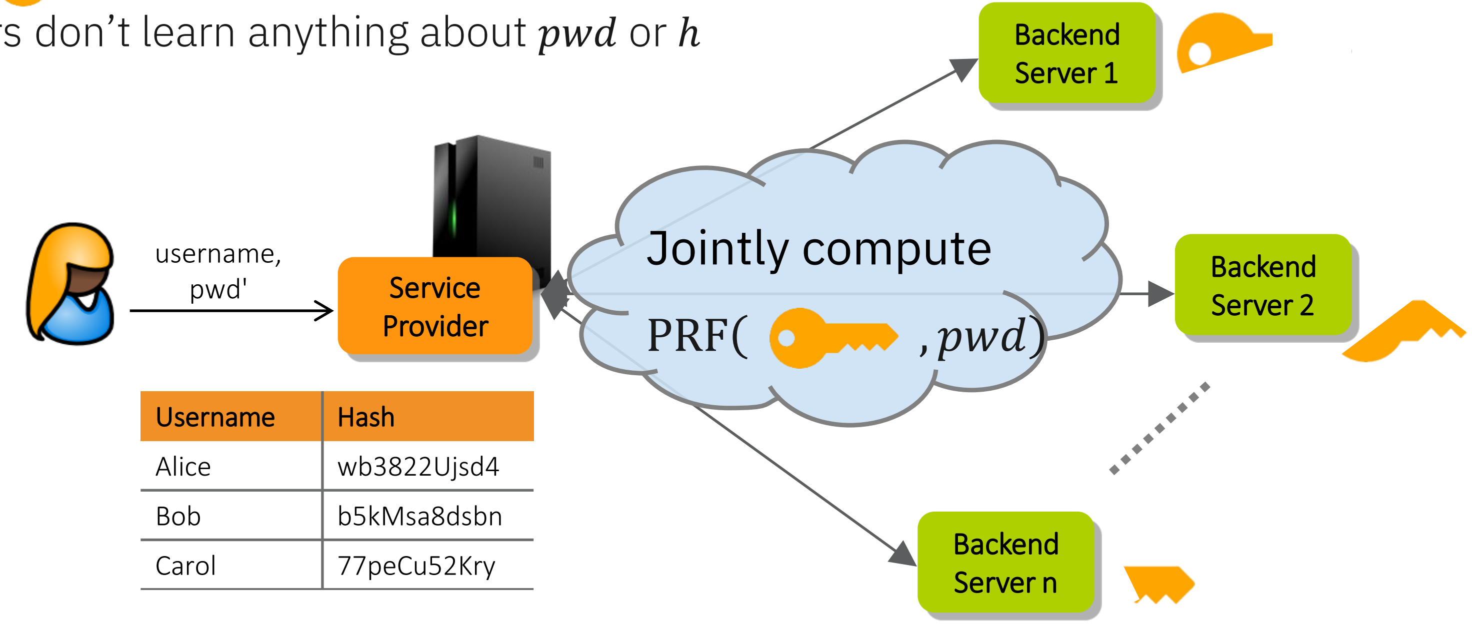


Username	Hash
Alice	wb3822Ujsd4
Bob	b5kMsa8dsbn
Carol	77peCu52Kry

[ECSJR'15] Everspaugh, Chatterjee, Scott, Juels, Ristenpart. *The Pythia PRF Service.* USENIX 2015.

Distributed Password Verification | High-Level Idea

- Replace $Hash$ by a secure $\mathrm{PRF}(\quad, pwd)$
- Split secret key into n shares
- $h = \mathrm{PRF}(\quad, pwd)$ computed distributed:
  - Servers don't learn anything about pwd or h

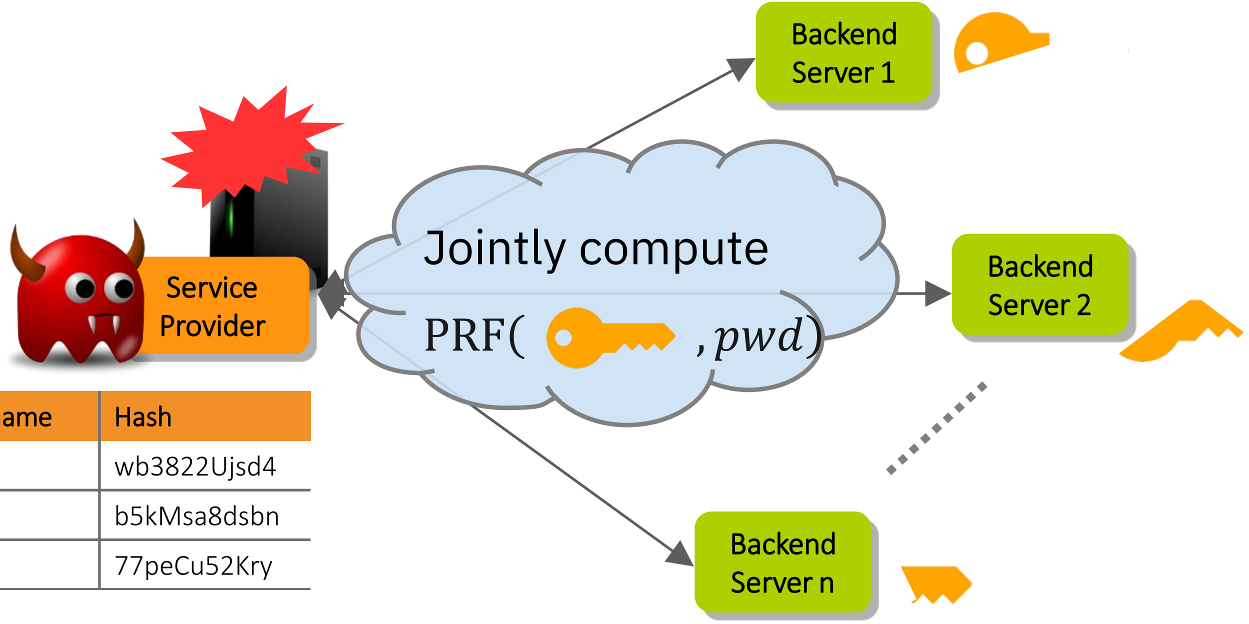


Username	Hash
Alice	wb3822Ujsd4
Bob	b5kMsa8dsbn
Carol	77peCu52Kry

[CLN'15] Camenisch, Lehmann, Neven. *Optimal Distributed Password Verification.* CCS 2015.

Distributed Password Verification | Security

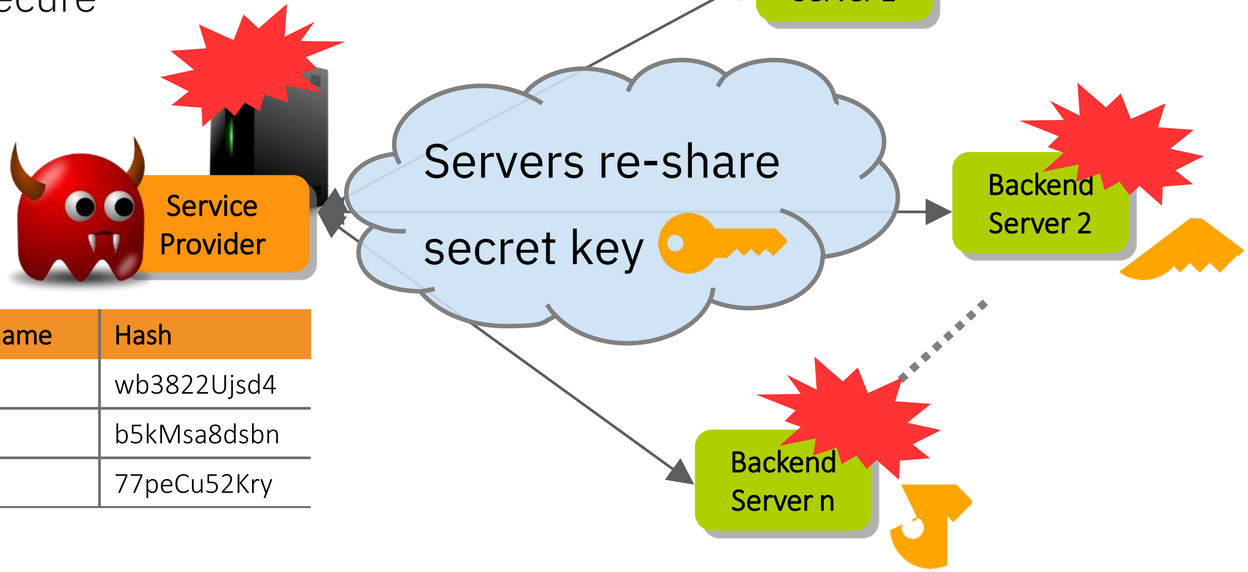
- Secret key has high-entropy, i.e., cannot be guessed
  - → Adversary needs backend servers (or full key) to verify password guesses
  - → Backend servers will stop verification if activity is suspicious



Username	Hash
Alice	wb3822Ujsd4
Bob	b5kMsa8dsbn
Carol	77peCu52Kry

Distributed Password Verification | Proactive Security

- Secret key gets re-shared periodically
  - → All previous key shares get useless
  - → Adversary must break into **all** servers at the **same time**
- As long as one server is not corrupted
  - → Passwords are secure



Username	Hash
Alice	wb3822Ujsd4
Bob	b5kMsa8dsbn
Carol	77peCu52Kry

DPV Protocol

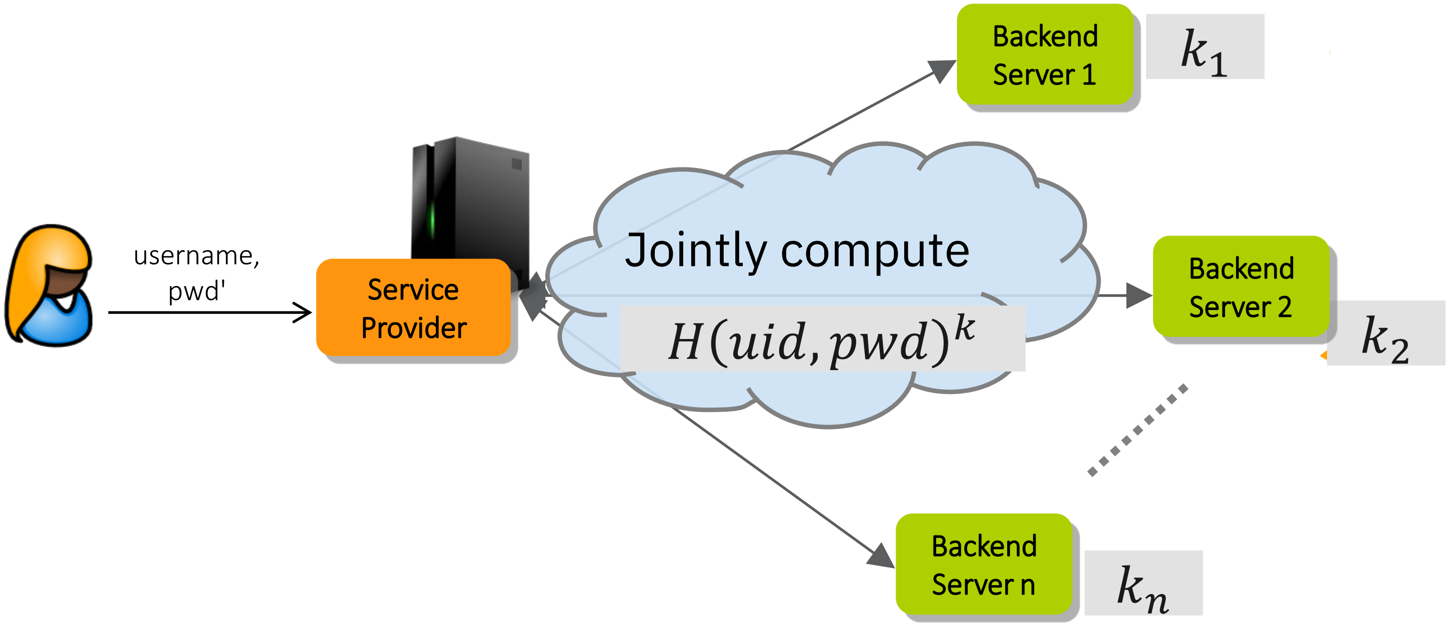
Optimal Distributed Password Verification. ACM CCS'15.
Camenisch, Lehmann, Neven.

Distributed Password Verification | Protocol

- Replace $Hash$ by a secure $H(uid, pwd)^k$
- Split secret key $k = k_1 + k_2 + \ldots + k_n \bmod q$

k = random element in Zq

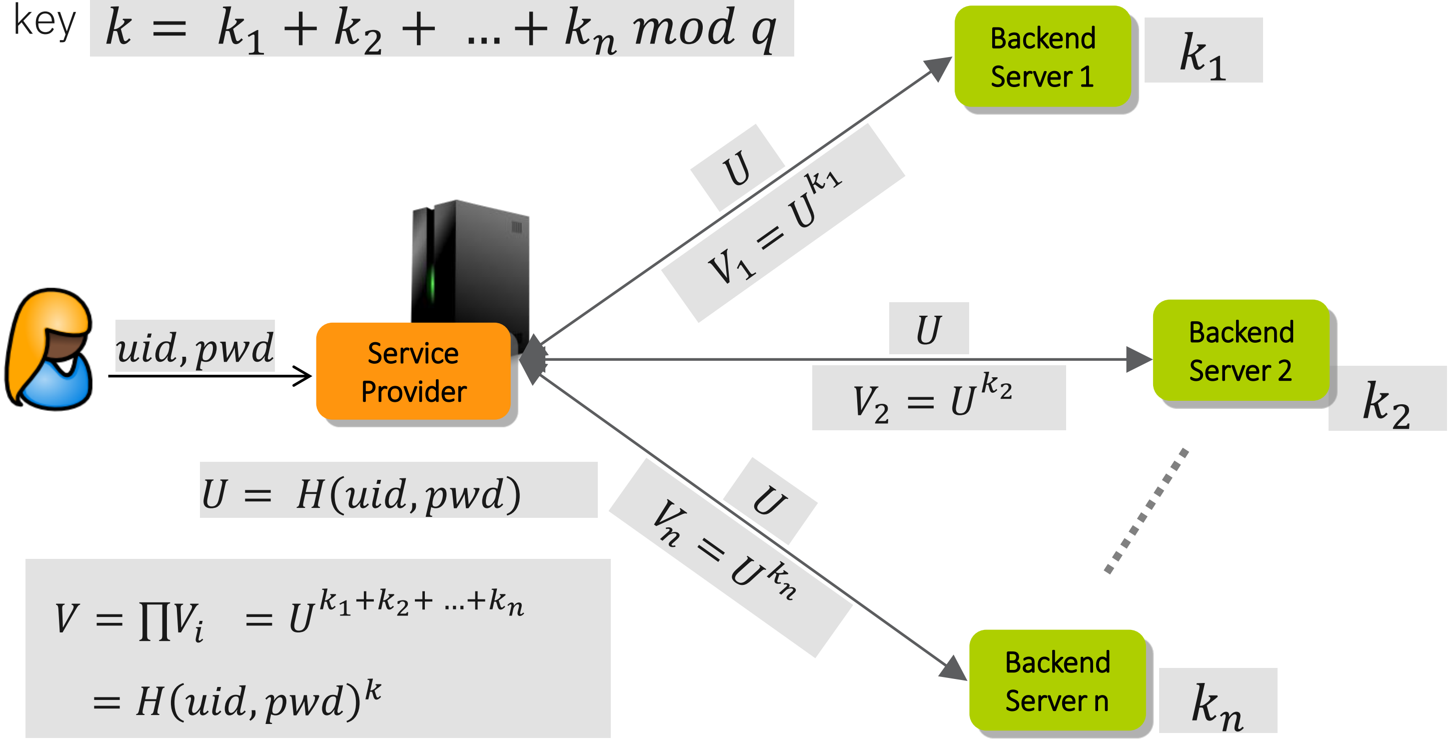
Cyclic group of prime order q



Naor, Pinkas, Reingold. *Distributed Pseudorandom Functions and KDCs*. Eurocrypt '99

Distributed Password Verification | Protocol

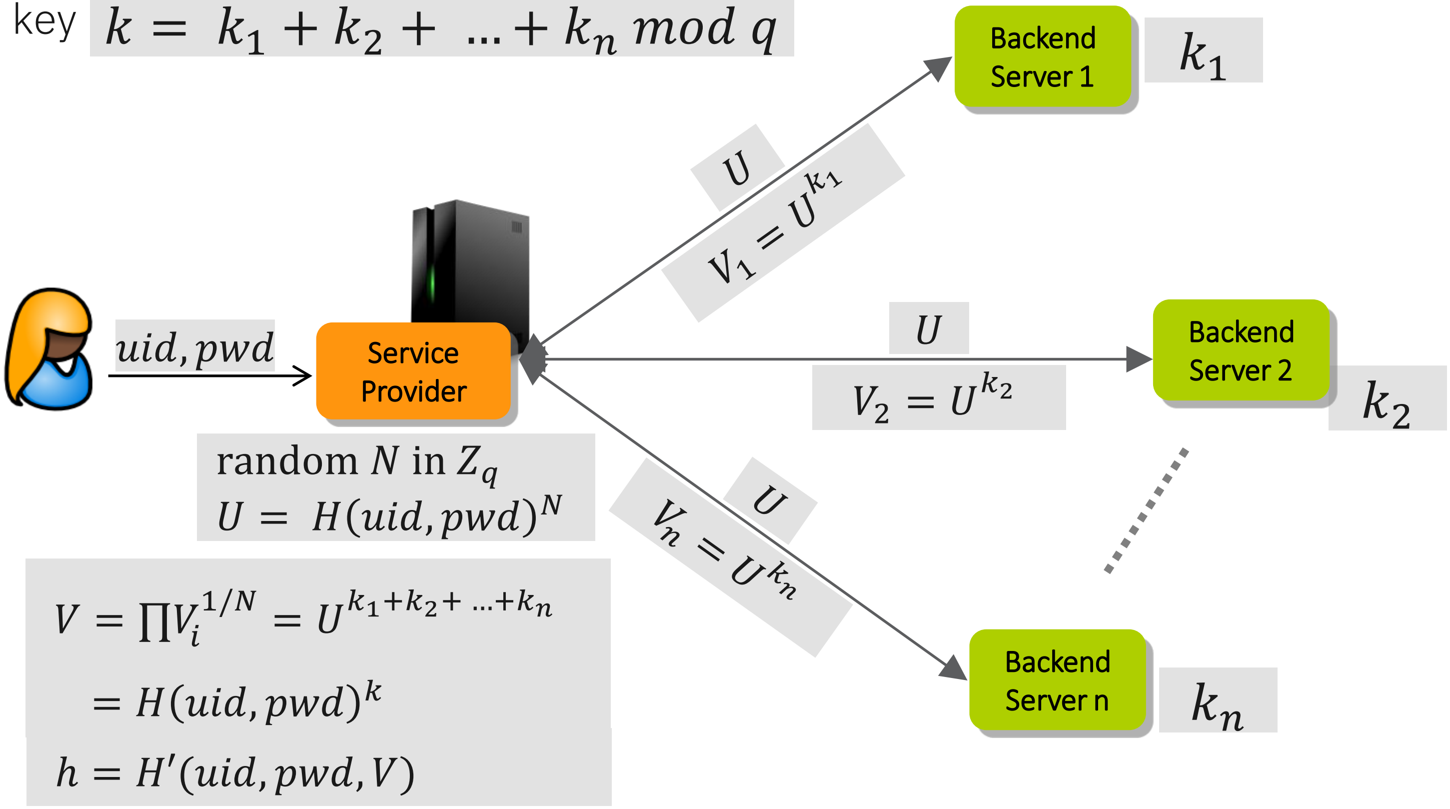
- Replace *Hash* by a secure $H(uid, pwd)^k$
- Split secret key $k = k_1 + k_2 + \ldots + k_n \bmod q$



Naor, Pinkas, Reingold. *Distributed Pseudorandom Functions and KDCs*. Eurocrypt '99

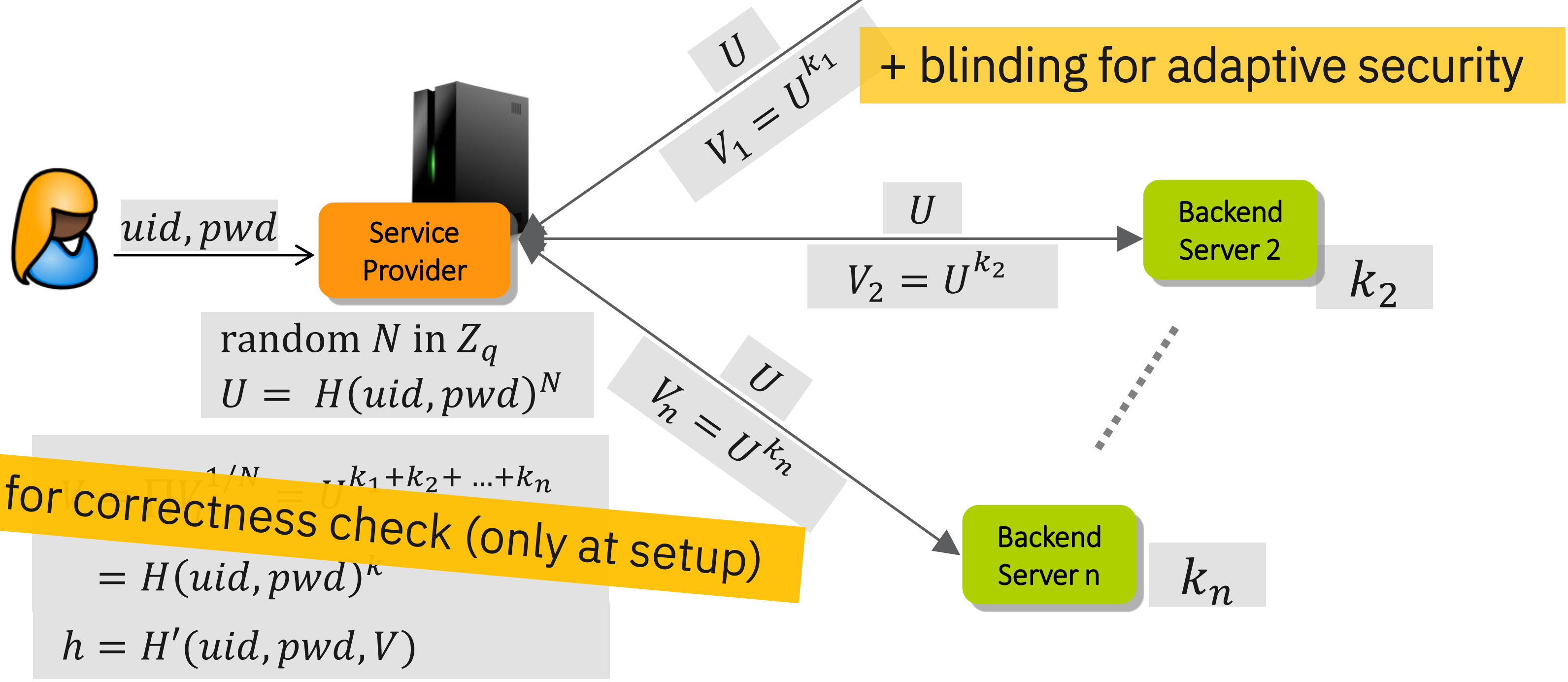
Distributed Password Verification | Protocol

- Replace *Hash* by a secure $H(uid, pwd)^k$
- Split secret key $k = k_1 + k_2 + \ldots + k_n \bmod q$



Distributed Password Verification | Protocol

- Replace *Hash* by a secure $H(uid, pwd)^k$
- Split secret key $k = k_1 + k_2 + \ldots + k_n \bmod q$



Distributed Password Verification | Protocol

- Proactive security & re-sharing of keys:

Agree on pseudorandom shares of zero:

$$\delta_1 + \delta_2 + \ldots + \delta_n = 0 \bmod q$$

$$k = k'_1 + k'_2 + \ldots + k'_n \bmod q$$

Backend Server 1 $\quad k'_1 = k_1 + \delta_1$

Backend Server 2 $\quad k'_2 = k_2 + \delta_2$

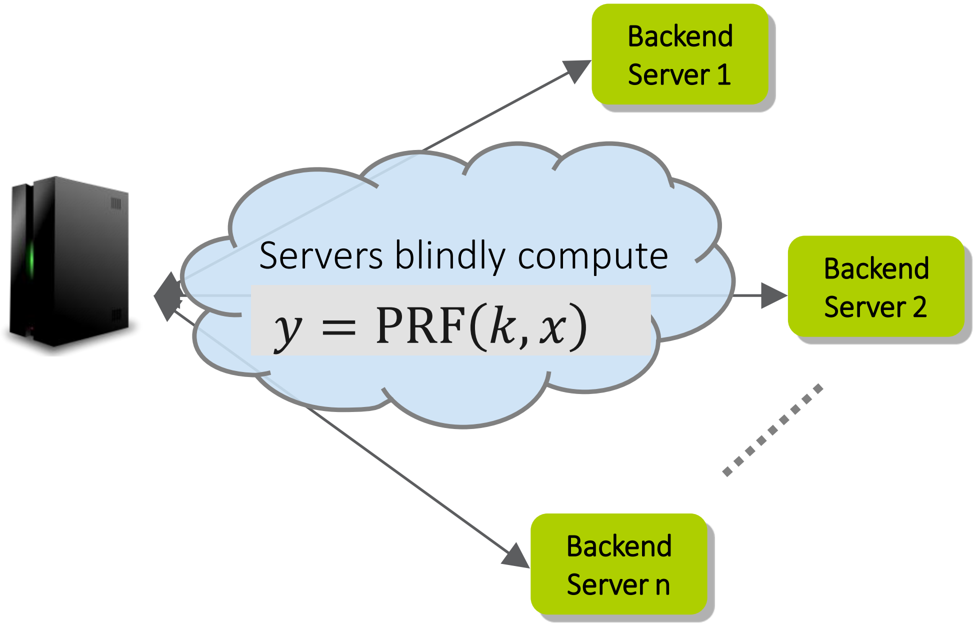
Backend Server n $\quad k'_n = k_n + \delta_n$

- No updates of "hash table" needed!

+ non-interactive protocol for computing δ_i (leveraging trusted setup & "secure" backup)

Distributed Password Verification
= Distributed OPRF (Oblivious PRF)

compute $y = \mathrm{PRF}(k, x)$ in a blind & distributed manner



Distributed Password Verification
= Distributed OPRF (Oblivious PRF)

compute $y = \mathrm{PRF}(k, x)$ in a blind & distributed manner

$k = \mathrm{KGen}(\tau)$

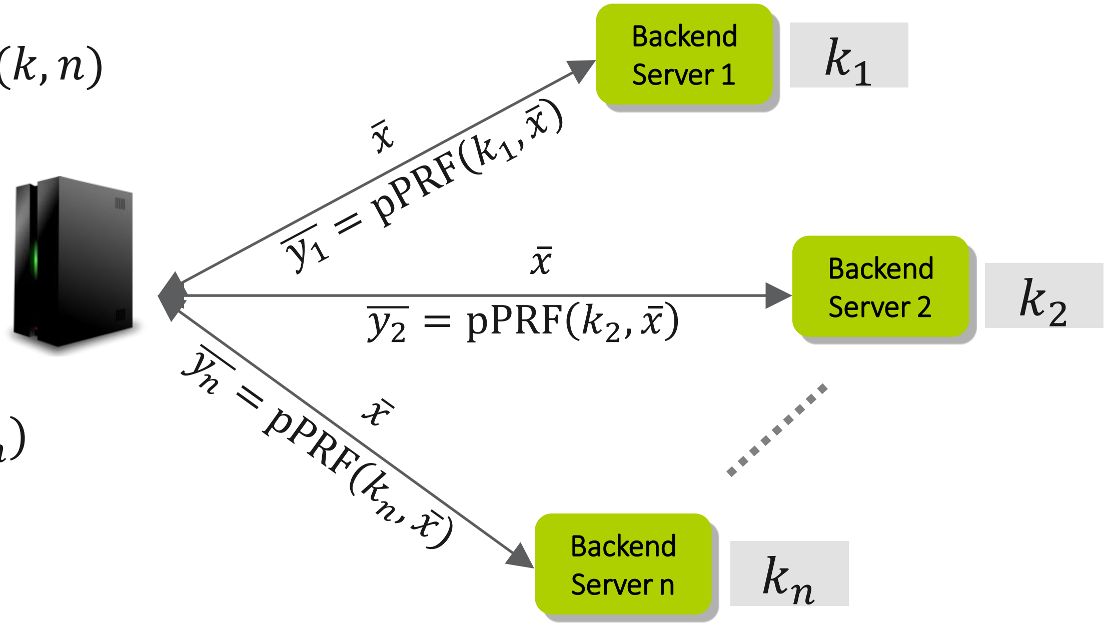
$k_1 + k_2 + \ldots + k_n = \mathrm{Share}(k, n)$

$\bar{x} = \mathrm{Blind}(x)$

$\bar{y} = \mathrm{Comb}(\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n)$

$y = \mathrm{Unblind}(\bar{y})$

s.t. $y = \mathrm{PRF}(k, x)$

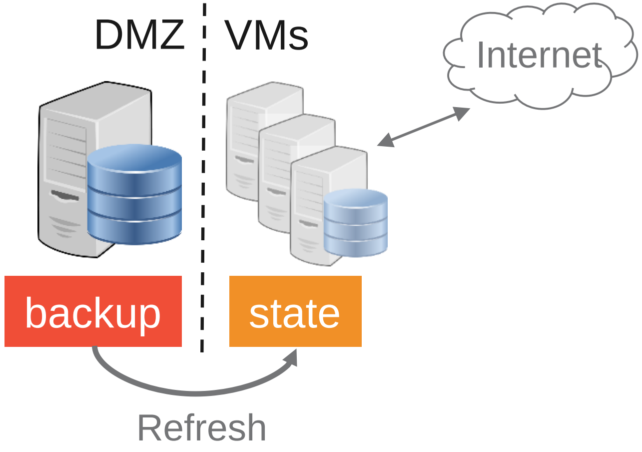


Distributed Password Verification | Security & Efficiency

- Efficient & round-optimal protocol
    - 1 round of communication
    - Login: one exponentiation per server (two for SP)
    - Non-interactive key refresh
    - Prototype implementation & evaluation (Ergon)
        - 3 backend servers, each 16 x 2.9Ghz core: 285 logins/second

- Provable security in very strong security model
    - Adaptive & active adversaries, UC Framework
    - One-More Gap DH (OMGDH), Random Oracle

- Password protection back where it belongs: on the server !



ROADMAP

- Password-Based Authentication
  How to make password checking systems *even* better

- Password-Authenticated Secret Sharing
  How to make cryptography accessible to end users

How to bridge cryptographic keys & humans

- Most cryptography relies on strong secret keys
  - Easy to manage for servers and devices … not so easy for humans

-----BEGIN PRIVATE KEY-----
MIICXgIBAAKBgQDHikastc8+I81zCg/qWW8dMr8mqvXQ3qbPAmu0RjxoZVI47tvs
kYlFAXOf0sPrhO2nUuooJngnHV0639iTTEYG1vckNaW2R6U5QTdQ5Rq5u+uV3pMk
7w7Vs4n3urQ6jnqt2rTXbC1DNa/PFeAZatbf7ffBBy0IGO0zc128IshYcwIDAQAB
AoGBALTNl2JxTvq4SDW/3VH0fZkQXWH1MM10oeMbB2qO5beWb11FGaOO77nGKfWc
bYgfp5Ogrql4yhBvLAXnxH8bcqqwORtFhlyV68U1y4R+8WxDNh0aevxH8hRS/1X5
031DJm1JlU0E+vStiktN0tC3ebH5hE+1OxbIHSZ+WOWLYX7JAkEA5uigRgKp8ScG
auUijvdOLZIhHWq7y5Wz+nOHUuDw8P7wOTKU34QJAoWEe771p9Pf/GTA/kr0BQnP
QvWUDxGzJwJBAN05C6krwPeryFKrKtjOGJIniIoY72wRnoNcdEEs3HDRhf48YWFo
riRbZylzzzNFy/gmzT6XJQTfktGqq+FZD9UCQGIJaGrxHJgfmpDuAhMzGsUsYtTr
iRox0D1Iqa7dhE693t5aBG010OF6MLqdZA1CXrn5SRtuVVaCSLZEL/2J5UcCQQDA
d3MXucNnN4NPuS/L9HMYJWD7lPoosaORcgyK77bSSNgk+u9WSjbH1uYIAIPSffUZ
bti+jc1dUg5wb+aeZlgJAkEAurrpmpqj5vg087ZngKfFGR5rozDiTsK5DceTV97K
a3Y+Nzl+XWTxDBWk4YPh2ZlKv402hZEfWBYxUDn5ZkH/bw==
-----END PRIVATE KEY-----

E.g., encrypted cloud storage (untrusted cloud)

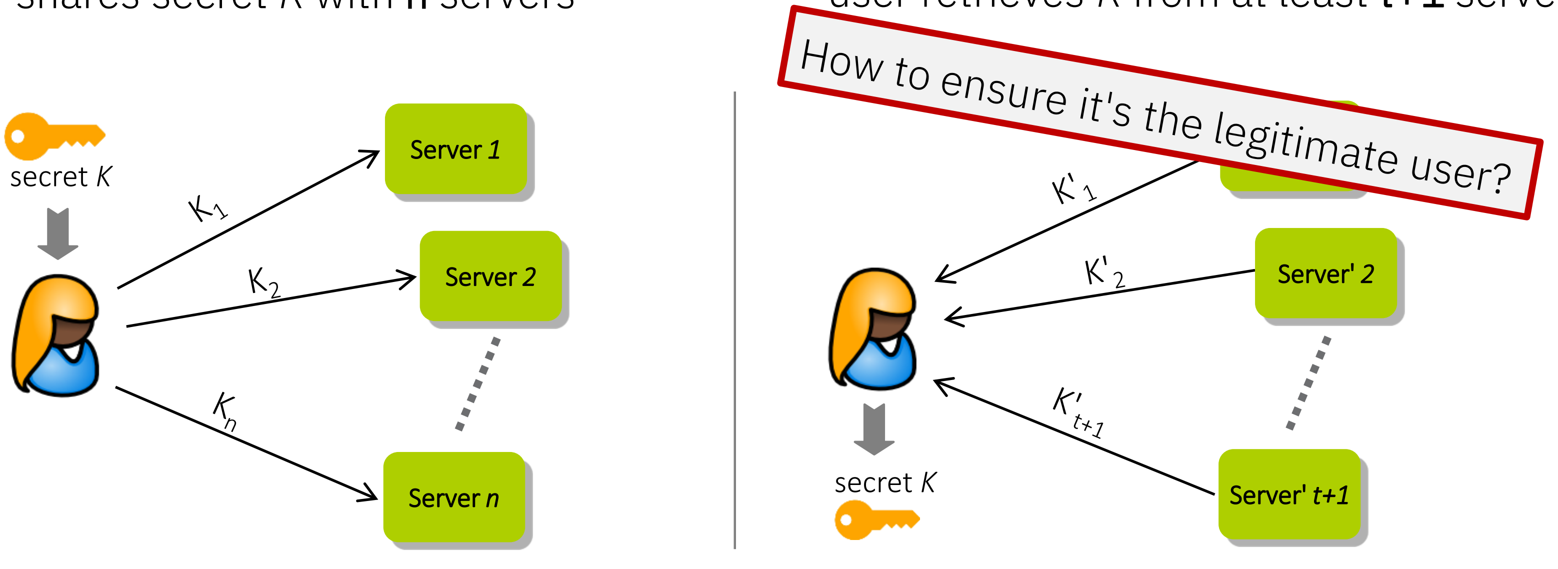
How to store the secret key?

- Access from many devices
- Trusted hardware inconvenient
- Device(s) can get broken or lost

Secret Sharing | Shamir' 79

user shares secret K with **n** servers

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



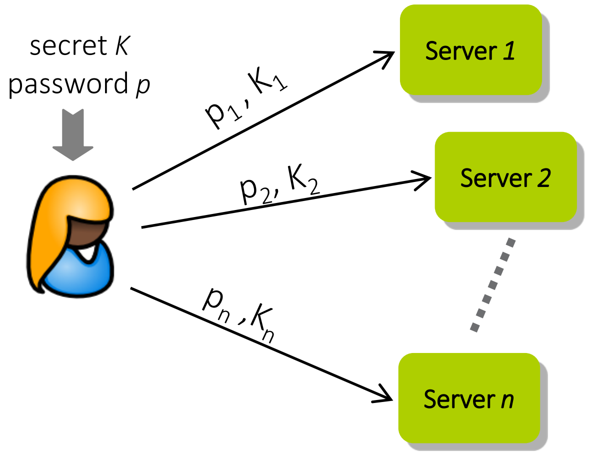
t+1 shares needed to reconstruct K

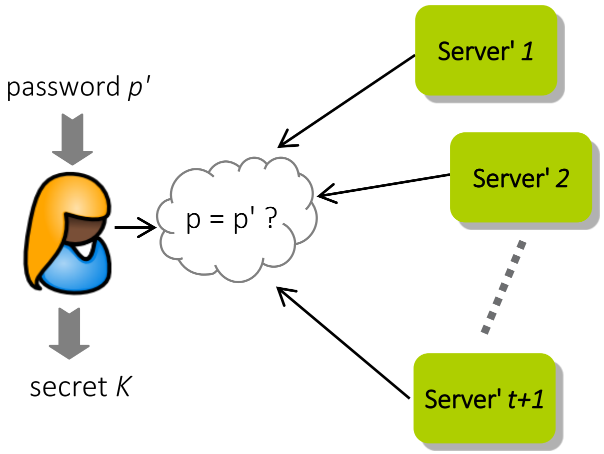
if at most t servers are corrupt → they don't learn anything about K

Password-Authenticated Secret Sharing | BJSL'11

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

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





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

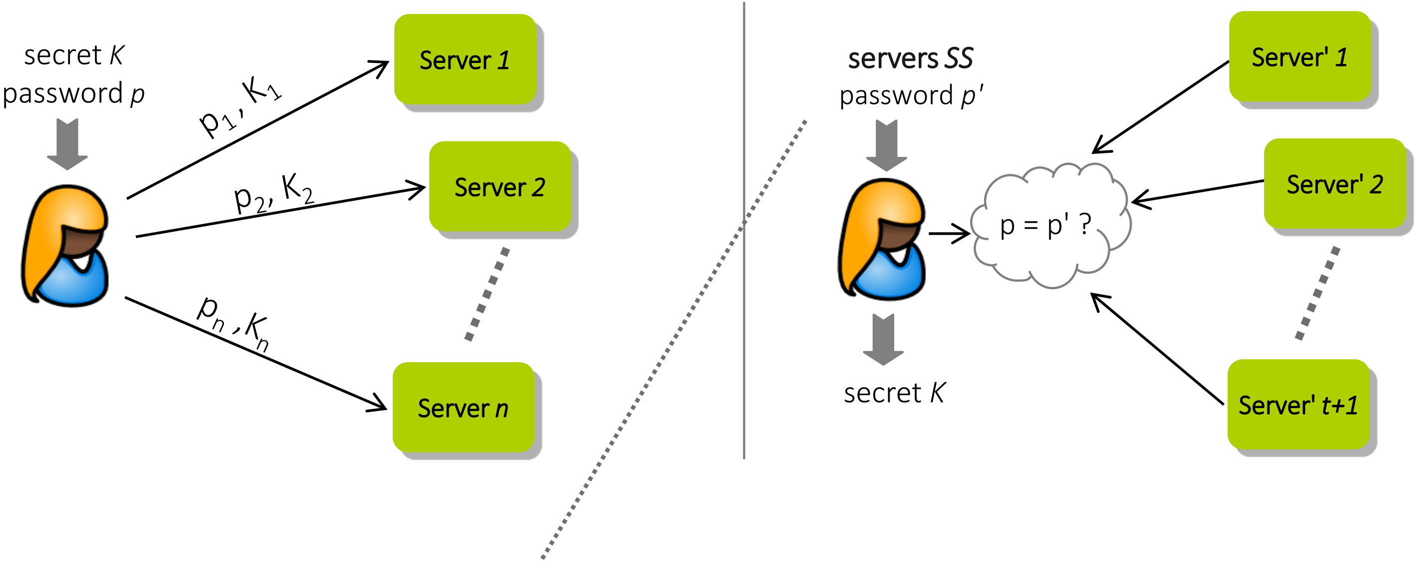
honest server throttle verification after too many (failed) attempts

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

Password-Authenticated Secret Sharing (TPASS/PPSS)

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

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

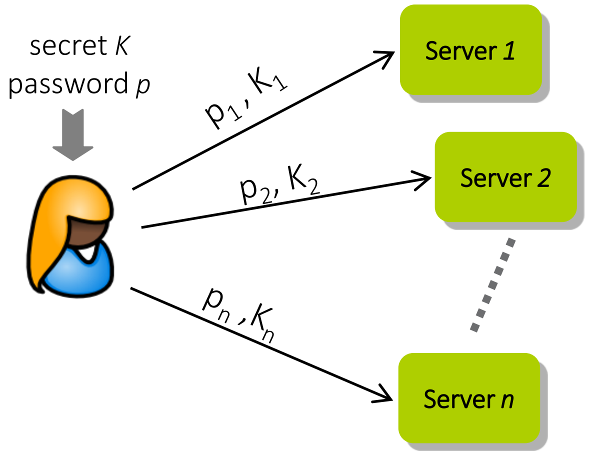


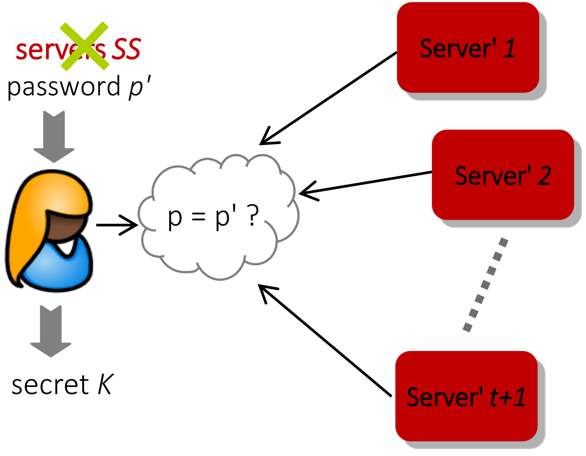
user has to remember the servers she trusted at setup

Password-Authenticated Secret Sharing (TPASS/PPSS)

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~~

[CLLN’14] Camenisch, Lehmann, Lysyanskaya, Neven. *Memento: How to Reconstruct your Secrets from a Single Password in a Hostile Environment.* Crypto 2014

Overview of TPASS Solutions

Scheme	Security Model	Assumption	Retrieval Rounds	Retrieval Exponentiation User	Retrieval Exponentiation Server
BJSL'11	Game	DDH-ROM	3	8t+17	16
CLLN'14	UC	DDH-ROM	5	14t+24	7t+28
JKK'14	Game	OMGDH-ROM	1	2t+3	3
ACNP'16	Game	OMGDH-ROM	1	?	?
JKKX'16	UC	OMGDH-ROM	1	t+2	1
JKKX'17	UC	TOMGDH-ROM	1	2	1

password-only

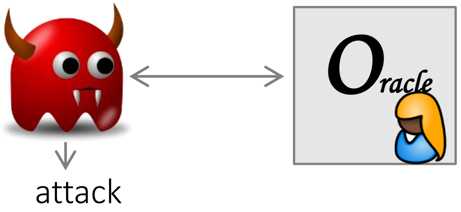
SECURITY MODELS

for password-based crypto

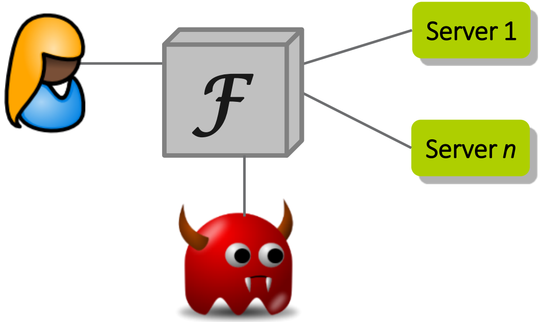
Provable Security

- Old days: security by obscurity
- Now: provable security = gold standard in cryptography
  - Formal security model  & formal security proof 
  - Also crucial for higher-level protocols: secure building blocks  secure protocol

Game-Based

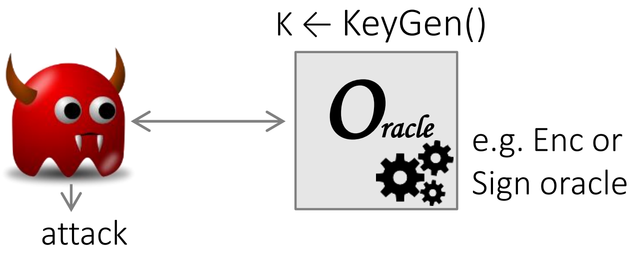


(UC) Ideal vs Real

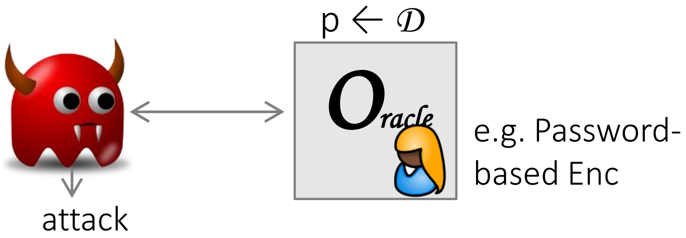


Challenge: Security Model including the User

- Game-based security notions most common
  - Oracle access to some secret key function
  - Secure if ∀Adv: Prob[attack] = negligible



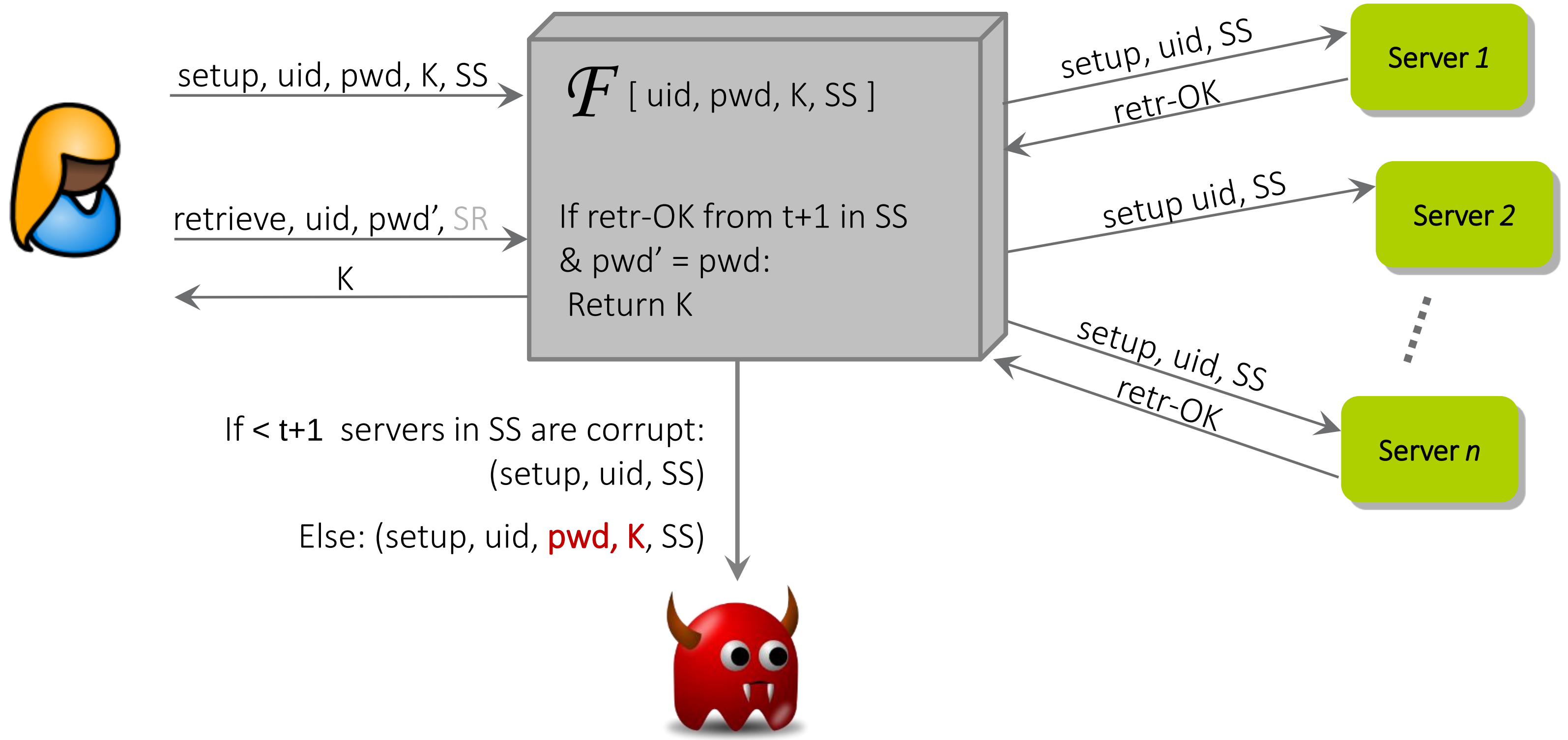
- User/Password-based cryptography
  - Adversary has black-box access “to the user”



Model	Reality
Passwords chosen at random from known, independent distribution	People reuse passwords, leak info about passwords
Honest user always uses correct password	Users make typos, “mix” passwords

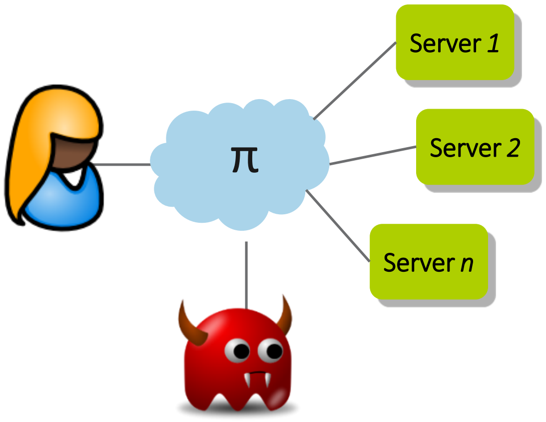
Universal Composability Framework | Canetti'00

- Security defined via ideal functionality $\mathcal{F}$ – $\mathcal{F}$ is "secure-by-design"



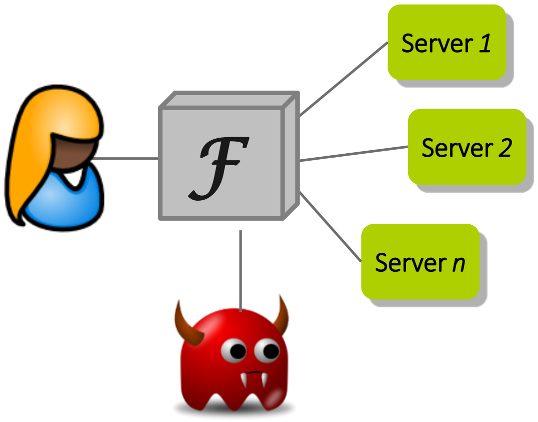
Universal Composability Framework | Canetti'00

- Security defined via ideal functionality $\mathcal{F}$ – $\mathcal{F}$ is "secure-by-design"



Real world

$\approx$

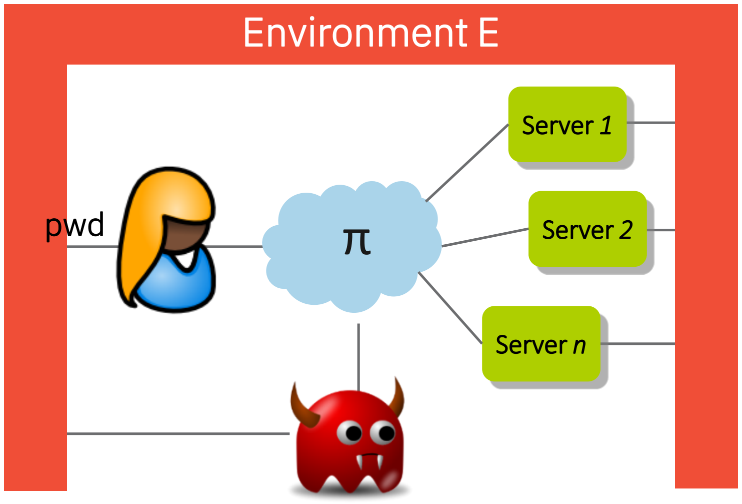


Ideal world

Universal Composability Framework | Canetti'00

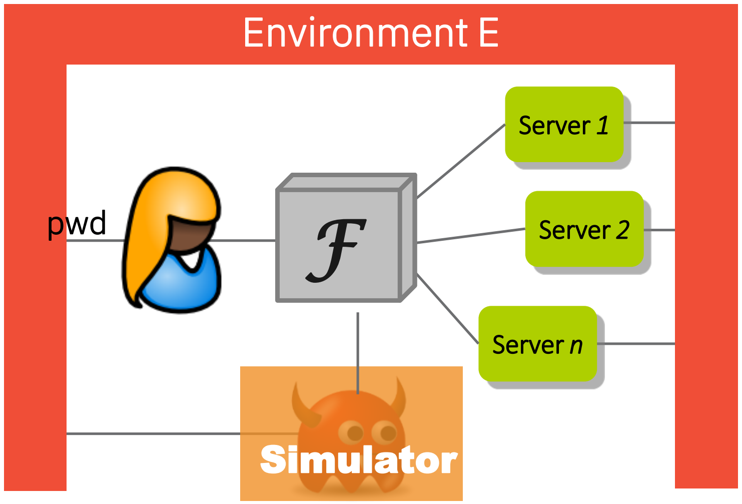
- Security defined via ideal functionality $\mathcal{F}$ – $\mathcal{F}$ is "secure-by-design"
- Protocol π securely implements $\mathcal{F}$ if $\forall$Adv $\exists$Sim such that $\forall$E: $\text{REAL}_{\pi,A,E} \approx \text{IDEAL}_{F,S,E}$

environment chooses passwords of honest users
→ no assumptions on pwd distributions & typos by honest users covered



Real world

≈



Ideal world

Overview of TPASS Solutions

Scheme	Security Model	Assumption	Retrieval Rounds	Retrieval Exponentiation User	Retrieval Exponentiation Server
BJSL'11	Game	DDH-ROM	3	8t+17	16
CLLN'14	UC	DDH-ROM	5	14t+24	7t+28
JKK'14	Game	OMGDH-ROM	1	2t+3	3
ACNP'16	Game	OMGDH-ROM	1	?	?
JKKX'16	UC	OMGDH-ROM	1	t+2	1
JKKX'17	UC	TOMGDH-ROM	1	2	1

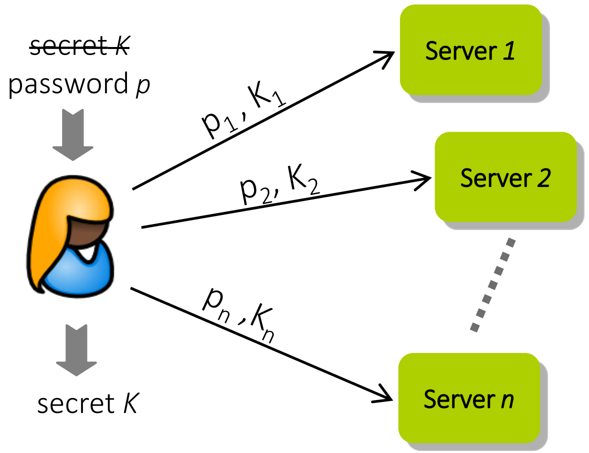
password-only

Disclaimer: security models vary

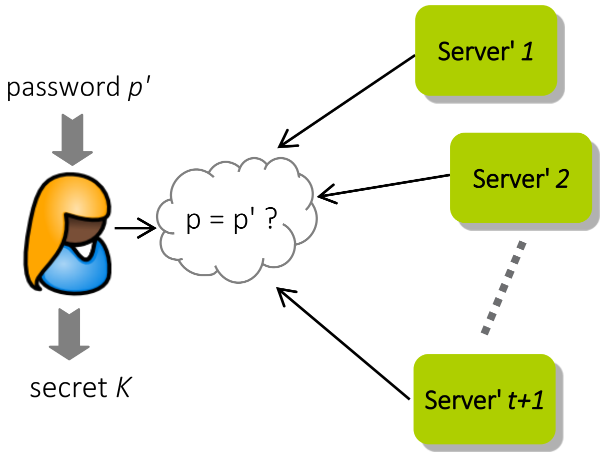
All based on OPRFs

TPASS by JKKX'17 | Slightly Different Setting

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



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



user **obtains** a random key *K* at setup

if < t+1 servers are corrupt → they don't learn anything about K
if ≥ t+1 servers are corrupt → they learn *K* (but its still a random key)

[JKKX'17] Jarecki, Kiayias, Krawczyk, Xu. *TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF*. ACNS 2017

Building Block: Threshold OPRF (T-OPRF)

compute $y = \mathrm{PRF}(k, x)$ in a blind & ~~distributed~~ threshold manner

$k = \mathrm{KGen}(\tau)$

$k_1 + k_2 + \ \ldots + k_n = \mathrm{Share}(k,$

any $t+1$ shares are sufficient to compute P

If <t+1 servers are corrupt:

T-OPRF outputs are indistinguisable from random

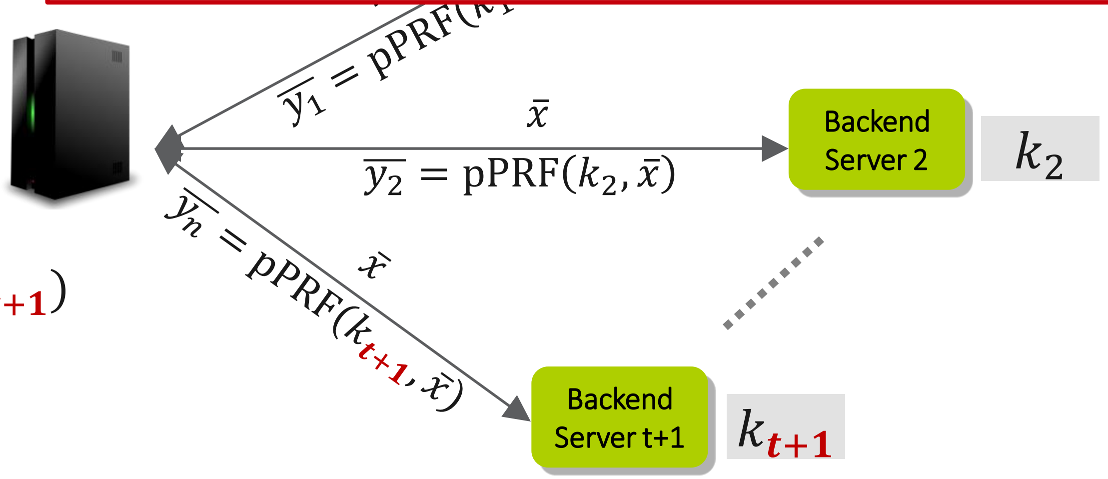
can only evaluate PRF with help of honest servers

$\bar{x} = \mathrm{Blind}(x)$

$\bar{y} = \mathrm{Comb}(\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_{t+1})$

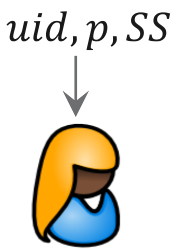
$y = \mathrm{Unblind}(\bar{y})$

s.t. $y = \mathrm{PRF}(k, x)$



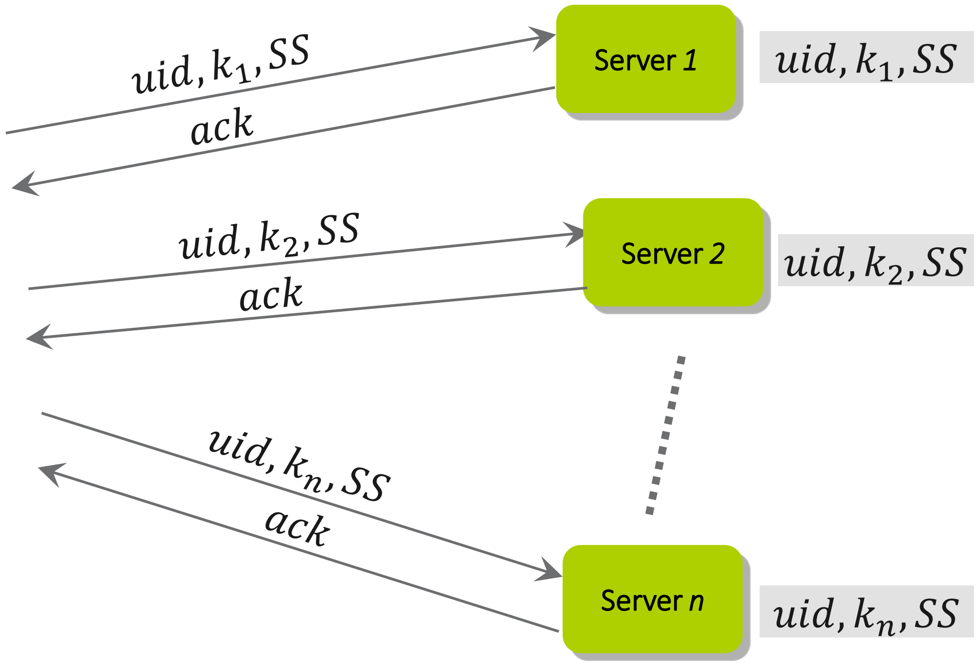
TPASS Protocol | Setup

- user obtains secret *K* protected by password *p* with *n* servers SS = $S_1, S_2, \ldots, S_n$



$k = \mathrm{PRF.KGen}(\tau)$
$(k_1, k_2, \ldots, k_n) = Share(k, t, n)$

if ack from all S in SS
  compute $y = \mathrm{PRF}(k, p)$
  compute $h = \mathrm{H}(y)$
  parse $h = (C, K)$



TPASS Protocol | Setup

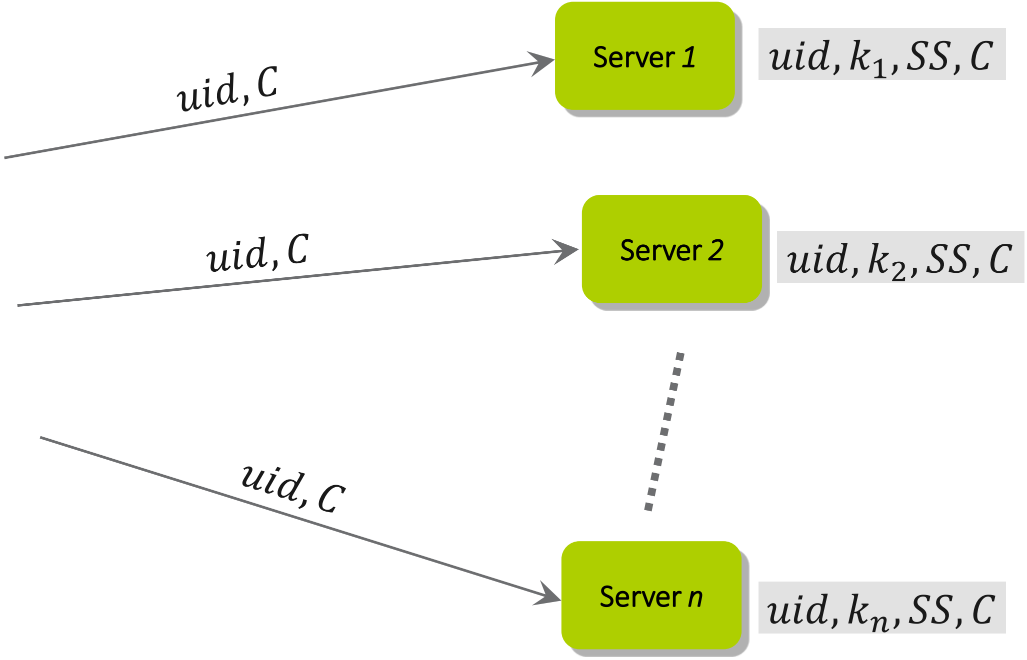
- user obtains secret K protected by password p with n servers SS = $S_1, S_2, \ldots, S_n$

uid, p, SS

$k = \mathrm{PRF.KGen}(\tau)$
$(k_1, k_2, \ldots, k_n) = Share(k, t, n)$

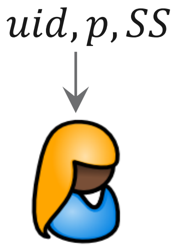
if ack from all S in SS
compute $y = \mathrm{PRF}(k, p)$
compute $h = \mathrm{H}(y)$
parse $h = (C, K)$

send C to all S & output K



TPASS Protocol | Setup

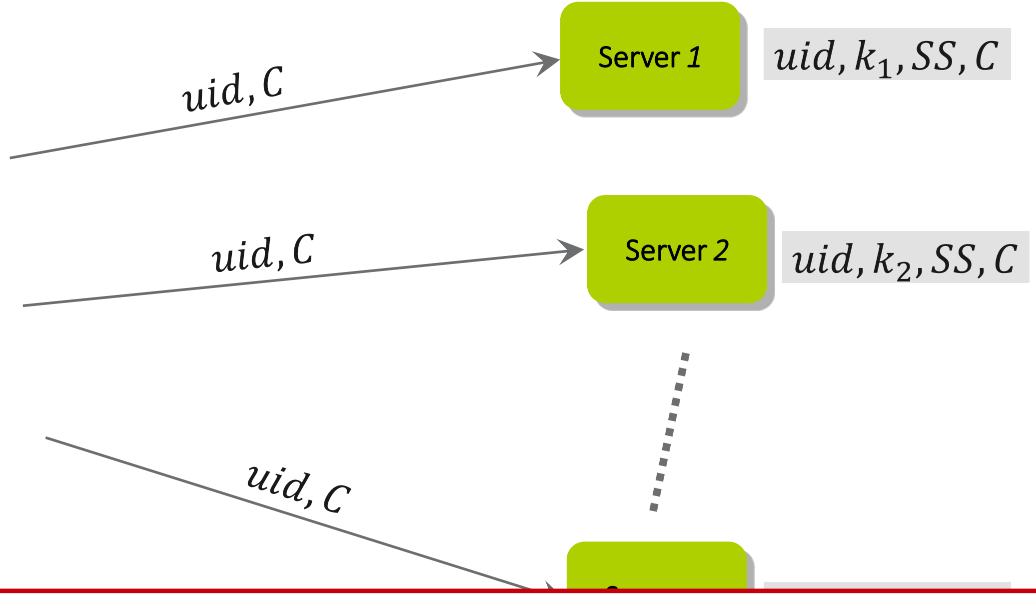
- user obtains secret *K* protected by password *p* with *n* servers SS = $S_1, S_2, \ldots, S_n$



$k = \mathrm{PRF.KGen}(\tau)$
$(k_1, k_2, \ldots, k_n) = Share(k, t, n)$

if ack from all S in SS
  compute $y = \mathrm{PRF}(k, p)$
  compute $h = \mathrm{H}(y)$
  parse $h = (C, K)$

send C to all S & out

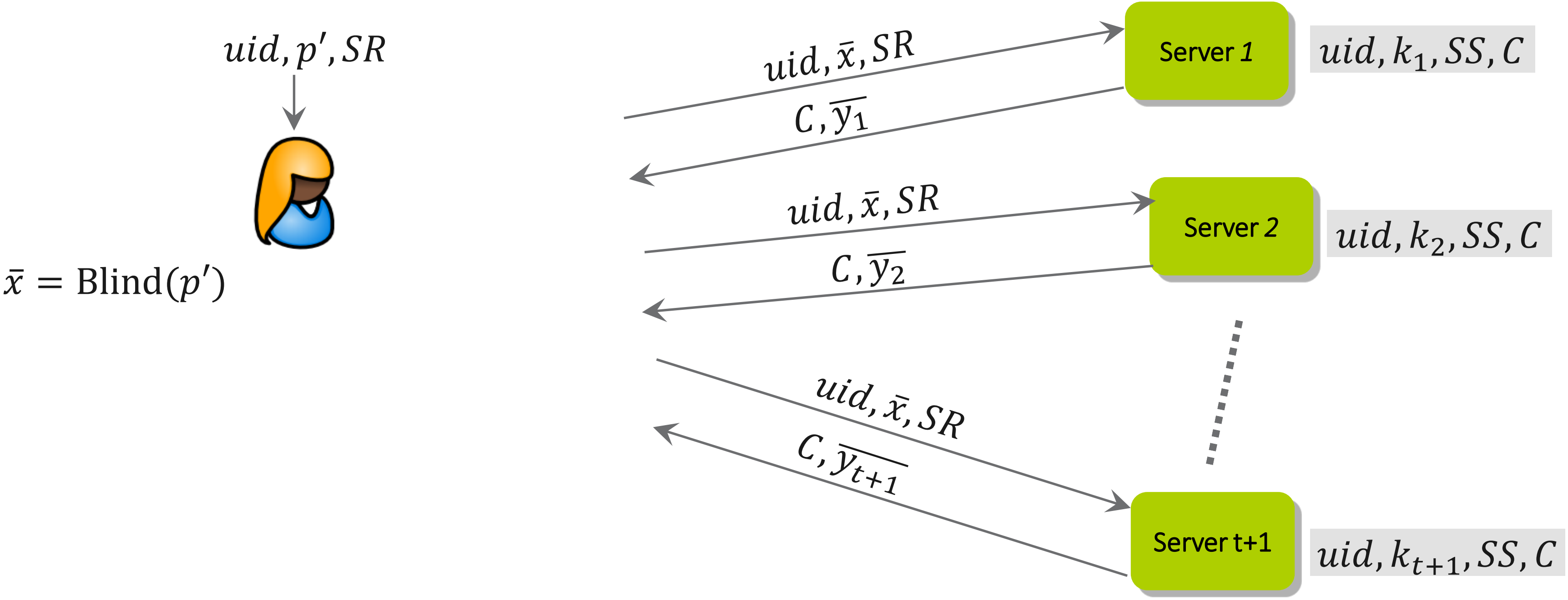


K is always a random key
If <t+1 servers are corrupt: Adv learns nothing about (p, K)
If ≥t+1 servers are corrupt: Adv can offline attack (p, K)

TPASS Protocol | Retrieval

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

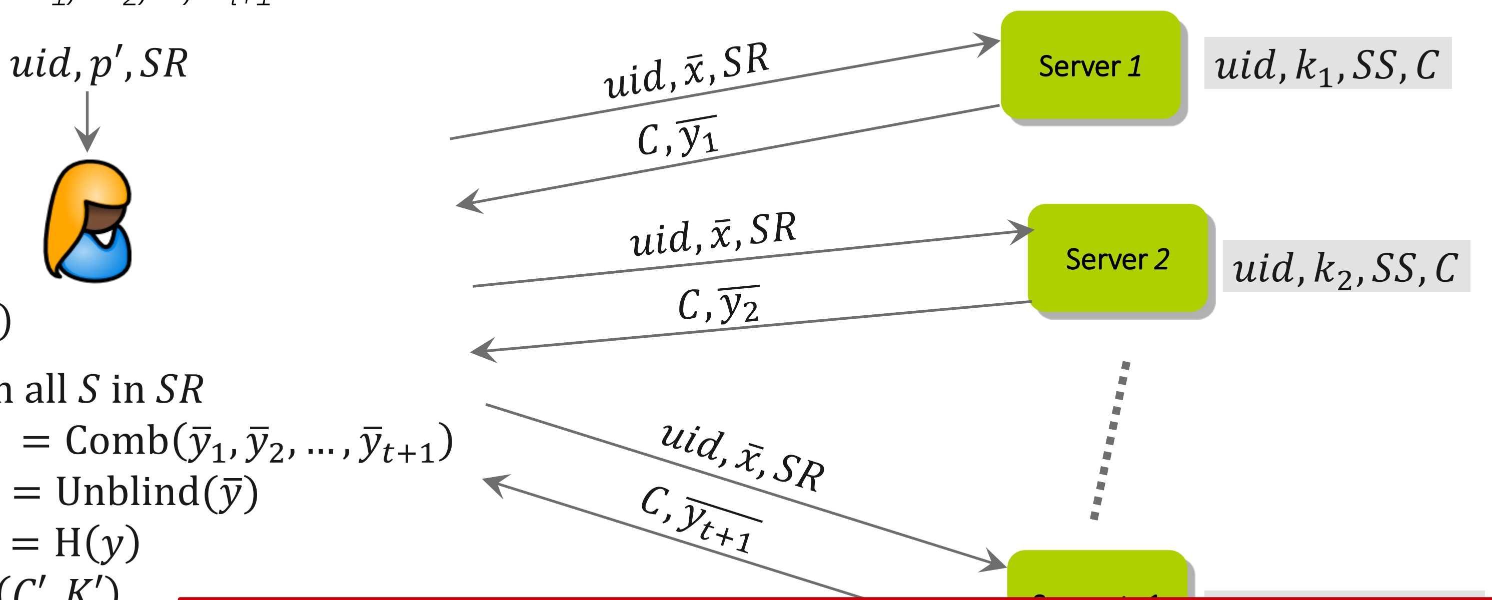
each S_i:
check that $SR \subset SS$
compute $\overline{y_i} = \mathrm{pPRF}(k_i, \bar{x})$



TPASS Protocol | Retrieval

- user retrieve her secret using password *pwd'* from *t+1* servers *SR = S'*$_1$, *S'*$_2$,…,*S'*$_{t+1}$

each S_i:
check that $SR \subset SS$
compute $\overline{y_i} = \mathrm{pPRF}(k_i, \bar{x})$



$\bar{x} = \mathrm{Blind}(p')$

if $(C, \overline{y_i})$ from all S in SR
  compute $\bar{y} = \mathrm{Comb}(\overline{y_1}, \overline{y_2}, \ldots, \overline{y_{t+1}})$
  compute $y = \mathrm{Unblind}(\bar{y})$
  compute $h = \mathrm{H}(y)$
  parse $h = (C', K')$

if $C' = C$ output K'
  else output $K' = \perp$

Security based on T-OPRF & ROM
Efficient T-OPRF from OMGDH & ROM (similar to our DORPF)

TPASS Protocol | Retrieval

- user retrieve her secret using password *pwd'* from *t+1* servers *SR* = *S'*$_1$, *S'*$_2$,…,*S'*$_{t+1}$

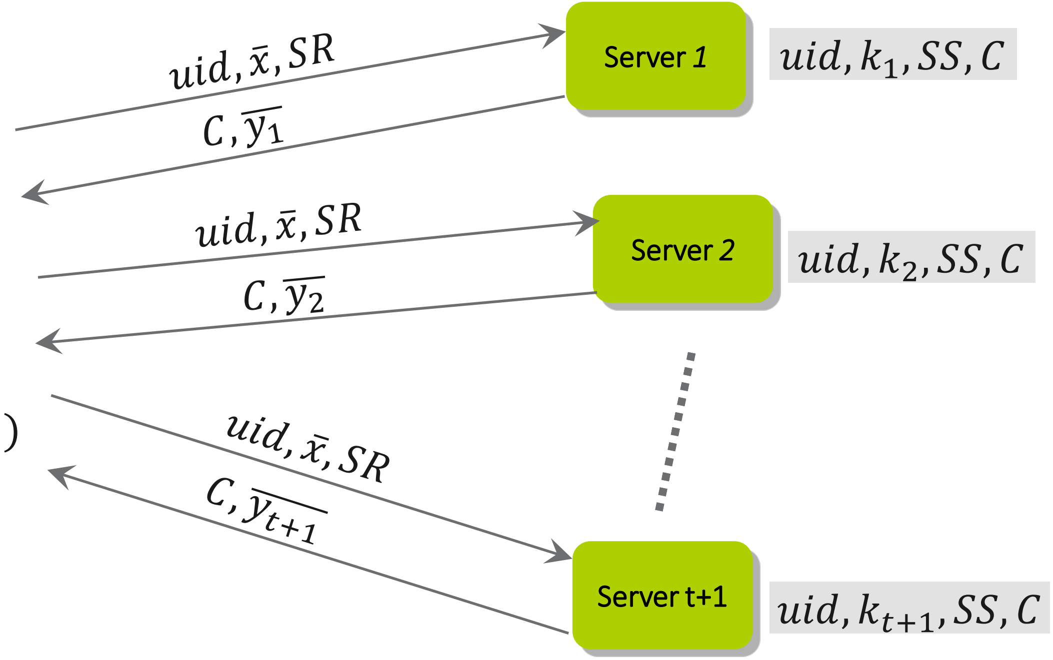
each S_i:
check that $SR \subset SS$
compute $\overline{y_i} = \text{pPRF}(k_i, \bar{x})$

uid, p', SR

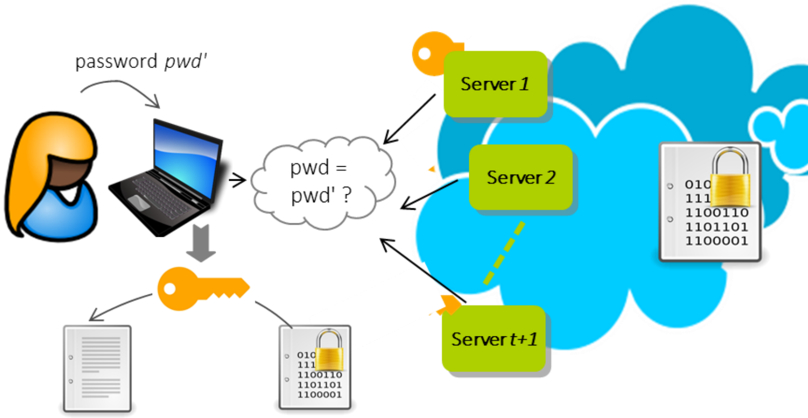
$\bar{x} = \text{Blind}(p')$

if $(C, \overline{y_i})$ from all S in SR
compute $\bar{y} = \text{Comb}(\bar{y}_1, \bar{y}_2, \dots, \bar{y}_{t+1})$
compute $y = \text{Unblind}(\bar{y})$
compute $h = \text{H}(y)$
parse $h = (C', K')$

if $C' = C$ output K'
else output $K' = \perp$



TPASS | Applications



- TPASS allows users to reconstruct strong secret key from weak password
  - Does not require trusted storage
  - Allows to bootstrap any cryptographic operation based on a strong key
    - Encrypted cloud storage, strong authentication, …
    - Bootstrap strong "passwords" from K, pwd= H(K,"iacr.org")
  - Reconstruction of secret key can be security risk – malware on device

- Less flexible, but more secure: protocols for joint password-based computations
  - Number of "solutions", most are vulnerable against offline attacks ☹
  - Distributed signing [CLNS16] – "Virtual Smartcard"

Password-Based Crypto | Summary

- Passwords are convenient & easy to use
- Low entropy makes them vulnerable to offline attacks
- Strong security from passwords requires multi-server solutions
  - Prevents offline attacks & detect online attacks

- UC-based definitions capture password use better than game-based models
- Highly-efficient solutions exist for a number of password-based primitives
- Lots of open research problems – Lets make crypto for people! ☺

Thanks! Questions?

anj@zurich.ibm.com