SUPPLEMENTARY DESCRIPTION

The supplemental materials include the definition of attribute-hiding, the zero-knowledge proof and correctness analysis of ABCT and ABCB scheme, the correctness analysis and security analysis of TIPFE scheme, and the correctness definition, zero-knowledge proof and security analysis of PriSign scheme.

A. ATTRIBUTE-HIDING

Attribute-Hiding. The attribute-hiding [29] is defined by experiment Exp^{ath-b} in Fig.1. The IPFE scheme satisfies attribute-hiding, if for any probability polynomial-time (PPT) adversary $\mathcal{A}$, there is a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{ath} = \left| \Pr\left[Exp^{ath-1}(\mathcal{A}, \lambda) = 1 \right] - \frac{1}{2} \right| \leqslant \epsilon(\lambda)$$

$Exp^{ath-b}(\mathcal{A}, \lambda)$:
1. $(sk, pk) \leftarrow \mathsf{IPFE.Setup}(1^\lambda)$;
2. $(M_0, M_1, \vec{u}, st) \leftarrow \mathcal{A}(pk)$ and $\vec{u} \in \Sigma$;
3. $\mathcal{A}$ requests policy keys for any characteristic vectors $\vec{v}_i$: $dk_{\vec{v}_i} \leftarrow \mathsf{IPFE.Extract}(sk, \vec{v}_i)$, $\langle \vec{u}, \vec{v}_i \rangle \neq 0$, where $i \in [1, N]$;
4. $C \quad \mathsf{IPFE.Enc}(pk, \vec{u}, M_b)$;
5. $b^* \leftarrow \mathcal{A}(st, C, \{dk_{\vec{v}_i}\}_{i \in [1,N]})$
6. If $b = b^*$, return 1;
7. Return 0.

Fig. 1: Attribute-Hiding

B. ABCT

In this section, we present the details of ZKSoK and the correctness analysis of the ABCT scheme.

B.1 Zero-Knowledge Signature of Knowledge

B.1.1 Details of Π_1.

The proof of $\Pi_1 = \mathsf{ZKSoK}\{usk : upk = g^{usk} \wedge utk = \widetilde{g}^{usk}\}$ is presented as below.

Prove: choose $z \xleftarrow{R} \mathbb{Z}_p^*$, compute $R = g^z, \widetilde{R} = \widetilde{g}^z$, $c = \mathsf{HASH}(upk, utk, R, \widetilde{R})$, $s = z - usk \cdot c$, and output $\Pi_1 = (c, s)$.

Verify: if $c = \mathsf{HASH}(upk, utk, g^s \cdot upk^c, \widetilde{g}^s \cdot utk^c)$, output 1, 0 otherwise.

B.1.2 Details of Π_2.

The proof of $\Pi_2 = \mathsf{ZKSoK}\{(usk, r_2, \{m_j\}_{j \in [1,q]/\mathcal{D}}) : \mu = \sigma_1'^{usk} \wedge \nu = \sigma_1'^{r_2} \wedge \kappa = \widetilde{X} \prod_{j \in [1,q]/\mathcal{D}} \widetilde{Y}_j^{m_j} \widetilde{g}^{r_2}\}(\mathsf{CTX})$ is presented as below.

Prove: choose $(z_k, z_2, \{t_j\}_{j \in [1,q]/\mathcal{D}}) \xleftarrow{R} \mathbb{Z}_p^*$, compute $R_1 = \sigma_1'^{z_k}, R_2 = \sigma_1'^{z_2}, R_3 = \prod_{j \in [1,q]/\mathcal{D}} \widetilde{Y}_j^{t_j} \widetilde{g}^{z_2}$, $c = \mathsf{HASH}(\sigma_1', \widetilde{X}, \{\widetilde{Y}_j\}_{j \in [1,q]/\mathcal{D}}, R_1, R_2, R_3, \mathsf{CTX})$, $s_k = z_k - c \cdot usk$, $s_2 = z_2 - c \cdot r_2$, $\{sm_j = t_j - c \cdot m_j\}_{j \in [1,q]/\mathcal{D}}$, and output $\Pi_2 = (c, s_k, s_2, \{sm_j\}_{j \in [1,q]/\mathcal{D}})$

Verify: if $c = \mathsf{HASH}(\sigma_1', \widetilde{X}, \{\widetilde{Y}_j\}_{j \in [1,q]/\mathcal{D}}, \sigma_1'^{s_k} \mu^c, \sigma_1'^{s_2} \nu^c, \prod_{j \in [1,q]/\mathcal{D}} \widetilde{Y}_j^{sm_j} \widetilde{g}^{s_2} (\kappa / \widetilde{X})^c, \mathsf{CTX})$, output 1, 0 otherwise.

B.2 Correctness Analysis

An ABCT scheme is correct if it satisfies: (1) the credential issued by the issuer can be verified by the user; (2) an honest user's credential presentations can be verified by the verifier; (3) the real identities of the users who computed credential presentations can be traced by the issuer.

For the first property, the equation $e(\sigma_2, \widetilde{g}) = e(\sigma_1, \widetilde{X} \widetilde{Y}_0^{usk} \prod_{j=1}^q \widetilde{Y}_j^{m_j})$ holds, since we have:

$$\begin{aligned} e(\sigma_2, \widetilde{g}) &= e((upk)^{r \cdot y_0} g^{r \cdot (x + \sum_{j=1}^q m_j y_j)}, \widetilde{g}) \\ &= e(g^{r(x + usk \cdot y_0 + \sum_{j=1}^q m_j y_j)}, \widetilde{g}) \\ &= e(g^r, \widetilde{g}^{x + usk \cdot y_0 + \sum_{j=1}^q m_j y_j}) \\ &= e(\sigma_1, \widetilde{X} \widetilde{Y}_0^{usk} \prod_{j=1}^q \widetilde{Y}_j^{m_j}) \end{aligned} \tag{1}$$

For the second property, the equation $e(\sigma_2' \nu, \widetilde{g}) = e(\sigma_1', \kappa \prod_{j \in \mathcal{D}} \widetilde{Y}_j^{m_j}) e(\mu, \widetilde{Y}_0)$ holds, since we have:

$$\begin{aligned} & e(\sigma_1', \kappa \prod_{j \in \mathcal{D}} \widetilde{Y}_j^{m_j}) e(\mu, \widetilde{Y}_0) \\ &= e(\sigma_1', \widetilde{X} \prod_{j \in [1,q]/\mathcal{D}} \widetilde{Y}_j^{m_j} \widetilde{g}^{r_2} \prod_{j \in \mathcal{D}} \widetilde{Y}_j^{m_j}) e(\sigma_1'^{usk}, \widetilde{Y}_0) \\ &= e(\sigma_1', \widetilde{X} \prod_{j \in [1,q]} \widetilde{Y}_j^{m_j} \widetilde{g}^{r_2}) e(\sigma_1', \widetilde{Y}_0^{usk}) \\ &= e(\sigma_1', \widetilde{X} \widetilde{Y}_0^{usk} \prod_{j \in [1,q]} \widetilde{Y}_j^{m_j}) e(\sigma_1', \widetilde{g}^{r_2}) \\ &= e(\sigma_2', \widetilde{g}) e(\nu, \widetilde{g}) \\ &= e(\sigma_2' \nu, \widetilde{g}) \end{aligned} \tag{2}$$

For the third property, the tracing is correct because we have:

$$e(\mu, \widetilde{g}) = e(\sigma_1'^{usk}, \widetilde{g}) = e(\sigma_1', utk) \tag{3}$$

Eqs. 1, Eqs. 2 and Eqs. 3 are used to show the correctness of the ABCT scheme.

C. ABCB

In this section, we present the details of ZKSoK and the correctness analysis of the ABCB scheme.

C.1 Zero-Knowledge Signature of Knowledge

C.1.1 Details of Π_3.

The proof of $\Pi_3 = \mathsf{ZKSoK}\{(d, r_1, \cdots, r_{l'}, m_1, \cdots, m_{l'}) : D = g^d, C_j = (g^{r_j}, D^{r_j} h^{m_j}) \ \forall j \in [1, l']\}$ is presented as below.

Prove: choose $(z_d, z_1, \cdots, z_{l'}, t_1, \cdots, t_{l'}) \xleftarrow{R} \mathbb{Z}_p^*$, compute $R_d = g^{z_d}, \{R_{j,0} = g^{z_j}, R_{j,1} = D^{z_j} h^{t_j}\}_{j \in [1,l']}$, $c = \mathsf{HASH}(D, h, \{C_{j,0}, C_{j,1}\}_{j \in [1,l']}, D_d, \{R_{j,0}, R_{j,1}\}_{j \in [1,l']})$, $s_d = z_d - c \cdot d, \{sr_j = z_j - c \cdot r_j, sm_j = t_j - c \cdot m_j\}_{j \in [1,l']}$, and output $\Pi_3 = (c, s_d, \{sr_j, sm_j\}_{j \in [1,l']})$.

Verify: if $c = \mathsf{HASH}(D, h, \{C_{j,0}, C_{j,1}\}_{j \in [1,l']}, g^{s_d} D^c, \{g^{sr_j} C_{j,0}^c, D^{sr_j} h^{sm_j} C_{j,1}^c\}_{j \in [1,l']})$, output 1, 0 otherwise.

C.1.2 Details of Π_4.

The proof of $\Pi_4 = \mathsf{ZKSoK}\{(r_2, \{m_j\}_{j\in[1,l]/\mathcal{D}}) : \nu' = \tau_1'^{r_2} \wedge \kappa' = \widetilde{A}\prod_{j\in[1,l]/\mathcal{D}} \widetilde{B}_j^{m_j}\widetilde{g}^{r_2}\}$ is presented as below.

Prove: choose $(z_2, \{t_j\}_{j\in[1,l]/\mathcal{D}}) \xleftarrow{R} \mathbb{Z}_p^*$, compute $R_2 = \tau_1'^{z_2}, R_3 = \prod_{j\in[1,l]/\mathcal{D}} \widetilde{B}_j^{t_j}\widetilde{g}^{z_2}$, $c = \mathsf{HASH}(\tau_1', \widetilde{A}, \{\widetilde{B}_j\}_{j\in[1,q]/\mathcal{D}}, R_2, R_3)$, $s_2 = z_2 - c\cdot r_2, \{sm_j = t_j - c\cdot m_j\}_{j\in[1,l]/\mathcal{D}}$, and output $\Pi_4 = (c, s_2, \{sm_j\}_{j\in[1,l]/\mathcal{D}})$

Verify: if $c = \mathsf{HASH}(\tau_1', \widetilde{A}, \{\widetilde{B}_j\}_{j\in[1,l]/\mathcal{D}}, \tau_1'^{s_2}\nu'^c, \prod_{j\in[1,l]/\mathcal{D}} \widetilde{B}_j^{sm_j}\widetilde{g}^{s_2}(\kappa'/\widetilde{A})^c)$, output 1, 0 otherwise.

C.2 Correctness Analysis

An ABCB scheme is correct if it satisfies: 1) the credential issued by the issuer can be verified by the user; 2) an honest user's credential presentations can be verified by the verifier.

For the first property, the equation $e(\tau_2, \widetilde{g}) = e(\tau_1, \widetilde{A}\prod_{j=1}^{l}\widetilde{B}_j^{m_j})$ holds, since we have:

$$\begin{aligned}
&e(\tau_2, \widetilde{g})\\
&= e(\tau_1'\tau_2'^{-d}, \widetilde{g})\\
&= e(h^a\prod_{j=1}^{l'} C_{j,1}^{b_j}\cdot h^{\sum_{j=l'+1}^{l} b_j m_j}(\prod_{j=1}^{l'} C_{j,0}^{b_j})^{-d}, \widetilde{g})\\
&= e((h^a\prod_{j=1}^{l'}(D^{r_j}h^{m_j})^{b_j}h^{\sum_{j=l'+1}^{l} b_j m_j}(\prod_{j=1}^{l'} g^{r_j b_j})^{-d}, \widetilde{g})\\
&= e(h^a\prod_{j=1}^{l} h^{m_j b_j}\prod_{j=1}^{l'} D^{r_j b_j}\prod_{j=1}^{l'} g^{r_j b_j(-d)}, \widetilde{g})\\
&= e(h^a\prod_{j=1}^{l} h^{m_j b_j}, \widetilde{g})\\
&= e(h, \widetilde{g}^{a+\sum_{j=1}^{l} m_j b_j})\\
&= e(\tau_1, \widetilde{A}\prod_{j=1}^{l}\widetilde{B}_j^{m_j})
\end{aligned} \tag{4}$$

For the second property, the equation $e(\tau_2'\nu', \widetilde{g}) = e(\tau_1', \kappa'\prod_{j\in\mathcal{D}}\widetilde{B}_j^{m_j})$ holds, since we have:

$$\begin{aligned}
&e(\tau_1', \kappa'\prod_{j\in\mathcal{D}}\widetilde{B}_j^{m_j})\\
&= e(\tau_1', \widetilde{A}\prod_{j\in[1,l]/\mathcal{D}}\widetilde{B}_j^{m_j}\widetilde{g}^{r_2}\prod_{j\in\mathcal{D}}\widetilde{B}_j^{m_j})\\
&= e(\tau_1', \widetilde{A}\prod_{j\in[1,l]}\widetilde{B}_j^{m_j}\widetilde{g}^{r_2})\\
&= e(\tau_1', \widetilde{A}\prod_{j\in[1,l]}\widetilde{B}_j^{m_j})e(\tau_1', \widetilde{g}^{r_2})\\
&= e(\tau_2', \widetilde{g})e(\nu', \widetilde{g})\\
&= e(\tau_2'\nu', \widetilde{g})
\end{aligned} \tag{5}$$

Eqs. 4 and Eqs. 5 are used to show the correctness of the ABCB scheme.

D. TIPFE

In this section, we present the details of correctness analysis and formal proof of the TIPFE scheme.

D.1 Correctness Analysis

A TIPFE scheme is correct if it satisfies: (1) the key extracted by each policymaker for verifiers is correct; (2) the aggregation of t shares of policy key is correct; (3) the decryption is correct if the attribute vector matches the policy i.e. $<\vec{u}, \vec{v}>= 0$.

For the first property, a received share of policy key is $dk_{id,\vec{v},i}$, and the verifier verifies it as $e(g, dk_{id,\vec{v},i}) = P_i^{\theta_{id}}\prod_{j=1}^{k} H_{i,j}^{v_j}$. This equation holds, since we have:

$$\begin{aligned}
e(g, dk_{id,\vec{v},i}) &= e(g, \widetilde{g}^{\sum_{j=1}^{k} s_{i,j}v_j + e_i\theta_{id}})\\
&= e(g, \widetilde{g}^{e_i})^{\theta_{id}}\prod_{j=1}^{k} e(g, \widetilde{g}^{s_{i,j}})^{v_j}\\
&= P_i^{\theta_{id}}\prod_{j=1}^{k} H_{i,j}^{v_j}
\end{aligned} \tag{6}$$

For the second property, a complete policy key should satisfy the equation $dk_{id,\vec{v}} = \widetilde{g}^{\sum_{j=1}^{k} s_j v_j + e\theta_{id}}$. This equation holds since we have:

$$\begin{aligned}
dk_{id,\vec{v}} &= \prod_{i\in[1,t]} dk_{id,\vec{v},i}^{\lambda_i}\\
&= \widetilde{g}^{\sum_{i\in[1,t]}\lambda_i(\sum_{j=1}^{k} s_{i,j}v_j + e_i\theta_{id})}\\
&= \widetilde{g}^{\sum_{i\in[1,t]}\sum_{j=1}^{k}\lambda_i s_{i,j}v_j + \sum_{i\in[1,t]}\lambda_i e_i\theta_{id}}\\
&= \widetilde{g}^{\sum_{j=1}^{k} s_j v_j + e\theta_{id}}
\end{aligned} \tag{7}$$

For the third property, if $<\vec{u}, \vec{v}>= 0$, decryption is successful since:

$$\begin{aligned}
&C_0\cdot(\frac{\prod_{j=1}^{k} C_{2,j}^{v_j}}{e(C_1, tk_{id,\vec{v}})})^{\theta_{id}^{-1}}\\
&= M\cdot P^r\cdot(\frac{(H_1^r G^{u_1})^{v_1}(H_2^r G^{u_2})^{v_2}\cdot(H_k^r G^{u_k})^{v_k}}{e(g^r, tk_{id,\vec{v}})})^{\theta_{id}^{-1}}\\
&= M\cdot(G^{e\cdot r\theta_{id}}\frac{G^{<\vec{u},\vec{v}>}G^{r(<\vec{s},\vec{v}>)}}{e(g,\widetilde{g})^{r(<\vec{s},\vec{v}>+e\theta_{id})}})^{\theta_{id}^{-1}}\\
&= M
\end{aligned} \tag{8}$$

Eqs. 6, Eqs. 7 and Eqs. 8 are used to show the correctness of the TIPFE scheme.

D.2 Security Analysis

***Theorem 4.1*:** Our TIPFE scheme satisfies attribute-hiding under the BDDH assumption.

Proof: Let $(g^{w_a}, g^{w_b}, \widetilde{g}^{w_a}, \widetilde{g}^{w_c}, e(g,\widetilde{g})^{w_z})$ be a BDDH challenge in $\mathcal{BL}$. $\mathcal{A}$ provides two distinct messages M_0 and M_1, and an attribute vector $\vec{u}$. The simulator $\mathcal{S}$ progresses as follows:

– $\mathcal{S}$ finds a $(k-1)$ basis of subspace $(\vec{u})^{\perp}$ and denotes the basis by $(\mu_1, \mu_2, \cdots, \mu_{k-1})$. $\mathcal{S}$ chooses $z_i \xleftarrow{R} \mathbb{Z}_p^*$ for $i\in[1, k-1]$.

– For $i\in[1,k]$, $\mathcal{S}$ computes $\nu_i = \alpha_i\vec{u} + \sum_{i=1}^{k-1}\lambda_i\mu_i$ to obtain the canonical basis $(\nu_1, \cdots, \nu_k)$, where $<\nu_i, \vec{u}>= \alpha_i\cdot||\vec{u}||^2$. Then, $\mathcal{S}$ computes $\alpha = \sum_{i=1}^{k}\nu_i\alpha_i = 1/||\vec{u}||^2\cdot\vec{u}$.

– $\mathcal{S}$ sets $G = e(g, \widetilde{g}^{w_c})$. $\forall i\in[1,k]$, $\mathcal{S}$ computes $H_i = e((g^{w_a})^{\alpha_i}g^{\sum_{j=1}^{k-1} z_j\lambda_j}, \widetilde{g}^{w_c}) = G^{a\alpha_i + \sum_{j=1}^{k-1} z_j\lambda_j}$. Note that $s_i = a\alpha_i + \sum_{j=1}^{k-1} z_j\lambda_j$.

– $\mathcal{S}$ chooses $e \xleftarrow{R} \mathbb{Z}_p^*$, computes $P = G^e$, and sets $pk = (G, P, H_1, H_2, \cdots, H_k)$.

– For a policy $\vec{v} = (v_1, \cdots, v_k) \in (\vec{u})^\perp$, $\mathcal{S}$ computes $\sum_{j=1}^k s_j v_j = \sum_{j=1}^k v_j z_j \lambda_j$. To achieve the threshold key distribution, $\mathcal{S}$ Chooses 2 polynomials of degree $t-1$: f_{sv}, f_e, where $f_{sv}(0) = \sum_{j=1}^k s_j v_j$, $f_e(0) = e$, and $\forall i \in [1,k]$ computes $sv_i = f_{sv}(i), e_i = f_e(i)$.

– For identity id and policy $\vec{v}$, $\mathcal{S}$ computes $\theta_{id} = \mathsf{HASH}(id)$ and returns $dk_{id,\vec{v},i} = \widetilde{g}^{sv_i + e_i \theta_{id}}$.

– In the challenge stage, for $b \in \{0,1\}$, $\mathcal{S}$ choose $r' \in \mathbb{Z}_p^*$ and returns $C^* = (C_0^*, C_1^*, C_2^*)$, where $C_0^* = M_b e(g^{w_b \cdot r'}, \widetilde{g}^{w_c}), C_1^* = g^{w_b \cdot r'}$, $C_{3,i}^* = e(g,\widetilde{g})^{w_z} e((g^{w_b \cdot r'})^{\sum_{j=1}^{k-1} z_j \lambda_j}, \widetilde{g}^{w_c}) G^{u_i}, i \in [1,k]$, which implicitly defines $r = r' \cdot b$.

If $w_a \cdot w_b \cdot w_c = w_z$, then we can see that (C_0^*, C_1^*, C_2^*) are identically distributed as in the $\mathsf{TIPFE.Enc}$ algorithm, because $C_0^* = M_b G^r, C_1^* = g^r, C_{3,i}^* = e(g,\widetilde{g})^{w_z} e((g^{w_b \cdot r'})^{\sum_{j=1}^{k-1} z_j \lambda_j} = H_i^r G^{u_i}, i \in [1,k]$. Otherwise, (C_0^*, C_1^*, C_2^*) are randoms for $\mathcal{A}$. Since if $\mathcal{A}$ can break the attribute-hiding with non-negligible probability, $\mathcal{S}$ can break the BDDH problem with equal probability.

E. PRISIGN

In this section, we present the details of correctness definition, ZKSoK, and security analysis of the PriSign scheme.

E.1 Formal Correctness Definition

Correctness. A PriSign scheme is correct if it satisfies:

$$Pr\left[\begin{array}{c|l} \begin{array}{c}\mathsf{PriSign.Trace}(msk, \mathcal{L}, pst) \to uid \\ \mathsf{PriSign.Verify}(vid, tok, tk_{vid,\vec{v}}, mpk) \to 1\end{array} & \begin{array}{l}\mathsf{PriSign.Setup}(1^\lambda, q, n, t, k) \to (pp, msk, mpk, \mathcal{L}); \\ \mathsf{PriSign.IssuerKeyGen}() \to (isk, ipk); \\ \mathsf{PriSign.IssuerReg}\langle \mathsf{I}(isk, ipk) \leftrightarrow \mathsf{CA}(msk, mpk)\rangle \to icred; \\ \mathsf{PriSign.PolMakKeyGen}(msk, mpk) \to (\{psk_i, pvk_i\}_{i\in[1,n]}); \\ \{\mathsf{PriSign.IssPolKey}(psk_i, vid, \vec{v}) \to tk_{vid,\vec{v},i}\}_{i\in[1,t]}; \\ \mathsf{PriSign.AggrPolKey}(vid, \vec{v}, \{tk_{vid,\vec{v},i}, mpk_i\}_{i\in[1,t]}) \to tk_{vid,\vec{v}}; \\ \mathsf{PriSign.UserKeyGen}() \to (usk, upk, utk); \\ \mathsf{PriSign.UserReg}\langle \mathsf{U}(uid, usk, upk, utk, \{m_j\}_{j\in[1,q]}) \leftrightarrow \mathsf{CA}(msk, mpk, \mathcal{L})\rangle \to (ucred, \mathcal{L}'); \\ \mathsf{PriSign.ObtTkt}\langle \mathsf{U}(usk, upk, ucred, \{m_j\}_{j\in[1,q]}, \mathcal{D}, \mathsf{CTX}) \leftrightarrow \mathsf{I}(isk, ipk, icred)\rangle \to (pst, tkt, dsid, VP); \\ \mathsf{PriSign.SignOn}(tkt, dsid, VP, \vec{u}, \vec{v}, mpk) \to tok. \end{array} \end{array}\right] = 1$$

E.2 Details of Π_5.

The proof of $\Pi_5 = \mathsf{ZKSoK}\{(a, b_1, \cdots, b_l) : \widetilde{A} = \widetilde{g}^a \wedge \widetilde{B}_i = \widetilde{g}^{b_i}, \forall 1 \leqslant i \leqslant l\}$ is presented as below.

Prove: choose $(z_a, z_1, \cdots, z_l) \xleftarrow{R} \mathbb{Z}_p^*$, compute $R_a = \widetilde{g}^{z_a}, \widetilde{R}_1 = \widetilde{g}^{z_1}, \cdots, \widetilde{R}_l = \widetilde{g}^{z_l}$, $c = \mathsf{HASH}(\widetilde{A}, \widetilde{B}_1, \cdots, \widetilde{B}_l, R_a, R_1, \cdots, R_l)$, $s_a = z_a - c \cdot a, s_1 = z_1 - c \cdot b_1, \cdots, s_l = z_l - c \cdot b_l$, and output $\Pi_5 = (c, s_a, s_1, \cdots, s_l)$.

Verify: if $c = \mathsf{HASH}(\widetilde{A}, \widetilde{B}_1, \cdots, \widetilde{B}_l, \widetilde{g}^{s_a} \cdot \widetilde{A}^c, \widetilde{g}^{s_1} \cdot \widetilde{B}_1^c, \cdots, \widetilde{g}^{s_l} \cdot \widetilde{B}_l^c)$, output 1, 0 otherwise.

E.3 Security Analysis

E.3.1 Unforgeability of credentials

***Theorem 4.3*:** Our PriSign scheme satisfies the unforgeability of credentials if the SDL assumption holds and the PS signature is unforgeable.

Proof: The PPT adversary $\mathcal{A}$ wins the experiment of unforgeability of credentials if he forges a credential presentation of an honest user or an unregistered user in $\mathsf{PriSign.ObtTkt}$ algorithm, therefore we distinguish two types of adversary:

Type-1: $\mathcal{A}$ forges $icred_{uid^*}$, where $\exists uid \in \mathcal{HU}$, and $usk_{uid} = usk_{uid^*}$;

Type-2: $\mathcal{A}$ forges $icred_{uid^*}$, where $\forall uid \in \mathcal{HU} \cup \mathcal{CU}$, and $usk_{uid} \neq usk_{uid^*}$;

where usk_{uid} is the private key of user uid and $icred_{uid}$ is the credential of user uid.

Lemma 1. If there is a type-1 adversary $\mathcal{A}$ that breaks the unforgeability of credentials with the probability ϵ, then there is a simulator $\mathcal{S}$ that breaks the SDL assumption with the probability $\frac{\epsilon}{N}$, where N is a bound on the number of honest users.

Proof: Let $(g, \widetilde{g}, g^w, \widetilde{g}^w)$ be a SDL challenge in $\mathcal{BL}$. $\mathcal{S}$ uses $\mathcal{BL}$ as the parameter, runs $(pp, msk, mpk, \mathcal{L}) \leftarrow \mathsf{PriSign.Setup}(1^\lambda, q, n, t, k)$, and send (pp, mpk) to $\mathcal{A}$. Since we are considering a type-1 adversary, there is an honest user with the identity uid^* that the adversary is trying to emulate. $\mathcal{S}$ makes a guess on $uid^* \in \mathcal{HU}$ and answers the oracles' queries as follows:

$\mathcal{O}_{\mathsf{Reg}_U}(uid, \{m_j\}_{j\in[1,q]})$: if $uid \neq uid^*$, then $\mathcal{S}$ proceeds as usual. Otherwise, it sets $upk = g^w$ and $utk = \widetilde{g}^w$, and then $\mathcal{S}$ completes the remaining operations by simulating ZKSoK Π_1.

$\mathcal{O}_{\mathsf{Cor}_U}(uid)$: if $uid \neq uid^*$, then $\mathcal{S}$ proceeds as usual. Otherwise, it returns $\perp$.

$\mathcal{O}_{\mathsf{Obt}}(uid, \mathcal{D}, \mathsf{CTX})$: if $uid \neq uid^*$, then $\mathcal{S}$ proceeds as usual. Otherwise, it completes the remaining operations by simulating ZKSoK Π_2.

$\mathcal{O}_{\mathsf{Reg}_I}, \mathcal{O}_{\mathsf{Obt}_I}$: $\mathcal{S}$ knows the secret keys of CA and the ticket issuer, so it can perfectly simulate these oracles.

If $\mathcal{S}$ guesses the uid^* successfully, then the simulations of the oracles is perfect, and the probability of its occurrence is $\frac{1}{N}$. In this case, if $\mathcal{A}$ successfully forges a credential presentation pst^* of user uid^*, then $\mathcal{S}$ can use the knowledge extractor of Π_2 to recover w, which is a valid solution to the SDL assumption. The probability of success for $\mathcal{S}$ is $\frac{\epsilon}{N}$.

Lemma 2. If there is a type-2 adversary $\mathcal{A}$ that breaks the unforgeability of credentials with the probability ϵ, then there is a simulator $\mathcal{S}$ that breaks the unforgeability of the PS signature with the same probability.

Proof: $\mathcal{S}$ runs the unforgeability game of the PS signature and so receives a public parameters $\mathcal{BL}$ and public key $pk =$

$(\widetilde{X}, \widetilde{Y}_0, \cdots, \widetilde{Y}_q)$. $\mathcal{S}$ sets $mpk_i = pk$, uses $\mathcal{BL}$ as the parameter, runs $(pp, msk, mpk, \mathcal{L}) \leftarrow \mathsf{PriSign.Setup}(1^\lambda, q, n, t, k)$, and sends (pp, mpk) to $\mathcal{A}$. $\mathcal{S}$ can query the signature oracle of PS, $\mathcal{O}_{PS.Sign}(\cdot)$, and answer oracles queries as follows:

$\mathcal{O}_{\mathsf{Reg}_U}(uid, \{m_j\}_{j\in[1,q]})$: $\mathcal{S}$ generates $(usk_{uid}, upk_{uid}, utk_{uid})$ for user uid, and submits $(usk_{uid}, m_1, \cdots, m_q)$ to the signature oracle $\mathcal{O}_{PS.Sign}(\cdot)$. It then receives a PS signature σ and sets $ucred = \sigma$.

$\mathcal{O}_{\mathsf{Reg}_I}$: $\mathcal{S}$ knows CA's secret keys used to issue credential for the ticket issuer, so it can perfectly simulate this oracle.

$\mathcal{O}_{\mathsf{Cor}_U}, \mathcal{O}_{\mathsf{Obt}}, \mathcal{O}_{\mathsf{Obt}_I}$: $\mathcal{S}$ knows the secret keys of the ticket issuer and all honest users, so it can perfectly simulate these oracles.

$\mathcal{S}$ can simulate these oracles perfectly and never aborts. Since we are considering a type-2 adversary, the adversary outputs a valid credential presentation $pst = (\{m_j\}_{j\in[1,q]}, \sigma_1', \sigma_2', \mu, \nu, \kappa, \Pi_2)$ with the probability ϵ at the end of the experiment; $\mathcal{S}$ can recover usk_{uid^*} using the extractor of Π_2 and $\forall uid \in \mathcal{HU} \cup \mathcal{CU}, usk_{uid^*} \neq usk_{uid}$. So $\mathcal{S}$ forges a valid PS signature (σ_1', σ_2') on the message $(usk_{uid^*}, \{m_j\}_{j\in[1,q]})$ with the probability ϵ.

E.3.2 Unforgeability of tokens

***Theorem 4.4*:** Our PriSign scheme satisfies the unforgeability of tokens if the PS signature is unforgeable.

Proof: $\mathcal{S}$ runs the unforgeability game of the PS signature and so receives the public parameters $\mathcal{BL}$ and public key $pk = (\widetilde{A}, \widetilde{B}_0, \widetilde{B}_1, B_0, B_1)$. $\mathcal{S}$ uses $\mathcal{BL}$ as the parameter, runs $(pp, msk, mpk, \mathcal{L}) \leftarrow \mathsf{PriSign.Setup}(1^\lambda, q, n, t, k)$, and sends (pp, mpk) to $\mathcal{A}$. $\mathcal{S}$ can query the signature oracle of PS $\mathcal{O}_{PS.Sign}(\cdot)$ and answer oracles queries as follows:

$\mathcal{O}_{\mathsf{Reg}_I}()$: $\mathcal{S}$ sets the ticket issuer's public key $ipk = pk$ and issues credential for the issuer.

$\mathcal{O}_{\mathsf{Obt}}(uid, \mathcal{D}, \mathsf{CTX})$: $\mathcal{S}$ generates $dsid$ and VP for a ticket, and submits $(dsid, VP)$ to the signature oracle $\mathcal{O}_{PS.Sign}(\cdot)$. It then receives a PS signature σ and sets the ticket $\tau = \sigma$.

$\mathcal{O}_{\mathsf{Iss}_V}, \mathcal{O}_{\mathsf{Reg}_U}, \mathcal{O}_{\mathsf{Cor}_U}, \mathcal{O}_{\mathsf{Sign}}$: $\mathcal{S}$ knows the secret keys of CA and all honest users, so it can perfectly simulate these oracles.

$\mathcal{S}$ can simulate these oracles perfectly and never aborts. At the end of the experiment, the adversary outputs a forged-yet-valid token $tok = (CM, CT)$ with the probability ϵ; $\mathcal{S}$ can recover $tok' = (\{dsid, VP\}, \tau_1', \tau_2')$ using the poly key of the verifier. Since $tok' \notin \mathcal{Q}_{\mathsf{TOK}}$, $\mathcal{S}$ forges a valid PS signature (τ_1', τ_2') on the message $(dsid, VP)$ with the probability ϵ.

E.3.3 Unlinkability of credentials

***Theorem 4.5*:** Our PriSign scheme satisfies the unlinkability of credentials if the DDH assumption holds in $\mathbb{G}_1$.

Proof: Let $(g, g^{w_1}, g^{w_2}, g^{w_3})$ be a DDH challenge in $\mathbb{G}_1$. $\mathcal{S}$ uses $(\mathbb{G}_1, g)$ as the parameter, runs $(pp, msk, mpk, \mathcal{L})$ $\mathsf{PriSign.Setup}(1^\lambda, q, n, t, k)$, and sends (pp, mpk) to $\mathcal{A}$. In the experiment, $\mathcal{S}$ makes a guess on $uid^* \in \mathcal{HU}$ that the adversary is trying to break, and answers the oracles' queries as follows:

$\mathcal{O}_{\mathsf{Reg}_U}(uid, \{m_j\}_{j\in[1,q]})$: If $uid \neq uid^*$, then $\mathcal{S}$ proceeds as usual. Otherwise, it assigns $upk_{uid^*} = g^{w_1}$. Then, it simulates the ZKSoK Π_1, chooses $r_1 \xleftarrow{R} \mathbb{Z}_p^*$ and computes $\sigma_1 = g^{r_1}, \sigma_2 = g^{w_1 y_0 r_1} g^{r_1 \cdot (x + \sum_{j\in[1,q]} m_j y_j)}$. We note that $\mathcal{S}$ can use global variables $\mathcal{Q}_{\mathsf{TKT}}$ to trace user identities directly without tracing keys.

$\mathcal{O}_{\mathsf{Cor}_U}(uid)$: If $uid \neq uid^*$, then $\mathcal{S}$ proceeds as usual. Otherwise, it returns $\perp$.

$\mathcal{O}_{\mathsf{Obt}}, \mathcal{O}_{\mathsf{Obt}_U}$: If $uid \neq uid^*$, then $\mathcal{S}$ proceeds as usual. Otherwise, it simulates the ZKSoK Π_2.

$\mathcal{O}_{\mathsf{Reg}_I}, \mathcal{O}_{\mathsf{Cor}_I}, \mathcal{O}_{\mathsf{Iss}_V}$: $\mathcal{S}$ knows the secret keys of CA, the ticket issuer and all verifiers, so it can perfectly simulate these oracles.

At some stage in the experiment, $\mathcal{A}$ outputs the identities uid_0 and uid_1 of two honest users along with an attribute disclosure policy $\mathcal{D}^*$. If $uid_b \neq uid^*$, then $\mathcal{S}$ aborts. Otherwise, it proceeds as follows.

$\mathcal{O}_{\mathsf{UnlCred}_b}(uid_0, uid_1, \mathcal{D}, \mathsf{CTX})$: $\mathcal{S}$ chooses $r_2 \xleftarrow{R} \mathbb{Z}_p^*$, computes $\sigma_1' = (g^{w_2})^{r_1}, \sigma_2' = \sigma_2^{w_2} = g^{w_3 y_0 r_1} (g^{w_2})^{r_1 \cdot (x + \sum_{j\in[1,q]} m_j y_j)}$, $\mu = (g^{w_3})^{r_1}, \nu = \sigma_1'^{r_2}, \kappa = \widetilde{X} \prod_{j\in[1,q]/\mathcal{D}} \widetilde{Y}_j^{m_j} \widetilde{g}^{r_2}$. Then, it simulates the ZKSoK Π_2 and completes the remaining operations.

If $w_1 \cdot w_2 = w_3$, then we can see that $(\sigma_1', \sigma_2', \mu, \nu, \kappa)$ are identically distributed as in the $\mathsf{PriSign.ObtTkt}$ algorithm. Otherwise, w_3 is random element, which means that σ_2' is random. Therefore, the behaviors of $\mathcal{A}$ can be used to solve the DDH assumption in $\mathbb{G}_1$, unless $\mathcal{S}$ aborts. The advantage of $\mathcal{S}$ is at least $\frac{\epsilon}{N}$, where N is a bound on the number of honest users.

E.3.4 Unlinkability of tokens

***Theorem 4.6*:** Our PriSign scheme satisfies the unlinkability of tokens if the ABCB scheme satisfies the unlinkability.

Proof: $\mathcal{S}$ runs the unlinkability game of the ABCB scheme and so receives the public parameters $\mathcal{BL}$ and public key $pk = (\widetilde{A}, \widetilde{B}_0, \widetilde{B}_1, B_0, B_1)$. $\mathcal{S}$ uses $\mathcal{BL}$ as the parameter, runs $(pp, msk, mpk, \mathcal{L}) \leftarrow \mathsf{PriSign.Setup}(1^\lambda, q, n, t, k)$, and sends (pp, mpk) to $\mathcal{A}$. $\mathcal{S}$ can query the oracles of ABCB and answer oracles queries as follows:

$\mathcal{O}_{\mathsf{Iss}_V}, \mathcal{O}_{\mathsf{Cor}_V}, \mathcal{O}_{\mathsf{Reg}_U}, \mathcal{O}_{\mathsf{Cor}_U}, \mathcal{O}_{\mathsf{Obt}_U}, \mathcal{O}_{\mathsf{Sign}}$: $\mathcal{S}$ knows the secret keys of CA and all users, so it can perfectly simulate these oracles.

$\mathcal{O}_{\mathsf{Reg}_I}()$: $\mathcal{S}$ sets the ticket issuer's public key $ipk = pk$ and issues credential for the issuer.

$\mathcal{O}_{\mathsf{Obt}}(uid, \mathcal{D}, \mathsf{CTX})$: $\mathcal{S}$ generates $dsid$ and VP for a ticket, and submits $(dsid, VP)$ to the ABCB's issuing oracle $\mathcal{O}_{\mathsf{ABCB.Issue}}(\cdot)$. It then receives a credential σ and sets the ticket $\tau = \sigma$.

At some stage in the experiment, $\mathcal{A}$ outputs two honest user uid_0 and uid_1 and a designated verifier vid with the policy $\vec{v}$.

$\mathcal{O}_{\mathsf{UnlTkt}_b}(uid_0, uid_1, \vec{u}, vid, \vec{v}, \mathsf{CTX})$: If $uid_0 \notin \mathcal{HU}$ or $uid_1 \notin \mathcal{HU}$, then $\mathcal{S}$ aborts. Otherwise, it submits uid_0, uid_1 to the challenge oracle $\mathcal{O}_{\mathsf{ABCB.Unl}_b}(\cdot)$, and then receives a token tok_b'. It encrypts tok_b' with $\vec{u}$ and returns the ciphertext tok_b.

$\mathcal{S}$ can simulate these oracles perfectly and never aborts. At the end of the experiment, the adversary outputs a guess b^* with the probability ϵ. Because the adversary knows the policy key, it can decrypts tok_b, then outputs tok_b' and make a guess. So $\mathcal{S}$ can return $\mathcal{A}$'s guess as its own, thus turning $\mathcal{A}$'s advantage in attacking PriSign system into an advantage in attacking ABCB scheme.