Chair of
Applied Cryptography

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Simple Password-Hardened Encryption Services

Russell W. F. Lai[1], Christoph Egger[1], Manuel Reinert[2],
Sherman S. M. Chow[3], Matteo Maffei[4], and Dominique Schröder[1]

[1]Friedrich-Alexander University Erlangen-Nuremberg
[2]Saarland University
[3]Chinese University of Hong Kong
[4]TU Wien

Overview

One-Package Solution for Data Security – Password-Hardened Encryption

What it does?

To protect sensitive *client* data ...

... stored in a *server* with password (or biometric / two-factor / etc) authentication ...

... even after the *server* is completely compromised...

... with minimal help from an external *rate-limiter*.

One-Package Solution for Data Security – Password-Hardened Encryption

What it does?

To protect sensitive *client* data ...

... stored in a *server* with password (or biometric / two-factor / etc) authentication ...

... even after the *server* is completely compromised...

... with minimal help from an external *rate-limiter*.

Security Features

- Eliminate offline (*e.g.*, dictionary) attacks
- Rate-limit online (*e.g.*, password guessing) attacks
- Obliviousness (Rate-Limiter learns nothing)
- Soundness (Rate-Limiter cannot cheat)
- Support key-rotation (required in PCI DSS)

PCI DSS: Payment Card Industry Data Security Standard

One-Package Solution for Data Security – Password-Hardened Encryption

What it does?

To protect sensitive *client* data ...
... stored in a *server* with password (or biometric / two-factor / etc) authentication ...
... even after the *server* is completely compromised...
... with minimal help from an external *rate-limiter*.

Security Features

- Eliminate offline (*e.g.*, dictionary) attacks
- Rate-limit online (*e.g.*, password guessing) attacks
- Obliviousness (Rate-Limiter learns nothing)
- Soundness (Rate-Limiter cannot cheat)
- Support key-rotation (required in PCI DSS)

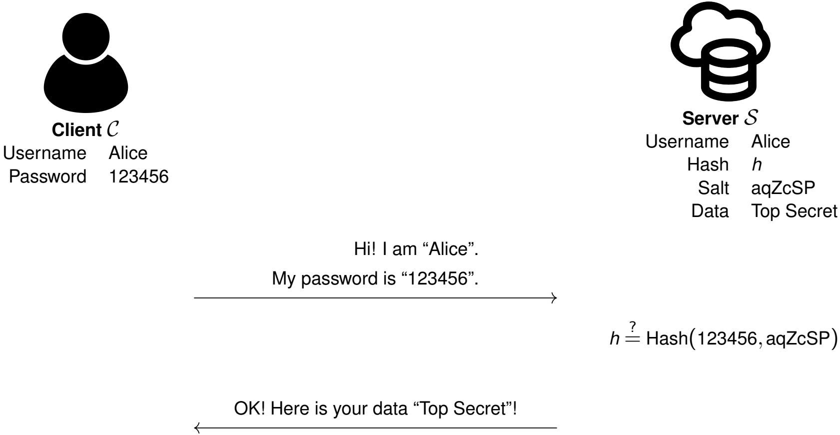
Practicality

- Simple and easy to implement
- Easy to convert from existing systems
- 250 logins per core per second

PCI DSS: Payment Card Industry Data Security Standard

Motivation

Password Authenticated Data Retrieval



Client $\mathcal{C}$

Username	Alice
Password	123456

Server $\mathcal{S}$

Username	Alice
Hash	h
Salt	aqZcSP
Data	Top Secret

Hi! I am “Alice”.
My password is “123456”.

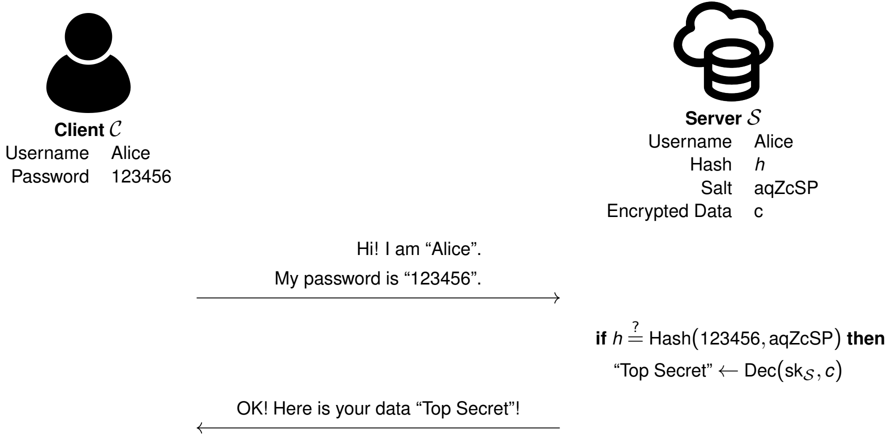
$$\longrightarrow$$

$$h \overset{?}{=} \mathsf{Hash}(123456, \mathsf{aqZcSP})$$

OK! Here is your data “Top Secret”!

$$\longleftarrow$$

Password Authenticated Encrypted Data Retrieval



Client $\mathcal{C}$

Username	Alice
Password	123456

Server $\mathcal{S}$

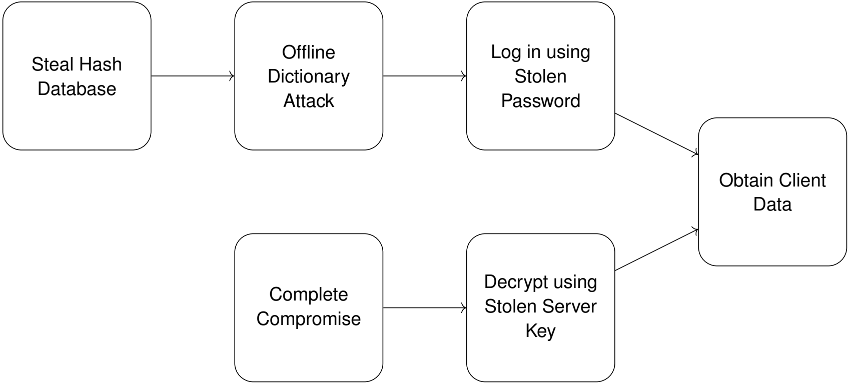
Username	Alice
Hash	h
Salt	aqZcSP
Encrypted Data	c

Hi! I am “Alice”.
My password is “123456”. $\longrightarrow$

if $h \overset{?}{=} \mathsf{Hash}(123456, \mathsf{aqZcSP})$ **then**
“Top Secret” $\leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathcal{S}}, c)$

$\longleftarrow$ OK! Here is your data “Top Secret”!

Issues and Solutions



Steal Hash Database

Offline Dictionary Attack

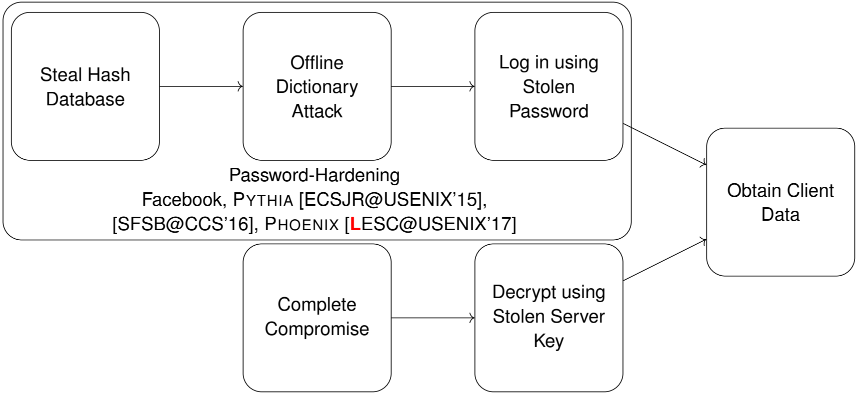
Log in using Stolen Password

Obtain Client Data

Complete Compromise

Decrypt using Stolen Server Key

Issues and Solutions


Steal Hash Database

Offline Dictionary Attack

Log in using Stolen Password

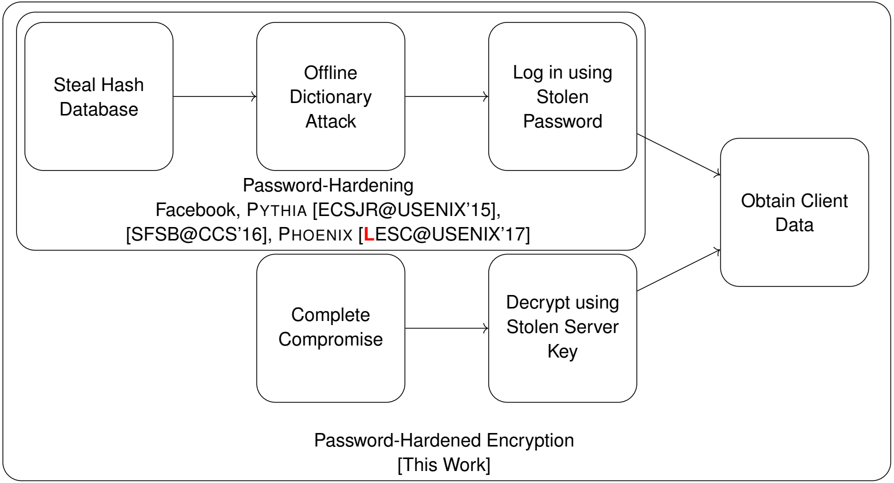
Obtain Client Data

Password-Hardening
Facebook, PYTHIA [ECSJR@USENIX'15],
[SFSB@CCS'16], PHOENIX [**L**ESC@USENIX'17]

Complete Compromise

Decrypt using Stolen Server Key

Issues and Solutions



Steal Hash Database

Offline Dictionary Attack

Log in using Stolen Password

Password-Hardening
Facebook, PYTHIA [ECSJR@USENIX'15],
[SFSB@CCS'16], PHOENIX [**L**ESC@USENIX'17]

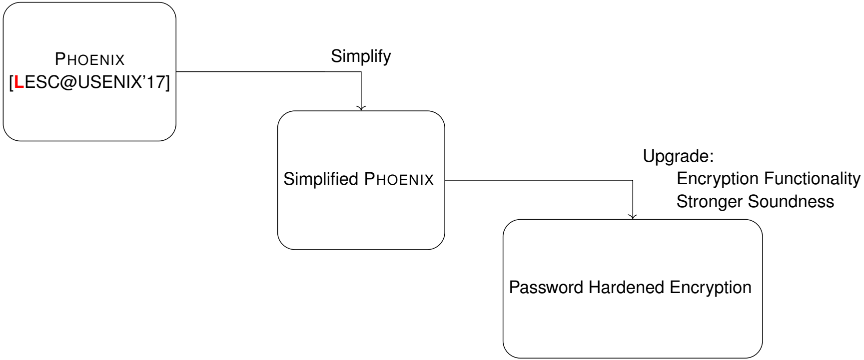
Obtain Client Data

Complete Compromise

Decrypt using Stolen Server Key

Password-Hardened Encryption
[This Work]

Roadmap



PHOENIX
[LESC@USENIX'17]

Simplify

Simplified PHOENIX

Upgrade:
Encryption Functionality
Stronger Soundness

Password Hardened Encryption

Password-Hardening

Ingredient: A Key-Homomorphic Pseudorandom Function (PRF)

Let $\mathbb{G}$ be a group of prime order q (written multiplicatively) where Decisional Diffie Hellman (DDH) is hard.

Let $H : \{0,1\}^* \to \mathbb{G}$ be a random oracle.

The function

$$\begin{aligned} \mathsf{PRF} : \mathbb{Z}_q \times \{0,1\}^* &\to \mathbb{G} \\ (\mathsf{key}, \mathsf{message}) &\mapsto H(\mathsf{message})^{\mathsf{key}} \end{aligned}$$

is pseudorandom under the DDH assumption.

PRF is key-homomorphic:

$$H(\mathsf{message})^{\mathsf{key}+\mathsf{key}'} = H(\mathsf{message})^{\mathsf{key}} \cdot H(\mathsf{message})^{\mathsf{key}'}$$

Simplified Phoenix [LESC@USENIX'17] – Registration

Client $\mathcal{C}$ **Server $\mathcal{S}$** **Rate-Limiter $\mathcal{R}$**

$\xrightarrow{\text{“Register”, “Alice”, “123456”}}$

$\mathsf{aqZcSP} \leftarrow_\$ \mathsf{Salts}$

$\xrightarrow{\text{“Register”}}$

$\mathsf{OjQZEe} \leftarrow_\$ \mathsf{Salts}$

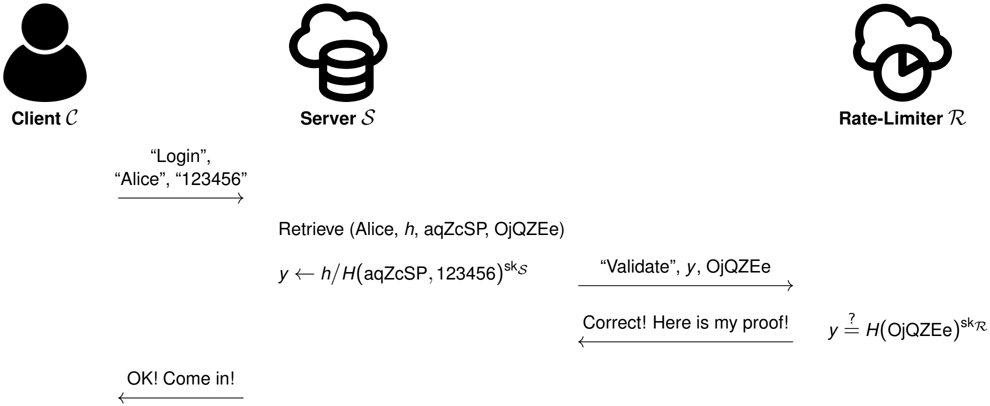
$y \leftarrow H(\mathsf{OjQZEe})^{\mathsf{sk}_\mathcal{R}}$

$\xleftarrow{y, \mathsf{OjQZEe}}$

$h \leftarrow H(\mathsf{aqZcSP}, 123456)^{\mathsf{sk}_\mathcal{S}} \cdot y$

Store (Alice, h, aqZcSP, OjQZEe)

Simplified PHOENIX [LESC@USENIX'17] – Login



Client $\mathcal{C}$

Server $\mathcal{S}$

Rate-Limiter $\mathcal{R}$

"Login",
"Alice", "123456"

$\longrightarrow$

Retrieve (Alice, h, aqZcSP, OjQZEe)

$y \leftarrow h / H(\text{aqZcSP}, 123456)^{\mathsf{sk}_{\mathcal{S}}}$

"Validate", y, OjQZEe

$\longrightarrow$

Correct! Here is my proof!

$\longleftarrow$

$y \overset{?}{=} H(\text{OjQZEe})^{\mathsf{sk}_{\mathcal{R}}}$

OK! Come in!

$\longleftarrow$

Simplified PHOENIX [LESC@USENIX'17] – Key-Rotation

$$h = H(\text{aqZcSP}, 123456)^{\mathsf{sk}_{\mathcal{S}}} \cdot H(\text{OjQZEe})^{\mathsf{sk}_{\mathcal{R}}}$$

Server $\mathcal{S}$
Key $\mathsf{sk}_{\mathcal{S}}$

Rate-Limiter $\mathcal{R}$
Key $\mathsf{sk}_{\mathcal{R}}$

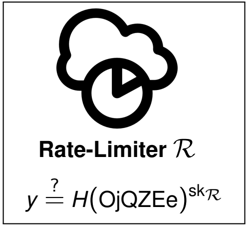
$$\begin{aligned} h' &= h^{\alpha} \cdot H(\text{OjQZEe})^{\beta} \\ &= H(\text{aqZcSP}, 123456)^{\alpha \cdot \mathsf{sk}_{\mathcal{S}}} \cdot H(\text{OjQZEe})^{\alpha \cdot \mathsf{sk}_{\mathcal{R}} + \beta} \\ &= H(\text{aqZcSP}, 123456)^{\mathsf{sk}'_{\mathcal{S}}} \cdot H(\text{OjQZEe})^{\mathsf{sk}'_{\mathcal{R}}} \end{aligned}$$

Server $\mathcal{S}$
Key $\mathsf{sk}'_{\mathcal{S}} = \alpha \cdot \mathsf{sk}_{\mathcal{S}}$

Rate-Limiter $\mathcal{R}$
Key $\mathsf{sk}'_{\mathcal{R}} = \alpha \cdot \mathsf{sk}_{\mathcal{R}} + \beta$

Simplified PHOENIX [LESC@USENIX'17] – What the rate-limiter does?

- Equality Check Functionality:
  Check equality of pseudorandom function values.
- Rate-limiting Policy:
  Refuse to respond if “OjQZEe” appears too frequently.



Rate-Limiter $\mathcal{R}$

$$y \overset{?}{=} H(\mathsf{OjQZEe})^{\mathsf{sk}_{\mathcal{R}}}$$

Simplified Phoenix [LESC@USENIX'17] – What the rate-limiter does?

- Equality Check Functionality:
  Check equality of pseudorandom function values.
- Rate-limiting Policy:
  Refuse to respond if “OjQZEe” appears too frequently.

Rate-Limiter $\mathcal{R}$

$$y \overset{?}{=} H(\text{OjQZEe})^{\mathsf{sk}_{\mathcal{R}}}$$

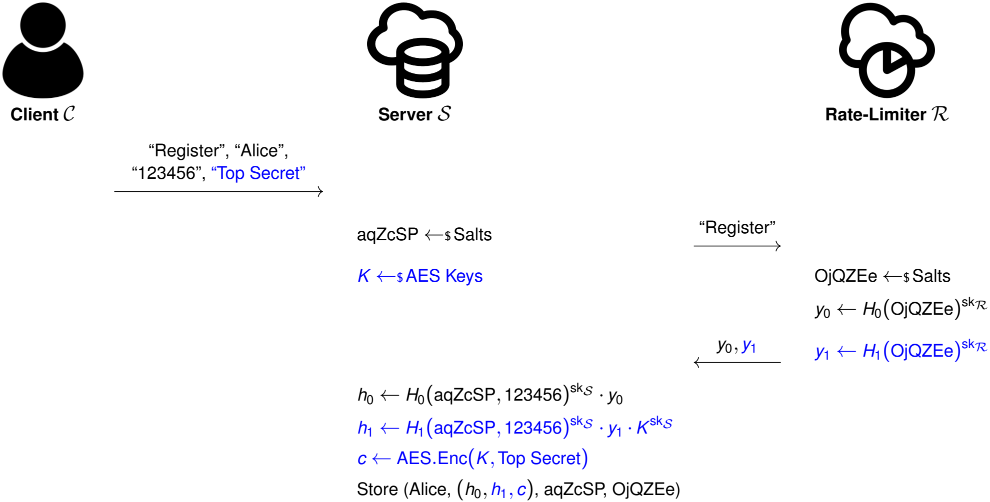
Idea: Upgrade to Password-Hardened Encryption

Conditional Decryption Functionality:

If “Check equality of pseudorandom function values” = True **then**
Partially decrypt ciphertext.

Password-Hardened Encryption

Password-Hardened Encryption – Registration



Client $\mathcal{C}$ **Server $\mathcal{S}$** **Rate-Limiter $\mathcal{R}$**

$\xrightarrow{\text{“Register”, “Alice”, “123456”, “Top Secret”}}$

$\text{aqZcSP} \leftarrow_\$ \text{Salts}$

$K \leftarrow_\$ \text{AES Keys}$

$\xrightarrow{\text{“Register”}}$

$\text{OjQZEe} \leftarrow_\$ \text{Salts}$

$y_0 \leftarrow H_0(\text{OjQZEe})^{\mathsf{sk}_\mathcal{R}}$

$y_1 \leftarrow H_1(\text{OjQZEe})^{\mathsf{sk}_\mathcal{R}}$

$\xleftarrow{y_0, y_1}$

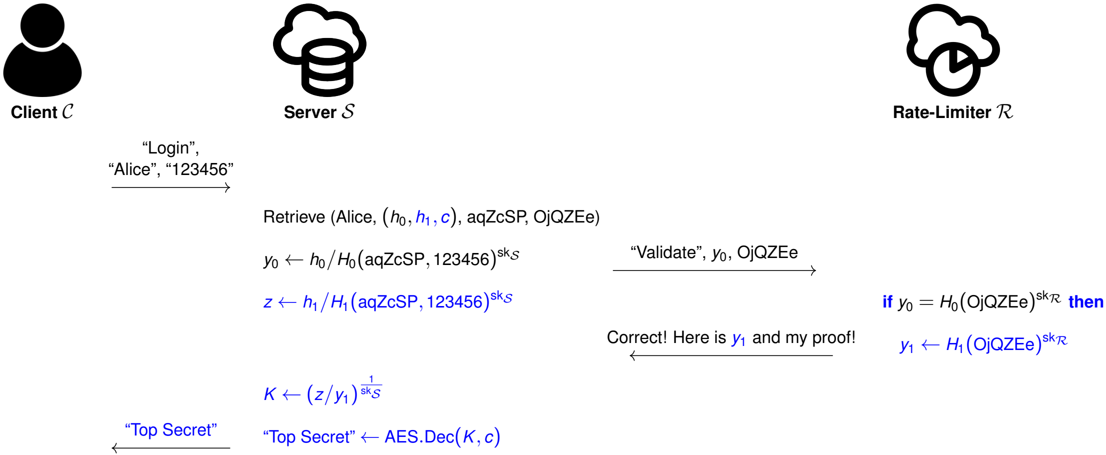
$h_0 \leftarrow H_0(\text{aqZcSP}, 123456)^{\mathsf{sk}_\mathcal{S}} \cdot y_0$

$h_1 \leftarrow H_1(\text{aqZcSP}, 123456)^{\mathsf{sk}_\mathcal{S}} \cdot y_1 \cdot K^{\mathsf{sk}_\mathcal{S}}$

$c \leftarrow \text{AES.Enc}(K, \text{Top Secret})$

Store (Alice, (h_0, h_1, c), aqZcSP, OjQZEe)

Password-Hardened Encryption – Login



Client $\mathcal{C}$ **Server $\mathcal{S}$** **Rate-Limiter $\mathcal{R}$**

“Login”, “Alice”, “123456” →

Retrieve (Alice, (h_0, h_1, c), aqZcSP, OjQZEe)

$y_0 \leftarrow h_0 / H_0(\text{aqZcSP}, 123456)^{\mathsf{sk}_\mathcal{S}}$

“Validate”, y_0, OjQZEe →

$z \leftarrow h_1 / H_1(\text{aqZcSP}, 123456)^{\mathsf{sk}_\mathcal{S}}$

if $y_0 = H_0(\text{OjQZEe})^{\mathsf{sk}_\mathcal{R}}$ **then**

$y_1 \leftarrow H_1(\text{OjQZEe})^{\mathsf{sk}_\mathcal{R}}$

← Correct! Here is y_1 and my proof!

$K \leftarrow (z / y_1)^{\frac{1}{\mathsf{sk}_\mathcal{S}}}$

“Top Secret” $\leftarrow$ AES.Dec(K, c)

← “Top Secret”

Password-Hardened Encryption – Security Features

Against Compromised Server

- Eliminate Offline Attacks
  - Password Hashes are masked by $\mathcal{R}$'s PRF
  - Compromised $\mathcal{S}$ must communicate with $\mathcal{R}$
- Rate-Limit Online Attacks (per Client)
  - $\mathcal{R}$ records the salt (*e.g.*, OjQZEe) in each login request
  - $\mathcal{R}$ refuses to respond if a client (a salt) tries to log in too frequently

Password-Hardened Encryption – Security Features

Against Compromised Rate-Limiter

- Obliviousness
    - Registration and login requests are completely independent of clients’ passwords and data
    - $\mathcal{R}$ learns nothing about clients’ passwords and data
- Soundness
    - $\mathcal{R}$ must prove for both valid and invalid requests

Proactive Security

- Key-Rotation
    - *e.g.*, periodically and when one party is (suspected to be) compromised
    - Due to the key-homomorphic PRF

Performance Evaluation

Setup

- 10 Core Intel Xeon E5-2640 CPU (both $\mathcal{S}$ and $\mathcal{R}$)
- Charm crypto prototyping library
- Falcon Web Framework
- HTTPS with keep-alive

Comparison (Rate-Limiter Throughput)

- $\approx 4x$ of PYTHIA [ECSJR@USENIX'15]
- $\approx 1.5x$ of PHOENIX [**L**ESC@USENIX'17]
- (Those are password-hardening without encryption!)

Performance Graphs

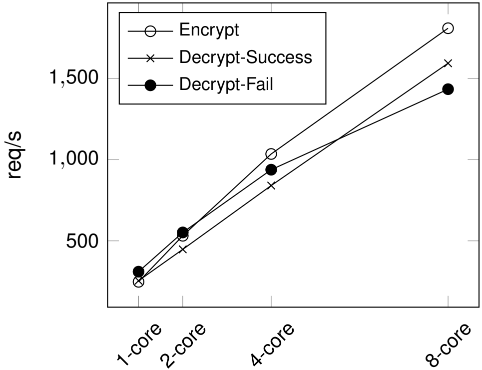


Figure: Server throughput in req/s

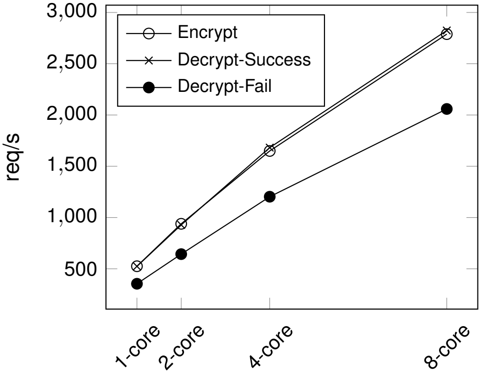


Figure: Rate-Limiter throughput in req/s

Conclusion

Simple Password-Hardened Encryption Services
– One-Package Solution for Data Security

Russell W. F. Lai
Friedrich-Alexander University Erlangen-Nuremberg
`russell.lai@cs.fau.de`

Questions and Answers

Why ...?

Why is it difficult to compromise both $\mathcal{S}$ and $\mathcal{R}$?

- Compromising two parties require twice the effort.
- We assume that $\mathcal{R}$ is built and maintained by security experts, so it is difficult to compromise.

Why not ...?

Why not password-authenticated key-exchange (PAKE)?

Different functionality. In PAKE, both parties know the password, and a fresh key is derived every time.

Why not password-protected secret-sharing (PPSS)?

- No existing scheme supports efficient key-rotation.
- PPSS is too strong: The user in PPSS (the counterpart of the server in PHE) has no secret key.