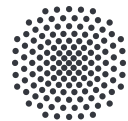
Universally Composable Password-Hardened Encryption

Behzad Abdolmaleki, Ruben Baecker, Paul Gerhart, Mike Graf, Mojtaba Khalili, Daniel Rausch, and Dominique Schröder



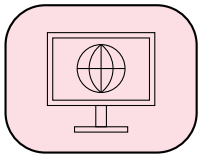
Why Threshold PHE?

Let’s add UC security…

…and find a (round-optimal) instantiation

A Single Point of Failure

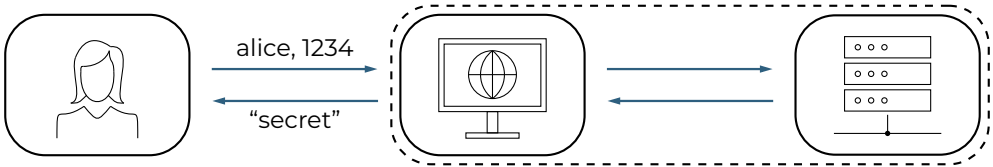
$\mathsf{H}(\text{“1234”}, 2181) = b481$



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

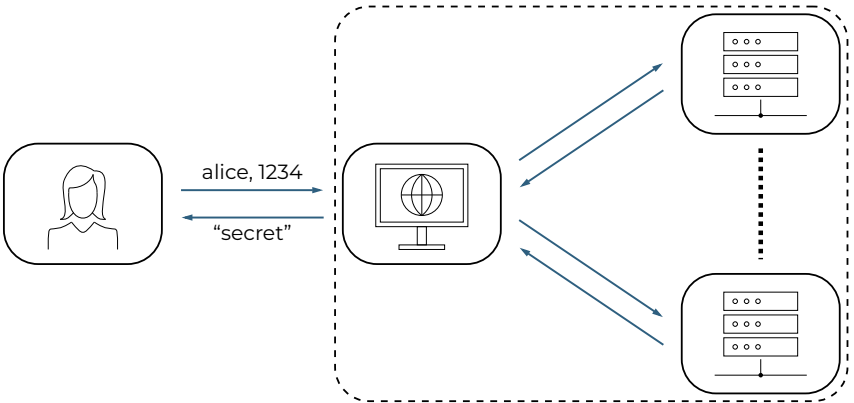
PHE: Distributing Trust

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



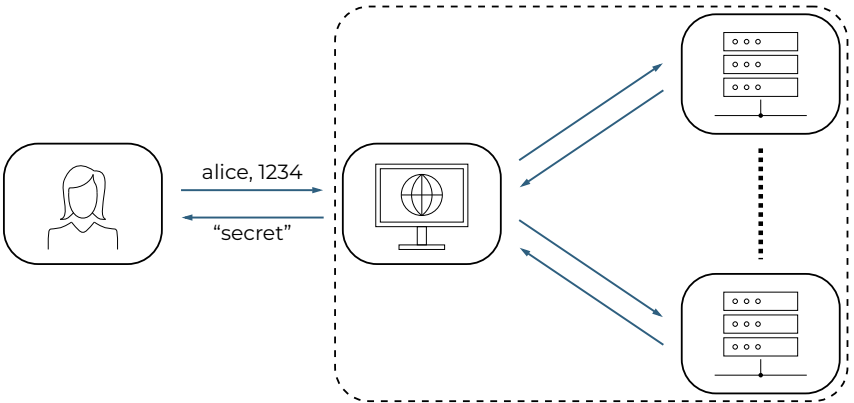
TPHE: Distributing Trust

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



TPHE: Distributing Trust even more

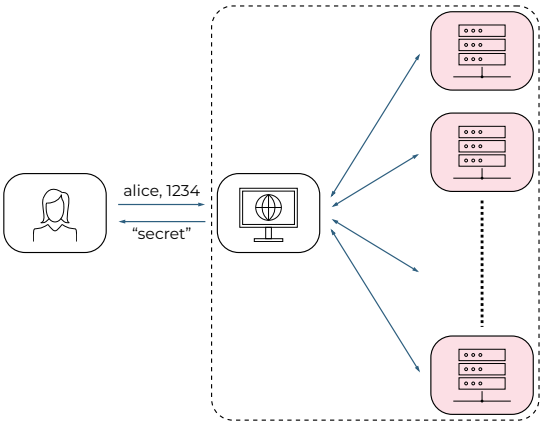
ECSJR'15 (USENIX), SFSB'16 (CCS), LESC'17 (USENIX), LERCMS'18 (USENIX), BELSSZ'20 (CCS)



TPHE: Security Against Corruption

Security against corruption of

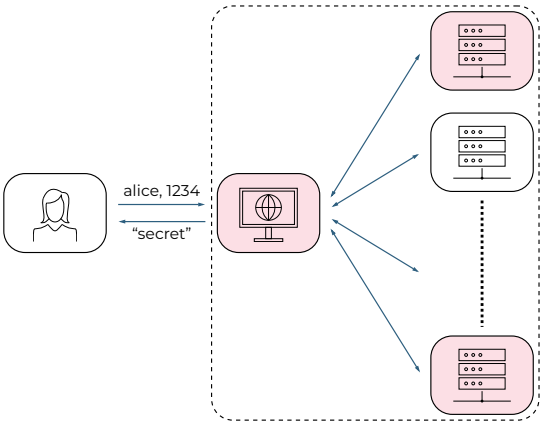
- **All** n ratelimiters
- The server **and** $t - 1$ ratelimiters



TPHE: Security Against Corruption

Security against corruption of

- **All** n ratelimiters
- The server **and** $t-1$ ratelimiters



Why Threshold PHE?

Let's add UC security...

...and find a (round-optimal) instantiation

Game-based vs Universally Composable

- PH(E) has a history of insufficient security models:
  LESC'17 discovered a gap in SFSB'16, and **B**GS'25 in LERCMS'18
- PH(E) is used in the context of complex eco-systems
  $\rightarrow$ standalone security is insufficient
- Human behaviour is not modeled in game-based definitions
  (typos, password re-usage, using personal information)

Game-based vs Universally Composable

- PH(E) has a history of insufficient security models:
  LESC'17 discovered a gap in SFSB'16, and **B**GS'25 in LERCMS'18
- PH(E) is used in the context of complex eco-systems
  $\rightarrow$ standalone security is insufficient
- Human behaviour is not modeled in game-based definitions
  (typos, password re-usage, using personal information)

Game-based vs Universally Composable

- PH(E) has a history of insufficient security models:
  LESC'17 discovered a gap in SFSB'16, and **B**GS'25 in LERCMS'18
- PH(E) is used in the context of complex eco-systems
  $\rightarrow$ standalone security is insufficient
- Human behaviour is not modeled in game-based definitions
  (typos, password re-usage, using personal information)

Game-based vs Universally Composable

- PH(E) has a history of insufficient security models:
  LESC'17 discovered a gap in SFSB'16, and **B**GS'25 in LERCMS'18
- PH(E) is used in the context of complex eco-systems
  $\rightarrow$ standalone security is insufficient
- Human behaviour is not modeled in game-based definitions
  (typos, password re-usage, using personal information)

$\rightarrow$ A security definition in the UC Framework avoids all those problems

Internal State of $\mathcal{F}_{\text{TPHE}}$

- storage(id)

  Maps id to a password, message pair (pw, m)

- retrieveRate$(\mathcal{R}_i, id)$

  Tracks the remaining quota of $\mathcal{R}_i$ for ciphertext stored at position id

Internal State of $\mathcal{F}_{\text{TPHE}}$

- storage(id)
  Maps id to a password, message pair (pw, m)
- retrieveRate$(\mathcal{R}_i, id)$
  Tracks the remaining quota of $\mathcal{R}_i$ for ciphertext stored at position id

Functional Interfaces of $\mathcal{F}_{\text{TPHE}}$

- $(\texttt{Store}, id, pw, m)$ from $\mathcal{S}$
  - Store (pw, m) in storage(id)
  - Leak $(\texttt{Store}, id, |m|)$ to $\mathcal{A}$
- $(\texttt{Retrieve}, id, pw')$ from $\mathcal{S}$
  - Store request with unique $rqid$
  - Leak $(\texttt{Retrieve}, id, rqid)$ to $\mathcal{A}$
- $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_i$
  - Increment retrieveRate$(\mathcal{R}_i, id)$
  - Notify $\mathcal{A}$

Functional Interfaces of $\mathcal{F}_{\text{TPHE}}$

- $(\texttt{Store}, id, pw, m)$ from $\mathcal{S}$
  - Store (pw, m) in storage(id)
  - Leak $(\texttt{Store}, id, |m|)$ to $\mathcal{A}$
- $(\texttt{Retrieve}, id, pw')$ from $\mathcal{S}$
  - Store request with unique $rqid$
  - Leak $(\texttt{Retrieve}, id, rqid)$ to $\mathcal{A}$
- $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_i$
  - Increment retrieveRate$(\mathcal{R}_i, id)$
  - Notify $\mathcal{A}$

Functional Interfaces of $\mathcal{F}_{\text{TPHE}}$

- $(\texttt{Store}, id, pw, m)$ from $\mathcal{S}$
  - Store (pw, m) in storage(id)
  - Leak $(\texttt{Store}, id, |m|)$ to $\mathcal{A}$
- $(\texttt{Retrieve}, id, pw')$ from $\mathcal{S}$
  - Store request with unique $rqid$
  - Leak $(\texttt{Retrieve}, id, rqid)$ to $\mathcal{A}$
- $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_i$
  - Increment retrieveRate$(\mathcal{R}_i, id)$
  - Notify $\mathcal{A}$

Adversarial Interfaces of $\mathcal{F}_{\text{TPHE}}$

- $(\texttt{FinishRetrieve}, rqid)$ from $\mathcal{A}$
  - Retrieve $(\texttt{Retrieve}, id, pw')$ from queue
  - Let $\mathcal{A}$ choose $RLset$ and ensure $\text{retrieveRate}(\mathcal{R}_i, id) \geq 1 \; \forall \mathcal{R}_i \in RLset$ and $|RLset| + n_c \geq \mathsf{t}$
  - Decrement $\text{retrieveRate}(\mathcal{R}_i, id) \; \forall \mathcal{R}_i \in RLset$
  - If $pw' == pw$, return $(\texttt{FinishRetrieve}, m)$ to $\mathcal{S}$; otherwise, return $(\texttt{FinishRetrieve}, \perp)$
- $(\text{ChangeCorruption}, P, corrupt)$ from $\mathcal{A}$
  - Set corruption status of P to $corrupt$
  - If $P = \mathcal{R}_i$ and $corrupt = \perp$, reset $\text{retrieveRate}(\mathcal{R}_i, id) \leftarrow 0 \quad \forall id \in \mathbb{N}$

Adversarial Interfaces of $\mathcal{F}_{\text{TPHE}}$

- $(\texttt{FinishRetrieve}, rqid)$ from $\mathcal{A}$
  - Retrieve $(\texttt{Retrieve}, id, pw')$ from queue
  - Let $\mathcal{A}$ choose $RLset$ and ensure $\text{retrieveRate}(\mathcal{R}_i, id) \geq 1 \; \forall \mathcal{R}_i \in RLset$ and $|RLset| + n_c \geq \mathsf{t}$
  - Decrement $\text{retrieveRate}(\mathcal{R}_i, id) \; \forall \mathcal{R}_i \in RLset$
  - If $pw' == pw$, return $(\texttt{FinishRetrieve}, m)$ to $\mathcal{S}$; otherwise, return $(\texttt{FinishRetrieve}, \perp)$
- $(\text{ChangeCorruption}, P, corrupt)$ from $\mathcal{A}$
  - Set corruption status of P to $corrupt$
  - If $P = \mathcal{R}_i$ and $corrupt = \perp$, reset $\text{retrieveRate}(\mathcal{R}_i, id) \leftarrow 0 \quad \forall id \in \mathbb{N}$

Adversarial Interfaces of $\mathcal{F}_{\text{TPHE}}$

- $(\texttt{PwGuessStart}, id)$ from $\mathcal{A}$
  - Ensure $\text{storage}(id) \neq \perp$
  - Ensure $\mathcal{S}$ is corrupt and perform same ratelimiting checks as for `FinishRetrieve`, including decrementing retrieveRate
  - Store new password guess token for the ciphertext stored under id
- $(\texttt{PwGuessFinish}, id, pw_{\mathcal{A}})$ from $\mathcal{A}$
  - Check that a password guess token for the ciphertext stored under id exists and delete it
  - Let (pw, m) be the entry of $\text{storage}(id)$
  - If $pw == pw_{\mathcal{A}}$, return $(\texttt{PwGuessFinish}, m)$ to $\mathcal{A}$; otherwise, return $(\texttt{PwGuessFinish}, \perp)$

Adversarial Interfaces of $\mathcal{F}_{\text{TPHE}}$

- $(\texttt{PwGuessStart}, id)$ from $\mathcal{A}$
  - Ensure $\text{storage}(id) \neq \perp$
  - Ensure $\mathcal{S}$ is corrupt and perform same ratelimiting checks as for `FinishRetrieve`, including decrementing retrieveRate
  - Store new password guess token for the ciphertext stored under id
- $(\texttt{PwGuessFinish}, id, pw_{\mathcal{A}})$ from $\mathcal{A}$
  - Check that a password guess token for the ciphertext stored under id exists and delete it
  - Let (pw, m) be the entry of $\text{storage}(id)$
  - If $pw == pw_{\mathcal{A}}$, return $(\texttt{PwGuessFinish}, m)$ to $\mathcal{A}$; otherwise, return $(\texttt{PwGuessFinish}, \perp)$

The Need for `PwGuessStart`

Assume 2-out-of-3 TPHE and the following calls to $\mathcal{F}_{\text{TPHE}}$

1. $(\texttt{Retrieve}, id, pw)$ from $\mathcal{S}$
2. $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_1$
3. $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_3$
4. $(\text{ChangeCorruption}, \mathcal{R}_i, \bot)$ from $\mathcal{A} \quad \forall i \in [3]$
   $\rightarrow$ resets all retrieveRate to 0
5. $(\texttt{FinishRetrieve}, rqid)$ from $\mathcal{A}$
   fails **unintendetly** because of insufficient decryption quota

The Need for `PwGuessStart`

Assume 2-out-of-3 TPHE and the following calls to $\mathcal{F}_{\text{TPHE}}$

1. $(\texttt{Retrieve}, id, pw)$ from $\mathcal{S}$
2. $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_1$
3. $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_3$
4. $(\text{ChangeCorruption}, \mathcal{R}_i, \bot)$ from $\mathcal{A} \quad \forall i \in [3]$
   $\rightarrow$ resets all retrieveRate to 0
5. $(\texttt{FinishRetrieve}, rqid)$ from $\mathcal{A}$
   fails **unintendetly** because of insufficient decryption quota

The Need for `PwGuessStart`

Assume 2-out-of-3 TPHE and the following calls to $\mathcal{F}_{\text{TPHE}}$

1. $(\texttt{Retrieve}, id, pw)$ from $\mathcal{S}$
2. $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_1$
3. $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_3$
4. $(\text{ChangeCorruption}, \mathcal{R}_i, \bot)$ from $\mathcal{A} \quad \forall i \in [3]$
   $\rightarrow$ resets all retrieveRate to 0
5. $(\texttt{FinishRetrieve}, rqid)$ from $\mathcal{A}$
   fails **unintendetly** because of insufficient decryption quota

The Need for `PwGuessStart`

Assume 2-out-of-3 TPHE and the following calls to $\mathcal{F}_{\text{TPHE}}$

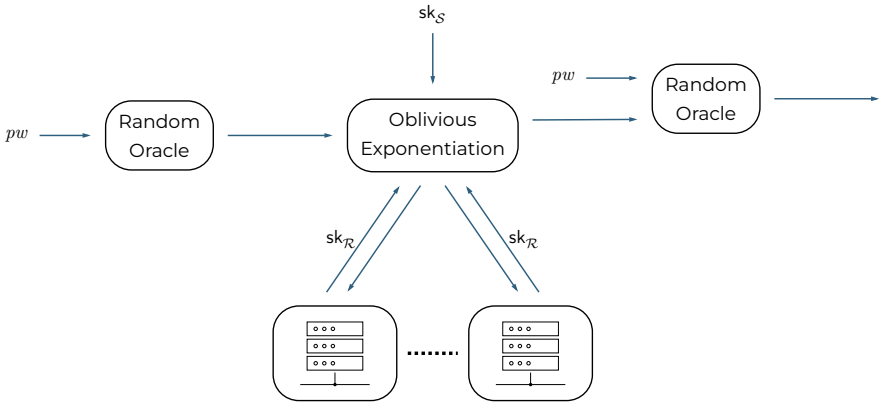
1. $(\texttt{Retrieve}, id, pw)$ from $\mathcal{S}$
2. $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_1$
3. $(\texttt{HelpRetrieve}, id)$ from $\mathcal{R}_3$
4. $(\text{ChangeCorruption}, \mathcal{R}_i, \bot)$ from $\mathcal{A} \quad \forall i \in [3]$
   $\rightarrow$ resets all retrieveRate to 0
5. $(\texttt{FinishRetrieve}, rqid)$ from $\mathcal{A}$
   fails **unintendetly** because of insufficient decryption quota

Why Threshold PHE?

Let’s add UC security…

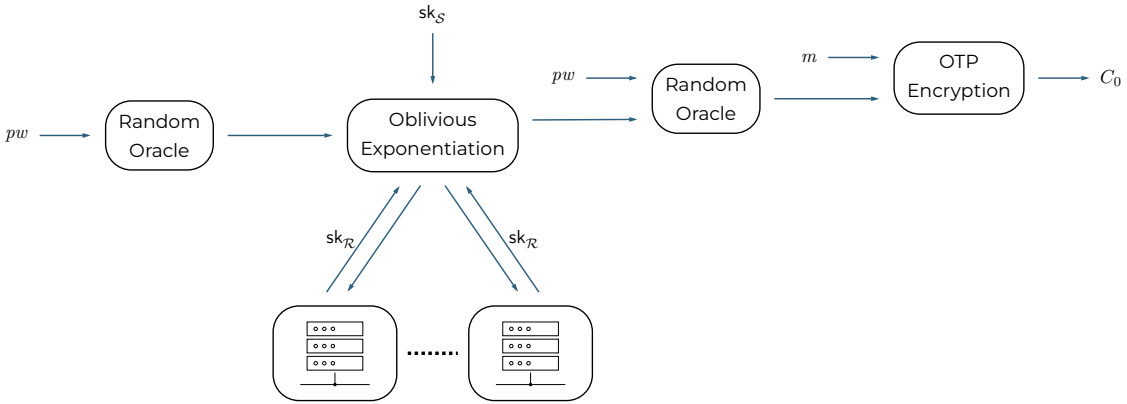
…and find a (round-optimal) instantiation

UCPY: A **Composable** TPHE Protocol



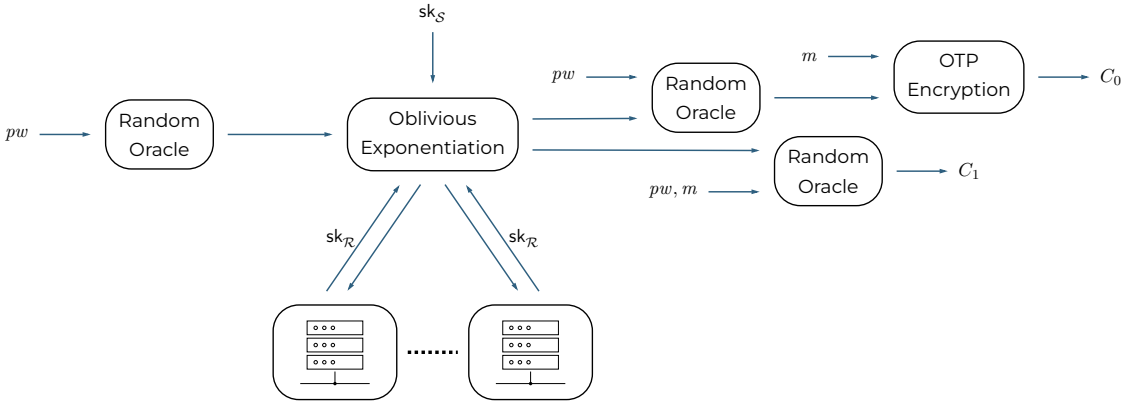
The Oblivious Exponentiation part is similar to **B**GRS'25 with a different secret sharing structure

UCPY: A **Composable** TPHE Protocol



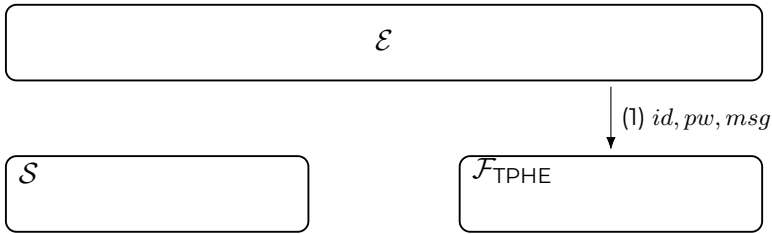
The Oblivious Exponentiation part is similar to **B**GRS'25 with a different secret sharing structure

UCPY: A **Composable** TPHE Protocol

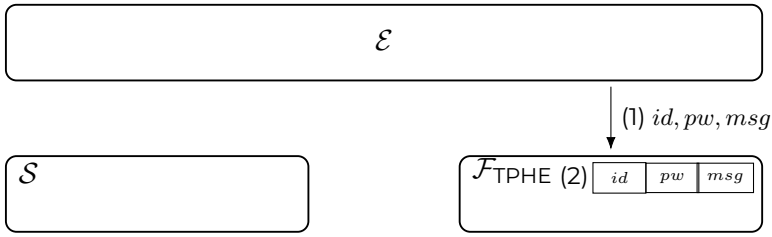


The Oblivious Exponentiation part is similar to **B**GRS'25 with a different secret sharing structure

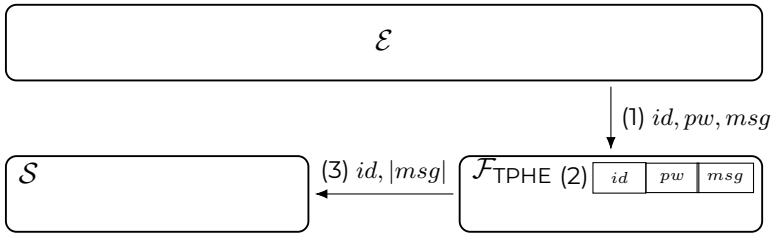
UCPY: Simulating **Encryption**



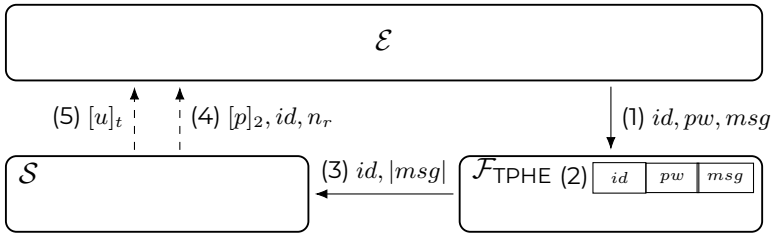
UCPY: Simulating **Encryption**



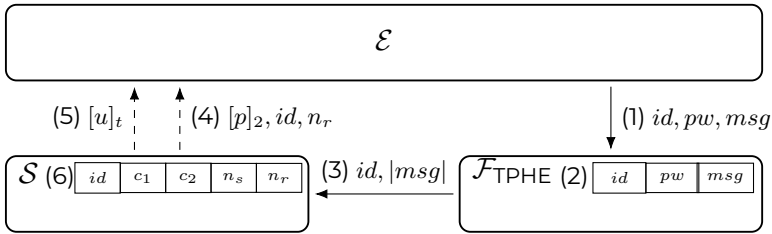
UCPY: Simulating **Encryption**



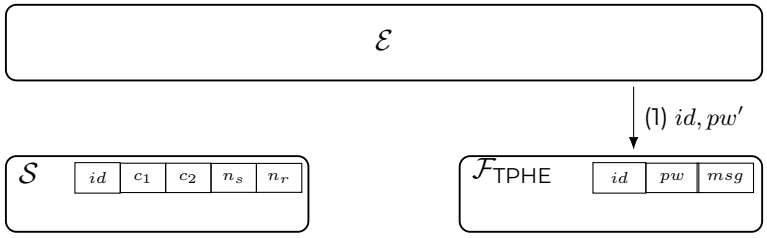
UCPY: Simulating **Encryption**



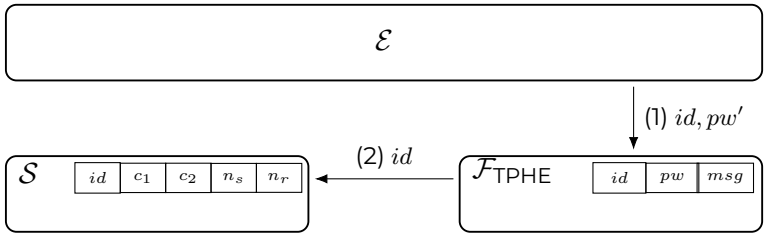
UCPY: Simulating **Encryption**



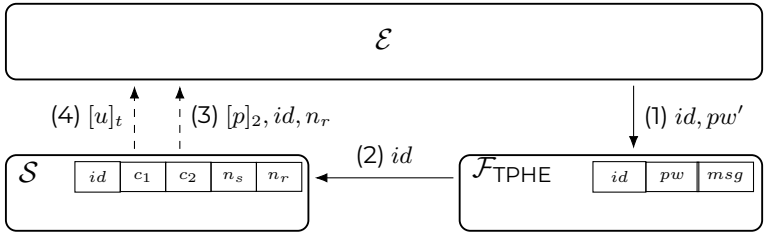
UCPY: Simulating **Decryption**



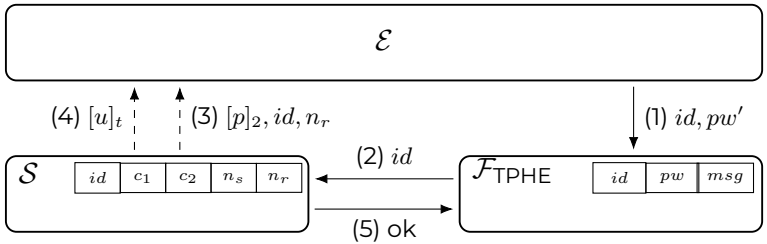
UCPY: Simulating **Decryption**



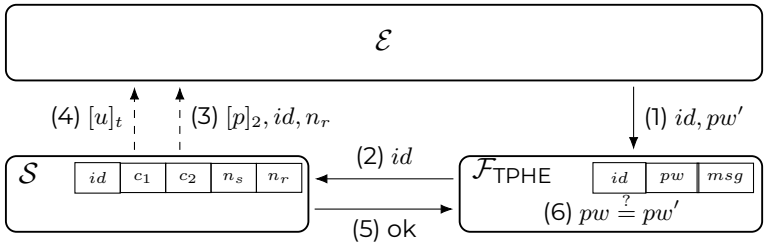
UCPY: Simulating **Decryption**



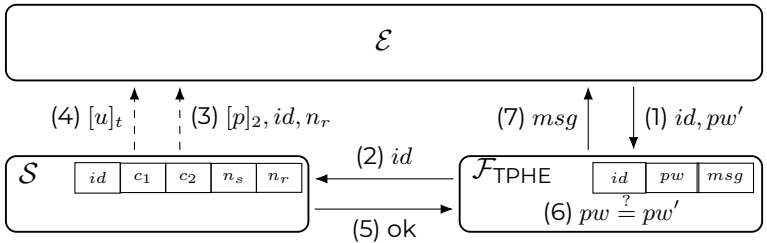
UCPY: Simulating **Decryption**



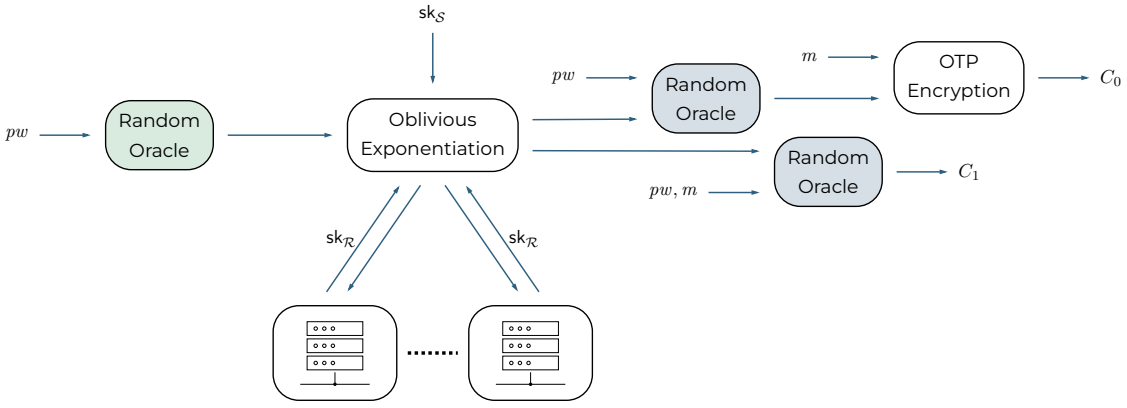
UCPY: Simulating **Decryption**



UCPY: Simulating **Decryption**



UCPY: Simulating **Random Oracles**



1. $\forall id \in \mathsf{storage}^{\mathsf{Real}} : (c_1^{\mathsf{Real}}, c_2^{\mathsf{Real}}, n^{\mathsf{Real}}) \sim (c_1^{\mathsf{Sim}}, c_2^{\mathsf{Sim}}, n^{\mathsf{Sim}}) \wedge [$
   $[\exists i \in \mathbb{N}\ s.t.\ (id \in \mathsf{correctMessageIDs}^{\mathsf{Ideal}} \implies \mathsf{storageHistory}^{\mathsf{Ideal}}[id, i+1] = \bot)$
   $\wedge\ \mathsf{H}_{\mathsf{OTP}}^{\mathsf{Real}}(\mathsf{sk}^{\mathsf{Real}} \cdot \mathsf{H}_1^{\mathsf{Real}}(pw^{\mathsf{Ideal}}, n^{\mathsf{Real}}) \cdot \mathsf{H}_2^{\mathsf{Real}}(id, n^{\mathsf{Real}}), pw^{\mathsf{Ideal}}, id, n^{\mathsf{Real}}) = c_1^{\mathsf{Real}} \oplus m^{\mathsf{Ideal}}$
   $\wedge\ \mathsf{H}_{\mathsf{MAC}}^{\mathsf{Real}}(\mathsf{sk}^{\mathsf{Real}} \cdot \mathsf{H}_1^{\mathsf{Real}}(pw^{\mathsf{Ideal}}, n^{\mathsf{Real}}) \cdot \mathsf{H}_2^{\mathsf{Real}}(id, n^{\mathsf{Real}}), m^{\mathsf{Ideal}}, pw^{\mathsf{Ideal}}, id, n^{\mathsf{Real}}) = c_2^{\mathsf{Real}}$
   $\wedge\ \mathsf{H}_{\mathsf{OTP}}^{\mathsf{Sim}}(\mathsf{sk}^{\mathsf{Sim}} \cdot \mathsf{H}_1^{\mathsf{Sim}}(pw^{\mathsf{Ideal}}, n^{\mathsf{Sim}}) \cdot \mathsf{H}_2^{\mathsf{Sim}}(id, n^{\mathsf{Sim}}), pw^{\mathsf{Ideal}}, id, n^{\mathsf{Sim}}) \in \{c_1^{\mathsf{Sim}} \oplus m^{\mathsf{Ideal}}, \bot\}$
   $\wedge\ \mathsf{H}_{\mathsf{MAC}}^{\mathsf{Sim}}(\mathsf{sk}^{\mathsf{Sim}} \cdot \mathsf{H}_1^{\mathsf{Sim}}(pw^{\mathsf{Ideal}}, n^{\mathsf{Sim}}) \cdot \mathsf{H}_2^{\mathsf{Sim}}(id, n^{\mathsf{Sim}}), m^{\mathsf{Ideal}}, pw^{\mathsf{Ideal}}, id, n^{\mathsf{Sim}}) \in \{c_2^{\mathsf{Sim}}, \bot\}]$
   $\vee [\exists (id^{\mathsf{Sim}}, n^{\mathsf{Sim}}, u, n^{\mathsf{Sim}}, pw^{\mathsf{Sim}}) \in \mathsf{injectedMessages}^{\mathsf{Sim}}]\ s.t.\ id = id^{\mathsf{Sim}}$
   $\wedge\ c_1^{\mathsf{Real}} = \mathsf{H}_{\mathsf{OTP}}^{\mathsf{Real}}(\mathsf{sk}^{\mathsf{Real}} \cdot \mathsf{H}_1^{\mathsf{Real}}(pw^{\mathsf{Sim}}, n^{\mathsf{Real}}) \cdot \mathsf{H}_2^{\mathsf{Real}}(id, n^{\mathsf{Real}}), pw^{\mathsf{Sim}}, id, n^{\mathsf{Real}}) \oplus m^{\mathsf{Sim}}$
   $\wedge\ c_2^{\mathsf{Real}} = \mathsf{H}_{\mathsf{MAC}}^{\mathsf{Real}}(\mathsf{sk}^{\mathsf{Real}} \cdot \mathsf{H}_1^{\mathsf{Real}}(pw^{\mathsf{Sim}}, n^{\mathsf{Real}}) \cdot \mathsf{H}_2^{\mathsf{Real}}(id, n^{\mathsf{Real}}), m^{\mathsf{Sim}}, pw^{\mathsf{Sim}}, id, n^{\mathsf{Real}})$
   $\wedge\ c_1^{\mathsf{Sim}} = \mathsf{H}_{\mathsf{OTP}}^{\mathsf{Sim}}(\mathsf{sk}^{\mathsf{Sim}} \cdot \mathsf{H}_1^{\mathsf{Sim}}(pw^{\mathsf{Sim}}, n^{\mathsf{Sim}}) \cdot \mathsf{H}_2^{\mathsf{Sim}}(id, n^{\mathsf{Sim}}), pw^{\mathsf{Sim}}, id, n^{\mathsf{Sim}}) \oplus m^{\mathsf{Sim}}$
   $\wedge\ c_2^{\mathsf{Sim}} = \mathsf{H}_{\mathsf{MAC}}^{\mathsf{Sim}}(\mathsf{sk}^{\mathsf{Sim}} \cdot \mathsf{H}_1^{\mathsf{Sim}}(pw^{\mathsf{Sim}}, n^{\mathsf{Sim}}) \cdot \mathsf{H}_2^{\mathsf{Sim}}(id, n^{\mathsf{Sim}}), m^{\mathsf{Sim}}, pw^{\mathsf{Sim}}, id, n^{\mathsf{Sim}})]$
   $\vee [\tilde{Z}(pw, m)\ s.t.$
   $c_1^{\mathsf{Real}} = \mathsf{H}_{\mathsf{OTP}}^{\mathsf{Real}}(\mathsf{sk}^{\mathsf{Real}} \cdot \mathsf{H}_1^{\mathsf{Real}}(pw, n^{\mathsf{Real}}) \cdot \mathsf{H}_2^{\mathsf{Real}}(id, n^{\mathsf{Real}}), pw, id, n^{\mathsf{Real}}) \oplus m$
   $\wedge\ c_2^{\mathsf{Real}} = \mathsf{H}_{\mathsf{MAC}}^{\mathsf{Real}}(\mathsf{sk}^{\mathsf{Real}} \cdot \mathsf{H}_1^{\mathsf{Real}}(pw, n^{\mathsf{Real}}) \cdot \mathsf{H}_2^{\mathsf{Real}}(id, n^{\mathsf{Real}}), m, pw, id, n^{\mathsf{Real}})$
   $\wedge\ c_1^{\mathsf{Sim}} = \mathsf{H}_{\mathsf{OTP}}^{\mathsf{Sim}}(\mathsf{sk}^{\mathsf{Sim}} \cdot \mathsf{H}_1^{\mathsf{Sim}}(pw, n^{\mathsf{Sim}}) \cdot \mathsf{H}_2^{\mathsf{Sim}}(id, n^{\mathsf{Sim}}), pw, id, n^{\mathsf{Sim}}) \oplus m$
   $\wedge\ c_2^{\mathsf{Sim}} = \mathsf{H}_{\mathsf{MAC}}^{\mathsf{Sim}}(\mathsf{sk}^{\mathsf{Sim}} \cdot \mathsf{H}_1^{\mathsf{Sim}}(pw, n^{\mathsf{Sim}}) \cdot \mathsf{H}_2^{\mathsf{Sim}}(id, n^{\mathsf{Sim}}), m, pw, id, n^{\mathsf{Sim}})]]$
   with $\mathsf{storage}^{\mathsf{Real}}[id] = (c_1^{\mathsf{Real}}, c_2^{\mathsf{Real}}, n^{\mathsf{Real}}),$
   $\mathsf{storage}^{\mathsf{Sim}}[id] = (c_1^{\mathsf{Sim}}, c_2^{\mathsf{Sim}}, n^{\mathsf{Sim}}),$
   $\mathsf{storageHistory}^{\mathsf{Ideal}}[id, i] = (pw^{\mathsf{Ideal}}, m^{\mathsf{Ideal}})$
2. $\forall id \in \mathsf{reqQueue}_{\mathsf{Enc}}^{\mathsf{Real}} : (pw^{\mathsf{Real}}, m^{\mathsf{Real}}, T^{\mathsf{Real}}) = (pw^{\mathsf{Ideal}}, m^{\mathsf{Ideal}}, T^{\mathsf{Sim}}) \wedge |m^{\mathsf{Real}}| = |m|^{\mathsf{Sim}} = |m^{\mathsf{Ideal}}|$
   $\wedge^i i^{\mathsf{Sim}} = i^{\mathsf{Ideal}} \wedge n^{\mathsf{Real}} \sim n^{\mathsf{Sim}}$
   with $\mathsf{reqQueue}_{\mathsf{Enc}}^{\mathsf{Real}}[id] = (pw^{\mathsf{Real}}, m^{\mathsf{Real}}, r, n^{\mathsf{Real}}, T^{\mathsf{Real}}),$
   $\mathsf{reqQueue}_{\mathsf{Enc}}^{\mathsf{Sim}}[id] = (i^{\mathsf{Sim}}, |m|^{\mathsf{Sim}}, n^{\mathsf{Real}}, T^{\mathsf{Real}}, p)$
   $\mathsf{storageHistory}^{\mathsf{Ideal}}[id, i^{\mathsf{Ideal}}| = (pw^{\mathsf{Ideal}}, m^{\mathsf{Ideal}})$ with i^{Ideal} max
3. $\forall \mathsf{retrieveCounter} \in \mathsf{reqQueue}_{\mathsf{Dec}}^{\mathsf{Real}} : \exists\ \mathsf{retrieveCounter}' \in \mathsf{reqQueue}_{\mathsf{Dec}}^{\mathsf{Ideal}}\ s.t.\ (pw'^{\mathsf{Real}}, T'^{\mathsf{Real}}) = (pw'^{\mathsf{Ideal}}, T'^{\mathsf{Sim}})$
   $\wedge (id^{\mathsf{Real}}, caller^{\mathsf{Real}}) = (id^{\mathsf{Sim}}, caller^{\mathsf{Sim}}) = (id^{\mathsf{Ideal}}, caller^{\mathsf{Ideal}}) \wedge (n^{\mathsf{Real}}, p^{\mathsf{Real}}) \sim (n^{\mathsf{Sim}}, p^{\mathsf{Sim}})$
   with $\mathsf{reqQueue}_{\mathsf{Dec}}^{\mathsf{Real}}[\mathsf{retrieveCounter}] = (id^{\mathsf{Real}}, pw^{\mathsf{Real}}, r, n^{\mathsf{Real}}, T^{\mathsf{Real}}, p^{\mathsf{Real}}, caller^{\mathsf{Real}}),$
   $\mathsf{reqQueue}_{\mathsf{Dec}}^{\mathsf{Sim}}[\mathsf{retrieveCounter}] = (id^{\mathsf{Sim}}, n^{\mathsf{Sim}}, T^{\mathsf{Sim}}, p^{\mathsf{Sim}}, caller^{\mathsf{Sim}})$
   $\mathsf{reqQueue}_{\mathsf{Dec}}^{\mathsf{Ideal}}[\mathsf{retrieveCounter}'] = (id^{\mathsf{Ideal}}, \mathsf{storageHistory}[id, i]^{\mathsf{Ideal}}, pw'^{\mathsf{Ideal}}, caller^{\mathsf{Ideal}})$
4. $\forall id \in \mathsf{retrieveRate}^{\mathsf{Real}}, rl \notin \mathsf{currentlyCorrupted} : \mathsf{retrieveRate}_{rl}^{\mathsf{Real}}[id] = \mathsf{retrieveRate}_{rl}^{\mathsf{Sim}}[id] \leq$
   $\leq \mathsf{retrieveRate}_{rl}^{\mathsf{Ideal}}[id] - |\{(rl, \_, id, \_, \_)\} \subseteq \mathsf{retrieveRequests}^{\mathsf{Sim}}|$
5. $\forall id, i \in \mathsf{storageHistory}^{\mathsf{Sim}} : \mathsf{H}_{\mathsf{OTP}}^{\mathsf{Sim}}(\mathsf{sk}^{\mathsf{Sim}} \cdot \mathsf{H}_1^{\mathsf{Sim}}(pw^{\mathsf{Ideal}}, n^{\mathsf{Sim}}) \cdot \mathsf{H}_2^{\mathsf{Sim}}(id, n^{\mathsf{Sim}}), pw^{\mathsf{Ideal}}, id, n^{\mathsf{Sim}}) \in \{c_1^{\mathsf{Sim}} \oplus m^{\mathsf{Ideal}}, \bot\}$
   $\wedge \mathsf{H}_{\mathsf{MAC}}^{\mathsf{Sim}}(\mathsf{sk}^{\mathsf{Sim}} \cdot \mathsf{H}_1^{\mathsf{Sim}}(pw^{\mathsf{Ideal}}, n^{\mathsf{Sim}}) \cdot \mathsf{H}_2^{\mathsf{Sim}}(id, n^{\mathsf{Sim}}), m^{\mathsf{Ideal}}, pw^{\mathsf{Ideal}}, id, n^{\mathsf{Sim}}) \in \{c_2^{\mathsf{Sim}}, \bot\}]$
   with $\mathsf{storageHistory}^{\mathsf{Sim}}[id, i] = (c_1^{\mathsf{Sim}}, c_2^{\mathsf{Sim}}, n^{\mathsf{Sim}}),$
   $\mathsf{storageHistory}^{\mathsf{Ideal}}[id, i] = (pw^{\mathsf{Ideal}}, m^{\mathsf{Ideal}})$

Fig. 27: Invariants for the notion of synchronization between the ideal and real world

Why Threshold PHE?

Let’s add UC security…

…and find a (round-optimal) instantiation

Why Threshold PHE?

Let’s add UC security...

...and find a (round-optimal) instantiation

We also found a gap in the proof of BELSSZ’20

A Gap in BELSSZ'20

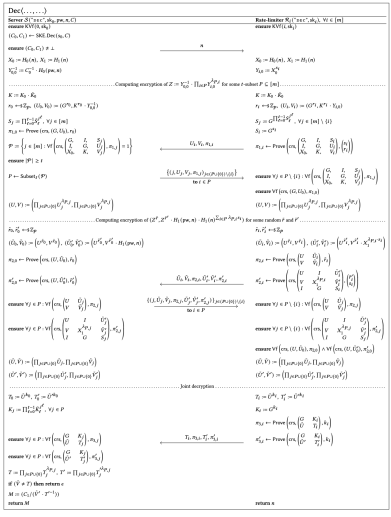


Figure 4: Decryption Protocol (Procedures for fine-grained rate-limiting in Figure 8)

A Gap in BELSSZ'20

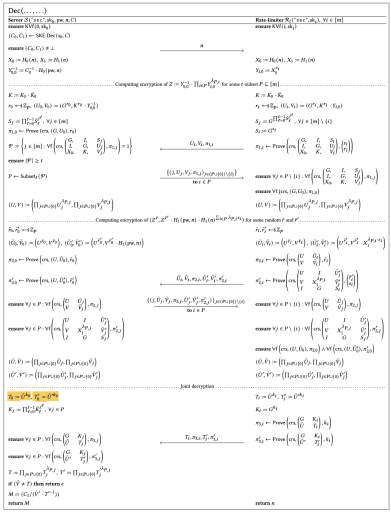


Figure 4: Decryption Protocol (Procedures for fine-grained rate-limiting in Figure 8)

A Gap in BELSSZ'20

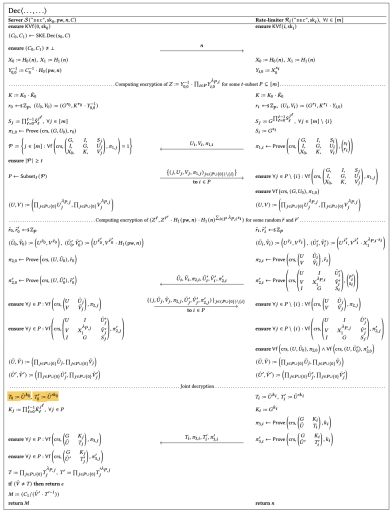


Figure 4: Decryption Protocol (Procedures for fine-grained rate-limiting in Figure 8)

$$T_0 := \tilde{U}^{k_0}, \; T_0' := \tilde{U}'^{k_0}$$

A Gap in BELSSZ'20

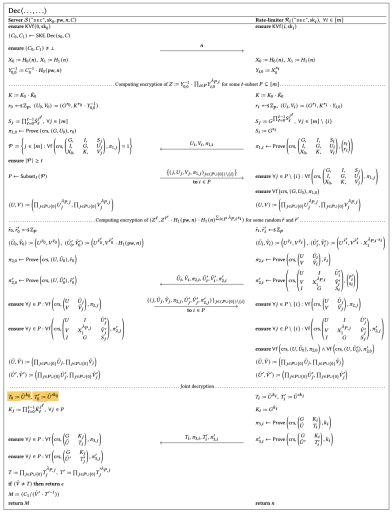


Figure 4: Decryption Protocol (Procedures for fine-grained rate-limiting in Figure 8)

$$T_0 := \tilde{U}^{k_0}, \quad T_0' := \tilde{U}'^{k_0}$$

$\mathcal{B}$ computes the views of all parties honestly except for the values U_0, V_0, T_0 and T_0'. For the q-th query, $\mathcal{B}$ sets $U_0 := G^{\tilde{\beta}}$ and $V_0 := G^{\gamma} \cdot G^{\beta \tilde{k}_0} \cdot Y_{0,0}^{-1}$. For other queries, $\mathcal{B}$ computes U_0 and V_0 honestly. For all queries, to compute T_0 and T_0', $\mathcal{B}$ runs the extractor of the NIZKPoK to extract the discrete logarithm $\tilde{u}$ and $\tilde{u}'$ such that $\tilde{U} = G^{\tilde{u}}$ and $\tilde{U}' = G^{\tilde{u}'}$. It then compute $T_0 := G^{\alpha \tilde{u}}$ and $T_0' := G^{\alpha \tilde{u}'}$.

Clearly, if $(G, G^{\alpha}, G^{\beta}, G^{\gamma})$ is a DH tuple, $\mathcal{B}$ simulates $\mathsf{Hyb}_{b,3,q-1}$ perfectly. Else, if $(G, G^{\alpha}, G^{\beta}, G^{\gamma})$ is a random tuple, $\mathcal{B}$ simulates $\mathsf{Hyb}'_{b,3,q}$ perfectly. The claim then follows.

A Gap in BELSSZ'20

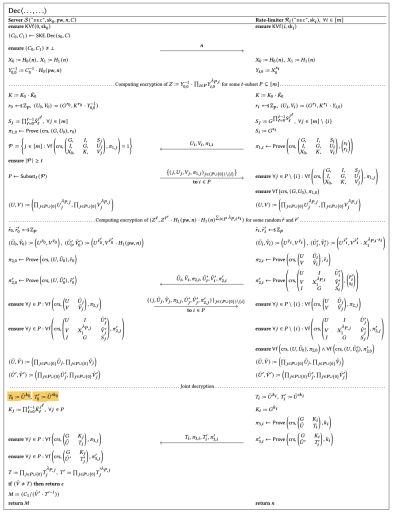


Figure 4: Decryption Protocol (Procedures for fine-grained rate-limiting in Figure 8)

$$T_0 := \tilde{U}^{k_0}, \quad T_0' := \tilde{U}'^{k_0}$$

$\mathcal{B}$ computes the views of all parties honestly except for the values U_0, V_0, T_0 and T_0'. For the q-th query, $\mathcal{B}$ sets $U_0 := G^{\tilde{\beta}}$ and $V_0 := G^{\gamma} \cdot G^{\beta k_0} \cdot Y_{0,0}^{-1}$. For other queries, $\mathcal{B}$ computes U_0 and V_0 honestly. For all queries, to compute T_0 and T_0', $\mathcal{B}$ runs the extractor of the NIZKPoK to extract the discrete logarithm $\tilde{u}$ and $\tilde{u}'$ such that $\tilde{U} = G^{\tilde{u}}$ and $\tilde{U}' = G^{\tilde{u}'}$. It then compute $T_0 := G^{\alpha\tilde{u}}$ and $T_0' := G^{\alpha\tilde{u}'}$.

Clearly, if $(G, G^{\alpha}, G^{\beta}, G^{\gamma})$ is a DH tuple, $\mathcal{B}$ simulates $\mathrm{Hyb}_{b,3,q-1}$ perfectly. Else, if $(G, G^{\alpha}, G^{\beta}, G^{\gamma})$ is a random tuple, $\mathcal{B}$ simulates $\mathrm{Hyb}'_{b,3,q}$ perfectly. The claim then follows.

A Gap in BELSSZ'20

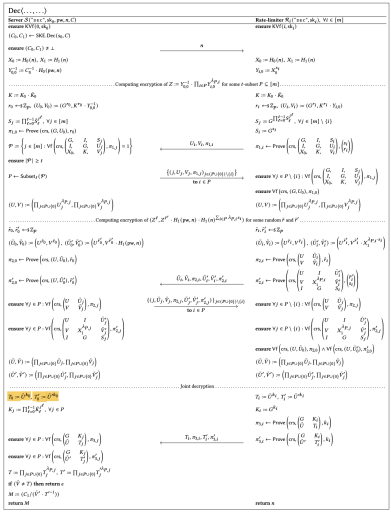


Figure 4: Decryption Protocol (Procedures for fine-grained rate-limiting in Figure 8)

$$T_0 := \tilde{U}^{k_0}, \quad T_0' := \tilde{U}'^{k_0}$$

$\mathcal{B}$ computes the views of all parties honestly except for the values U_0, V_0, T_0 and T_0'. For the q-th query, $\mathcal{B}$ sets $U_0 := G^{\hat{\beta}}$ and $V_0 := G^{\gamma} \cdot G^{\beta k_0} \cdot Y_{0,0}^{-1}$. For other queries, $\mathcal{B}$ computes U_0 and V_0 honestly. For all queries, to compute T_0 and T_0', $\mathcal{B}$ runs the extractor of the NIZKPoK to extract the discrete logarithm $\tilde{u}$ and $\tilde{u}'$ such that $\tilde{U} = G^{\tilde{u}}$ and $\tilde{U}' = G^{\tilde{u}'}$. It then compute $T_0 := G^{\alpha\tilde{u}}$ and $T_0' := G^{\alpha\tilde{u}'}$.

Clearly, if $(G, G^{\alpha}, G^{\beta}, G^{\gamma})$ is a DH tuple, $\mathcal{B}$ simulates $\mathrm{Hyb}_{b,3,q-1}$ perfectly. Else, if $(G, G^{\alpha}, G^{\beta}, G^{\gamma})$ is a random tuple, $\mathcal{B}$ simulates $\mathrm{Hyb}'_{b,3,q}$ perfectly. The claim then follows.

- $\mathrm{Hyb}_{b,1}$ is mostly identical to $\mathrm{Hyb}_{b,0}$, except that all zero-knowledge proofs are simulated by running the simulator of the NIZKPoK scheme. It it straightforward to show that, for all $b \in \{0,1\}$,

$$\left| \Pr\left[\mathrm{Hyb}_{b,0} = 1\right] - \Pr\left[\mathrm{Hyb}_{b,1} = 1\right] \right| \leq \mathrm{negl}(\lambda)$$

using the zero-knowledge property of the NIZKPoK scheme.

A Gap in BELSSZ'20

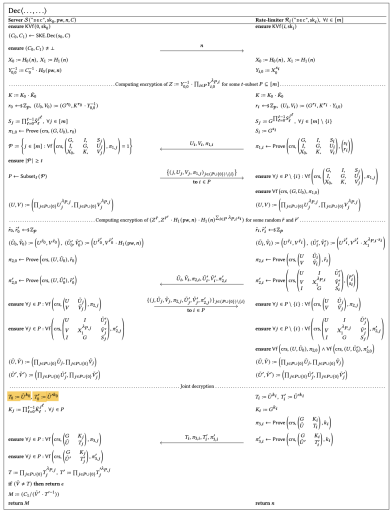


Figure 4: Decryption Protocol (Procedures for fine-grained rate-limiting in Figure 8)

$$T_0 := \tilde{U}^{k_0}, \; T_0' := \tilde{U}'^{k_0}$$

$\mathcal{B}$ computes the views of all parties honestly except for the values U_0, V_0, T_0 and T_0'. For the q-th query, $\mathcal{B}$ sets $U_0 := G^{\hat{\beta}}$ and $V_0 := G^{\gamma} \cdot G^{\beta k_0} \cdot Y_{0,0}^{-1}$. For other queries, $\mathcal{B}$ computes U_0 and V_0 honestly. For all queries, to compute T_0 and T_0', $\mathcal{B}$ runs the extractor of the NIZKPoK to extract the discrete logarithm $\tilde{u}$ and $\tilde{u}'$ such that $\tilde{U} = G^{\tilde{u}}$ and $\tilde{U}' = G^{\tilde{u}'}$. It then compute $T_0 := G^{\alpha\tilde{u}}$ and $T_0' := G^{\alpha\tilde{u}'}$.

Clearly, if $(G, G^{\alpha}, G^{\beta}, G^{\gamma})$ is a DH tuple, $\mathcal{B}$ simulates $\mathsf{Hyb}_{b,3,q-1}$ perfectly. Else, if $(G, G^{\alpha}, G^{\beta}, G^{\gamma})$ is a random tuple, $\mathcal{B}$ simulates $\mathsf{Hyb}'_{b,3,q}$ perfectly. The claim then follows.

- $\mathsf{Hyb}_{b,1}$ is mostly identical to $\mathsf{Hyb}_{b,0}$, except that all zero-knowledge proofs are simulated by running the simulator of the NIZKPoK scheme. It it straightforward to show that, for all $b \in \{0,1\}$,

$$\left| \Pr\left[\mathsf{Hyb}_{b,0} = 1\right] - \Pr\left[\mathsf{Hyb}_{b,1} = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

using the zero-knowledge property of the NIZKPoK scheme.

A Gap in BELSSZ'20

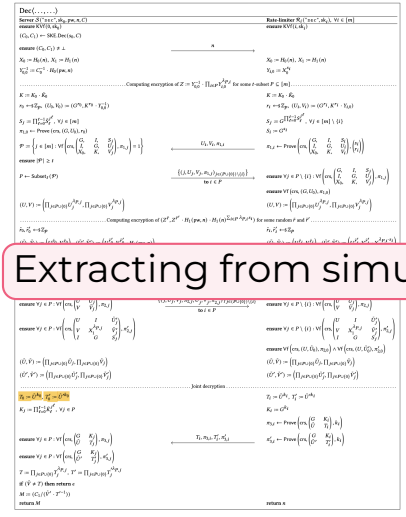


Figure 4: Decryption Protocol (Procedures for fine-grained rate-limiting in Figure 8)

$$T_0 := \tilde{U}^{k_0}, \; T_0' := \tilde{U}'^{k_0}$$

$\mathcal{B}$ computes the views of all parties honestly except for the values U_0, V_0, T_0 and T_0'. For the q-th query, $\mathcal{B}$ sets $U_0 := G^{\hat{\beta}}$ and $V_0 := G^{\gamma} \cdot G^{\beta \hat{k}_0} \cdot Y_{0,0}^{-1}$. For other queries, $\mathcal{B}$ computes U_0 and V_0 honestly. For all queries, to compute T_0 and T_0', $\mathcal{B}$ runs the extractor of the NIZKPoK to extract the discrete logarithm $\tilde{u}$ and $\tilde{u}'$ such that $\tilde{U} = G^{\tilde{u}}$ and $\tilde{U}' = G^{\tilde{u}'}$. It then compute $T_0 := G^{\alpha\tilde{u}}$ and $T_0' := G^{\alpha\tilde{u}'}$.

Extracting from simulated proofs is impossible!

- $\mathrm{Hyb}_{b,1}$ is mostly identical to $\mathrm{Hyb}_{b,0}$, except that all zero-knowledge proofs are simulated by running the simulator of the NIZKPoK scheme. It it straightforward to show that, for all $b \in \{0,1\}$,

$$\left| \Pr\left[\mathrm{Hyb}_{b,0} = 1\right] - \Pr\left[\mathrm{Hyb}_{b,1} = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

using the zero-knowledge property of the NIZKPoK scheme.

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

UCPY: A Hybrid Protocol