Robust PUF-Based Multi-Factor Authentication: Resist Machine Learning Attacks and Support Account Recovery and Reset (Appendix File)

Liufu Zhu, Peizhen Hong and Ding Wang

Abstract—This appendix file consists of three parts. Appendix A shows the formal security analysis of our scheme under the ROM model, Appendix B presents the UC security analysis of our scheme and Appendix C provides an informal security analysis.

Index Terms—Appendix file.

APPENDIX A: GAME-BASED ANALYSIS UNDER THE ROM

Theorem 1. Assume that there is a probabilistic polynomial-time adversary $\mathcal{A}$ that executes q_{Ext} times of Extract(θ_U, θ_{GW}, θ_S), q_S times of Send(θ_U, θ_{GW}, θ_S, M), q_{H_i} times of $H_i(\cdot)_{i\in[1,3]}$, and q_{enc} authenticated encryption queries. Then $\text{Adv}_{\mathcal{A}} \leq C^{'} q_{Send}^{\eta^{'}} + \sum_{1\leq i\leq 3} \frac{q_{H_i}^2}{2^{l_i+1}} + \frac{(q_{Send}+q_{Ext})^2}{2p} + \frac{q_{\text{ae}}^2}{2^{l_{ae}}} + \frac{q_{Ext}^2}{2p} + \frac{N\lambda_{asw_i}}{G_{asw_i}} + \sum_{1\leq i\leq 3} \frac{q_{Send}+q_{Ext}}{2^{l_i}} + \frac{\varepsilon_{cdh}}{q_{H_3}} + \varepsilon_{ae} + \varepsilon_{puf}$, where C', η', λ_{asw_i} and G_{asw_i} are constants related to the password and secret questions; l_i is the length of $H_i(\cdot)_{i\in[1,3]}$; l_{enc} is the length of the ciphertext; and ε_{cdh} denotes the advantage in solving the CDH problem; ε_{ae} denotes the advantage in breaking the authenticated encryption scheme; ε_{puf} denotes the advantage in modeling PUFs' responses.

Security Model. We evaluate the security of the proposed scheme in the random oracle model, where all hash functions are modeled as publicly accessible oracles. Let θ_U, θ_{GW}, and θ_S denote the communication instances of the user, the authentication gateway, and the server, respectively. We assume a probabilistic polynomial-time adversary $\mathcal{A}$ who has full control over the network and is equipped with a set of oracles $\Delta = \{\Omega_U, \Omega_S, \Omega_{GW}\}$ to launch the following queries.

(1) **Extract($\theta_U, \theta_{GW}, \theta_S$):** This query models passive eavesdropping, where $\mathcal{A}$ can intercept and record all messages transmitted over the public channel.
(2) **Send($\theta_U, \theta_{GW}, \theta_S, M$):** This query captures active attacks, where $\mathcal{A}$ can inject a crafted message M to any instance and obtain the corresponding response.
(3) **Reveal(θ_U, θ_S):** If the user instance θ_U and the server instance θ_S have successfully established a session key sk, this oracle returns sk to $\mathcal{A}$; otherwise, it outputs $\perp$.
(4) **Corrupt(θ_U, b):** This oracle simulates the compromise of authentication factors, allowing $\mathcal{A}$ to obtain any two of the three factors, but not all three simultaneously. It also captures secret question guessing attacks during the account recovery phase.
   - **Corrupt($\theta_U, 1$):** $\mathcal{A}$ obtains PW and BIO.
   - **Corrupt($\theta_U, 2$):** $\mathcal{A}$ obtains PW and PUF.
   - **Corrupt($\theta_U, 3$):** $\mathcal{A}$ obtains PUF and BIO.
(5) **Hash($\cdot$):** This oracle returns a random value for each unique query, and all queries are recorded in lists $List_{h_i}$ to maintain consistency.
(6) **Test1(θ_U, θ_S):** It captures the semantic security of the session key. A fair coin b is tossed; if $b = 1$, the real sk is returned; if $b = 0$, a random string sk' is returned. It requires that Corrupt(θ_U, b) and Reveal(θ_U, θ_S) have not been issued to the target instances.
(7) **Test2(θ_U, θ_{GW}):** This oracle challenges the security of the account recovery and reset process. A fair coin c is tossed; if $c = 1$, the actual $PW||BIO||C_a$ is returned; if $c = 0$, random strings $PW'||BIO'||C_a^{'}$ are returned. It requires that Corrupt(θ_U, b) and Reveal(θ_U, θ_{GW}) have not been issued to the target instances.

Definition 1. If the adversary $\mathcal{A}$ can perform $Execute(\Delta)$, $Send(\Delta, M)$, $Corrupt(\Delta, b)$, $Hash(\cdot)$, $Test1(\Delta)$ and $Test2(\Delta)$ queries in polynomial time under the ROM model, then it can break the session key security of the proposed protocol with advantage $Adv_{\mathcal{A}}$ as follows:

$$Adv_{\mathcal{A}} \leq \sum_{1\leq i\leq 3} \frac{q_{H_i}^2}{2^{l_i+1}} + \frac{(q_{Send}+q_{Ext})^2}{2p} + \frac{q_{\text{ae}}^2}{2^{l_{ae}}} + \frac{q_{Ext}^2}{2p} + \frac{N\lambda_{asw_i}}{G_{asw_i}}$$

Manuscript received 9 August 2026. This work was supported by the National Natural Science Foundation of China under Grant No.62572259, and by the Natural Science Foundation of Tianjin, China under Grant No.25JCJQJC00230, and by the Fundamental Research Funds for the Central Universities, Nankai University under Grant No.63263258.

Liufu Zhu and Ding Wang are with the College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China, with the State Key Laboratory of Cryptology, Beijing 100878, China, with Key Laboratory of Data and Intelligent System Security, Nankai University, Ministry of Education, Tianjin 300350, China, with the Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China; also with the Beijing Key Laboratory of Post-Quantum Cryptography, Beijing 100083, China. Peizhen Hong is with the College of Electronics Information and Optical Engineering, Nankai University, Tianjin 300350, China.

$$+\sum_{1\le i\le 3}\frac{q_{Send}+q_{Ext}}{2^{l_i}}+\frac{\varepsilon_{cdh}}{q_{H_3}}+\varepsilon_{ae}+C^{'}q_{Send}^{\eta^{'}}+\varepsilon_{puf}.$$

Proof. A sequence of games G_i is defined to prove the security of our scheme. $\mathcal{A}$ wins the games if and only if she guesses b in each game $Game_i$ successfully. The proof is presented as follows.

$Game_1$: This game simulates the real word attacks without performing any queries under the ROM model. Therefore, the advantage of $\mathcal{A}$ breaking the scheme is $Adv_{\mathcal{A}} = |2Pr[win_{\mathcal{A}}^{G_1}]-1|$.

$Game_2$: This game is identical to G_1 except that hash lists $List_{h_i}$ ($i \in [1,3]$) have been created to record the requests and responses. Specifically, if a value x has been queried before and a tuple (x,y) has been recorded in the list $List_{h_i}$, it will return the response y directly. Otherwise, it will choose y at random, record the tuple (x,y) into the list and then return y to $\mathcal{A}$ as a response. We have that $Pr[win_{\mathcal{A}}^{G_2}] = Pr[win_{\mathcal{A}}^{G_1}]$.

$Game_3$: This game is identical to G_2 except that it will terminate immediately if a collision exists in the $Hash(\cdot)$ queries, or a collision exists in transcripts, or a collision exists in the authenticated encryption ciphertext. The probability of collision in $Hash(\cdot)$ queries is at most $\frac{q_{H_i}^2}{2^{l_i+1}}$, the probability of collision in transcripts is at most $\frac{(q_{Send}+q_{Ext})^2}{2p}$, and the probability of the collision in the authenticated ciphertext is $\frac{q_{\mathsf{ae}}^2}{2^{l_{ae}}}$. The advantage is

$$Pr[win_{\mathcal{A}}^{G_3}]-Pr[win_{\mathcal{A}}^{G_2}] \le \sum_{1\le i\le 3}\frac{q_{H_i}^2}{2^{l_i+1}}+\frac{(q_{Send}+q_{Ext})^2}{2p}+\frac{q_{\mathsf{ae}}^2}{2^{l_{ae}}}.$$

$Game_4$: This game is identical to G_3 except that it can perform $Execute(\Delta)$ queries. However, only the publicly transmitted message can be provided to $\mathcal{A}$. Notice that, due to random values r_{us} and r_{su}, it is difficult to guess b efficiently. The advantage of $\mathcal{A}$ to break the scheme is the same as G_3, that is, $|Pr[win_{\mathcal{A}}^{G_4}]-Pr[win_{\mathcal{A}}^{G_3}]| \le \frac{q_{Ext}^2}{2p}+\varepsilon_{ae}$.

$Game_5$: It is the same as G_4, except that it will abort if $\mathcal{A}$ guesses the correct hash values without querying the random oracle $H_i(\cdot)$ after q_{Ext} and q_{Send} queries to the oracle $Execute(\Delta)$, and $Send(\Delta, M)$. We have $Pr[win_{\mathcal{A}}^{G_5}]-Pr[win_{\mathcal{A}}^{G_4}] \le \sum_{1\le i\le 3}\frac{q_{Send}+q_{Ext}}{2^{l_i}}$.

$Game_6$: This game simulates the malicious behavior where the adversary $\mathcal{A}$ can compromise the secret key of the authentication gateway, which means that $\mathcal{A}$ is allowed to conduct $Corrupt(\Omega_{AGW},1)$ queries to achieve x of $\mathcal{A}$ and attempts to break the forward security (compute the previously established session key). Note that, although in possession of x, it is infeasible for $\mathcal{A}$ to establish the session key sk because r_{su} is random and confidential to $\mathcal{A}$. In this case, if $\mathcal{A}$ can successfully guess the session key, then an adversary $\mathcal{A}_1$ can be constructed to solve the CDH problem in G. Here we present a detailed analysis.

Assume that the adversary $\mathcal{A}_1$ is given an instance of the CDH problem that is (g, g^a, g^b), then $\mathcal{B}_1$ attempts to compute g^{ab}. During the simulation, let $g^{r_{us}} = g^a$, $g^{r_{su}} = g^b$; an adversary $\mathcal{B}_2$ is given a ciphertext (CT_1, σ_1), (CT_2, σ_2) and attempts to recover the plaintext. Since $\mathcal{A}$ has compromised the secret key x of $\mathcal{GW}$, $\mathcal{A}$ can compute the secret $k_{ug} = (g^{r_{ug}})^x$, and decrypts CT_1 to obtain the secret value $g^{r_{us}}$. The adversary $\mathcal{A}$ can obtain $g^{r_{su}}$ through the publicly transmitted message. Now, we suppose that $\mathcal{A}$ can efficiently distinguish the session key sk. That is, given the message $(g, g^{r_{us}} = g^a, g^{r_{su}})$, $\mathcal{A}$ can distinguish the hash value $H(g^{r_{us}} \,||\, g^{r_{su}} \,||\, g^{r_{us}r_{su}} \,||\, HID \,||\, SID)$, which means that $\mathcal{A}$ has queried q_{H_3} the random oracle $H_3(\cdot)$ with request (g^{ab}, g^a, g^b). Therefore, the adversary $\mathcal{A}_1$ can solve the CDH problem using the value $g^{r_{us}r_{su}} = (g^{ab})$. Therefore, in this case, we have $|Pr[win_{\mathcal{A}}^{G_6}] - Pr[win_{\mathcal{A}}^{G_5}]| \le \frac{\varepsilon_{cdh}}{q_{H_3}}$.

$Game_7$: This game is equal to G_6, except that $\mathcal{A}$ attempts to perform password guessing attacks, biometric guessing attacks, and secret question guessing attacks. Since secret questions are implemented during the fallback authentication phase for account recovery and reset, we assume that the adversary $\mathcal{A}$ can compromise only one authentication factor each time. Note that the premise for making secret question guesses is that the email address has been given to $\mathcal{A}$ and $\mathcal{A}$ can obtain Q from the authentication gateway.

Firstly, it is required that if $Corrupt(\Omega_U, 1)$ is queried, $Corrupt(\Omega_U, 2)$ is forbidden to query. Through q_{Ext} and q_{Send} queries of the oracle $Execute(\Delta)$, and $Send(\Delta, M)$, the probability of guessing the correct password is no more than $C^{'}q_{Send}^{\eta^{'}}$, where $C^{'}$ and $\eta^{'}$ are constants related to the size of password space. Secondly, it is required that if $Corrupt(\Omega_U, 2)$ is queried, $Corrupt(\Omega_U, 1)$ is forbidden to query. The advantage of $\mathcal{A}$ guessing the biometrics is $\frac{1}{2^{N_b}}$, where N_b is the entropy that biological features can provide. Finally, if PUFs have been compromised by $\mathcal{A}$ and can infer the generation patterns of challenges and responses by the specific device through a large number of training samples, then it has an advantage ε_{puf} to recover the challenge C_a generated by the $\mathcal{GW}$. We have found through experimental analysis (shown in the main file) that the proposed scheme can resist machine learning attacks, so ε_{puf} is negligible. Thus, we have $|Pr[win_{\mathcal{A}}^{G_7}] - Pr[win_{\mathcal{A}}^{G_6}]| \le C^{'}\eta^S + \frac{1}{2^{N_b}} + \varepsilon_{puf}$.

$Game_8$: This game is the same as G_7. However, it replaces the random oracle H_3 with a self-selected random oracle $H_3^{'}$, and replace the values $g^{r_{us}}, g^{r_{su}}, M_3$ with random values. Due to the randomness of r_{us}, r_{su}, we have that $Pr[win_{\mathcal{A}}^{G_8}] - Pr[win_{\mathcal{A}}^{G_7}] \le \frac{1}{2^p}$.

$Game_9$: This game is the same as G_8. However, $\mathcal{A}$ attempts to recover the $\mathcal{U}$'s account $BIO||PW$ and then tries to build the session key sk. Due to the security of the personal knowledge questions Q, we have that $\mathcal{A}$ can guess the correct answer to each question by advantage no more than $\frac{\lambda_{asw_i}}{G_{asw_i}}$. Therefore, we have that $Pr[win_{\mathcal{A}}^{G_9}] - Pr[win_{\mathcal{A}}^{G_8}] \le \frac{N\lambda_{asw_i}}{G_{asw_i}}$.

From the analysis of G_1 to G_9, the advantage of $\mathcal{A}$ to break the proposed protocol can be summarized as

$$Adv_{\mathcal{A}} \leq \sum_{1\leq i\leq 3} \frac{q_{H_i}^2}{2^{l_i+1}} + \frac{(q_{Send}+q_{Ext})^2}{2p} + \frac{q_{\text{ae}}^2}{2^{l_{ae}}} + \frac{q_{Ext}^2}{2p} + \frac{N\lambda_{asw_i}}{G_{asw_i}}$$

$$+ \sum_{1\leq i\leq 3} \frac{q_{Send}+q_{Ext}}{2^{l_i}} + \frac{\varepsilon_{cdh}}{q_{H_3}} + \varepsilon_{ae} + C^{'} q_{Send}^{\eta^{'}} + \varepsilon_{puf}.$$

APPENDIX B: UC SECURITY ANALYSIS

In this section, we prove the universally composable (UC) security of our scheme (expressed as Π). The ideal functionality $\mathcal{F}_{\Pi}$ is presented in Fig. 1 and Fig. 2; It is parameterized with a security κ, dictionary DX, and values $count_{rec}$, and $count_{ret}$. It stores and manages a database EDB to record the account information, then interacts with the simulator $\mathcal{SIM}$, the user $\mathcal{U}$, the authentication gateway $\mathcal{GW}$, and the server $\mathcal{S}$. We claim the following,

Theorem 2. The proposed protocol Π UC-realizes the functionality $\mathcal{F}_{\Pi}$ under the $\{\mathcal{F}_{\text{ROM}}, \mathcal{F}_{\text{VOPRF}}, \mathcal{F}_{\text{AE}}, \mathcal{F}_{\text{CRS}}\}$-hybrid model.

We construct a simulator $\mathcal{SIM}$ (see Fig. 3, Fig. 4 and Fig. 5) to show that there exists no environment $\mathcal{E}$ can tell whether it interacts with Π and $\mathcal{A}$ in the hybrid model or with $\mathcal{F}_{\Pi}$ and $\mathcal{SIM}$ in the ideal world. Generally speaking, the proof states that the executions in the hybrid model are identical to the executions in the ideal world. We omit the details of interactions with $\mathcal{A}$ and $\mathcal{F}_{\text{ROM}}$, $\mathcal{F}_{\text{VOPRF}}$, $\mathcal{F}_{\text{AE}}$, $\mathcal{F}_{\text{CRS}}$ functionalities, because $\mathcal{SIM}$ can obtain information from $\mathcal{A}$'s interfaces.

$\mathcal{SIM}$ initializes $DX_{\mathcal{SIM}}$ and $EDB_{\mathcal{SIM}}$ to record secret values and account information. We prove that $\mathcal{SIM}$'s behavior ensures that the environment $\mathcal{E}$'s view in the hybrid model is identically distributed with its view in the ideal world. We first discuss the case in which the user $\mathcal{U}$ is corrupted and the server $\mathcal{S}$ is honest, then analyze the case in which the user $\mathcal{U}$ is honest but the server $\mathcal{S}$ is corrupted, and finally we analyze the case where the authentication gateway $\mathcal{GW}$ is corrupted and the user $\mathcal{U}$ and the server $\mathcal{S}$ are honest. Since the user registration is executed through a secure channel, and $\mathcal{SIM}$ can simulate perfectly using the correct factors when the user is corrupted before registration, we only need to analyze the security of the other phases.

Simulation overview. Without loss of generality, we assume a dummy adversary $\mathcal{A}$ who is controlled by the environment $\mathcal{E}$ and merely passes messages to or from $\mathcal{E}$. In the ideal world, $\mathcal{SIM}$ can obtain messages from $\mathcal{A}$'s interfaces to functionalities $\mathcal{F}_{\text{ROM}}$, $\mathcal{F}_{\text{VOPRF}}$ and $\mathcal{F}_{\text{AE}}$. It interacts with the ideal functionality $\mathcal{F}_{\Pi}$, which interacts with the environment $\mathcal{E}$ via the dummy honest party playing the role of the user $\mathcal{U}$, the role of the server $\mathcal{S}$, and the role of the authentication gateway $\mathcal{GW}$. The environment $\mathcal{E}$ can also instruct $\mathcal{A}$ to send inputs on behalf of the corrupt or compromised party. There are two types of interactions between $\mathcal{SIM}$ and $\mathcal{A}$, corresponding to three different interfaces $\mathcal{A}$ has in the real world. First, there is the network interface, i.e., messages that Π sends over plain channels. This interface is used for sending public messages, including $\{N_u, M_1, CT_1, \pi_1, T_1, N_s, M_2, CT_2, \pi_2, T_2, N_{us}, M_3, T_3\}$. Second, $\mathcal{A}$ can communicate with functionalities $\mathcal{F}_{\text{ROM}}$, $\mathcal{F}_{\text{VOPRF}}$, and $\mathcal{F}_{\text{AE}}$, which are emulated by $\mathcal{SIM}$, allowing $\mathcal{SIM}$ to learn the adversarial inputs from the interfaces of these functionalities.

Description of $\mathcal{F}_{\Pi}$. In this section, we provide an overview of $\mathcal{F}_{\Pi}$, including its interactions with party $\mathcal{U}$, $\mathcal{S}$, $\mathcal{GW}$, and $\mathcal{SIM}$.

Init: $\mathcal{F}_{\Pi}$ receives an initialization request from a server $\mathcal{S}$ and $\mathcal{GW}$, it sends $(\text{INIT}, sid, \mathcal{S})$ and $(\text{INIT}, sid, \mathcal{GW})$ to $\mathcal{SIM}$. Then, $\mathcal{SIM}$ simulates an instance of $\mathcal{F}_{\text{CRS}}$, sets $pk_{\mathcal{GW}}$ and $pk_{\mathcal{S}}$ as public parameters.

$\mathcal{U}$ Registration: $\mathcal{F}_{\Pi}$ receives registration request $(\text{REGIS}, sid, \mathcal{GW}, PW, BIO, ID, EML, Q)$ from $\mathcal{U}$, it records the factors PW, ID, BIO, EML, Q in EDB. $\mathcal{F}_{\Pi}$ sends a registration message to $\mathcal{SIM}$. If $\mathcal{U}$ has been corrupted, $\mathcal{SIM}$ can obtain all registration data from $\mathcal{F}_{\Pi}$. Otherwise, $\mathcal{F}_{\Pi}$ only sends the leakage information $L_1(PW, ID, BIO, C_a)$ and $L_2(EML, Q)$ to $\mathcal{SIM}$. If $\mathcal{SIM}$ responds with an error message $(\text{VOPRF}, sid, error)$, $\mathcal{F}_{\Pi}$ return error to $\mathcal{U}$ and aborts. Otherwise, $\mathcal{F}_{\Pi}$ send a registration response $(sid, \text{RESPONSE})$ to $\mathcal{U}$ and records PW in the list $List_{reuse}$.

$\mathcal{U}$ Login and Authentication: $\mathcal{F}_{\Pi}$ receives an authentication request $(\text{LOGAUTHUSER}, sid, PW^{'}, ID^{'}, BIO^{'}, C_a^{'})$ from $\mathcal{U}$ for a potential account. $\mathcal{F}_{\Pi}$ records $(PW^{'}, ID^{'}, BIO^{'}, C_a^{'})$ in DX. If $\mathcal{U}$ is corrupted, $\mathcal{F}_{\Pi}$ send $(\text{AUTHENUSER}, sid, PW^{'}, ID^{'}, BIO^{'}, C_a^{'})$ to $\mathcal{SIM}$. Otherwise, $\mathcal{F}_{\Pi}$ send the leakage message $(\text{AUTHENUSER}, sid, L_3(PW^{'}, ID^{'}, BIO^{'}, C_a^{'}))$ to $\mathcal{SIM}$. Then, $\mathcal{F}_{\Pi}$ generates authentication credentials $(sid, AuthUrToAgw)$ and sends it to $\mathcal{U}$.

$\mathcal{GW}$ Authentication: $\mathcal{F}_{\Pi}$ receives an authentication request $(\text{AUTHENAGW}, sid, req_{gs})$ from $\mathcal{GW}$. $\mathcal{F}_{\Pi}$ records req_{gs} in DX. If $\mathcal{GW}$ is corrupted, $\mathcal{F}_{\Pi}$ sends $(\text{AUTHENAGW}, sid, req_{gs})$ to $\mathcal{SIM}$. Otherwise, $\mathcal{F}_{\Pi}$ send $(\text{AUTHENAGW}, sid)$ to $\mathcal{SIM}$. Then, $\mathcal{F}_{\Pi}$ generates credentials $(sid, AuthAgwToSr)$ and sends it to $\mathcal{GW}$.

$\mathcal{S}$ Authentication: $\mathcal{F}_{\Pi}$ receives an authentication request $(\text{AUTHENSR}, sid, req_{su})$ from $\mathcal{S}$. $\mathcal{F}_{\Pi}$ record req_{us} in DX. If $\mathcal{S}$ is corrupted, $\mathcal{F}_{\Pi}$ send $(\text{AUTHENSR}, sid, req_{us})$ to $\mathcal{SIM}$. Otherwise, $\mathcal{F}_{\Pi}$ send $(\text{AUTHENSR}, sid)$ to $\mathcal{SIM}$. Then, $\mathcal{F}_{\Pi}$ generates authentication credentials $(sid, AuthSrToUr)$ and sends it to $\mathcal{U}$.

New Key: $\mathcal{F}_{\Pi}$ receives a new session key message $(\text{NEWKEY}, sid, \mathcal{U}, \mathcal{S}, sk^*)$ from $\mathcal{SIM}$, where sk^* is determined by $\mathcal{SIM}$. $\mathcal{F}_{\Pi}$ checks if $\mathcal{U}$ or $\mathcal{S}$ is corrupted. If so, $\mathcal{F}_{\Pi}$ set $sk = sk^*$. If $\mathcal{U}$ and $\mathcal{S}$ $\mathcal{F}_{\Pi}$ are both honest, and there exist session key $sk^{'}$ that has been sent to $\mathcal{U}$ and $\mathcal{S}$, $\mathcal{F}_{\Pi}$ sets $sk = sk^{'}$. Otherwise, $\mathcal{F}_{\Pi}$ chooses a random session key $sk \in_R \{0,1\}^l$. Finally, $\mathcal{F}_{\Pi}$ marks this session COMPLETED and sends sk to $\mathcal{U}$ and $\mathcal{S}$.

Guessing Test: $\mathcal{F}_{\Pi}$ receives a guessing message $(\text{TESTPASS}, sid, PW^*, ID^*, BIO^*, C_a^*)$ from $\mathcal{SIM}$ after $\mathcal{A}$ performs an attempt to guess the correct factors. If there exists a record (PW, ID, BIO, C_a) in EDB which is marked completed, $\mathcal{F}_{\Pi}$ check if $PW^* = PW$, $ID^* = ID$, $BIO^* \approx BIO$, and $C_a^* = C_a$. If the above equations hold, $\mathcal{F}_{\Pi}$ sends CORRECT to $\mathcal{SIM}$ and marks this session compromised. Otherwise, $\mathcal{F}_{\Pi}$ sends WRONG to $\mathcal{SIM}$ and marks this session interrupted.

Account Recovery: $\mathcal{F}_{\Pi}$ receives an account recovery request $(\text{RECOVERY}, sid, EML^*, GID^*, ASW^*)$ from $\mathcal{U}$.

If there exists no account in EDB associated with EML^* and GID^*, $\mathcal{F}_{\Pi}$ ignores this message and sends nothing to $\mathcal{U}$. Otherwise, it increments $count_{rec} = count_{rec} + 1$ and sends a recovery message to $\mathcal{SIM}$ as follows: If $\mathcal{U}$ is corrupted, it sends (RECOVER, sid, $\mathcal{U}$, EML^*, GID^*,, ASW^*, $count_{rec}$) to $\mathcal{U}$. Otherwise, it sends (RECOVERY, sid, $\mathcal{U}$, $count_{rec}$) to $\mathcal{SIM}$. Furthermore, if $\mathcal{SIM}$ sends an error message (VOPRF, sid, $error$), $\mathcal{F}_{\Pi}$ aborts this session and exits. Otherwise, $\mathcal{F}_{\Pi}$ checks whether $EML^* = EML$ and $ASW^* = ASW$. If it holds and $\mathcal{U}$ is compromised, $\mathcal{F}_{\Pi}$ sends (PW, BIO, C_a) to $\mathcal{SIM}$, and "Successful Recovery" to $\mathcal{SIM}$. Otherwise, $\mathcal{F}_{\Pi}$ sends message $\perp$ to $\mathcal{U}$ and message "Failed Recovery" to $\mathcal{SIM}$.

Account Reset: $\mathcal{F}_{\Pi}$ receives an account reset request (RESET, sid, PW^{new}, EML^*, GID^*, BIO^{new}) from $\mathcal{U}$. It increments $count_{ret} = count_{ret} + 1$. If $\mathcal{U}$ is corrupted, $\mathcal{F}_{\Pi}$ sends (RESET, sid, $\mathcal{U}$, PW^{new}, BIO^{new}, $count_{ret}$) to $\mathcal{SIM}$. Otherwise, it sends (RESET, sid, $\mathcal{U}$, $count_{ret}$) to $\mathcal{SIM}$. If $\mathcal{SIM}$ sends REJECT, then $\mathcal{F}_{\Pi}$ aborts this session and exits. Otherwise, it sends $(sid, \pi(PW, UID, BIO, T_i))$ to $\mathcal{U}$. If there exists no account in EDB associated with EML^* and GID^*, $\mathcal{F}_{\Pi}$ ignores this message and sends nothing to $\mathcal{U}$. Otherwise, $\mathcal{F}_{\Pi}$ checks whether $EML^* = EML$, $ASW^* = ASW$ and $PW^{new} \in List_{reuse}$. If the above conditions are met,$\mathcal{F}_{\Pi}$ sends ALLOW to $\mathcal{U}$ and "Successful Reset" to $\mathcal{SIM}$. Otherwise, $\mathcal{F}_{\Pi}$ sends REJECT to $\mathcal{U}$ and "Failed Reset" to $\mathcal{SIM}$.

Corrupt: $\mathcal{F}_{\Pi}$ receives a corrupt message (Corrupt, sid, $\mathcal{P}_i$) from $\mathcal{SIM}$, and sends the internal state (sid, $state$) of the party $\mathcal{P}_i$ to $\mathcal{SIM}$.

We now describe how the simulator $\mathcal{SIM}$'s behavior ensures that environment $\mathcal{E}$'s view in the $\{\mathcal{F}_{\text{ROM}}, F_{\text{VOPRF}}, \mathcal{F}_{\text{AE}}, F_{\text{CRS}}\}$-hybrid model is identically distributed with its view in the ideal world. For any request to interact with $\mathcal{F}_{\text{ROM}}$, $\mathcal{F}_{\text{VOPRF}}$ and $\mathcal{F}_{\text{AE}}$, $\mathcal{SIM}$ receives inputs or randomly chooses some inputs, and then simulates the interactions between the emulated functionalities and parties. Here, we omit the case where $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{GW}$ are all honest or corrupted because $\mathcal{SIM}$ can simulate messages perfectly.

Case 1: corrupted $\mathcal{U}$

When $\mathcal{SIM}$ receives messages (INIT, sid, $\mathcal{S}$) and (INIT, sid, $\mathcal{GW}$) from $\mathcal{F}_{\Pi}$, it simulates an instance of F_{CRS}, chooses random values x and s to simulate the secret of of $\mathcal{GW}$ and $\mathcal{S}$. Let x and s be the secret key of $\mathcal{GW}$ and $\mathcal{S}$, and let $pk_{GW} = g^x$ and $pk_S = g^s$ be the public parameters. $\mathcal{SIM}$ simulates the instance of F_{ROM} to generate hash responses.

When $\mathcal{SIM}$ receives a registration message (REGIS, sid, $\mathcal{U}$, $\mathcal{GW}$, PW, ID, PUF, BIO) which indicates a new account registration request, and the message $\{C_1, C_2, C_3, C_4, name, ct_1, ct_2, ct_3, List_{pw}\}$. $\mathcal{SIM}$ stores (PW, ID, C_a, BIO) in $EDB_{\mathcal{SIM}}$. Then, $\mathcal{SIM}$ emulates the instance of the functionality $\mathcal{F}_{\text{ROM}}$, and generates HPW for PW and BIO. Then, it obtains r_1 and ID via the interface of $\mathcal{F}_{\text{ROM}}$ and generates HID for $\mathcal{A}$. It chooses a random challenge C_a for $\mathcal{A}$ which satisfies that $R_2 = sPUF(wPUF(C_a) \,||\, \sigma)$, and $R_a = H_1(R_1 \oplus R_2)$; a value PID_u satisfies $PID_u = H_1(H_2(R_a) \,||\, H_1(HPW) \; mod \; n_p) \oplus C_1$. $\mathcal{SIM}$ sends $\{HPW, HID, R_a\}$ to $\mathcal{A}$. It ensures that it's infeasible for $\mathcal{E}$ to distinguish whether these interactions are conducted in the real world or the ideal world.

$\mathcal{SIM}$ emulates an instance of $\mathcal{F}_{\text{VOPRF}}$ upon receiving the message (EVAL, sid, $\mathcal{U}$) and (EVAL, sid, $\mathcal{GW}$, m_2). If an $\mathcal{F}_{\text{VOPRF}}$ evaluation is successful, $\mathcal{SIM}$ receives (EVALCOMPLETE, sid, V_1) and (EVALCOMPLETE, sid, V_2) from $\mathcal{F}_{\text{VOPRF}}$. Otherwise, if $\mathcal{SIM}$ receives message (EVAL, sid, $error$), it sends (VOPRF, sid, $error$) to $\mathcal{F}_{\Pi}$ and aborts. Since these values are generated through direct emulation of $\mathcal{F}_{\text{VOPRF}}$, it is distributed identically to a value V_1, V_2 that is generated directly through a real instance of $\mathcal{F}_{\text{VOPRF}}$. The environment $\mathcal{E}$'s view is identical. Otherwise, the environment $\mathcal{E}$ can be transformed into an adversary $\mathcal{A}$ to break the security of VOPRF. Then, $\mathcal{SIM}$ generates δ_2 satisfying $\delta_2 = ct_2 \oplus (PW \,||\, BIO \,||\, C_a)$; emulates the instance of $\mathcal{F}_{\text{ROM}}$ with inputs (EML, ASW) and (PW_i, σ) to obtain ct_3 and HPW_i and sends (ct_3, HPW_i) to $\mathcal{A}$. These transmitted messages are uniformaly distributed and difficult for $\mathcal{E}$ to distinguish.

Upon receiving message (AUTHFUR, sid, $\mathcal{GW}$, $PW^{'}$, $BIO^{'}$), $\mathcal{SIM}$ sends a test message (TEST, sid, $\mathcal{U}$, $PW^{'}$, $BIO^{'}$) to $\mathcal{F}_{\Pi}$. If it receives "CORRECT" from $\mathcal{F}_{\Pi}$, $\mathcal{SIM}$ extracts the related values C_1, C_2, V, R_2, K, PID_u stored in $EDB_{\mathcal{SIM}}$, emulates the instance of $\mathcal{F}_{\text{ROM}}$ to generate $H(H(R_a) \,||\, H(HPW) \; mod \; n_p) = PID_u \oplus C_1$, $H(HPW \,||\, PID_u) = K \oplus C_2$. Let $V^* = V$. Otherwise, set $count_{Fail} += 1$. If the number of login and authentication attempts exceeds the threshold, this session will terminate. $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{ROM}}$ and generates HPW for $PW^{'}$ and $BIO^{'}$. Then, $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{AE}}$ with inputs $(k_{ug}, g^{r_{us}}, CID, CAK)$ to generate the values (CT_1, π_1); emulates the instance of $\mathcal{F}_{\text{ROM}}$ to generate M_1. We have that it is infeasible for $\mathcal{E}$ to distinguish these simulated messages from real messages.

Upon receiving message (RECOVERY, sid, $\mathcal{S}$, EML, GID), $\mathcal{SIM}$ sets $count_{rec} += 1$ and emulates the instance of $\mathcal{F}_{\text{ROM}}$ with inputs (sid, EML, GID) to generate m_1 for $\mathcal{A}$. Then, $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{VOPRF}}$ with inputs (EVAL, sid, m_1). If it receives a error message (VOPRF, sid, $eror$), $\mathcal{SIM}$ aborts this session and exits. Otherwise, it emulates the instance of $\mathcal{F}_{\text{ROM}}$ with inputs (sid, EML, V_1) to generate $(name, \delta_1)$, where $\delta_1 = \delta_3 \oplus ct_2$. Further, when receiving $(sid, EML^{'}, Q^{'}, ASW^{'})$ from $\mathcal{A}$, $\mathcal{SIM}$ sends a test message (TEST, $sid, \mathcal{U}, EML^{'}, Q^{'}, ASW^{'}$) to $\mathcal{F}_{\Pi}$. If it receives "CORRECT" from $\mathcal{F}_{\Pi}$, $\mathcal{SIM}$ will receive a valid message (RECOVERYCOMPLETE, sid, $PW \,||\, BIO \,||\, C_a$) from $\mathcal{F}_{\Pi}$, then it sets $\delta_2 = (PW \,||\, BIO \,||\, C_a) \oplus ct_2$. $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{ROM}}$ to generate m_2, and emulates the instance of $\mathcal{F}_{\text{VOPRF}}$ when receiving (EVAL, sid, m_2). If it receives an error message (VOPRF, sid, $eror$), $\mathcal{SIM}$ aborts this session and exits. If EML and ASW are valid, PW, BIO and C_a can be correctly returned to the user. Otherwise, random values $PW^{'}$, $BIO^{'}$ and $C_a^{'}$ will be returned. It is difficult for $\mathcal{E}$ to distinguish these interactions.

Upon receiving (RESET, sid, PW^{new}, BIO^{new}, EML, ASW) from $\mathcal{F}_{\Pi}$, $\mathcal{SIM}$ sends a test message (TEST, sid, $\mathcal{U}$, $EML^{'}$, $Q^{'}$, $ASW^{'}$) to $\mathcal{F}_{\Pi}$ and sets $count_{rec} += 1$. If it receives "CORRECT" from $\mathcal{F}_{\Pi}$, it extracts ct_3, $List_{pw}$ from $\mathcal{EDB}_{\mathcal{SIM}}$, emulates the instance of $\mathcal{F}_{\text{ROM}}$ with inputs (sid, EML, ASW) to generate ct_3^* where $ct_3^* = ct_3$. If $\mathcal{SIM}$ receives message Successful from $\mathcal{F}_{\Pi}$, it updates the password $PW \rightarrow PW^{new}$ stored in $\mathcal{DX}_{\mathcal{SIM}}$. Otherwise,

$\mathcal{SIM}$ makes no change. We have that the environment $\mathcal{E}$'s view is identical in both the real and ideal worlds.

Case 2: corrupted $\mathcal{GW}$

Registration. When $\mathcal{SIM}$ receives a registration message (REGIS, $sid, \mathcal{U}, \mathcal{GW}, L_1(PW, UID, BIO), L_2(EML, GID, Q)$, which indicates a new account registration request, $\mathcal{SIM}$ chooses dummy values PW^*, UID^*, and BIO^* according to the leakage $L_1(PW, UID, BIO)$. It generates the random values HPW and MID, then sends them to $\mathcal{A}$. $\mathcal{SIM}$ sends a corruption message (CORRUPT, sid, $\mathcal{GW}$) to $\mathcal{F}_{\Pi}$, receives the secret key x of $\mathcal{GW}$ and records x in $\mathcal{DX}_{\mathcal{SIM}}$. $\mathcal{SIM}$ receives the challege C_a from the interface of $\mathcal{F}_{\text{SMT}}$, then choose a random response R_a to $\mathcal{A}$. When $\mathcal{SIM}$ receives $\{sid, C_1, C_2, C_3, C_4, C_5\}$ from $\mathcal{F}_{\Pi}$, it computes $PID_u = HID \oplus C_2$, and $a_u = C_1 \oplus C_3$; emulates the instance of $\mathcal{F}_{\text{ROM}}$ to generates $H_2(H_2(R_a) \,||\, H_1(HPW)\ mod\ n_p) = C_1 \oplus PID_u$, $H_1(x \,||\, PID_u \,||\, Num_{ret}) = (C_a \,||\, R_a) \oplus C_5$ and tuples $H_1(HPW \,||\, PID_u)$,$H_1(x \,||\, HID \,||C_a))$ satisfying $C_2 = H_1(HPW \,||\, PID_u) \oplus H_1(x \,||\, HID \,||C_a)$. $\mathcal{SIM}$ records PID_u, a_u in $\mathcal{EDB}_{\mathcal{SIM}}$.

$\mathcal{SIM}$ chooses dummy values EML^*, GID^*, Q^*, ASW^* according to the leakage function $L_2(EML, GID, Q)$, and generates random message m_1 and m_2. $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{VOPRF}}$ with message $(EVAL, sid, EML, GID)$ to generate V_1; then generates V_2 with inputs $(EVAL, sid, EML, V_1)$. If an error message (VOPRF, sid, $error$) was returned, $\mathcal{SIM}$ aborts this session and exits. Otherwise, it generates random value $name$, ct_1, ct_2, ct_3 and $List_{pw}$ to $\mathcal{A}$. Due to the pseudorandomness of VOPRF value, it is infeasible for $\mathcal{E}$ to distinguish the simulated messages and real messages.

Upon receiving message (AUTHEUSER, sid, $\mathcal{GW}$, $L_3(PW^{'}, C_a^{'}, BIO^{'})$), $\mathcal{SIM}$ chooses dummy values PW^*, C_a^*, and BIO^* according to the leakage $L_3(PW^{'}, C_a^{'}, BIO^{'})$. $\mathcal{SIM}$ extracts messages (PID_u, R_a, HID) from $\mathcal{EDB}_{\mathcal{SIM}}$. $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{ROM}}$, and computes $PID_u = H_2(H_1(R_a) \,||\, H_1(HPW)\ mod\ n_p) \oplus C_1$, $HID = C_4 \oplus PID_u$, $K = H_1(x \,||\, HID \,||\, C_a) = H_1(HPW \,||\, PID_u) \oplus C_2$. Then, it chooses random values r_{ug}, emulates the instance of $\mathcal{F}_{\text{AE}}$ to generate random values (CT_1, π_1); set random values $CID\ (= HID \oplus H(N_u \,||\, k_{ug}))$, $CAK\ (= (a_u^* \,||\, K) \oplus H_3(N_u \,||\, k_{ug}))$; choose a random value r_{us} and emulates the instance of $\mathcal{F}_{\text{ROM}}$ to generate M_1. Then it sends CT_1, π_1, $g^{r_{ug}}$, M_1, and T_1 to $\mathcal{GW}$. Due to the randomness of r_3, r_4, r_u and the security of AE, it is infeasible for $\mathcal{E}$ to distinguish the simulated messages and real messages.

Upon receiving message (AUTHEAGW, sid, $\mathcal{S}$, K_S), $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{AE}}$ and returns $(g^{r_{us}} \,||\, g^{r_{ug}} \,||\, HID)$ to $\mathcal{GW}$. $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{AE}}$ with inputs $(k_{gs}, g^{r_{us}}, g^{r_{gs}}, HID)$ and emulates the instance of $\mathcal{F}_{\text{ROM}}$ with inputs $(K_S, k_{gs}, g^{r_{gs}}, T_2)$, then generates (CT_2, π_2) and M_2 for $\mathcal{GW}$. Upon receiving message (AUTHESR, sid,$\mathcal{U}$), $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{AE}}$ to simulates the decryption and verification for $\mathcal{S}$, then generates $(g^{r_{us}}, g^{r_{gs}}, HID)$, VALID and sends them to $\mathcal{S}$. Furthermore, it chooses a random value r_{su} and emulates the instance of $\mathcal{F}_{\text{ROM}}$ to generate sk and M_3. $\mathcal{SIM}$ sends $g^{r_{su}}$, M_3, SID and T_3 to $\mathcal{S}$. Due to the security of the authenticated encryption scheme AE, and the difficulty of CDH problem, it is difficult for $\mathcal{E}$ to distinguish the simulated messages.

Upon receiving message (RECOVER, sid, $\mathcal{GW}$, $L_4(EML, Q, GID)$, $count_{rec}$), $\mathcal{SIM}$ chooses random values EML and GID and emulates the instance of $\mathcal{F}_{\text{VOPRF}}$ by creating the message (EVAL, sid, $\mathcal{GW}$, EML, GID). if an error message is sent to $\mathcal{F}_{\Pi}$ during this emulation, then $\mathcal{SIM}$ aborts and exits. Otherwise, upon receiving (RECOVER, $name$, Q) from $\mathcal{F}_{\Pi}$, $\mathcal{SIM}$ retrieves $(name, ct_1, ct_2)$ from $DX_{\mathcal{SIM}}$, chooses δ_1 at random, and sends $(name, \delta_1)$ to $\mathcal{GW}$. Upon receiving (RESTORE, sid, $\perp$), $\mathcal{SIM}$ emulates the instance of $\mathcal{F}_{\text{VOPRF}}$ by creating the message (EVAL, sid, $\mathcal{GW}$, $\perp$), if an error message is sent to $\mathcal{F}_{\Pi}$ during this emulation, $\mathcal{SIM}$ aborts and exits.

Upon receiving (RESET, sid, $L_5(PW^{new}, BIO^{new}, EML, Q, GID)$) from $\mathcal{F}_{\Pi}$, $\mathcal{SIM}$ chooses random values PW^{new} and BIO^{new}, emulates the instance of $\mathcal{F}_{\text{ROM}}$ to generate HPW^{new} and sets $count_{ret} += 1$. It chooses random value r_3 and computes secret key $k_{ret} = g^{xr_3}$, emulates the instance of $\mathcal{F}_{\text{AE}}$ with inputs $(sid, ct_3*, HPW^{new}, B_3^{new})$ to generate (CT_4, π_4). Then it sends CT_4, π_4, g^{r_3} to $\mathcal{A}$. Due to the security of the authenticated encryption scheme AE, and the difficulty of CDH problem, it is difficult for $\mathcal{E}$ to distinguish the simulated messages.

APPENDIX C: INFORMAL SECURITY ANALYSIS

This section provides the informal analysis of our scheme, showing that it can resist known attacks such as password guessing attacks, user impersonation attacks, node capture attacks, and achieves multi-factor security, account recovery, and forward security.

(1) Resist Password Guessing Attacks. An adversary $\mathcal{A}$ can extract messages from the public channel. Assuming that the biometrics BIO and the PUF challenge C_a have been exposed to $\mathcal{A}$ according to the assumption of the multi-factor security, which enables $\mathcal{A}$ to derive σ by calculating $Res(BIO, \theta)$ and R_1 and R_2 through $R_1 = wPUF(C_a)$, $R_2 = sPUF(C_a \,||\, \sigma)$. Furthermore, $\mathcal{A}$ attempts to guess PW and ID. Since $PID_u = H_2(H_2(R_a) \,||\, H_1(HPW)\ mod\ n_p) \oplus C_1$, $K = H_1(x \,||\, HID \,||\, C_a) = H_1(HPW \,||\, PID_u) \oplus C_2$, $HID = C_4 \oplus PID_u$. Due to the application of the fuzzy verifier technique, it implies that there exist almost $\frac{|HID| \cdot |PW|}{N_0} \approx 2^{32}$ candidate pairs of (HID, PW) to thwart $\mathcal{A}$, where $N_0 \approx 2^8$, $|HID| \approx 10^6$ and $|PW| \approx 10^6$ represent the size of identity space and password space respectively. It is computationally infeasible to pinpoint the correct identity HID and password PW from the candidates without interacting with $\mathcal{GW}$. It is difficult for $\mathcal{A}$ to launch password guessing attacks.

(2) Resist Node Capture Attacks. In node capture attacks, the first observation is that the compromised server will not weaken the security of other servers because the publicly transimitted message $\{N_s, M_2, CT_2, \pi_2, T_2\}$ and $\{N_{us}, M_3, SID, T_3\}$ will not reveal the secret key of the server. The second one demonstrates that even if the preshared session key k_{gs} is exposed, it is difficult for $\mathcal{A}$ to recover the current session key sk. Following the construction of the proposed scheme, we have $k_{gs} = pk_s^{r_{gs}}$, $N_s = g^{r_{gs}}$, without the knowledge of the secret r_{gs}, it is

difficult for $\mathcal{A}$ to compute other servers' k_{gs} because of the difficulty of the discrete logarithmic problem. Additionally, since r_{gs} is chosen randomly by $\mathcal{GW}$ for each session, it is infeasible to extract the random value r_{gs} through the publicly transmitted message. As a result, $\mathcal{A}$ can't retrieve the message $g^{r_{us}}$ to compute the previous session key even if it compromises the $\mathcal{S}$.

(3) Resist Impersonation Attacks. Suppose that $\mathcal{A}$ eavesdrops, intercepts and further modifies the messages to impersonate the legitimate party. There exist three cases to consider: **(1)** impersonating $\mathcal{U}$; **(2)** impersonating $\mathcal{S}$; **(3)** impersonating $\mathcal{GW}$. In the first case, $\mathcal{A}$ intercepts $\{CT_1, \pi_1, N_u, M_1, T_1\}$ and attempts to impersonate $\mathcal{U}$. Observing apparently, it is difficult for $\mathcal{A}$ to obtain PID_u, HID, and K without knowing the password PW, the biometrics BIO, and the PUFs' response R_a. Additionally, despite passing the authentication using the original messages, it is computationally infeasible to rebuild the session key without knowing the random values r_{us} and r_{su}. The same reason holds for the second case, without the knowledge of the secret s and K_S, $\mathcal{A}$ cannot impersonate $\mathcal{S}$. According to the above analysis, although the server has been compromised, it is difficult for $\mathcal{A}$ to compute the session key sk without the knowledge of r_{us} and r_{su}. Finally, it is impossible to impersonate $\mathcal{GW}$ because $\mathcal{A}$ doesn't know x and K_S.

(4) Resist Privileged Insider Attacks. Suppose that an insider $\mathcal{A}$ obtains $\mathcal{U}$'s registration request $\{HPW, HID, R_2\}$ from $\mathcal{GW}$. However, $\mathcal{A}$ is unable to recover preshared key k_{gu} without the knowledge of PW and BIO. Furthermore, $Gen(\cdot)$ and $Res(\cdot)$ are unavailable for $\mathcal{GW}$ to further compute σ_b from BIO. The application of the fuzzy verifier technique makes it computationally infeasible for $\mathcal{A}$ to guess the targeted identity and password (ID, PW).

(5) Resist Replay Attacks. Due to the deployment of timestamps, it can prevent replay attacks. When $|T_i^{'} - T_i| > \Delta T$, this session will be aborted immediately. In addition, the proposed protocol is mutually authenticated among three parties, where **(1)** $\mathcal{GW}$ checks the equation $M_1 = M_1^{'}$ to authenticate $\mathcal{U}$; **(2)** $\mathcal{S}$ verifies the equation $M_2 = M_2^{'}$ to authenticate $\mathcal{GW}$; **(3)** $\mathcal{U}$ checks $M_3 = M_3^{'}$ to authenticate $\mathcal{S}$. Failures at any of the above steps will lead to termination.

(6) Resist Known Session-Specific Temporary Information (KSSTI) Attacks. KSSTI attacks describe that $\mathcal{A}$ attempts to calculate the session key using random numbers like r_{us}, r_{su}. It is required that only one random value can be obtained by $\mathcal{A}$, that is, $\mathcal{A}$ is restricted to be the legitimate user, or the server during each KSSIT attack. According to the construction of the scheme, the session key is calculated as $sk = H_3(g^{r_{us}} \,||\, g^{r_{su}} \,||\, g^{r_{us}r_{su}} \,||\, HID \,||\, SID)$, showing that it is infeasible to rebuild the session key in the case where only one of the random values is exposed without the knowledege of other random values and the dummy identity HID.

(8) Resist Key Compromise Impersonation (KCI) Attacks. Assuming the leakage of the long-term secret key x, $\mathcal{A}$ attempts to impersonate the legitimate user. According to the construction of our scheme, we have that the publicly transmitted message are N_u, CT_1, π_1. $\mathcal{A}$ can compute $k_{ug} = N_u^x$ and decrypt $AE.Dec(k_{ug}, CT_1) = g^{r_{us}} \,||\, CID \,||\, CAK)$. However, without the knowledge of R_a, it is infeasible for $\mathcal{A}$ to compute $M_1 = H_3(k_{ug} \,||\, K \,||\, CID \,||\, CAK \,||\, R_a||T_1)$. Additionally, $\mathcal{A}$ can decrypt CT_1 to obtain $g^{r_{us}}$ but cannot compute the session key $sk = H_3(g^{r_{us}} \,||\, g^{r_{su}} \,||\, g^{r_{us}r_{su}} \,||\, HID \,||\, SID \,||\, T_3)$ from the message $g^{r_{us}}$ and $g^{r_{su}}$ due to the difficulty of the CDH problem. Therefore, $\mathcal{A}$ cannot impersonate a user using a KCI attack.

(9) Perfect Forward Security. Notice that, the session key is computed as $sk = H_3(g^{r_{us}} \,||\, g^{r_{su}} \,||\, g^{r_{us}r_{su}} \,||\, HID \,||\, SID \,||\, T_3)$, where r_{us} and r_{su} are chosen randomly and transmitted securely in ciphertext CT_1 and CT_3. Since r_{us} and r_{su} are chosen randomly for each session, it is computationally infeasible for $\mathcal{A}$ to compute the previous sk even if the current random values are compromised. Additionally, according to (8), $\mathcal{A}$ cannot compute sk when it knows the long-term secret key x.

(10) Account Recovery and Reset. When forgetting or leaking accounts, our scheme enables users to recover or reset their accounts. During the registration phase, $\mathcal{U}$ computes $\delta_2 = H_1(\, V_2 \,||\, r_2)$, $\delta_3 = EML \,||\, Q \,||\, r_2 \,||\, r_{ret}$, $ct_1 = \delta_1 \oplus \delta_3$, and $ct_2 = \delta_2 \oplus (PW \,||\, BIO \,||\, C_a)$, $ct_3 = H_1(EML \,||\, ASW \,||\, GID||r_{ret})$ using her email address EML, secret questions and answers $(Q,\ ASW)$. Finally, $\mathcal{U}$ stores the account information to the server $\mathcal{S}$. Once forgetting or losing her account, $\mathcal{U}$ is required to provide the correct email address EML and answer the secret questions Q correctly to obtain the answer ASW. Furthermore, $\mathcal{U}$ can recover the account $\delta_3 = \delta_1 \oplus ct_1$, $\delta_2 = H_1(\, V_2 \,||\, r_2)$, and recovers the account as $PW \,||\, BIO \,||\, C_a = \delta_2 \oplus ct_2$. When $\mathcal{U}$ decides to reset her account, she computes $ct_3^* = H_1(EML \,||\, GID \,||\, ASW \,||\, r_{ret})$ which will be sent to $\mathcal{GW}$ secretly. If $ct_3^* = ct_3$, $\mathcal{U}$ is allowed to reset her account as $C_5^{new} = C_5 \oplus H_1(x \,||\, PID_u \,||\, Num_{ret} - 1) \oplus H_1(x \,||\, PID_u \,||\, Num_{ret})$.

(11) Password Reuse Resistance. According to the construction of our scheme, $\mathcal{U}$ chooses some dummy passwords PW_1, PW_2, ..., PW_t, and let $PW_i = PW$ which denotes passwords that are easily guessed and computes $HPW_i = H_1(PW_i \,||\, \sigma)$, B_{1i}, B_{2i}, B_{3i}. Afterwards, $\mathcal{U}$ applies bloom filter BF to map B_{3i} into a list $List_u$. When $\mathcal{U}$ decides to reset her account, she needs to compute $HPW^{new} = H_1(PW^{new} \,||\, \sigma^{new})$, computes B_3^{new} and sends it to $\mathcal{GW}$. Then, $\mathcal{GW}$ applies bloom filter BF to checks whether B_3^{new} has been recorded in $List_u$. If a previously used password is submitted, $\mathcal{U}$ cannot perform this operation.

(12) Anonymity and Untraceability. Our scheme provides anonymity for $\mathcal{U}$ through generating pseudo-identity $HID = H_1(ID \,||\, r_1)$ to hide the real identity of $\mathcal{U}$ and is hidden in the form of $C_4 = HID \oplus PID_u$. Without the knowledge of PW, BIO and R_a, it is difficult to compute PID_u to obtain HID in each session.

Functionality F_{Π}

The functionality $\mathcal{F}_{\Pi}$ is parameterized with a security κ, dictionary DT, and values $count_{log}$, and $count_{rec}$, which are initialized to zero, to record the number of account login and recovery. It stores and manages a database EDB, then interacts with the simulator $\mathcal{SIM}$, the user $\mathcal{U}$, the authentication gateway $\mathcal{GW}$ and the server $\mathcal{S}$ via the following queries.

- Upon receiving (INIT,sid) from $\mathcal{GW}$, check if it is the first INIT query for sid:
  (1) If so, record (initrec, sid, $\mathcal{GW}$) and send (INIT, sid, $\mathcal{GW}$) to $\mathcal{SIM}$.
  (2) Else, ignore this query.
  (3) Else, if $\mathcal{GW}$ is corrupted, send (INIT,sid, $state$, $\mathcal{GW}$) to $\mathcal{SIM}$.
- Upon receiving (INIT,sid) from $\mathcal{S}$, check if it is the first INIT query for sid:
  (1) If so, record (initrec, sid, $\mathcal{S}$) and send (INIT,sid, $\mathcal{S}$) to $\mathcal{SIM}$.
  (2) Else, ignore this query.
  (3) Else, if $\mathcal{S}$ is corrupted, send (INIT,sid, $state$, $\mathcal{S}$) to $\mathcal{SIM}$.
- Upon receiving (REGIS,sid, $\mathcal{GW}$, PW,ID,BIO,EML,Q,ASW) from $\mathcal{U}$,
  (1) Record (regrec, $\mathcal{U}$, PW, ID, BIO, EML, Q, GID, ASW).
  (2) If $\mathcal{U}$ is corrupted, send message (REGIS, sid, $\mathcal{GW}$, PW, UID, BIO, EML, Q, ASW) to $\mathcal{SIM}$.
  (3) Else, send (REGIS, sid, $\mathcal{GW}$, $L_1(PW, ID, BIO)$, $L_2(EML, Q, ASW)$) to $\mathcal{SIM}$. // $L_1(PW, UID, BIO)$ denotes the leakage of PW, ID and BIO, $L_2(EML, Q, ASW)$ denotes the leakage of EML, Q and ASW.
  (4) If $\mathcal{SIM}$ responds with message (VOPRF, sid, errror), return error to $\mathcal{U}$ and exit.
  (5) Choose dummy values $PW_i^{'}$, then generate list $List_{pw}$ for PW and all $PW_i^{'}$. Record $List_{pw}$ in EDB.
  (6) Send the registration response $(sid, \text{RESPONSE})$ to $\mathcal{U}$.
- Upon receiving (LOGAUTHUSER,sid, $PW^{'}$, $BIO^{'}$) from $\mathcal{U}$,
  (1) If it is the first LOGAUTHUREQSER message for sid, then record (authrec, $\mathcal{U}$, $\mathcal{GW}$, PW, BIO).
  (2) Send an authentication message to $\mathcal{SIM}$.
  (3) If $\mathcal{U}$ is corrupted, send message (LOGAUTHUSER, sid, $\mathcal{U}$, $\mathcal{GW}$, $PW^{'}$, $BIO^{'}$) to $\mathcal{SIM}$.
  (4) Else, send (LOGAUTHUSER, sid,$\mathcal{U}$, $\mathcal{GW}$, $L_3(PW^{'}, BIO^{'})$) to $\mathcal{SIM}$. // $L_3(PW^{'}, BIO^{'})$ denotes the leakage of $PW^{'}$ and $BIO^{'}$
  (5) Send an authentication credential $(sid, \text{LOGAUTHREPUR})$ to $\mathcal{U}$.
- Upon receiving (AUTHAGW,sid, $\mathcal{GW}$, $\mathcal{S}$, $SECRET_{gs}$) from $\mathcal{GW}$,
  (1) If it is the first AUTHREQAGW message for sid, then record (authrec, $\mathcal{GW}$, $\mathcal{S}$, $SECRET_{gs}$).
  (2) Send an authentication message to $\mathcal{SIM}$.
  (3) If $\mathcal{GW}$ is corrupted, send (AUTHAGW, sid, $\mathcal{GW}$, $\mathcal{S}$, $SECRET_{gs}$) to $\mathcal{SIM}$.
  (4) Else, send (AUTHENAGW, sid, $\mathcal{GW}$, $\mathcal{S}$) to $\mathcal{SIM}$.
  (5) Send an authentication credential $(sid, \text{AUTHREPAGW})$ to $\mathcal{GW}$.
- Upon receiving (AUTHREQSR,sid, $\mathcal{S}$, $\mathcal{U}$, $SECRET_{su}$) from $\mathcal{S}$,
  (1) If it is the first AUTHSR message for sid, then record (authrec, $\mathcal{S}$, $\mathcal{U}$, $SECRET_{SU}$).
  (2) Send an authentication message to $\mathcal{SIM}$.
  (3) If $\mathcal{S}$ is corrupted, send message (AUTHREQSR, sid, $\mathcal{S}$, $\mathcal{U}$, $SECRET_{SU}$) to $\mathcal{SIM}$.
  (4) Else, send (AUTHENSERVER, sid, $\mathcal{S}$, $\mathcal{U}$) to $\mathcal{SIM}$.
  (5) Send an authentication credential $(sid, \text{AUTHREPSR})$ to $\mathcal{S}$.

Fig. 1: The ideal functionality $\mathcal{F}_{\Pi}$, part 1 of 2.

Functionality F_{Π}

The second part of the ideal functionality $\mathcal{F}_{\Pi}$.

- Upon receiving (NEWKEY,sid,$\mathcal{U}$, $\mathcal{S}$, sk^*) from $\mathcal{SIM}$, if there is a record $(sid,\mathcal{U},\mathcal{S})$ not marked COMPLETED, then do the following:
  (1) If the record is COMPLETED, or $\mathcal{U}$ or $\mathcal{S}$ is corrupted, set $sk = sk^*$. // sk is chosen by $\mathcal{A}$
  (2) Else if the record is FRESH, a record $(sid, sk^{'})$ was sent to $\mathcal{U}$ and $\mathcal{S}$, and at that time there was a record $(sid,\mathcal{U},\mathcal{S})$ marked FRESH, set $sk = sk^{'}$.
  (3) Else select $sk \in_R \{0,1\}^l$. // a random and fresh session key is chosen
  (4) Finally mark $(sid,\mathcal{U},\mathcal{S})$ COMPLETED and send (sid, sk) to $\mathcal{U}$, and $\mathcal{S}$.
- Upon receiving (TEST,sid, PW^*, BIO^*) from $\mathcal{SIM}$, do the following:// capture guessing attacks
  (1) If the record is a record (regrec, $\mathcal{U}$, PW, ID, BIO, $\perp$, $\perp$) marked COMPLETED, do: if $PW^* = PW$ and $BIO \approx BIO^*$, return "CORRECT" to $\mathcal{A}$ and mark sid as compromised; else return "WRONG" and mark sid as interrupted.
- Upon receiving (RECOVER,sid,$\mathcal{S}$,$EML^{'}$,$BIO^{'}$,$GID^{'}$,$ASW^{'}$) from $\mathcal{U}$,
  (1) If $\mathcal{U}$ is corrupted, send (RESTORE, sid,$\mathcal{U}$, $EML^{'}$, $BIO^{'}$, $GID^{'}$, $ASW^{'}$) to $\mathcal{SIM}$. Otherwise, send (RESTORE, sid, $\mathcal{U}$,$L_5(EML^{'}, GID^{'}, Q^{'}, BIO^{'}, ASW^{'})$) to $\mathcal{SIM}$.
  (3) If $\mathcal{SIM}$ responds with (sid, VOPRF, error) , send error to $\mathcal{U}$ and exit.
  (4) If there exist an account (PW, BIO, EML, Q, GID, ASW) labeled with $name$ where $EML^{'} = EML$, $GID^{'} = GID$, and $Q^{'} = Q$, retrieve $(ASW, PW||BIO)$ from $EDB[name]$ and set $count_{rec}+ = 1$.
  (5) If $ASW^{'} = ASW$, $BIO = BI0^{'}$ and $\mathcal{U}$ is compromised, then send $(PW, BIO), count_{rec}+ = 1$ to $\mathcal{SIM}$.
  (6) If $ASW^{'} = ASW$, $BIO = BI0^{'}$ and $\mathcal{U}$ is honest, then send (PW, BIO) to $\mathcal{U}$.
  (7) Else, send "Failed Recovery" and $count_{rec}+ = 1$ to $\mathcal{U}$.
- Upon receiving (RESET, sid, $\mathcal{U}$, $EML^{'}$, $BIO^{'}$, $ASW^{'}$, PW^{new}) from $\mathcal{U}$,
  (2) If $\mathcal{U}$ is corrupted, send (RESET, sid, $\mathcal{U}$, $EML^{'}$, $BIO^{'}$, $ASW^{'}$, PW^{new}) to $\mathcal{SIM}$. Otherwise, send (RESET, sid, $L_6(EML^{'}, Q^{'}, BIO^{'}, ASW^{'})$) to $\mathcal{SIM}$.
  (4) If there exist an account ($List_{pw}$, BIO, EML, Q, ASW) labeled with $name$ where $EML^{'} = EML$, and $Q^{'} = Q$, set $count_{rec}+ = 1$.
  (5) If $ASW^{'} = ASW$, $BIO = BI0^{'}$ and $PW^{new} \notin List_{pw}$, then set $PW \rightarrow PW^{new}$.
  (5) If $ASW^{'} = ASW$, $BIO = BI0^{'}$ and $PW^{new} \in List_{pw}$, then return (REUSE) to $\mathcal{U}$.
  (5) Else, send (REJECT) to $\mathcal{U}$.
- Upon receiving (CORRUPT,sid,P_i) from $\mathcal{SIM}$, // P_i denotes the party $\{\mathcal{U}, \mathcal{GW}, \mathcal{S}\}$
  (1) Send the internal state $(sid, state)$ of P_i to $\mathcal{SIM}$.

Fig. 2: The ideal functionality $\mathcal{F}_{\Pi}$, part 2 of 2.

Simulator for $\mathcal{F}_{\Pi_1}$

The simulator creates the database $EDB_{\mathcal{SIM}}$ and $DX_{\mathcal{SIM}}$ to record the account information and some secret message. The simulator $\mathcal{SIM}$ can obtain the secret inputs of parties through the interface of $\mathcal{F}_{\text{AE}}$ and $\mathcal{F}_{\text{VOPRF}}$.

- Upon receiving message (INIT, sid), $\mathcal{SIM}$ emulates $\mathcal{F}_{\text{ROM}}$, $\mathcal{F}_{\text{CRS}}$, and $\mathcal{F}_{\text{VOPRF}}$ by internally running a copy of these functionalities as described in Fig. 9, Fig. 6, and Fig. 8. $\mathcal{SIM}$ sends message (INIT, sid) to $\mathcal{F}_{\text{VOPRF}}$ and sends $(\text{PARAMETER}, sid)$ to $\mathcal{F}_{\text{CRS}}$. Then, receives $(\text{PARAMETERCOMPLETE}, sid, G, g, p, \{H_i\}_{1\leq i\leq 5})$.
- Upon receiving message (REGIS, sid, PW, ID, BIO,EML,Q,ASW), record $(PW, ID, BIO, EML, Q, ASW)$ in $DT_{\mathcal{SIM}}$,// $\mathcal{U}$ is corrupted
  (1) Run $Gen(BIO)$ to obtain (σ, θ), choose random value r_1 and generate (HID, HPW).
  (2) Choose random values x, C_a, and PID_u as $\mathcal{GW}$, and compute C_1, C_2, C_3, C_4, C_5 according to the scheme using HPW, MID, PID_u, C_a, and x. Record x in $DX_{\mathcal{SIM}}$ and $(C_1, C_2, C_3, C_4, C_5, PID_u, C_a, R_a)$ in $EDB_{\mathcal{SIM}}$.
  (3) Upon receiving $(\text{EVAL}, sid, \mathcal{GW}, m_1)$ and $(\text{EVAL}, sid, \mathcal{GW}, m_2)$, extracts the random value x from $DX_{\mathcal{SIM}}$. Then, emulate an instance of $\mathcal{F}_{\text{VOPRF}}$. If an error was sent to $\mathcal{F}_{\Pi}$, abort this session and exit. Otherwise, generate $(V1, V_2)$ via the instance of $\mathcal{F}_{\text{VOPRF}}$.
  (4) Choose a random value r_2, compute $\delta_2, \delta_3, ct_1, ct_2, ct_3$ according to the scheme using EML, BIO, ASW and r_2. Then, record $name, ct_1, ct_2, ct_3$ in $EDB_{\mathcal{SIM}}$.
- Upon receiving message (REGIS, sid, $\mathcal{GW}$, $L_1(PW, ID, BIO)$, $L_2(EML, Q)$),// $\mathcal{U}$ is honest
  (1) Choose dummy identity, password and biometrics as $ID^{'}$, $PW^{'}$, $BIO^{'}$, generate HID and HPW at random.
  (2) If $\mathcal{GW}$ is corrupted, choose the challenge C_a, $PID_u \in_R \{0,1\}^*$, send C_a to UR. Upon receiving R_2, compute C_1, C_2 , C_3, C_4, and C_5 according to the scheme using HPW, HID, PID_u, R_a, and x.
  (3) If $\mathcal{GW}$ is honest, generate C_1, C_2 , C_3, C_4, and C_5 at random.
  (4) Upon receiving $(\text{EVAL}, sid, \mathcal{GW})$ from $\mathcal{U}$, if an error was sent to $\mathcal{F}_{\Pi}$, abort this session and exit. Otherwise, if $\mathcal{GW}$ is corrupted, receive x from $DX_{\mathcal{SIM}}$, then emulate an instance of $\mathcal{F}_{\text{VOPRF}}$ to generate $(V1, V_2)$.
  (5) Compute $(name, ct_1, ct_2, ct_3)$ at random and record $name, ct_1, ct_2, ct_3$ in $EDB_{\mathcal{SIM}}$.
- Upon receiving (LOGAUTHUR, $sid, \mathcal{U}$, $\mathcal{GW}$, $PW^{'}$, $BIO^{'}$, $C_a^{'}$), // $\mathcal{U}$ is corrupted
  (1) Send a message $(\text{TEST}, sid, \mathcal{U}, PW, BIO, C_a)$ to $\mathcal{F}_{\Pi}$. When receiving "CORRECT" from $\mathcal{F}_{\Pi}$, replace the dummy values with $PW^{'}$, $BIO^{'}$ and $C_a^{'}$. Otherwise, when receiving "WRONG" from $\mathcal{F}_{\Pi}$, make no change.
  (2) Retrieve C_1, C_2, C_3, C_4 from $DX_{\mathcal{SIM}}$. Emulate the instance of $\mathcal{F}_{\text{ROM}}$ to simulate HPW, and compute $PID_u = H_2(H_1(R_a) \mid\mid H_1(HPW) \ mod \ n_p) \oplus C_1$, $HID = C_4 \oplus PID_u$, where $R_a = H_1(R_1 \oplus R_2)$ generated by C_a.
  (3) Choose random value r_{ug}, and generate N_u, M_1, CID, and CAK using these values. Further, emulate the instance of $\mathcal{F}_{\text{AE}}$ to generate the message (CT_1, π_1) for values $g^{r_{us}}, CID, CAK$ where r_{us} is chosen randomly. Send CT_1, π_1, N_u, M_1 to $\mathcal{GW}$.
  (4) When receiving (N_{us}, M_3, SID) from $\mathcal{S}$, compute sk and record it in $DX_{\mathcal{SIM}}$. Send (NEWKEY,sid, $\mathcal{U}$, $\mathcal{S}$, sk) to $\mathcal{F}_{\Pi}$.
- Upon receiving (AUTHUSR, $sid, \mathcal{U}$, $\mathcal{GW}$, $L_3(PW^{'}, BIO^{'}, C_a^{'})$), // $\mathcal{U}$ is honest
  (1) Choose dummy values $PW^{''}$, $BIO^{''}$ and $C_a^{''}$ according to the function L_3, and set $R_a^{''}$ at random first.
  (2) If $\mathcal{GW}$ is corrupted, retrieve x and C_a from $DX_{\mathcal{SIM}}$; then, replace $C_a^{''}$ with C_a and then generate R_1 using the correct C_a. Otherwise, make no change.
  (3) Choose random values r_{ug}, r_{us}, and generate CT_1, π_1, N_u, M_1 using $PW^{''}$, $BIO^{''}$, C_a, and R_a. Send CT_1, π_1, N_u, M_1 to $\mathcal{GW}$.
  (4) If $\mathcal{S}$ is corrupted, perform decryption to obtain $g^{r_{us}}$ and HID from CT_2; choose a random value r_{su}, compute sk, and record it in $DX_{\mathcal{SIM}}$; then generate N_{us}, and M_3 using the known values. Otherwise, choose random values $N_{us} \in_R$, sk, and M_3. Finally, send (NEWKEY,sid, $\mathcal{U}$, $\mathcal{S}$, sk) to $\mathcal{F}_{\Pi}$.

Fig. 3: The simulator of $\mathcal{F}_{\Pi_1}$, part 1 of 3.

Simulator for $\mathcal{F}_{\Pi}$

The second part of the simulator $\mathcal{SIM}$ for $\mathcal{F}_{\Pi}$.

- Upon receiving (AUTHAGW,sid, $\mathcal{GW}$, $\mathcal{S}$, K_S), // $\mathcal{GW}$ is corrupted
  (1) Retrieve x from $DX_{\mathcal{SIM}}$.
  (2) Compute the secret key $k_{ug} = N_u^x$, emulate the instance of $\mathcal{F}_{\text{AE}}$ and then perform the decryption to get $g^{r_{us}}$ and HID.
  (3) Choose random value r_{gs} to generate N_s, M_2,and $k_{gs} = pk_s^{r_{gs}}$, emulate the instance of $\mathcal{F}_{\text{AE}}$ and then perform the encryption to obtain CT_2, π_2.
- Upon receiving (AUTHAGW,sid, $\mathcal{GW}$, $\mathcal{S}$), // $\mathcal{GW}$ is honest
  (1) If $\mathcal{S}$ is corrupted, retrieve s, K_S from $DX_{\mathcal{SIM}}$. Choose a random value r_{gs}, and compute N_s, k_{gs}.
  (2) If there exist a record $(\perp, \perp, HID)$ in $EDB_{\mathcal{SIM}}$, extract HID. Otherwise, choose random value $g^{r_{us}} \in_R G$, $HID \in_R \{0,1\}^*$, emulate the instance of $\mathcal{F}_{\text{AE}}$ to generate (CT_2, π_2).
- Upon receiving (AUTHSERVER, sid, $\mathcal{S}, \mathcal{U}$, s), // $\mathcal{S}$ is corrupted
  (1) Record s in $DX_{\mathcal{SIM}}$ if $\mathcal{S}$ has not been corrupted before. Generate the secret key k_{gs},emulate the instance of $\mathcal{F}_{\text{AE}}$ to obtain $g^{r_{us}}, HID = AE.Dec(k_{gs}, CT_2)$.
  (2) Choose a random value r_{su} and compute the session key as sk.
  (3) Record sk in $DX_{\mathcal{SIM}}$ and send (NEWKEY,$sid, \mathcal{U}$, $\mathcal{S}$, sk) to $\mathcal{F}_{\Pi}$.
- Upon receiving (AUTHSR,sid, $\mathcal{S}, \mathcal{U}$), // $\mathcal{S}$ is honest
  (1) If $\mathcal{GW}$ is corrupted, retrieve $g^{r_{su}}$, HID from $DX_{\mathcal{SIM}}$. Otherwise, choose random values $(g^{r_{us}})' \in_R G$, $HID \in_R \{0,1\}^*$.
  (2) Choose a random value r_{su}, generate the session key sk, M_3 using these values. Record sk in $DX_{\mathcal{SIM}}$ and send (NEWKEY,$sid, \mathcal{U}$, $\mathcal{S}$, sk) to $\mathcal{F}_{\Pi}$.
- Upon receiving (RECOVER, sid, $\mathcal{GW}$, EML, GID, ASW), // $\mathcal{U}$ is corrupted
  (1) If $\mathcal{GW}$ is corrupted, retrieve $x, name, ct_1, ct_2$ from $DX_{\mathcal{SIM}}$ and $EDB_{\mathcal{SIM}}$. Otherwise, use the previously generated dummy values. Then, increment the value $count_{rec} = count_{rec} + 1$, and record it into $DX_{\mathcal{SMI}}$.
  (2) Emulate the instance of $\mathcal{F}_{\text{ROM}}$ with query $EML||GID$ and generate the response m_1. Emulate the instance of $\mathcal{F}_{\text{VOPRF}}$ when receiving (EVAL, sid, m_1) and generate the response V_1. If an error message $(\text{VOPRF}, error)$ is returned, abort and exit.
  (3) Upon receiving $name$, check if there exists a record $(name, ct_1, ct_2)$. If so, return $(name, ct_1, ct_2)$ to $\mathcal{U}$. Else, return $\perp$.
  (4) Emulate the instance of $\mathcal{F}_{\text{ROM}}$ with query $name||EML||ASW$ and generate the response m_2. Record ASW in $DX_{\mathcal{SIM}}$. Emulate the instance of $\mathcal{F}_{\text{VOPRF}}$ when receiving (EVAL, sid, m_2) and generate the response V_2. If an error message $(\text{VOPRF}, error)$ is returned, abort and exit.
  (5) Upon receiving $(\text{RECOVER}, sid, PW||BIO||C_a)$ from $\mathcal{F}_{\Pi}$, set $\delta_2 = (PW||BIO||C_a) \oplus ct_2$, and then emulate the instance of $\mathcal{F}_{\text{ROM}}$ with query $ASW||V_2$.

Fig. 4: The simulator of $\mathcal{F}_{\Pi_1}$, part 2 of 3.

Simulator for $\mathcal{F}_{\Pi}$

The third part of the simulator $\mathcal{SIM}$ for $\mathcal{F}_{\Pi}$.

- Upon receiving (RECOVER,sid,$\mathcal{GW}$), // $\mathcal{U}$ is honest
  - (1) If $\mathcal{GW}$ is corrupted, retrieve x, $name$, ct_1, ct_2 from $DX_{\mathcal{SIM}}$ and $EDB_{\mathcal{SIM}}$. Choose dummy values EML and GID, and generate $(name, \delta_1)$ by emulating the instance of $\mathcal{F}_{\text{ROM}}$. Then, send $name$ to $\mathcal{GW}$.
  - (2) Otherwise, generate random value $name$, choose random answers ASW and interact with $\mathcal{GW}$.
- Upon receiving (RESET, sid, $\mathcal{U}$, HID, PW^{new}, BIO, ASW, $count_{ret}$), // $\mathcal{U}$ is corrupted
  - (1) If $\mathcal{GW}$ is corrupted, receive x, PID_u, $name$, C_5, and ct_3 and update related values stored into $DX_{\mathcal{SIM}}$ and $EDB_{\mathcal{SIM}}$ previously. Otherwise, retrieve the previously generated dummy values, and record $count_{ret}$ into $EDB_{\mathcal{SMI}}$.
  - (2) If receiving OK from $\mathcal{F}_{\Pi}$, replace PW with PW^{new} and update C_5 with $C_5^{new} = C_5 \oplus H_1(x||PID_u||count_{ret}-1) \oplus H_1(x||PID_u||count_{ret})$. Otherwise, make no change.
- Upon receiving (RESET, sid, $\mathcal{U}$, $count_{ret}$), // $\mathcal{U}$ is honest
  - (1)
  - (1) If $\mathcal{GW}$ is corrupted, receive x, PID_u, $name$, C_5, and ct_3 and update related values stored into $DX_{\mathcal{SIM}}$ and $EDB_{\mathcal{SIM}}$ previously. Otherwise, retrieve the previously generated dummy values, and record $count_{ret}$ into $EDB_{\mathcal{SMI}}$.
  - (2) If receiving OK from $\mathcal{F}_{\Pi}$, update C_5 with $C_5^{new} = C_5 \oplus H(x||PID_u||count_{ret}-1) \oplus H_1(x||PID_u||count_{ret}-1)$. Otherwise, make no change.
- Upon receiving (CORRUPT, sid, $\mathcal{S}$) from $\mathcal{E}$,
  - (1) Send (CORRUPT, sid, $\mathcal{S}$) to $\mathcal{F}_{\Pi}$.
  - (2) Upon receiving (sid, K_S) from $\mathcal{F}_{\Pi}$ and $(sid, k_f, name, ct_1, ct_2)$ from $\mathcal{F}_{\text{VOPRF}}$, record (K_S, k_f) in $DX_{\mathcal{SIM}}$, and record $(name, ct_1, ct_2)$ in $EDB_{\mathcal{SIM}}$.
- Upon receiving (CORRUPT, sid, $\mathcal{GW}$) from $\mathcal{E}$,
  - (1) Send (CORRUPT, sid, $\mathcal{GW}$) to $\mathcal{F}_{\Pi}$.
  - (2) Upon receiving (sid, x, PID_u, K_S) from $\mathcal{F}_{\Pi}$, record $(x, PID_u, , K_S)$ in $DX_{\mathcal{SIM}}$.
- Upon receiving (CORRUPT, sid, $\mathcal{U}$) from $\mathcal{E}$,
  - (1) Send (CORRUPT, sid, $\mathcal{U}$) to $\mathcal{F}_{\Pi}$.
  - (2) Upon receiving $(sid, PW^*, UID^*, BIO^*, C_a^*, EML^*, GID^*, ASW^*)$ from $\mathcal{F}_{\Pi}$, record $(PW^*, UID^*, BIO^*, C_a^*, EML^*, GID^*, ASW^*)$ in $DX_{\mathcal{SIM}}$.

Fig. 5: The simulator of $\mathcal{F}_{\Pi_1}$, part 3 of 3.

Functionality $\mathcal{F}_{\text{CRS}}$

$\mathcal{F}_{\text{CRS}}$ proceeds as follows, when parameterized by a distribution D.

- Upon receiving the query (PARAMETER, sid), check if it is the first input from the party:
  - (1) If so, choose a value $d \in_R D$ and send d back to the activating party.
  - (2) Otherwise, in each other activation, return the value d to the activating party.

Fig. 6: Ideal functionality $\mathcal{F}_{\text{CRS}}$. The CRS model operates under the assumption that all participating parties have access to a shared, trusted random string (CRS) prior to protocol execution. This string is generated via a trusted or verifiable process and serves as a universal public parameter within the protocol.

Functionality $\mathcal{F}_{\text{ROM}}$

$\mathcal{F}_{\text{ROM}}$ proceeds as follows,

- Upon receiving message (HASHQUERY, sid, x),
  - (1) If there is a record (x, y) in the table T_h, then output (HASHCOMPLETE, sid, y).
  - (2) Else, if $x \in$ M, choose $y \in$ Y at random, record (x, y) in T_h and output (HASHCOMPLETE, sid, y).
  - (3) Else, if $x \notin$ M, output (HASHCOMPLETE, sid, $\perp$).

Fig. 7: Ideal functionality $\mathcal{F}_{\text{ROM}}$. A table T_h is initialized to record query and response.

Functionality $\mathcal{F}_{\text{VOPRF}}$

$\mathcal{F}_{\text{VOPRF}}$ is parameterized by a security parameter κ with output length l which is polynomial in κ, it interacts with a simulator $\mathcal{SIM}$, user $\mathcal{U}$ and server $\mathcal{S}$ via the following queries:

- Upon receiving message (KEYGEN, sid) from $\mathcal{S}$, output (KEYGEN, sid, $\mathcal{S}$) to $\mathcal{SIM}$.
- Upon receiving message (PARAMETER, sid, $\mathcal{S}$, π, M) from $\mathcal{SIM}$,
  (1) If there exists a record (paramrec, sid, $\mathcal{S}$, π, M), ignore this query. // π identifies the server's PRF key
  (2) Otherwise, set params($\mathcal{S}$) $= (\pi, M)$, set ticket(π) $= 0$, and hist(π) $= \perp$. If $\mathcal{S}$ is honest, send (PARAMETER, sid, π) to $\mathcal{S}$, else parse M as a polynomial-size circuit with l-bit output and insert (π, M) in CorruptParams. // ticket(π) identifies the number of the uncompleted execution with parameter π between the user and server.
- Upon receiving message (EVAL, sid,$\mathcal{S}$,x) from $\mathcal{U}$ for $\mathcal{S}$, record (evalrec, sid, $\mathcal{U}$, x) and forward (EVAL, sid, $\mathcal{U}$, $\mathcal{S}$) to $\mathcal{SIM}$.
- Upon receiving (SENDERCOMPLETE, sid, $\mathcal{S}$) from $\mathcal{SIM}$ for honest $\mathcal{S}$, output (SENDERCOMPLETE, sid, $\mathcal{S}$) to $\mathcal{SIM}$ and set ticket(π) += 1. // ticket(π)+ = 1 denotes that $\mathcal{S}$ has completed one execution with π
- Upon receiving (USERCOMPLETE, sid, $\mathcal{U}$, π, flag) from $\mathcal{SIM}$, recover (evalrec, sid, $\mathcal{U}$, x).
  (1) If flag $= \top$ and $< \pi >=$ param($\mathcal{S}$) for some honest $\mathcal{S}$, then: if ticket(π) ≤ 0 ignore the USERCOMPLETE query of $\mathcal{SIM}$, otherwise proceed as follows. If hist(π) includes a record (x,$\rho_1^{'}$, $\rho_2^{'}$), set $\rho_1 = \rho_1^{'}$, $\rho_2 = \rho_2^{'}$, else sample ρ_1 and ρ_2 at random from $\{0,1\}^l$ and enter (x,ρ_1, ρ_2) into hist(π), Set ticket(π) $-= 1$ and output (EVALCOMPLETE, sid,π, ρ_1, ρ_2) to $\mathcal{U}$.// ticket(π)$- = 1$ denotes that user has completed one execution with π
  (2) Else, if flag $= \perp$, then return (EVALCOMPLETE, sid,π,ρ_1, ρ_2) to $\mathcal{U}$.
  (3) Else, if flag $= \top$ and π is such that (π,M) $\in$ CorruptParams for some M, compute $\rho = M(x)$ and enter (x,ρ_1, ρ_2) in hist(π). Output (EVALCOMPLETE, sid,π,ρ_1, ρ_2) to $\mathcal{U}$.

Fig. 8: Ideal functionality $\mathcal{F}_{\text{VOPRF}}$ for the protocol VOPRF. The VOPRF evaluation cycles through three activations of the functionality: KEYGEN represents the key generation for the server. EVAL that defines the input to a VOPRF computation and two activations, SENDERCOMPLETE and USERCOMPLETE, that signal the completion of the VOPRF computation by a server and user, respectively.

Functionality $\mathcal{F}_{\text{AE}}$

The functionality $\mathcal{F}_{\text{AE}}$ is parameterized by a security parameter κ, interacts with the sender $\mathcal{S}$, the receiver $\mathcal{R}$, and the simulator $\mathcal{SIM}$ using the following queries.

- Upon receiving $(\text{KEYGEN}, sid, \mathcal{S})$ from $\mathcal{R}$,
  (1) If $sid \neq (\mathcal{S}/\mathcal{R}, sid^{'})$, ignore.
  (2) Else, send $(\text{KEYGEN}, sid, \mathcal{S}, \mathcal{R})$ to $\mathcal{SIM}$.
  (3) Upon receiving OK from $\mathcal{SIM}$, generate K at random and send (sid, K) to $\mathcal{S}$ and $\mathcal{R}$. When receiving REJECT from $\mathcal{SIM}$, send $\perp$ to $\mathcal{S}$ and $\mathcal{R}$.
- Upon receiving $(\text{AUTHENC}, sid, M)$ from $\mathcal{S}$,
  (1) Send $(\text{AUTHENC}, sid, \mathcal{S})$ to $\mathcal{SIM}$.
  (2) If there exists no (sid, K), then perform $(\text{KEYGEN}, sid, \mathcal{S})$ first.
  (3) Else, generate (CT, σ) for M using K.
  (4) Record $((CT, \sigma), M)$ in database $DATA$.
  (5) Send (CT, σ) to $\mathcal{S}$.
- Upon receiving $(\text{AUTHDEC}, sid, (CT^*, \sigma^*))$ from $\mathcal{R}$,
  (1) Send $(\text{AUTHDEC}, sid, (CT^*, \sigma^*), \mathcal{R})$ to $\mathcal{SIM}$.
  (2) Upon receiving OK from $\mathcal{SIM}$, check if there exist record $((CT, \sigma), M)$ in $DATA$ where $((CT, \sigma), M) = (CT^*, \sigma^*)$. If so, retrieve M and VALID from $DATA$. Else, choose a random message M, then record $((CT^*, \sigma^*), M)$ in database $DATA$.
  (3) When receiving REJECT from $\mathcal{SIM}$, send $\perp$ to $\mathcal{R}$.
  (4) If there exist a record $((CT, \sigma), M)$ in $DATA$, where $CT = CT^*$, $\sigma \neq \sigma^*$, mark (CT^*, σ^*) as INVALID.
  (5) Send (M, VALID) or INVALID to $\mathcal{R}$.

Fig. 9: Ideal functionality $\mathcal{F}_{\text{AE}}$. A data $DATA$ is created to record the ciphertext and authentication tag.