Password-Hardened Encryption Revisited

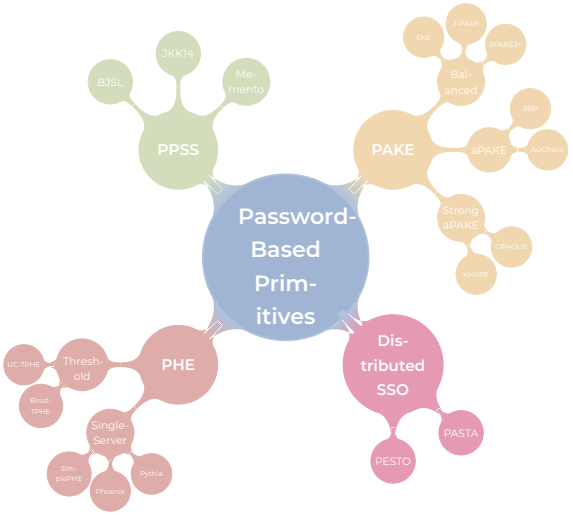
Ruben Baecker, Paul Gerhart, and Dominique Schröder

Why Password Hardened Encryption?

We found an attack...

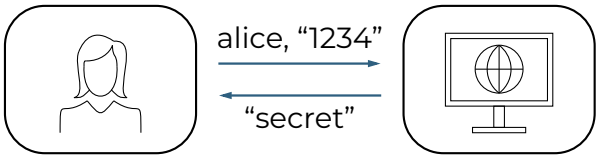
...caused by an insufficient security model

Password-Based Primitives



The Problem with **Database Breaches**

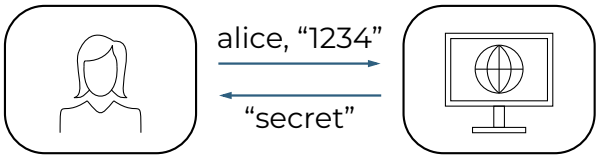
$\mathsf{H}(\text{“1234”}, 2181) = b481$



un	s	h	data
alice	2181	$b481$	“secret”

The Problem with **Database Breaches**

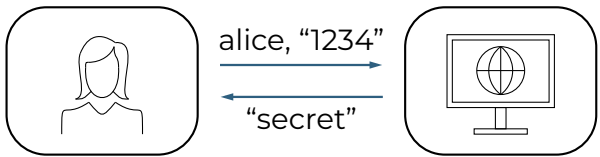
$\mathsf{H}(\text{“1234”}, 2181) = b481$



un	s	h	data
alice	2181	$b481$	“secret”

Encryption at Rest?

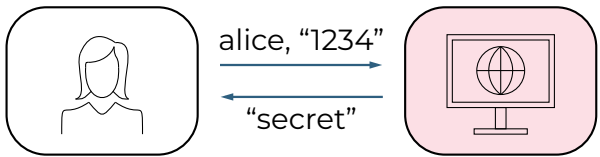
$\mathsf{H}(\text{“1234”}, 2181) = b481$



un	s	h	data
alice	0935	$d390$	“gsrtbe”

Encryption at Rest?

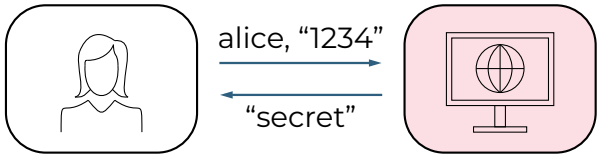
$\mathsf{H}(\text{“1234”}, 2181) = b481$



un	s	h	data
alice	0935	$d390$	“gsrtbe”

Encryption at Rest?

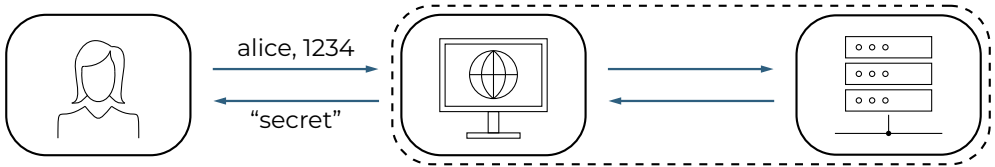
$\mathsf{H}(\text{“1234”}, 2181) = b481$



un	s	h	data
alice	2181	$b481$	“secret”

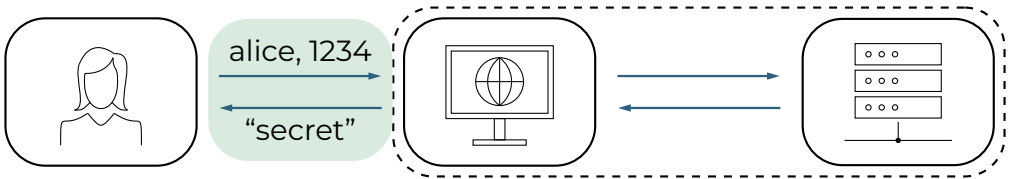
PHE: Distributing Trust

ECSJR'15 (USENIX), SFSB'16 (CCS), LESC'17 (USENIX), LERCMS'18 (USENIX), BELSSZ'20 (CCS)



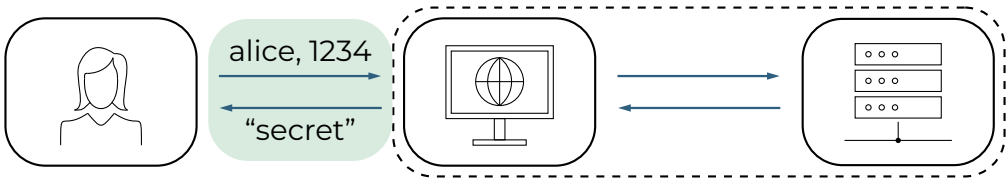
PHE: **Oblivious** To The User

- The user does not have to do crypto
- The user-server interface is unchanged
- PHE protects **ALL** users



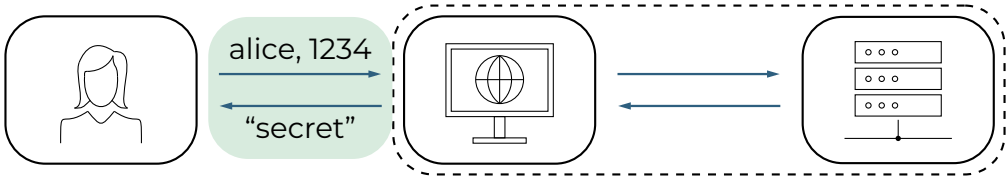
PHE: **Oblivious** To The User

- The user does not have to do crypto
- The user-server interface is unchanged
- PHE protects **ALL** users



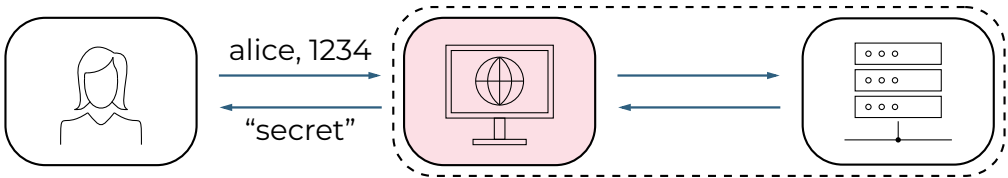
PHE: **Oblivious** To The User

- The user does not have to do crypto
- The user-server interface is unchanged
- PHE protects **ALL** users



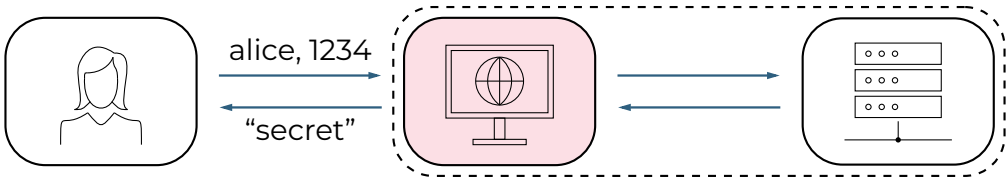
PHE: Security Against **Corrupt Servers**

- Online interaction with ratelimiter necessary for password validation and decryption
- Ratelimiter rate-limits authentication attempts on a **per-user** basis



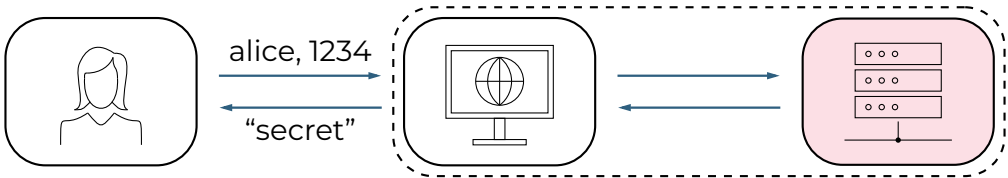
PHE: Security Against **Corrupt Servers**

- Online interaction with ratelimiter necessary for password validation and decryption
- Ratelimiter rate-limits authentication attempts on a **per-user** basis



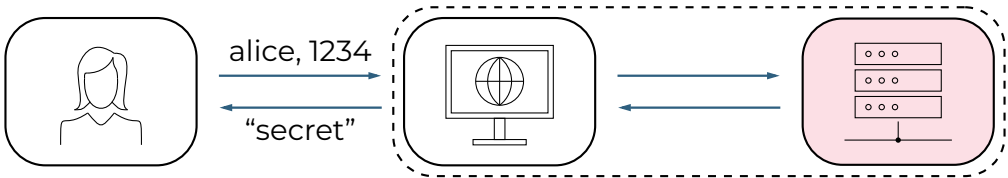
PHE: Security Against **Corrupt Ratelimiters**

- Ratelimiter learns **nothing** about the password or the encrypted message
- Ratelimiter cannot convince the server of a wrong outcome
- Ratelimiter operates in a **zero-trust** fashion
- Ratelimiter only requires **constant** storage



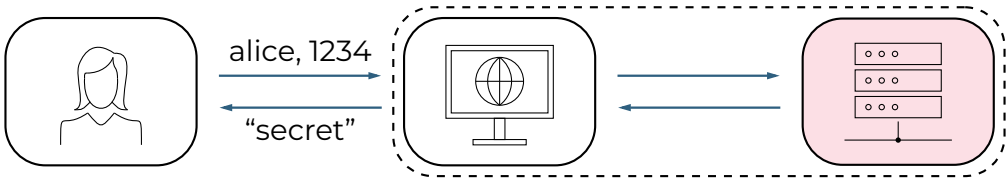
PHE: Security Against **Corrupt Ratelimiters**

- Ratelimiter learns **nothing** about the password or the encrypted message
- Ratelimiter cannot convince the server of a wrong outcome
- Ratelimiter operates in a **zero-trust** fashion
- Ratelimiter only requires **constant** storage



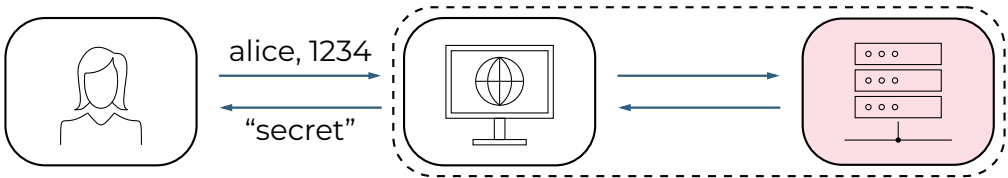
PHE: Security Against **Corrupt Ratelimiters**

- Ratelimiter learns **nothing** about the password or the encrypted message
- Ratelimiter cannot convince the server of a wrong outcome
- Ratelimiter operates in a **zero-trust** fashion
- Ratelimiter only requires **constant** storage



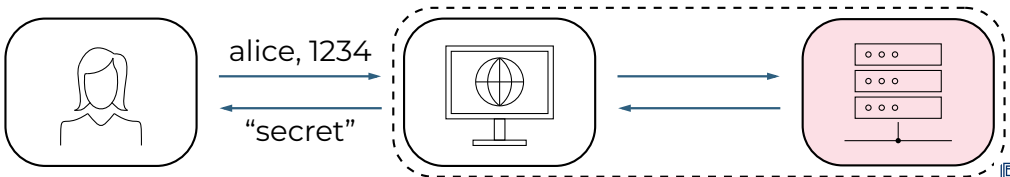
PHE: Security Against **Corrupt Ratelimiters**

- Ratelimiter learns **nothing** about the password or the encrypted message
- Ratelimiter cannot convince the server of a wrong outcome
- Ratelimiter operates in a **zero-trust** fashion
- Ratelimiter only requires **constant** storage



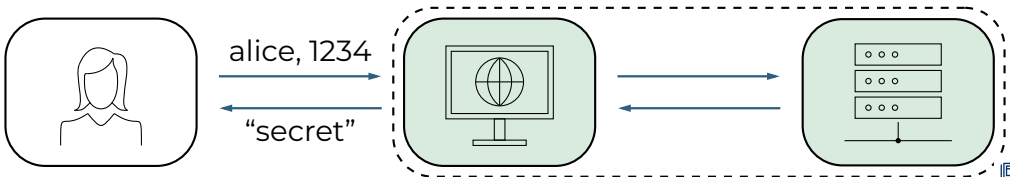
PHE: **Restoring Security** After Corruption

- After one corruption, we're back to a **single point of failure**
- Key rotation resets corruptions
- Provides security even against **alternating corruptions**
- Key rotation is best practice in the industry and required by PCI-DSS and NIST standards
- Requires no user interaction and is of constant communication size



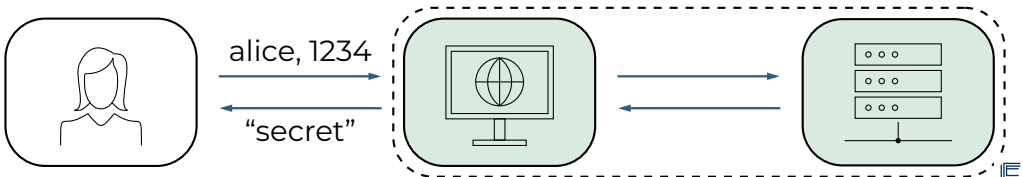
PHE: **Restoring Security** After Corruption

- After one corruption, we're back to a **single point of failure**
- Key rotation resets corruptions
- Provides security even against **alternating corruptions**
- Key rotation is best practice in the industry and required by PCI-DSS and NIST standards
- Requires no user interaction and is of constant communication size



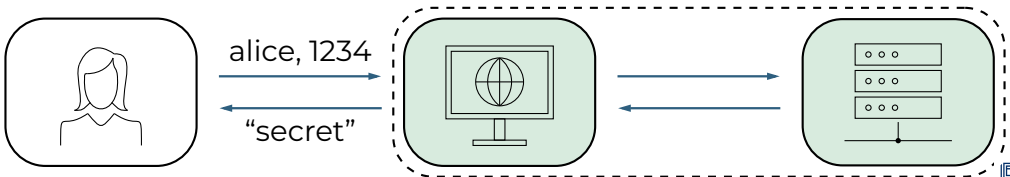
PHE: **Restoring Security** After Corruption

- After one corruption, we're back to a **single point of failure**
- Key rotation resets corruptions
- Provides security even against **alternating corruptions**
- Key rotation is best practice in the industry and required by PCI-DSS and NIST standards
- Requires no user interaction and is of constant communication size



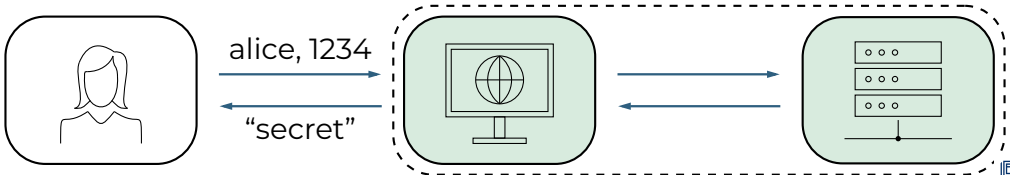
PHE: **Restoring Security** After Corruption

- After one corruption, we're back to a **single point of failure**
- Key rotation resets corruptions
- Provides security even against **alternating corruptions**
- Key rotation is best practice in the industry and required by PCI-DSS and NIST standards
- Requires no user interaction and is of constant communication size



PHE: **Restoring Security** After Corruption

- After one corruption, we're back to a **single point of failure**
- Key rotation resets corruptions
- Provides security even against **alternating corruptions**
- Key rotation is best practice in the industry and required by PCI-DSS and NIST standards
- Requires no user interaction and is of constant communication size



Why Password Hardened Encryption?

We found an attack...

...caused by an insufficient security model

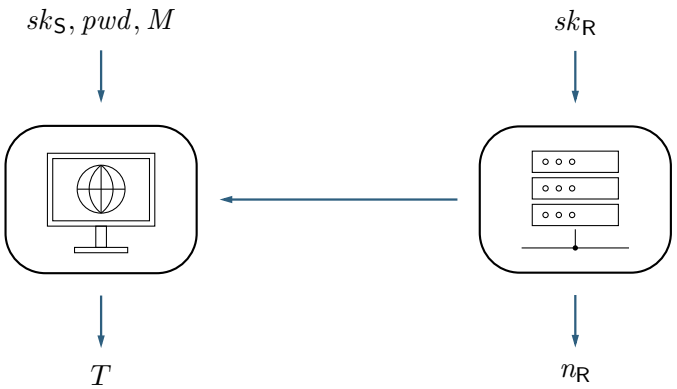
Simple PHE – LERCMS'18

$$T = (t_0, t_1, n_{\mathsf{S}}, n_{\mathsf{R}})$$
$$t_0 = F^0_{sk_{\mathsf{S}}}(pwd, n_{\mathsf{S}}) \cdot F^0_{sk_{\mathsf{R}}}(n_{\mathsf{R}})$$
$$t_1 = F^1_{sk_{\mathsf{S}}}(pwd, n_{\mathsf{S}}) \cdot F^1_{sk_{\mathsf{R}}}(n_{\mathsf{R}}) \cdot M$$

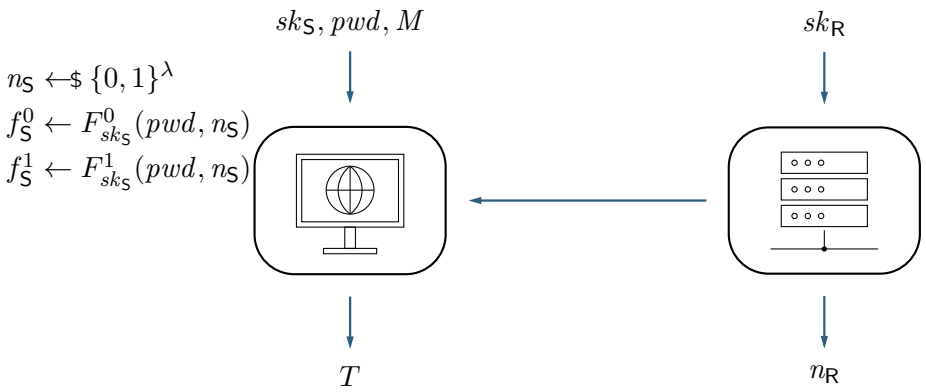
- t_0 is used for the password check
- t_1 hides the message
- commercialized by Virgil Security

FAU

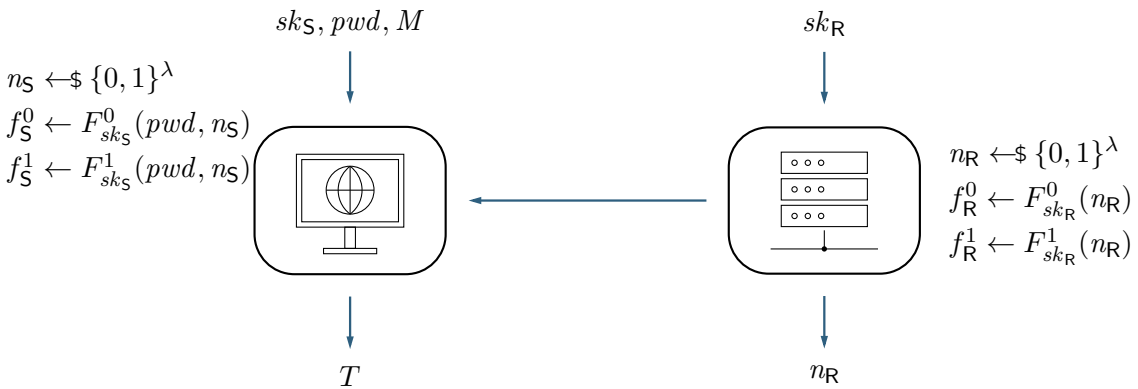
Simple PHE **Encryption**



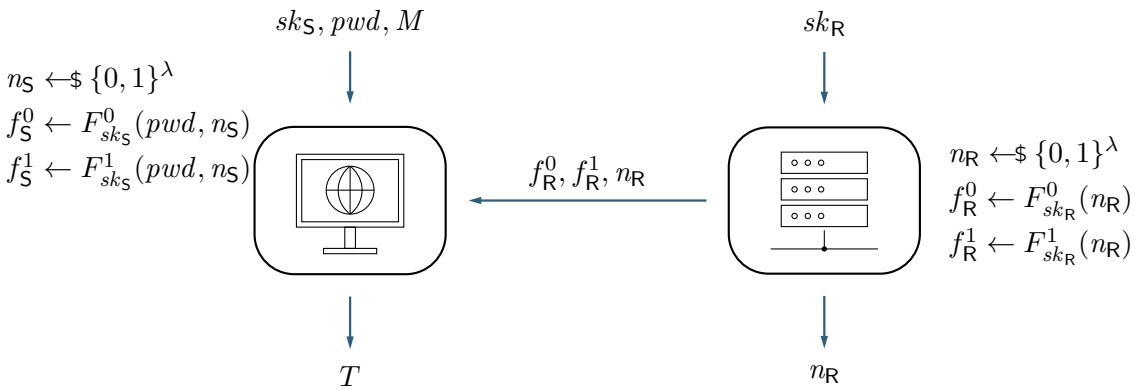
Simple PHE **Encryption**



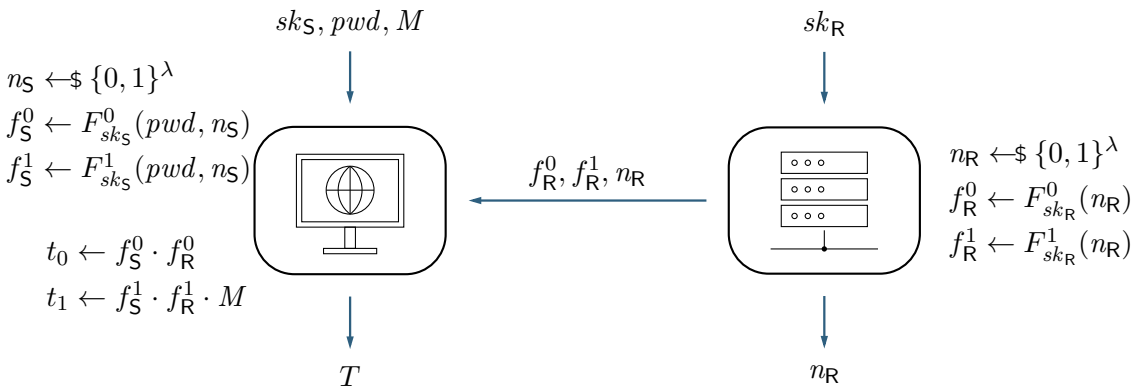
Simple PHE **Encryption**



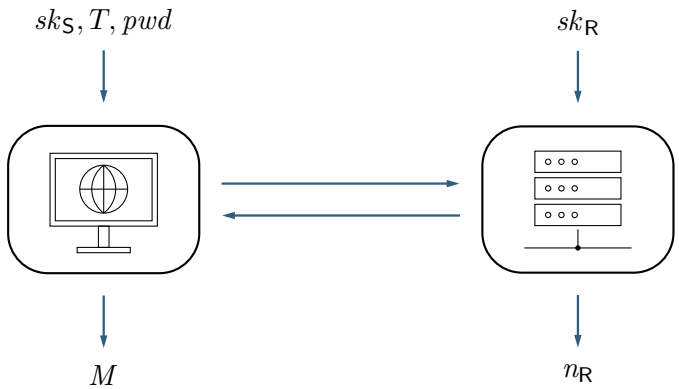
Simple PHE **Encryption**



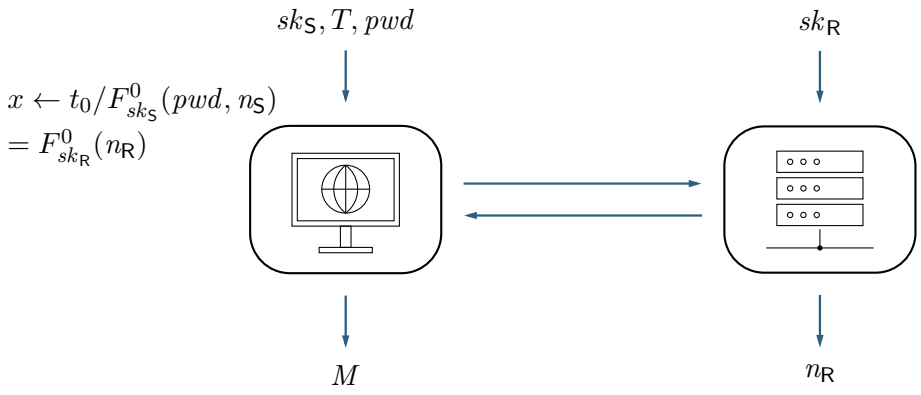
Simple PHE **Encryption**



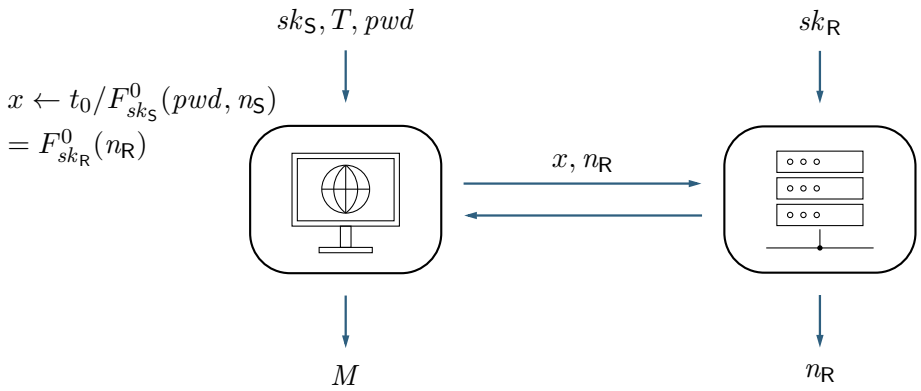
Simple PHE **Decryption**



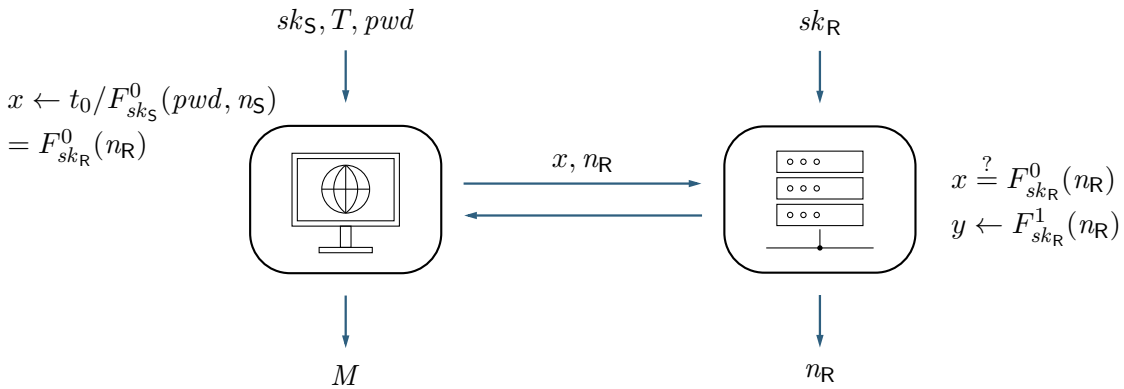
Simple PHE **Decryption**



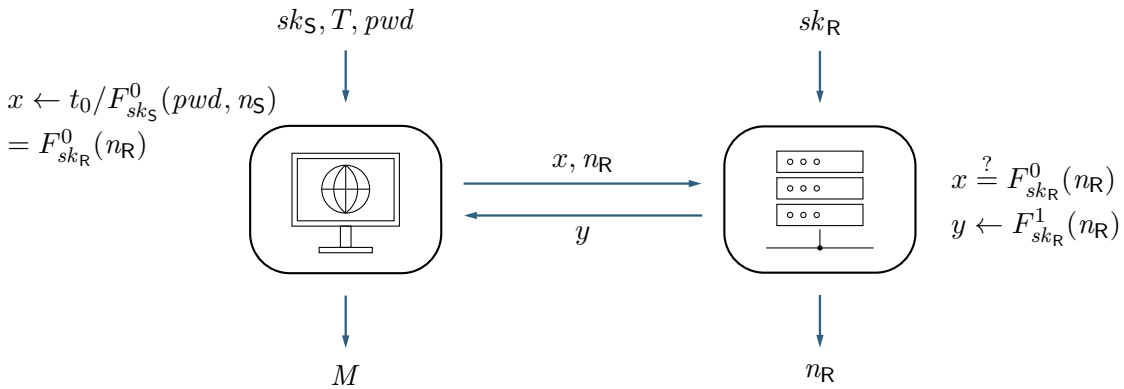
Simple PHE **Decryption**



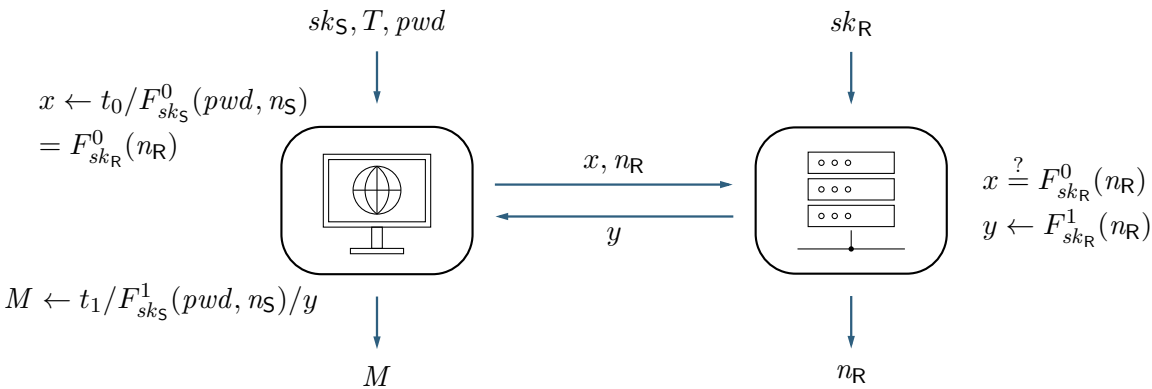
Simple PHE **Decryption**



Simple PHE **Decryption**



Simple PHE **Decryption**



The Attack: Malicious Ratelimiter

Step 0: The target user is already enrolled with the secure password secPW

$$t_0^{\mathsf{hon}} = F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n_\mathsf{S}^{\mathsf{hon}}) \cdot F^0_{sk_\mathsf{R}}(n_\mathsf{R}^{\mathsf{hon}})$$
$$t_1^{\mathsf{hon}} = F^1_{sk_\mathsf{S}}(\mathsf{secPW}, n_\mathsf{S}^{\mathsf{hon}}) \cdot F^1_{sk_\mathsf{R}}(n_\mathsf{R}^{\mathsf{hon}}) \cdot M$$

Step 1: Corrupt the ratelimiter and enroll malicious user with known password malPW reusing $n_\mathsf{R}^{\mathsf{hon}}$

$$t_0^{\mathsf{mal}} = F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n_\mathsf{S}^{\mathsf{mal}}) \cdot F^0_{sk_\mathsf{R}}(n_R^{\mathsf{hon}})$$
$$t_1^{\mathsf{mal}} = F^1_{sk_\mathsf{S}}(\mathsf{malPW}, n_\mathsf{S}^{\mathsf{mal}}) \cdot F^1_{sk_\mathsf{R}}(n_R^{\mathsf{hon}}) \cdot \hat{M}$$

The Attack: Malicious Ratelimiter

Step 0: The target user is already enrolled with the secure password secPW

$$t_0^{\mathsf{hon}} = F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n_\mathsf{S}^{\mathsf{hon}}) \cdot F^0_{sk_\mathsf{R}}(n_\mathsf{R}^{\mathsf{hon}})$$
$$t_1^{\mathsf{hon}} = F^1_{sk_\mathsf{S}}(\mathsf{secPW}, n_\mathsf{S}^{\mathsf{hon}}) \cdot F^1_{sk_\mathsf{R}}(n_\mathsf{R}^{\mathsf{hon}}) \cdot M$$

Step 1: Corrupt the ratelimiter and enroll malicious user with known password malPW reusing $n_\mathsf{R}^{\mathsf{hon}}$

$$t_0^{\mathsf{mal}} = F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n_\mathsf{S}^{\mathsf{mal}}) \cdot F^0_{sk_\mathsf{R}}(n_R^{\mathsf{hon}})$$
$$t_1^{\mathsf{mal}} = F^1_{sk_\mathsf{S}}(\mathsf{malPW}, n_\mathsf{S}^{\mathsf{mal}}) \cdot F^1_{sk_\mathsf{R}}(n_R^{\mathsf{hon}}) \cdot \hat{M}$$

The Attack: Malicious Ratelimiter

Step 0: The target user is already enrolled with the secure password secPW

$$t_0^{\mathsf{hon}} = F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n_\mathsf{S}^{\mathsf{hon}}) \cdot F^0_{sk_\mathsf{R}}(n_\mathsf{R}^{\mathsf{hon}})$$
$$t_1^{\mathsf{hon}} = F^1_{sk_\mathsf{S}}(\mathsf{secPW}, n_\mathsf{S}^{\mathsf{hon}}) \cdot F^1_{sk_\mathsf{R}}(n_\mathsf{R}^{\mathsf{hon}}) \cdot M$$

Step 1: Corrupt the ratelimiter and enroll malicious user with known password malPW reusing $n_\mathsf{R}^{\mathsf{hon}}$

$$t_0^{\mathsf{mal}} = F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n_\mathsf{S}^{\mathsf{mal}}) \cdot F^0_{sk_\mathsf{R}}(n_R^{\mathsf{hon}})$$
$$t_1^{\mathsf{mal}} = F^1_{sk_\mathsf{S}}(\mathsf{malPW}, n_\mathsf{S}^{\mathsf{mal}}) \cdot F^1_{sk_\mathsf{R}}(n_R^{\mathsf{hon}}) \cdot \hat{M}$$

The Attack: Malicious Server

Step 2: Release the ratelimiter and corrupt the server → honest key rotation

Step 3: Extract $F^0_{sk_R}(n_R^{hon})$ from $t_0^{mal} = F^0_{sk_S}(\mathsf{malPW}, n_S^{mal}) \cdot F^0_{sk_R}(n_R^{hon})$

$$F^0_{sk_R}(n_R^{hon}) \leftarrow F^0_{sk_S}(\mathsf{malPW}, n_S^{mal}) \cdot F^0_{sk_R}(n_R^{hon}) / F^0_{sk_S}(\mathsf{malPW}, n_S^{mal})$$

Step 4: Extract $F^0_{sk_S}(\mathsf{secPW}, n_S^{hon})$ from $t_0^{hon} = F^0_{sk_S}(\mathsf{secPW}, n_S^{hon}) \cdot F^0_{sk_R}(n_R^{hon})$

$$F^0_{sk_S}(\mathsf{secPW}, n_S^{hon}) \leftarrow F^0_{sk_S}(\mathsf{secPW}, n_S^{hon}) \cdot F^0_{sk_R}(n_R^{hon}) / F^0_{sk_R}(n_R^{hon})$$

Step 5: Brute-force secPW from $F^0_{sk_S}(\mathsf{secPW}, n_S^{hon})$

FAU

The Attack: Malicious Server

Step 2: Release the ratelimiter and corrupt the server $\rightarrow$ honest key rotation

Step 3: Extract $F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$ from $t_0^{\mathsf{mal}} = F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_R)$

$$F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R}) \leftarrow F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_R) / F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S})$$

Step 4: Extract $F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S})$ from $t_0^{\mathsf{hon}} = F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$

$$F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \leftarrow F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R}) / F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$$

Step 5: Brute-force secPW from $F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S})$

FAU

The Attack: Malicious Server

Step 2: Release the ratelimiter and corrupt the server $\rightarrow$ honest key rotation

Step 3: Extract $F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$ from $t^{\mathsf{mal}}_0 = F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_R)$

$$F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R}) \leftarrow F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_R) / F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S})$$

Step 4: Extract $F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S})$ from $t^{\mathsf{hon}}_0 = F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$

$$F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \leftarrow F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R}) / F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$$

Step 5: Brute-force secPW from $F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S})$

FAU

The Attack: Malicious Server

Step 2: Release the ratelimiter and corrupt the server $\rightarrow$ honest key rotation

Step 3: Extract $F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$ from $t_0^{\mathsf{mal}} = F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_R)$

$$F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R}) \leftarrow F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_R) / F^0_{sk_\mathsf{S}}(\mathsf{malPW}, n^{\mathsf{mal}}_\mathsf{S})$$

Step 4: Extract $F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S})$ from $t_0^{\mathsf{hon}} = F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$

$$F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \leftarrow F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S}) \cdot F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R}) / F^0_{sk_\mathsf{R}}(n^{\mathsf{hon}}_\mathsf{R})$$

Step 5: Brute-force secPW from $F^0_{sk_\mathsf{S}}(\mathsf{secPW}, n^{\mathsf{hon}}_\mathsf{S})$

FAU

Why Password Hardened Encryption?

We found an attack...

...caused by an insufficient security model

Treating Attack Vectors in **Isolation**

- **Hiding:** A corrupt server does not learn an encrypted message
- **Partial Obliviousness:** A corrupt ratelimiter does not learn an encrypted message
- **Soundness:** A corrupt ratelimiter cannot convince the server of an incorrect outcome
- **Forward Security:** An updated ciphertext after key rotation is indistinguishable from a newly generated one

Treating Attack Vectors in **Isolation**

- **Hiding:** A corrupt server does not learn an encrypted message
- **Partial Obliviousness:** A corrupt ratelimiter does not learn an encrypted message
- **Soundness:** A corrupt ratelimiter cannot convince the server of an incorrect outcome
- **Forward Security:** An updated ciphertext after key rotation is indistinguishable from a newly generated one

Treating Attack Vectors in **Isolation**

- **Hiding:** A corrupt server does not learn an encrypted message
- **Partial Obliviousness:** A corrupt ratelimiter does not learn an encrypted message
- **Soundness:** A corrupt ratelimiter cannot convince the server of an incorrect outcome
- **Forward Security:** An updated ciphertext after key rotation is indistinguishable from a newly generated one

Treating Attack Vectors in **Isolation**

- **Hiding:** A corrupt server does not learn an encrypted message
- **Partial Obliviousness:** A corrupt ratelimiter does not learn an encrypted message
- **Soundness:** A corrupt ratelimiter cannot convince the server of an incorrect outcome
- **Forward Security:** An updated ciphertext after key rotation is indistinguishable from a newly generated one

A **Unified** Hiding Definition

$\mathcal{A}$ does not learn an encrypted message

- as a **corrupt server**
  (previously covered by Hiding)
- as a **corrupt ratelimiter**
  (previously covered by Partial Obliviousness)
- using **alternating corruption** patterns
  (previously only partially covered by Forward Security)
- while **deviating from the protocol**
  (previously covered by Soundness)

A **Unified** Hiding Definition

$\mathcal{A}$ does not learn an encrypted message

- as a **corrupt server**
  (previously covered by Hiding)
- as a **corrupt ratelimiter**
  (previously covered by Partial Obliviousness)
- using **alternating corruption** patterns
  (previously only partially covered by Forward Security)
- while **deviating from the protocol**
  (previously covered by Soundness)

A **Unified** Hiding Definition

$\mathcal{A}$ does not learn an encrypted message

- as a **corrupt server**
  (previously covered by Hiding)
- as a **corrupt ratelimiter**
  (previously covered by Partial Obliviousness)
- using **alternating corruption** patterns
  (previously only partially covered by Forward Security)
- while **deviating from the protocol**
  (previously covered by Soundness)

FAU

A **Unified** Hiding Definition

$\mathcal{A}$ does not learn an encrypted message

- as a **corrupt server**
  (previously covered by Hiding)
- as a **corrupt ratelimiter**
  (previously covered by Partial Obliviousness)
- using **alternating corruption** patterns
  (previously only partially covered by Forward Security)
- while **deviating from the protocol**
  (previously covered by Soundness)

FAU

A **Unified** Hiding Definition

$\mathcal{A}$ does not learn an encrypted message

- as a **corrupt server**
  (previously covered by Hiding)
- as a **corrupt ratelimiter**
  (previously covered by Partial Obliviousness)
- using **alternating corruption** patterns
  (previously only partially covered by Forward Security)
- while **deviating from the protocol**
  (previously covered by Soundness)

FAU

Why Password Hardened Encryption?

We found an attack...

...caused by an insufficient security model

Why Password Hardened Encryption?

We found an attack...

...caused by an insufficient security model

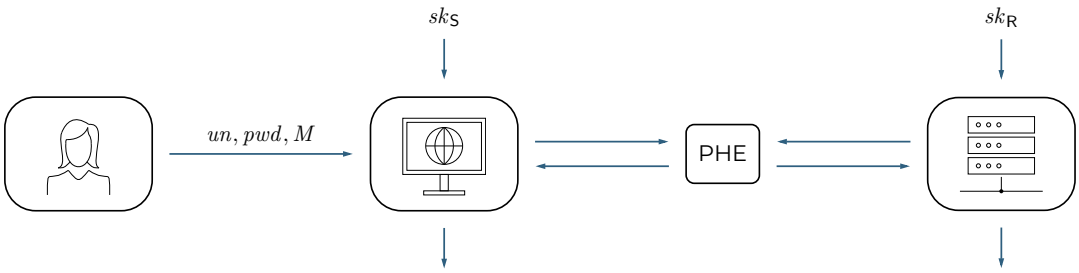
We also proposed the fastes PHE ever

HildeGUARD

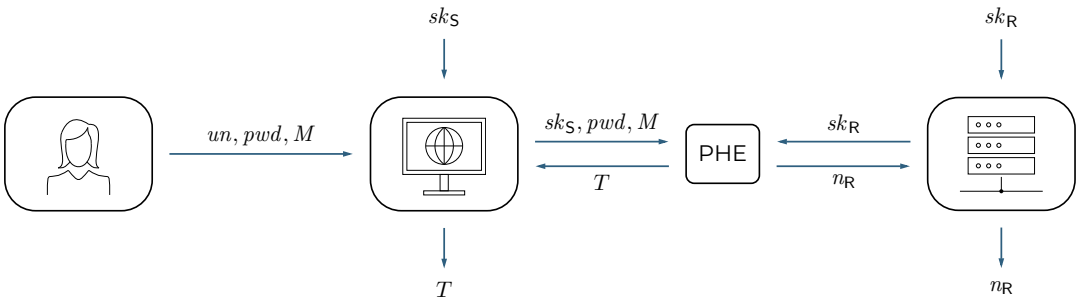
Two conceptual changes enable faster PHE

FAU

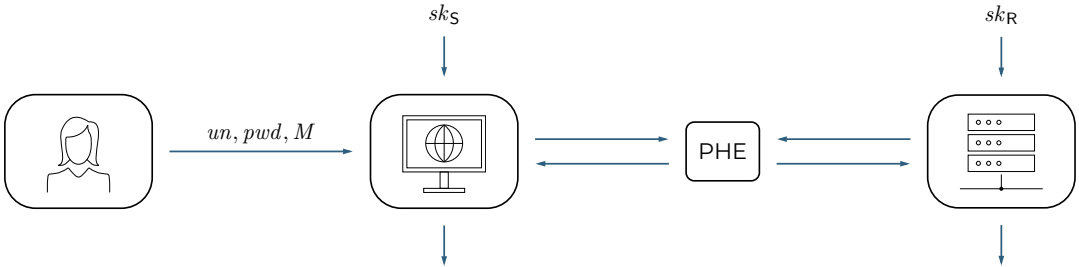
Standard PHE



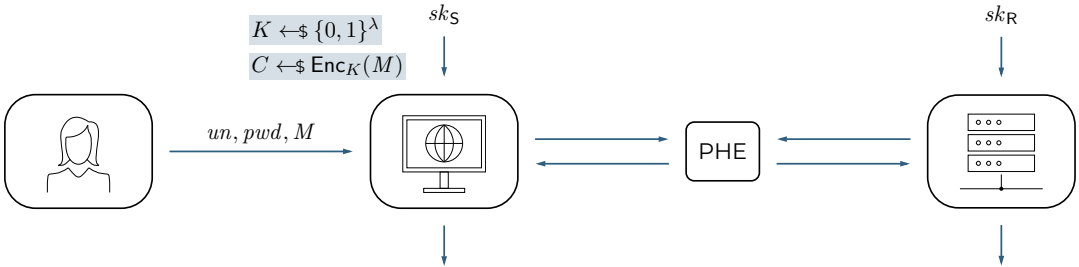
Standard PHE



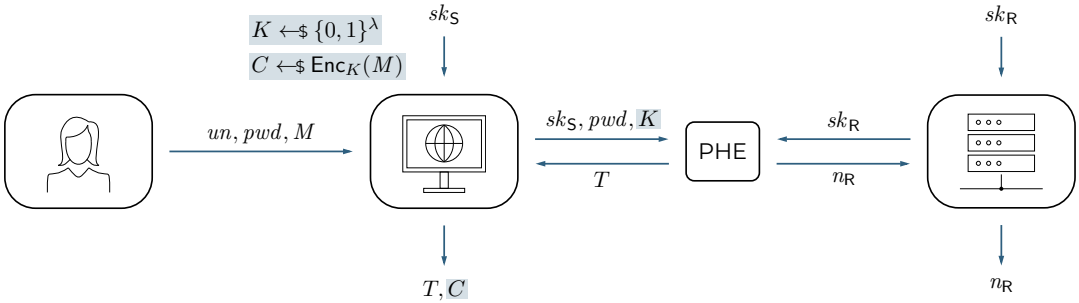
How PHE is used **in practice**



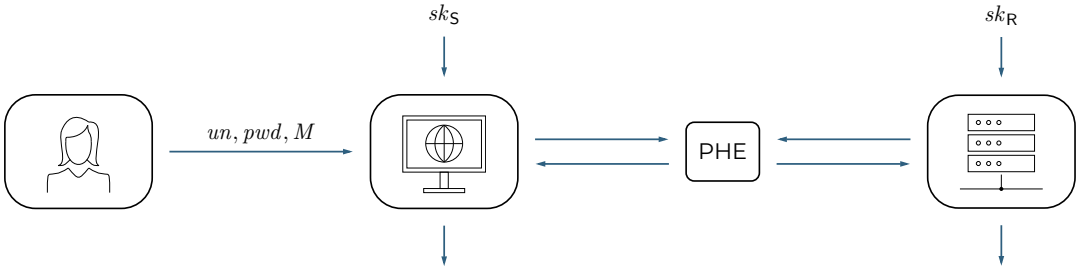
How PHE is used **in practice**



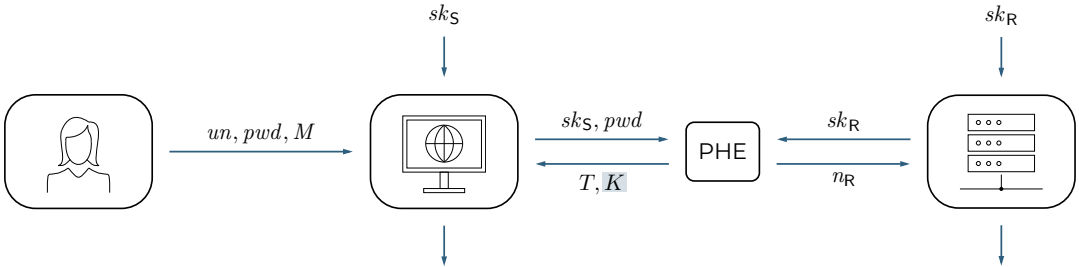
How PHE is used in practice



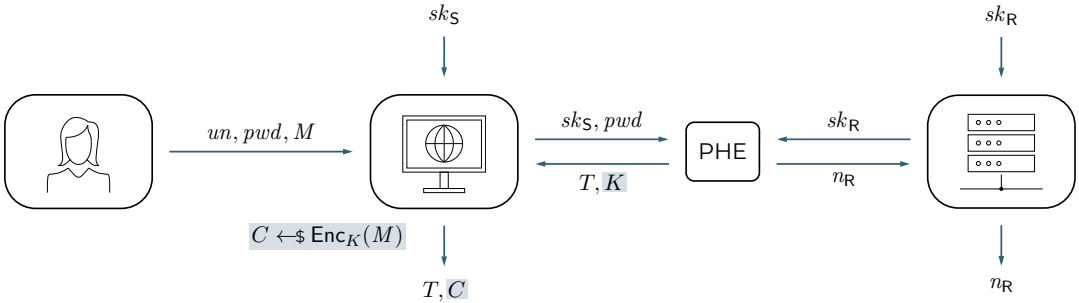
Encapsulation-style PHE



Encapsulation-style PHE



Encapsulation-style PHE



Soundness vs. Binding

Soundness (LERCMS'18)

- $\mathcal{A}$ has **two-time** oracle access
- $\mathcal{A}$ wins if **different** $M \neq M'$ are returned for the **same** pwd
- or if the **same** M is returned for **different** $pwd \neq pwd'$

Binding (our work)

- $\mathcal{A}$ has **arbitrary** oracle access
- $\mathcal{A}$ wins if **different** $M \neq M'$ are returned for the **same** pwd

Soundness vs. Binding

Soundness (LERCMS'18)

- $\mathcal{A}$ has **two-time** oracle access
- $\mathcal{A}$ wins if **different** $M \neq M'$ are returned for the **same** pwd
- or if the **same** M is returned for **different** $pwd \neq pwd'$

Binding (our work)

- $\mathcal{A}$ has **arbitrary** oracle access
- $\mathcal{A}$ wins if **different** $M \neq M'$ are returned for the **same** pwd

Soundness vs. Binding

Soundness (LERCMS'18)

- $\mathcal{A}$ has **two-time** oracle access
- $\mathcal{A}$ wins if **different** $M \neq M'$ are returned for the **same** pwd
- or if the **same** M is returned for **different** $pwd \neq pwd'$

Binding (our work)

- $\mathcal{A}$ has **arbitrary** oracle access
- $\mathcal{A}$ wins if **different** $M \neq M'$ are returned for the **same** pwd

Why Binding suffices

- Correct password:
  - Correct key: **Intended** behaviour
  - Incorrect key: Prevented by **binding**
  - No key: **Trivially** achievable
    (also not covered by other soundness definitions)
- Incorrect password:
  - Correct key: Prevented by **hiding**
  - Incorrect key: Prevented by **binding**
  - No key: **Intended** behaviour

Why Binding suffices

- Correct password:
  - Correct key: **Intended** behaviour
  - Incorrect key: Prevented by **binding**
  - No key: **Trivially** achievable
    (also not covered by other soundness definitions)
- Incorrect password:
  - Correct key: Prevented by **hiding**
  - Incorrect key: Prevented by **binding**
  - No key: **Intended** behaviour

Why Binding suffices

- Correct password:
  - Correct key: **Intended** behaviour
  - Incorrect key: Prevented by **binding**
  - No key: **Trivially** achievable
    (also not covered by other soundness definitions)
- Incorrect password:
  - Correct key: Prevented by **hiding**
  - Incorrect key: Prevented by **binding**
  - No key: **Intended** behaviour

Why Binding suffices

- Correct password:
  - Correct key: **Intended** behaviour
  - Incorrect key: Prevented by **binding**
  - No key: **Trivially** achievable
    (also not covered by other soundness definitions)
- Incorrect password:
  - Correct key: Prevented by **hiding**
  - Incorrect key: Prevented by **binding**
  - No key: **Intended** behaviour

Why Binding suffices

- Correct password:
  - Correct key: **Intended** behaviour
  - Incorrect key: Prevented by **binding**
  - No key: **Trivially** achievable
    (also not covered by other soundness definitions)
- Incorrect password:
  - Correct key: Prevented by **hiding**
  - Incorrect key: Prevented by **binding**
  - No key: **Intended** behaviour

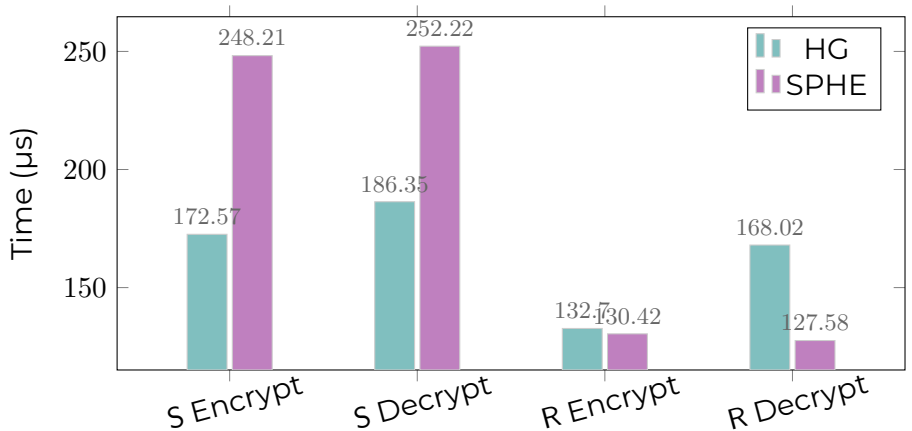
Why Binding suffices

- Correct password:
  - Correct key: **Intended** behaviour
  - Incorrect key: Prevented by **binding**
  - No key: **Trivially** achievable
    (also not covered by other soundness definitions)
- Incorrect password:
  - Correct key: Prevented by **hiding**
  - Incorrect key: Prevented by **binding**
  - No key: **Intended** behaviour

Performance Evaluation

Implemented in Rust using curve25519_dalek

Tested on MacBook Pro (M3 Pro with 36GB unified memory)



Thanks for your attention!

ruben-baecker.de

Password-Hardened Encryption Revisited
Ruben Baecker, Paul Gerhart, and Dominique Schröder
ia.cr/2025/1453

Universally Composable Password-Hardened Encryption
Behzad Abdolmaleki, Ruben Baecker, Paul Gerhart, Mike Graf, Mojtaba Khalili, Daniel Rausch, and Dominique Schröder
ia.cr/2025/1647

HildeGUARD: The Decryption Protocol

A slightly simplified enrollment record

$$T = \mathsf{Enc}_{sk_\mathsf{S}}(t_0, t_1, n)$$
$$t_0 = F^0_{sk_\mathsf{R}}(n) \cdot \mathsf{H}(pwd, n)$$
$$t_1 = F^1_{sk_\mathsf{R}}(n) \cdot K$$

Decryption:

- S obtains $F^0_{sk_\mathsf{R}}(n)$ from t_0 using pwd; R recomputes $F^0_{sk_\mathsf{R}}(n)$
- Perform a PAKE using $F^0_{sk_\mathsf{R}}(n)$ as the password
- R uses session key to encrypt $F^1_{sk_\mathsf{R}}(n)$; S uses session key to decrypt $F^1_{sk_\mathsf{R}}(n)$
- S uses $F^1_{sk_\mathsf{R}}(n)$ to obtain K from t_1

HildeGUARD: The Decryption Protocol

A slightly simplified enrollment record

$$T = \mathsf{Enc}_{sk_\mathsf{S}}(t_0, t_1, n)$$
$$t_0 = F^0_{sk_\mathsf{R}}(n) \cdot \mathsf{H}(pwd, n)$$
$$t_1 = F^1_{sk_\mathsf{R}}(n) \cdot K$$

Decryption:

- S obtains $F^0_{sk_\mathsf{R}}(n)$ from t_0 using pwd; R recomputes $F^0_{sk_\mathsf{R}}(n)$
- Perform a PAKE using $F^0_{sk_\mathsf{R}}(n)$ as the password
- R uses session key to encrypt $F^1_{sk_\mathsf{R}}(n)$; S uses session key to decrypt $F^1_{sk_\mathsf{R}}(n)$
- S uses $F^1_{sk_\mathsf{R}}(n)$ to obtain K from t_1

HildeGUARD: The Decryption Protocol

A slightly simplified enrollment record

$$T = \mathsf{Enc}_{sk_\mathsf{S}}(t_0, t_1, n)$$
$$t_0 = F^0_{sk_\mathsf{R}}(n) \cdot \mathsf{H}(pwd, n)$$
$$t_1 = F^1_{sk_\mathsf{R}}(n) \cdot K$$

Decryption:

- S obtains $F^0_{sk_\mathsf{R}}(n)$ from t_0 using pwd; R recomputes $F^0_{sk_\mathsf{R}}(n)$
- Perform a PAKE using $F^0_{sk_\mathsf{R}}(n)$ as the password
- R uses session key to encrypt $F^1_{sk_\mathsf{R}}(n)$; S uses session key to decrypt $F^1_{sk_\mathsf{R}}(n)$
- S uses $F^1_{sk_\mathsf{R}}(n)$ to obtain K from t_1

HildeGUARD: The Decryption Protocol

A slightly simplified enrollment record

$$T = \mathsf{Enc}_{sk_\mathsf{S}}(t_0, t_1, n)$$
$$t_0 = F^0_{sk_\mathsf{R}}(n) \cdot \mathsf{H}(pwd, n)$$
$$t_1 = F^1_{sk_\mathsf{R}}(n) \cdot K$$

Decryption:

- S obtains $F^0_{sk_\mathsf{R}}(n)$ from t_0 using pwd; R recomputes $F^0_{sk_\mathsf{R}}(n)$
- Perform a PAKE using $F^0_{sk_\mathsf{R}}(n)$ as the password
- R uses session key to encrypt $F^1_{sk_\mathsf{R}}(n)$; S uses session key to decrypt $F^1_{sk_\mathsf{R}}(n)$
- S uses $F^1_{sk_\mathsf{R}}(n)$ to obtain K from t_1

HildeGUARD: The Decryption Protocol

A slightly simplified enrollment record

$$T = \mathsf{Enc}_{sk_\mathsf{S}}(t_0, t_1, n)$$
$$t_0 = F^0_{sk_\mathsf{R}}(n) \cdot \mathsf{H}(pwd, n)$$
$$t_1 = F^1_{sk_\mathsf{R}}(n) \cdot K$$

Decryption:

- S obtains $F^0_{sk_\mathsf{R}}(n)$ from t_0 using pwd; R recomputes $F^0_{sk_\mathsf{R}}(n)$
- Perform a PAKE using $F^0_{sk_\mathsf{R}}(n)$ as the password
- R uses session key to encrypt $F^1_{sk_\mathsf{R}}(n)$; S uses session key to decrypt $F^1_{sk_\mathsf{R}}(n)$
- S uses $F^1_{sk_\mathsf{R}}(n)$ to obtain K from t_1