Simple Password-Hardened Encryption Services

Russell W. F. Lai\textsuperscript{1}, Christoph Egger\textsuperscript{1}, Manuel Reinert\textsuperscript{2}, Sherman S. M. Chow\textsuperscript{3}, Matteo Maffei\textsuperscript{4}, and Dominique Schröder\textsuperscript{1}

\textsuperscript{1}Friedrich-Alexander University Erlangen-Nuremberg
\textsuperscript{2}Saarland University
\textsuperscript{3}Chinese University of Hong Kong
\textsuperscript{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

Russel W. F. Lai
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 C**
- Username: Alice
- Password: 123456

**Server S**
- Username: Alice
- Hash: $h$
- Salt: aqZcSP
- Data: Top Secret

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

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

OK! Here is your data “Top Secret”!
Hi! I am "Alice". My password is "123456".

if \( h \neq \text{Hash}(123456, aqZcSP) \) then

"Top Secret" \( \leftarrow \text{Dec}(s_kS, c) \)

OK! Here is your data "Top Secret"!
Issues and Solutions

- Steal Hash Database
- Offline Dictionary Attack
- Log in using Stolen Password
- Obtain Client Data
- Decrypt using Stolen Server Key
- Complete Compromise

Password-Hardening:
- Facebook, PHYTHIA
- [ECSJR@USENIX'15], [SFSB@CCS'16], PHOENIX
- [RESTC@USENIX'17]

Password-Hardened Encryption:
- [This Work]
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 [LESC@USENIX’17]

Obtain Client Data

Complete Compromise → Decrypt using Stolen Server Key

[This Work]
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 [LESC@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 $G$ be a group of prime order $q$ (written multiplicatively) where Decisional Diffie Hellman (DDH) is hard.

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

The function

$$\text{PRF} : \mathbb{Z}_q \times \{0, 1\}^* \rightarrow G$$

$$(\text{key}, \text{message}) \mapsto H(\text{message})^{\text{key}}$$

is pseudorandom under the DDH assumption.

PRF is key-homomorphic:

$$H(\text{message})^{\text{key} + \text{key}'} = H(\text{message})^{\text{key}} \cdot H(\text{message})^{\text{key}'}$$
Simplified PHOENIX [LESC@USENIX’17] – Registration

Client $C$

Server $S$

Rate-Limiter $\mathcal{R}$

```
"Register", "Alice", "123456"
```

```
aqZcSP \leftarrow \$ Salts
```

```
OjQZEe \leftarrow \$ Salts
```

```
y, OjQZEe
```

```
h \leftarrow H(aqZcSP, 123456)^{sk_S} \cdot y
```

Store (Alice, $h$, aqZcSP, OjQZEe)
Simplified PHOENIX [LESC@USENIX’17] – Login

Client $C$

Server $S$

Rate-Limiter $\mathcal{R}$

"Login", "Alice", "123456"

$y \leftarrow h/H(aqZcSP, 123456)^{sk_S}$

"Validate", $y$, $OjQZEe$

Correct! Here is my proof! $y = H(OjQZEe)^{sk_R}$

OK! Come in!
Simplified PHOENIX [LESC@USENIX’17] – Key-Rotation

\[ h = H(aqZcSP, 123456)^{sk_S} \cdot H(OjQZEe)^{sk_R} \]

\[ h' = h^\alpha \cdot H(OjQZEe)^\beta \]
\[ = H(aqZcSP, 123456)^{\alpha \cdot sk_S} \cdot H(OjQZEe)^{\alpha \cdot sk_R + \beta} \]
\[ = H(aqZcSP, 123456)^{sk'_S} \cdot H(OjQZEe)^{sk'_R} \]

Server \( S \)
Key \( sk_S \)

Rate-Limiter \( R \)
Key \( sk_R \)

Server \( S \)
Key \( sk'_S = \alpha \cdot sk_S \)

Rate-Limiter \( R \)
Key \( sk'_R = \alpha \cdot sk_R + \beta \)
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.

\[ y \leftarrow H(OjQZEe)^{skR} \]
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.

Idea: Upgrade to Password-Hardened Encryption

Conditional Decryption Functionality:

If “Check equality of pseudorandom function values” = True
  Partially decrypt ciphertext.
Password-Hardened Encryption
Password-Hardened Encryption – Registration

Client $C$

Server $S$

Rate-Limiter $\mathcal{R}$

"Register", “Alice”, “123456”, “Top Secret”

$aqZcSP \leftarrow$ Salts

$K \leftarrow$ AES Keys

$h_0 \leftarrow H_0(aqZcSP, 123456)^{sk_S} \cdot y_0$

$h_1 \leftarrow H_1(aqZcSP, 123456)^{sk_S} \cdot y_1 \cdot K^{sk_S}$

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

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

"Register"

$OjQZEe \leftarrow$ Salts

$y_0 \leftarrow H_0(OjQZEe)^{sk_{\mathcal{R}}}$

$y_1 \leftarrow H_1(OjQZEe)^{sk_{\mathcal{R}}}$
Password-Hardened Encryption – Login

Client $C$ → Server $S$ → Rate-Limiter $R$

```
"Login", "Alice", "123456"
```

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

$y_0 \leftarrow h_0 / H_0(aqZcSP, 123456)^{sk_S}$

$z \leftarrow h_1 / H_1(aqZcSP, 123456)^{sk_S}$

```
"Validate", $y_0$, OjQZEe
```

Correct! Here is $y_1$ and my proof!

```
if $y_0 = H_0(OjQZEe)^{sk_R}$ then

$y_1 \leftarrow H_1(OjQZEe)^{sk_R}$
```

```
K \leftarrow (z/y_1)^{1/\pi}
```

```
"Top Secret"
```

```
"Top Secret" \leftarrow AES.Dec(K, c)
```

Simple Password-Hardened Encryption Services

Russel W. F. Lai
Password-Hardened Encryption – Security Features

Against Compromised Server

• Eliminate Offline Attacks
  • Password Hashes are masked by $R$’s PRF
  • Compromised $S$ must communicate with $R$
• Rate-Limit Online Attacks (per Client)
  • $R$ records the salt (e.g., OjQZEe) in each login request
  • $R$ 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 [LESC@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 $S$ and $R$?**

- Compromising two parties require twice the effort.
- We assume that $R$ 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.