Phoenix: Rebirth of a Cryptographic Password-Hardening Service

Russell W.F. Lai [1,2]
Christoph Egger [1]
Dominique Schröder [1]
Sherman S.M. Chow [2]

[1]Friedrich-Alexander-Universität Erlangen-Nürnberg
[2]Chinese University of Hong Kong

August 17, 2017

Scheme Design

Password Authentication - before 1976

User

Username	Alice
Password	123456

Database

Username	Alice
Password	pw

I am "Alice".

My password is "123456".

$\longrightarrow$

Password Authentication - before 1976

User

Username	Alice
Password	123456

Database

Username	Alice
Password	pw

I am "Alice".

My password is "123456".

→

Validation Algorithm

$\mathsf{pw} \stackrel{?}{=} 123456$

Password Authentication - before 1976

User

Username	Alice
Password	123456

Database

Username	Alice
Password	pw

I am "Alice".

My password is "123456".

$\longrightarrow$

Validation Algorithm

$pw \stackrel{?}{=} 123456$

OK!

$\longleftarrow$

Password Authentication - after 1976

User

Username	Alice
Password	123456

Database

Username	Alice
Hash	h
Salt	aqZcSP

I am “Alice”.

My password is “123456”. →

Validation Algorithm

$h \stackrel{?}{=} H(123456, \mathsf{aqZcSP})$

OK! ←

Problem I - Weak Passwords

Worst Passwords of 2016 by TeamsID

- 4% of users use “123456” as password
- 25% of users use the top 25 worst passwords
- Users are stubborn
  - ~~Choose stronger passwords~~
  - ~~Use crypto~~

Problem II - Stolen Passwords

List of data breaches

From Wikipedia, the free encyclopedia

For a broader coverage related to this topic, see Data breach.

This is a list of **data breaches**, using data compiled from various sources, including press reports, government ne
breaches occur continually. Breaches of large organizations where the number of records is still unknown are alsc

Most breaches occur in North America. It is estimated that the average cost of a data breach will be over $150 mil
breaches.[3] Vigilante.pw lists over 2,100 websites which have had their databases breached, containing over 2

Entity	Year	Records
Yahoo	2013	1,000,000,000
Yahoo	2014	500,000,000
Friend Finder Networks	2016	412,214,295
Massive American business hack including 7-Eleven and Nasdaq	2012	160,000,000
Adobe Systems	2013	152,000,000
eBay	2014	145,000,000
Heartland	2009	130,000,000
Rambler.ru	2012	98,167,935
TK / TJ Maxx	2007	94,000,000
AOL	2004	92,000,000

Data breaches in 2004–2017 (Wikipedia)

Problem II - Stolen Passwords

List of data breaches

From Wikipedia, the free encyclopedia

For a broader coverage related to this topic, see Data breach.

This is a list of **data breaches**, using data compiled from various sources, including press reports, government ne
breaches occur continually. Breaches of large organizations where the number of records is still unknown are alsc

Most breaches occur in North America. It is estimated that the average cost of a data breach will be over $150 mil
breaches.[3] Vigilante.pw lists over 2,100 websites which have had their databases breached, containing over 2

Entity	Year	Records
Yahoo	2013	1,000,000,000
Yahoo	2014	500,000,000
Friend Finder Networks	2016	412,214,295
Massive American business hack including 7-Eleven and Nasdaq	2012	160,000,000
Adobe Systems	2013	152,000,000
eBay	2014	145,000,000
Heartland	2009	130,000,000
Rambler.ru	2012	98,167,935
TK / TJ Maxx	2007	94,000,000
AOL	2004	92,000,000

Data breaches in 2004–2017 (Wikipedia)

Cost of Data Breach (IBM - 2017 Study)

- Average Cost per Data Breach: $3.62 million
- Average Cost per Stolen Record $141

Problem II - Stolen Passwords

List of data breaches

From Wikipedia, the free encyclopedia

For a broader coverage related to this topic, see Data breach.

This is a list of **data breaches**, using data compiled from various sources, including press reports, government ne
breaches occur continually. Breaches of large organizations where the number of records is still unknown are alsc

Most breaches occur in North America. It is estimated that the average cost of a data breach will be over $150 mil
breaches.[3] Vigilante.pw lists over 2,100 websites which have had their databases breached, containing over 2

Entity	Year	Records
Yahoo	2013	1,000,000,000
Yahoo	2014	500,000,000
Friend Finder Networks	2016	412,214,295
Massive American business hack including 7-Eleven and Nasdaq	2012	160,000,000
Adobe Systems	2013	152,000,000
eBay	2014	145,000,000
Heartland	2009	130,000,000
Rambler.ru	2012	98,167,935
TK / TJ Maxx	2007	94,000,000
AOL	2004	92,000,000

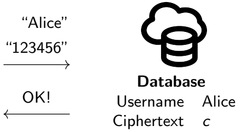
Data breaches in 2004–2017 (Wikipedia)

Cost of Data Breach (IBM - 2017 Study)

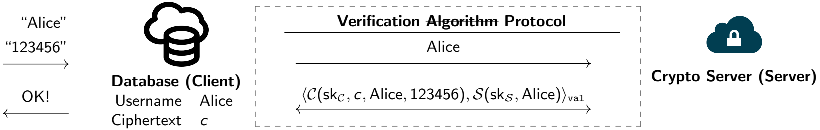
- Average Cost per Data Breach: $3.62 million
- Average Cost per Stolen Record $141

Service providers have incentives to change!

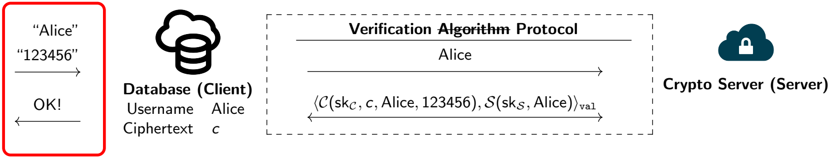
Password Hardening Services (Facebook, PYTHIA)



Password Hardening Services (Facebook, PYTHIA)



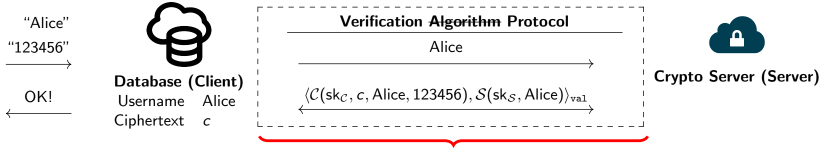
Password Hardening Services (Facebook, PYTHIA)



Key Features

- Seamless to end user (Alice)

Password Hardening Services (Facebook, Pythia)

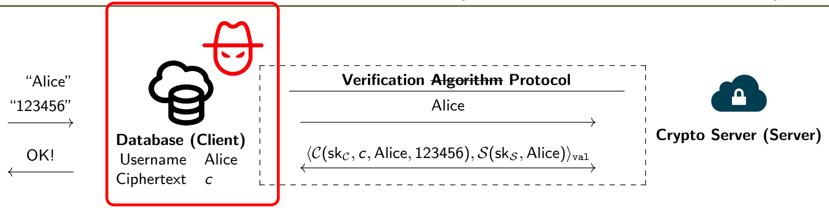


Messages from client are independent of passwords!

Key Features

- Seamless to end user (Alice)
- Obliviousness: (Malicious) server does not learn password

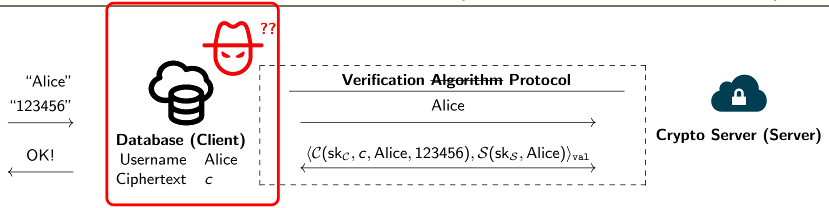
Password Hardening Services (Facebook, PYTHIA)



Key Features

- Seamless to end user (Alice)
- Obliviousness: (Malicious) server does not learn password
- Hiding: (Compromised) client cannot verify password by itself
  – eliminate offline attacks

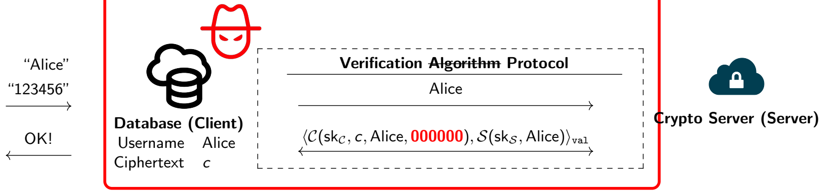
Password Hardening Services (Facebook, PYTHIA)



Key Features

- Seamless to end user (Alice)
- Obliviousness: (Malicious) server does not learn password
- Hiding: (Compromised) client cannot verify password by itself
  - eliminate offline attacks

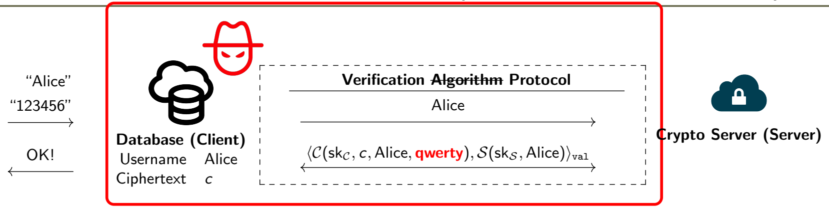
Password Hardening Services (Facebook, PYTHIA)



Key Features

- Seamless to end user (Alice)
- Obliviousness: (Malicious) server does not learn password
- Hiding: (Compromised) client cannot verify password by itself
  - – eliminate offline attacks
- Rate-Limiting (per Username): (Compromised) client cannot submit too many requests
  - – mitigate online attacks

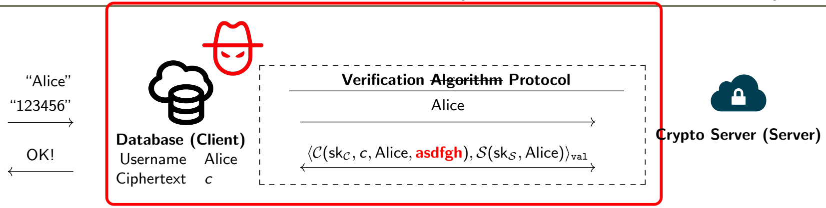
Password Hardening Services (Facebook, PYTHIA)



Key Features

- Seamless to end user (Alice)
- Obliviousness: (Malicious) server does not learn password
- Hiding: (Compromised) client cannot verify password by itself
  - eliminate offline attacks
- Rate-Limiting (per Username): (Compromised) client cannot submit too many requests
  - mitigate online attacks

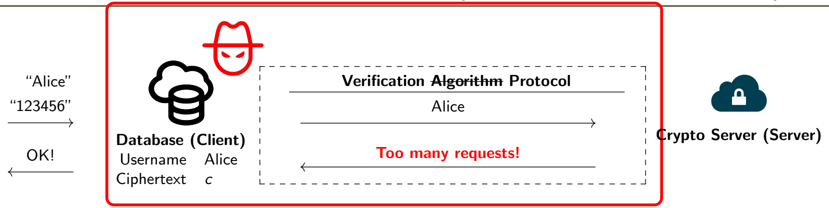
Password Hardening Services (Facebook, PYTHIA)



Key Features

- Seamless to end user (Alice)
- Obliviousness: (Malicious) server does not learn password
- Hiding: (Compromised) client cannot verify password by itself
  - eliminate offline attacks
- Rate-Limiting (per Username): (Compromised) client cannot submit too many requests
  - mitigate online attacks

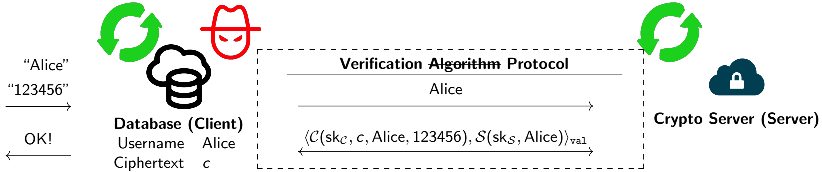
Password Hardening Services (Facebook, PYTHIA)



Key Features

- Seamless to end user (Alice)
- Obliviousness: (Malicious) server does not learn password
- Hiding: (Compromised) client cannot verify password by itself
  - – eliminate offline attacks
- Rate-Limiting (per Username): (Compromised) client cannot submit too many requests
  - – mitigate online attacks

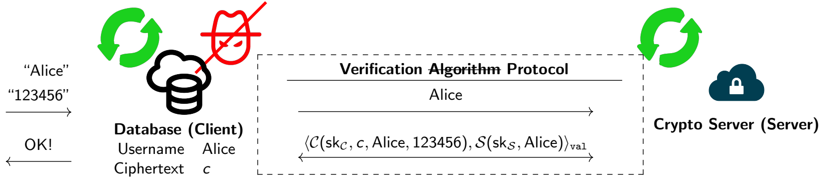
Password Hardening Services (Facebook, PYTHIA)



Key Features

- **Key-Rotation**
  - Update both keys if either party is compromised
  - Bring the entire system to a fresh state
  - Update ciphertexts without knowing passwords (seamless to end user)

Password Hardening Services (Facebook, PYTHIA)



Key Features

- **Key-Rotation**
  - Update both keys if either party is compromised
  - Bring the entire system to a fresh state
  - Update ciphertexts without knowing passwords (seamless to end user)

The Crypto Server

Crypto Server

- Key generation independent of client
  - Can be set up by any third party company / organization
  - One server can serve multiple clients

The Crypto Server

Crypto Server

- Key generation independent of client
  - Can be set up by any third party company / organization
  - One server can serve multiple clients
- Only stores:
  - One secret key per client
  - One counter per end user for rate-limiting (deleted after the current time interval)

The Crypto Server

Crypto Server

- Key generation independent of client
  - Can be set up by any third party company / organization
  - One server can serve multiple clients
- Only stores:
  - One secret key per client
  - One counter per end user for rate-limiting (deleted after the current time interval)
- Can be split into multiple servers (future work)

False Friends (Similar but different notions)

Common Feature

- To distribute the task of verifying passwords to multiple servers

False Friends (Similar but different notions)

Common Feature

- To distribute the task of verifying passwords to multiple servers

Distributed Password Verification

(CCS'15) Camenisch, Lehmann, and Neven

- Joint key generation between client and server
- Rate-limiting at client only

False Friends (Similar but different notions)

Common Feature

- To distribute the task of verifying passwords to multiple servers

Distributed Password Verification

(CCS'15) Camenisch, Lehmann, and Neven

- Joint key generation between client and server
- Rate-limiting at client only

Password-Authenticated Key Exchange (PAKE) / Password-Protected Secret Sharing (PPSS)

- Crypto servers need to store a secret share per end user
- No / inefficient key rotations

Literature on Password Hardening (PH)

Partially-Oblivious Pseudorandom Functions (PO-PRF)
(USENIX'15) Everspaugh, Chatterjee, Scott, Juels, and Ristenpart

- Formalized PO-PRF
- Construction (PYTHIA) based on pairing

Literature on Password Hardening (PH)

Partially-Oblivious Pseudorandom Functions (PO-PRF)

(USENIX'15) Everspaugh, Chatterjee, Scott, Juels, and Ristenpart

- Formalized PO-PRF
- Construction (PYTHIA) based on pairing

Partially-Oblivious Commitments (PO-Com)

(CCS'16) Schneider, Fleischhacker, Schröder, and Backes

- Formalized PO-Com
  - Security definitions too weak for PH (not covering online attacks)
- Construction without pairing
  - $2\times$ faster than PYTHIA when used for PH

This Work

- Finally, formalized PH
  - Revisit and strengthen model of Schneider *et al.*

This Work

- Finally, formalized PH
  - Revisit and strengthen model of Schneider *et al.*
- Formalized key-rotation

This Work

- Finally, formalized PH
  - Revisit and strengthen model of Schneider *et al.*
- Formalized key-rotation
- Devastating online attacks against scheme of Schneider *et al.*
  - Attack 1: Enable offline-dictionary attack after one validation request
  - Attack 2: Extract password after one validation request
  - The attacks defeat the purpose of external crypto server
  - The attacks are outside of their security model

This Work

- Finally, formalized PH
  - Revisit and strengthen model of Schneider *et al.*
- Formalized key-rotation
- Devastating online attacks against scheme of Schneider *et al.*
  - Attack 1: Enable offline-dictionary attack after one validation request
  - Attack 2: Extract password after one validation request
  - The attacks defeat the purpose of external crypto server
  - The attacks are outside of their security model
- Extremely simple construction (still without pairing)
  - Simple enough for real-world use – easy to understand and implement
  - Proven secure under strengthened security model
  - 1.5$\times$ faster than scheme of Schneider *et al.*
  - 3$\times$ faster than PYTHIA

From Salted Hash to PHOENIX (Intuitive Description)

Database (Client)

Username	un
Hash	$H(\text{un}, \text{pw}, n_{\mathcal{C}})$
Client Nonce	$n_{\mathcal{C}}$

From Salted Hash to PHOENIX (Intuitive Description)

Database (Client)

Username	un
PRF Value	$\mathsf{PRF}_{\mathsf{sk}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})$
Client Nonce	$n_{\mathcal{C}}$

From Salted Hash to PHOENIX (Intuitive Description)

Database (Client)

Username	un
Product PRF Value	$\mathsf{PRF}_{\mathsf{sk}_\mathcal{S}}(\mathsf{un}, n_\mathcal{S}) \cdot \mathsf{PRF}_{\mathsf{sk}_\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})$
Client Nonce	$n_\mathcal{C}$
Server Nonce	$n_\mathcal{S}$

Enrollment Protocol

Username un →

← PRF Value $\mathsf{PRF}_{\mathsf{sk}_\mathcal{S}}(\mathsf{un}, n_\mathcal{S})$, Server Nonce $n_\mathcal{S}$

Crypto Server (Server)

From Salted Hash to PHOENIX (Intuitive Description)

Database (Client)

Username	un
Ciphertext	$\mathsf{Enc}\left(\mathsf{pk}_{\mathcal{S}}, \begin{matrix}\mathsf{PRF}_{\mathsf{sk}_{\mathcal{S}}}(\mathsf{un}, n_{\mathcal{S}}) \cdot \\ \mathsf{PRF}_{\mathsf{sk}_{\mathcal{C}}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})\end{matrix}\right)$
Client Nonce	$n_{\mathcal{C}}$
Server Nonce	$n_{\mathcal{S}}$

Enrollment Protocol

Username un →

PRF Value $\mathsf{PRF}_{\mathsf{sk}_{\mathcal{S}}}(\mathsf{un}, n_{\mathcal{S}})$

Server Nonce $n_{\mathcal{S}}$ ←

Crypto Server (Server)

From Salted Hash to PHOENIX (Intuitive Description)

Database (Client)

Username	un
Ciphertext	$\mathsf{Enc}\left(\mathsf{pk}_{\mathcal{S}}, \begin{array}{c}\mathsf{PRF}_{\mathsf{sk}_{\mathcal{S}}}(\mathsf{un}, n_{\mathcal{S}})\cdot \\ \mathsf{PRF}_{\mathsf{sk}_{\mathcal{C}}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})\end{array}\right)$
Client Nonce	$n_{\mathcal{C}}$
Server Nonce	$n_{\mathcal{S}}$

Enrollment Protocol

Username un →

← PRF Value $H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}$, Server Nonce $n_{\mathcal{S}}$

Crypto Server (Server)

Homomorphic Encryption

$$\mathsf{Enc}(\mathsf{pk}, m \cdot m') \approx \mathsf{Enc}(\mathsf{pk}, m) \cdot \mathsf{Enc}(\mathsf{pk}, m')$$

Key-Homomorphic PRF

$$\mathsf{PRF}_{\mathsf{sk}\cdot\mathsf{sk}'}(m) = \mathsf{PRF}_{\mathsf{sk}}(m) \cdot \mathsf{PRF}_{\mathsf{sk}'}(m)$$

From Salted Hash to PHOENIX (Intuitive Description)

Database (Client)

Username	un
	$(g^r, h^r \cdot H_{\mathcal{S}}(\mathrm{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\mathrm{un}, \mathrm{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}})$
Client Nonce	$n_{\mathcal{C}}$
Server Nonce	$n_{\mathcal{S}}$

Enrollment Protocol

Username un →

← PRF Value, Server Nonce $n_{\mathcal{S}}$

Crypto Server (Server)

e.g., ElGamal (and variants)

$$\mathrm{sk} = s,\ \mathrm{pk} = h = g^s$$

$$c = (g^r, h^r \cdot m)$$

$$(g^{r+r'}, h^{r+r'} \cdot m \cdot m') = (g^r, h^r \cdot m) \cdot (g^{r'}, h^{r'} \cdot m')$$

e.g., Naor-Pinkas-Reingold

$$\mathrm{sk} = k$$

$$y = H(m)^k$$

$$H(m)^{k+k'} = H(m)^k \cdot H(m)^{k'}$$

PHOENIX Validation (Intuitive Description)

Database (Client)

Crypto Server (Server)

Username	un
Ciphertext	$(g^r, h^r \cdot H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}})$
Client Nonce	$n_{\mathcal{C}}$
Server Nonce	$n_{\mathcal{S}}$

PHOENIX Validation (Intuitive Description)

Database (Client)

Crypto Server (Server)

Username	un
Ciphertext	$(g^r, h^r \cdot H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}} \cdot H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}})$
Client Nonce	$n_\mathcal{C}$
Server Nonce	$n_\mathcal{S}$

Validation Protocol

$$h^r \cdot H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}} = \frac{h^r \cdot H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}} \cdot H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}}}{H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}}}$$

PHOENIX Validation (Intuitive Description)

Database (Client)

Crypto Server (Server)

Username	un
Ciphertext	$(g^r, h^r \cdot H_{\mathcal{S}}(\mathrm{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\mathrm{un}, \mathrm{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}})$
Client Nonce	$n_{\mathcal{C}}$
Server Nonce	$n_{\mathcal{S}}$

Validation Protocol

$$h^r \cdot H_{\mathcal{S}}(\mathrm{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} = \frac{h^r \cdot H_{\mathcal{S}}(\mathrm{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\mathrm{un}, \mathrm{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}}{H_{\mathcal{C}}(\mathrm{un}, \mathrm{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}}$$

$$\xrightarrow{(g^r, h^r \cdot H_{\mathcal{S}}(\mathrm{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}), \mathrm{un}, n_{\mathcal{S}}}$$

PHOENIX Validation (Intuitive Description)

Database (Client)

Crypto Server (Server)

Username	un
Ciphertext	$(g^r, h^r \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}})$
Client Nonce	$n_{\mathcal{C}}$
Server Nonce	$n_{\mathcal{S}}$

Validation Protocol

$$h^r \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} = \frac{h^r \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}}{H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}}$$

$$\xrightarrow{(g^r, h^r \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}), \text{un}, n_{\mathcal{S}}}$$

Is un requested too many times?

Is the ciphertext valid?

PHOENIX Validation (Intuitive Description)

Database (Client)

Crypto Server (Server)

Username	un
Ciphertext	$(g^r, h^r \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}})$
Client Nonce	$n_{\mathcal{C}}$
Server Nonce	$n_{\mathcal{S}}$

Validation Protocol

$$h^r \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} = \frac{h^r \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}}{H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}}$$

$\xrightarrow{(g^r, h^r \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}), \text{un}, n_{\mathcal{S}}}$

Is un requested too many times?

Is the ciphertext valid?

$\xleftarrow{\text{Prove(The ciphertext is valid!)}}$

Why it works?

- Obliviousness:
  - Nothing about the password is sent to the server

$$\xrightarrow{(g^r, h^r \cdot H_{\mathcal{S}}(\mathrm{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}), \mathrm{un}, n_{\mathcal{S}}}$$

Why it works?

- Obliviousness:
  - Nothing about the password is sent to the server

$$\xrightarrow{(g^r, h^r \cdot H_{\mathcal{S}}(\mathrm{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}), \mathrm{un}, n_{\mathcal{S}}}$$

- Hiding:
  - PRF values of the passwords are encrypted to the server
  - Client cannot decrypt by itself
  - Validity check binds $(\mathrm{un}, n_{\mathcal{S}})$ with $H_{\mathcal{S}}(\mathrm{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}$ in c
  - Online attacks require guessing pw to remove $H_{\mathcal{C}}(\mathrm{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}$ from c

PHOENIX Key-Rotation (Intuitive Description)

Database (Client)

Crypto Server (Server)

$$g^{s \cdot r} \cdot H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}$$

Phoenix Key-Rotation (Intuitive Description)

Database (Client)

Crypto Server (Server)

$$g^{s\cdot r} \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}$$

$$\downarrow \hat{}\,\alpha$$

$$g^{\alpha\cdot s\cdot r} \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{\alpha\cdot k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{\alpha\cdot k_{\mathcal{C}}}$$

$$\downarrow \cdot(g^{r})^{\beta} \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{\gamma}$$

$$g^{(\alpha\cdot s+\beta)\cdot r} \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{\alpha\cdot k_{\mathcal{S}}+\gamma} \cdot H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{\alpha\cdot k_{\mathcal{C}}}$$

$$||$$

$$g^{s'\cdot r} \cdot H_{\mathcal{S}}(\text{un}, n_{\mathcal{S}})^{k'_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\text{un}, \text{pw}, n_{\mathcal{C}})^{k'_{\mathcal{C}}}$$

Evaluation and Deployment

Evaluation

In Comparison

Current password hashing recommendations suggest up to one second single-core computing time

Context

We have all three (python based) implementations running on Amazon AWS single-core instances

Questions

- How long does the user have to wait for password verification
- Do we need many servers to support Password-Hardening
- What are the practical implications

Evaluation

How long must the end user wait for to log in?

Evaluation

How long must the end user wait for to log in?

≈ 8 *ms!* (+ round trip time)

	Frankfurt HTTP	Frankfurt HTTPS	Frankfurt HTTPS keep-alive	Ireland HTTP	Ireland HTTPS	Ireland HTTPS keep-alive
RTT (64 bytes)		1.2			23	
PYTHIA enroll/validate	17.93	25.28	16.01	62.03	113.79	38.56
Schneider *et al.* enroll	9.80	22.86	8.14	53.72	111.40	30.89
Schneider *et al.* validate	12.30	25.65	10.73	56.32	115.32	33.49
PHOENIX enroll	5.43	17.93	3.89	50.30	107.25	26.52
PHOENIX validate	9.74	22.78	8.06	53.92	113.02	30.73

Latency in millisecond (ms)

Evaluation

How many requests can the server entertain in one second?

Evaluation

How many requests can the server entertain in one second?

Over 370*!*

	HTTPS keep-alive	HTTPS
parameter	2,607.16	807.50
PYTHIA enroll/validate	128.50	125.75
Schneider *et al.* enroll	380.37	278.51
Schneider *et al.* validate	221.75	183.92
PHOENIX enroll	1,557.81	697.66
PHOENIX validate	371.34	275.42

Requests per second

Practical Deployment

Hybrid Scheme

Can we make use of memory-hard functions like Argon2 or scrypt?

Practical Deployment

Hybrid Scheme

Can we make use of memory-hard functions like Argon2 or scrypt?

- Use a memory-hard function instead of a traditional hash function for the PRF
- Even if the attacker has compromised both Client and Server, she has to use the memory-hard function for dictionary attacks

Naor-Pinkas-Reingold

sk $= k$
PRF value $H(x)^k$

Practical Deployment

Hybrid Scheme

Can we make use of memory-hard functions like Argon2 or scrypt?

- Use a memory-hard function instead of a traditional hash function for the PRF
- Even if the attacker has compromised both Client and Server, she has to use the memory-hard function for dictionary attacks

Naor-Pinkas-Reingold

$\mathsf{sk} = k$

PRF value $Sha256(x)^k$

Practical Deployment

Hybrid Scheme

Can we make use of memory-hard functions like Argon2 or scrypt?

- Use a memory-hard function instead of a traditional hash function for the PRF
- Even if the attacker has compromised both Client and Server, she has to use the memory-hard function for dictionary attacks

Naor-Pinkas-Reingold

sk $= k$
PRF value $Argon2(x)^k$

Practical Deployment

Availability

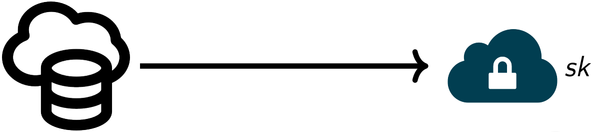
What happens when the Crypto Server is unavailable?

Practical Deployment

Availability

What happens when the Crypto Server is unavailable?

- Server only holds key-pair per Client and Rate-Limiting information
- Several Crypto Servers can host the key-pair for availability **but** keys are then located on several machines

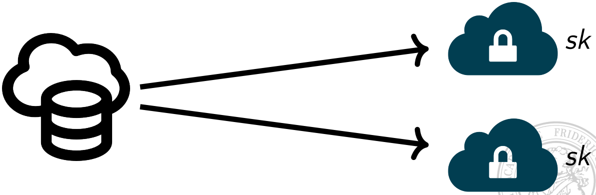


Practical Deployment

Availability

What happens when the Crypto Server is unavailable?

- Server only holds key-pair per Client and Rate-Limiting information
- Several Crypto Servers can host the key-pair for availability **but** keys are then located on several machines

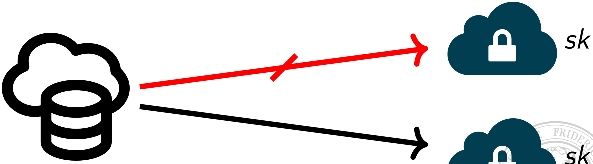


Practical Deployment

Availability

What happens when the Crypto Server is unavailable?

- Server only holds key-pair per Client and Rate-Limiting information
- Several Crypto Servers can host the key-pair for availability **but** keys are then located on several machines

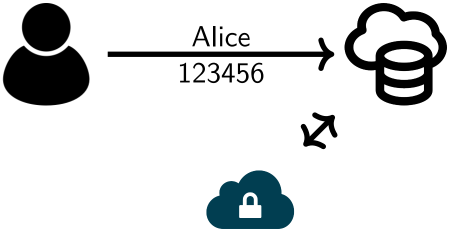


DDoS and Rate-Limiting

There are two scenarios where Rate-Limiting might be triggered:

Client has been compromised

- It is acceptable when users fail to log in

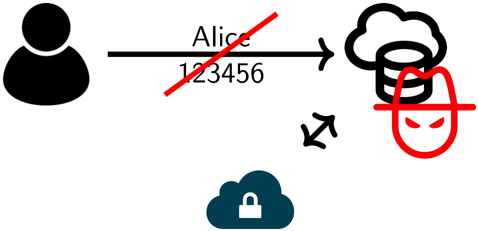


DDoS and Rate-Limiting

There are two scenarios where Rate-Limiting might be triggered:

Client has been compromised

- It is acceptable when users fail to log in

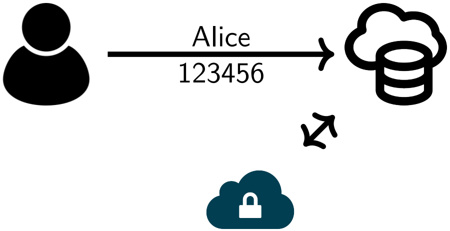


DDoS and Rate-Limiting

There are two scenarios where Rate-Limiting might be triggered:

External Attacker

- Honest users should only be slightly inconvenienced
- Crypto Server has little information to distinguish users
- Client is honest and can therefore help

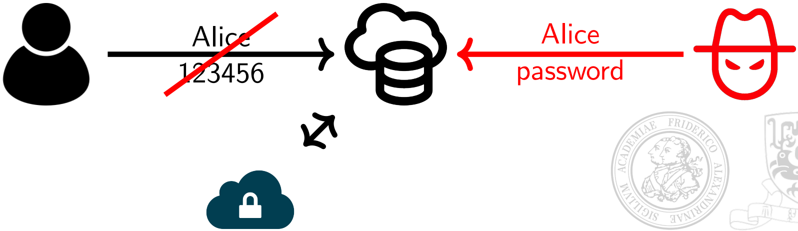


DDoS and Rate-Limiting

There are two scenarios where Rate-Limiting might be triggered:

External Attacker

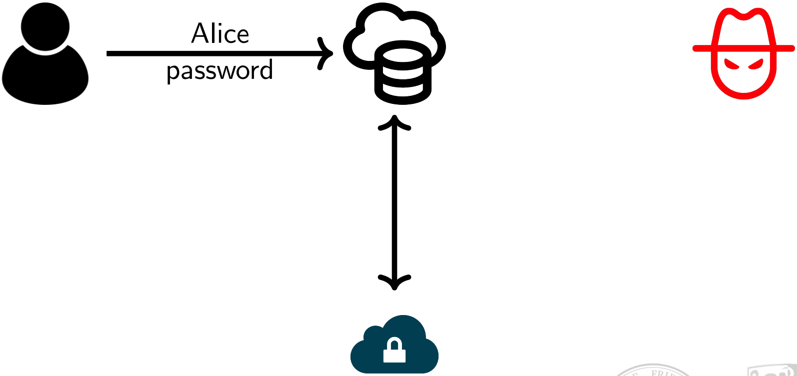
- Honest users should only be slightly inconvenienced
- Crypto Server has little information to distinguish users
- Client is honest and can therefore help



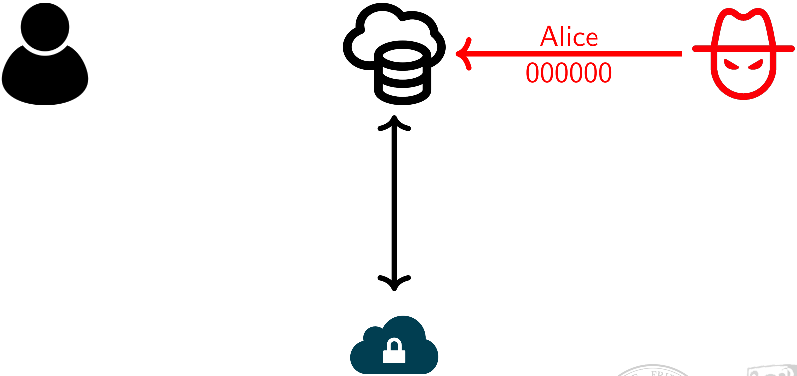
Rate-Limiting external clients

- *Warn* the Client about upcoming Rate-Limiting:
  once a soft limit is exceeded
  there is a limited number of additional tries available
  the client needs to make sure only the honest user gets to use these
- Client then takes extra measures, for example
  - Send an E-Mail / SMS / ... with an one-time code to the user
  - Add Puzzles to the login screen

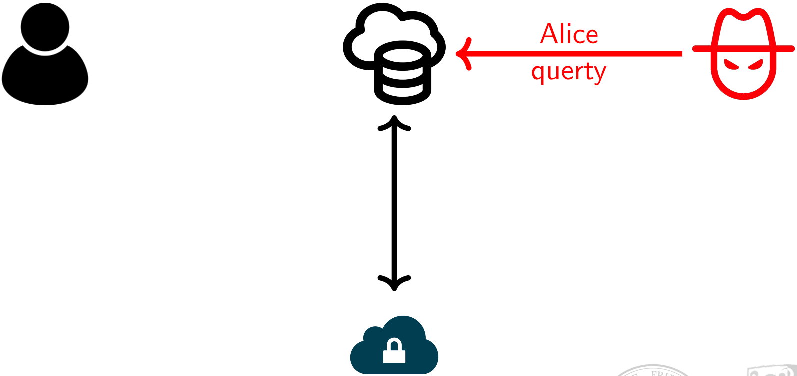
Rate-Limiting external clients



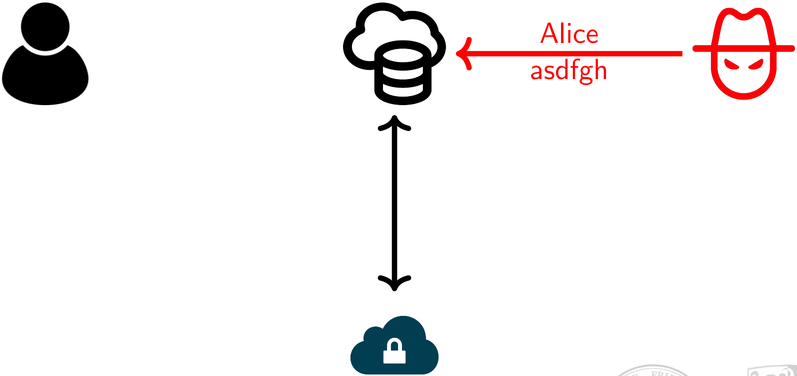
Rate-Limiting external clients



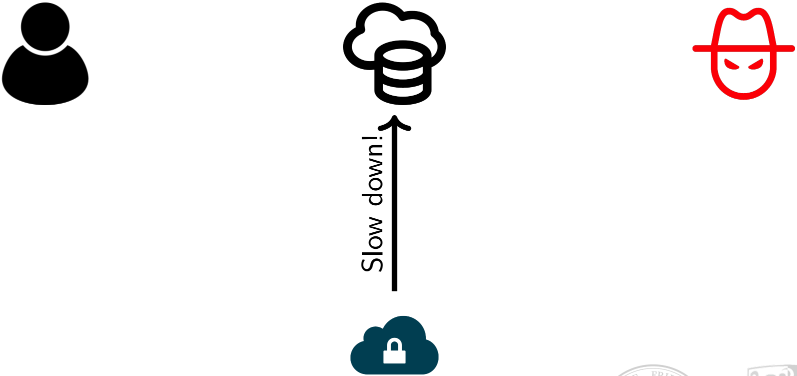
Rate-Limiting external clients



Rate-Limiting external clients

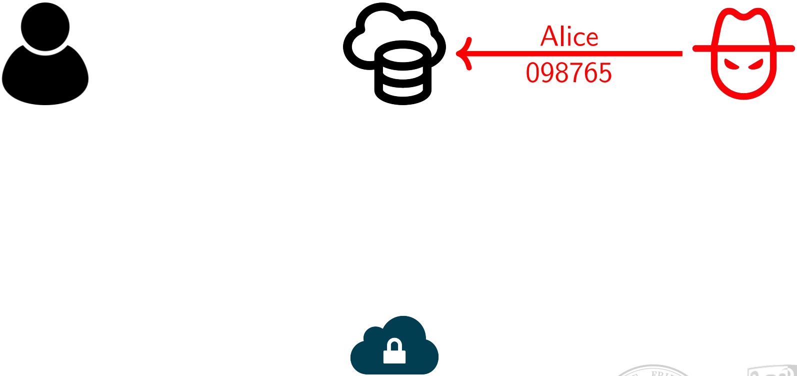


Rate-Limiting external clients

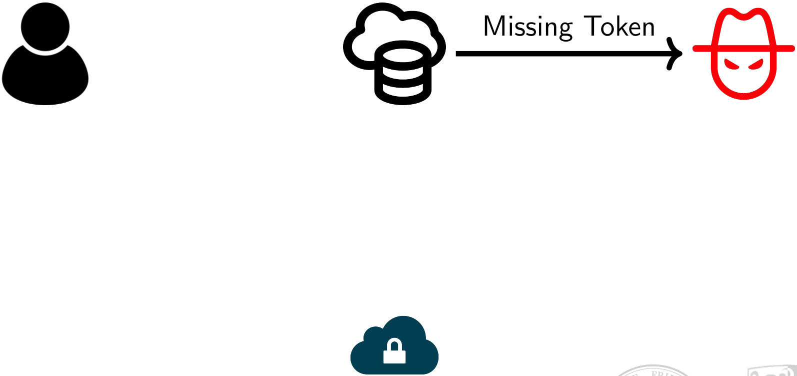


Rate-Limiting external clients

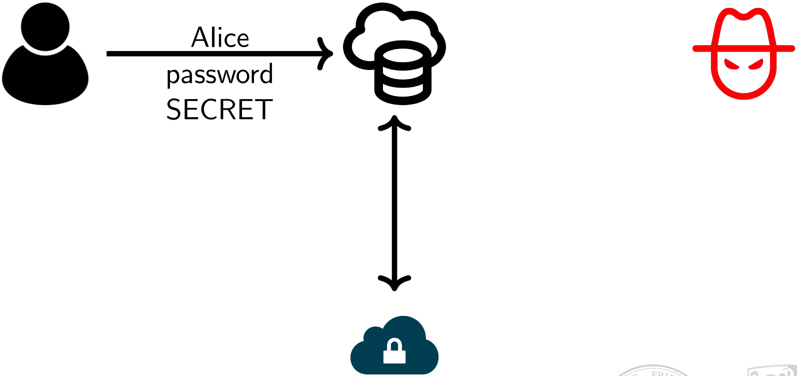
Rate-Limiting external clients



Rate-Limiting external clients



Rate-Limiting external clients



Upgrade Path

- Both, salted hash and Phoenix need a database field to store data
- Algorithm-ID often already stored alongside the salt and hash
- Upgrade users once they log in

Username	Password	Data
Alice	`12:PeADRGbk:gaG4s[...]2BwM=`	...
Bob	`12:q79JVDSo:IIRBz[...]/9L4=`	...
Carol	`5:3V+ToDHL:FCozKw/gxP/9YZ+Pdr7pcg==`	...

Upgrade Path

- Both, salted hash and Phoenix need a database field to store data
- Algorithm-ID often already stored alongside the salt and hash
- Upgrade users once they log in

Username	Password	Data
Alice	12:PeADRGbk:gaG4s[...]2BwM=	...
Bob	13:MxTfsL[...]phh+8i3DU==	...
Carol	5:3V+ToDHL:FCozKw/gxP/9YZ+Pdr7pcg==	...

Concluding Remark

Summary

- Strengthened model of password-hardening
- Formal treatment to key-rotation
- Devastating attack to an existing scheme
- PHOENIX: The most efficient scheme to date

Concluding Remark

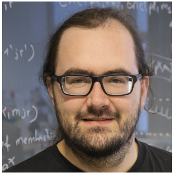
Summary

- Strengthened model of password-hardening
- Formal treatment to key-rotation
- Devastating attack to an existing scheme
- PHOENIX: The most efficient scheme to date

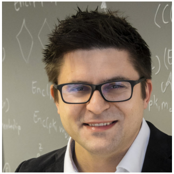
On Going Projects

- Extended functionality – Derive key upon successful validation
- Anonymize end user while retaining rate-limiting
- Deployment by start-up company

Russell W. F. Lai
FAU Nuremberg $\leftarrow$
Chinese U. Hong Kong



Christoph Egger
FAU Nuremberg



Dominique Schröder
FAU Nuremberg

Sherman S. M. Chow
Chinese U. Hong Kong