Proofs for the t-out-of-n Setting and Weak Partial Blindness

1 Formal Proof

In this section, we provide the security proof in the style of [1] of our protocol in the t-out-of-n setting with weak partial blindness, adopting modifications in sections 3.2 (t-out-of-n) and 4.1 (weak partial blindness) of our original submission. We present the ideal functionality in t-out-of-n setting with weak partial blindness in Fig. 1, the protocols of key generation phase, the signing phase without weak partial blindness, and the weak partial blindness signing in Fig. 2, Fig. 3, and Fig. 4, respectively. Besides, we explain the difference between the functionality in Fig. 1 and the one defined in [1] in section 2.

In the t-out-of-n setting, $\mathcal{T}$ denotes the set of signing parties. The parts that are mainly distinct from those in the n-out-of-n setting are marked by *.

The same as [1], our weak, partially blind signing protocol supports the application: coalescing of credentials from multiple authorities, as well. Specifically, a user recognized by multiple credential authorities wants to authenticate to a service by providing a joint statement about who these authorities believe the user to be and what each of them (individually) authorizes the user to do. On the other hand, the user may not wish to reveal to one authority their relationship with another authority. Both [1] and our weak, partially blind signing protocol allow the user to privately prove different predicates to each authority. The user requests a vector of messages to be signed in a weak-blind manner, with each message representing an authority's credential. Each authority independently verifies its message by checking equality with a fixed string. This results in a compact credential that combines the user's relationships with all authorities.

For the weak partially blind signing phase, the Pedersen commitment is formed as

$$B_0 := G_1 + s_0 \cdot H_1 + \sum_{i\in[\ell]} m_i \cdot H_{i+1}$$

which information-theoretically hides the message $\mathbf{m}$ using the randomness s_0. The relation that the user, namely the client, proves to each authority is the conjunction of an arbitrary predicate and accumulated discrete logarithm, which is defined as

$$\begin{aligned}\mathcal{R}_{\mathsf{dl}\wedge\phi} := \{((X, B_1, \cdots, B_\ell), (x_1, \cdots, x_\ell))|\\ X = G_1 + \sum_{i\in\ell} x_i \cdot B_i \wedge \phi(x_1, \cdots, x_\ell) = 1\}\end{aligned}$$

where ϕ varies for each signing party, meaning it differs for each authority involved in credential coalescing.

Our weak partially blind signing scheme utilizes the zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{dl}\wedge\phi}}$ for predicates $\{\phi_i\}_{i\in\mathcal{T}}$. The realization of this functionality can be found in [1]. Although the description of $\mathcal{R}_{\mathsf{dl}\wedge\phi}$ in [1] differs from ours—specifically, the value X in their relation is computed as $\sum_{i\in\ell} x_i \cdot B_i$ rather than $G_1 + \sum_{i\in\ell} x_i \cdot B_i$—we can still leverage their realization. This is possible because the structures of the witnesses are consistent, allowing us to invoke their functionality by inputting $((X - G_1, B_1, \cdots, B_\ell), (x_1, \cdots, x_\ell))$. Below is our security theorem.

Theorem 1. *(**t-out-of-n** setting) Let public parameter* $\mathsf{pp} := (\mathcal{G}, \mathsf{e}, \ell)$ *where* $\mathcal{G} \leftarrow \mathsf{BilinearGen}(1^\lambda)$. *If CL encryption is IND-CPA secure and assume the existence of (programable) random oracle* $\mathcal{RO}(\cdot)$ *with output domain* $\mathcal{D}_\mathcal{H}$, *then the protocol* $\mathsf{SET\text{-}BBS+}$ *securely realizes* $\mathcal{F}_{\mathsf{BBS+}}(t)$ *in the* $(\mathcal{F}_{\mathsf{COM}}, \mathcal{F}_{\mathsf{GenPub}}, \mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}\wedge\mathsf{dl}}}, \mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}rand}}}, \mathcal{F}_{\mathsf{PVSS}}(t), \mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{dl}\wedge\phi}})$-*hybrid model in the presence of a malicious static adversary under the security parameters* λ *and* λ_d.

Prior to the formal proof, we first give a proof sketch below, including the key generation and signing. We don't include weak partially blind signing in the sketch because it has a very similar simulation to the original BBS+ signing.

Proof Sketch. In the key generation phase, we fix a specific honest party P_k such that the simulator $\mathcal{S}$ doesn't know his BBS+ secret key sharing. Upon receiving the public key $(\mathbf{H}, X)$ from $\mathcal{F}_{\mathsf{BBS+}}(t)$, $\mathcal{S}$ can call corrupted parties to obtain their public (i.e., $X_i, \mathbf{H}_i$) and secret key shares (i.e. x_i) via $\mathcal{F}_{\mathsf{COM}}$, $\mathcal{F}_{\mathsf{GenPub}}$, and $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}\wedge\mathsf{dl}}}$, thus can adjust the public

The functionality $\mathcal{F}_{\mathsf{BBS+}}(t)$ operates with n parties, denoted as $P_1, \cdots, P_n$, and interacts with a client $\mathcal{C}$. An ideal adversary is represented by $\mathcal{S}$, who can corrupt up to $t-1$ parties at the beginning (static corruption). During the key generation phase, $\mathcal{S}$ may instruct $\mathcal{F}_{\mathsf{BBS+}}(t)$ to abort. During any signing phase, $\mathcal{S}$ may instruct $\mathcal{F}_{\mathsf{BBS+}}(t)$ to abort or output a "failure" to $\mathcal{C}$ instead of a signature. $\mathcal{F}_{\mathsf{BBS+}}(t)$ supports two functions working as follows:
Keygen: On receiving $(\mathsf{Kgen}, \mathsf{sid})$ from some party P_i for $i \in [n]$ with fresh sid, send $(\mathsf{Kgen}, \mathsf{sid}, i)$ to $\mathcal{S}$. On receiving $(\mathsf{Kgen}, \mathsf{sid})$ from all parties,

- Obtain $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Kgen}(1^\lambda)$.
- Send $(\mathsf{pub\text{-}key}, \mathsf{sid}, \mathsf{pk})$ to $\mathcal{S}$ and to all parties as adversarially-delayed private output.
- Store $(\mathsf{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ in memory and ignore any furthur calls related to sid.

Sign: On receiving $(\mathsf{Sign}, \mathsf{sid}, \mathsf{ssid}, \mathbf{m} \in \mathbb{Z}_q^\ell, \mathcal{T})$ from a client $\mathcal{C}$ with a fresh sid, if a record of the form $(\mathsf{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ does not exist in memory or $|\mathcal{T}| < t$, this call is ignored. Otherwise, send $(\mathsf{Sign}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathbf{m}, \mathcal{T})$ to all parties indexed by $\mathcal{T}$. Upon receiving $(\mathsf{accept}, \mathsf{sid}, \mathsf{ssid})$ from all parties indexed by $\mathcal{T}$, $\mathcal{F}_{\mathsf{BBS+}}(t)$ executes as follows,

- Obtain $(A, e, s) \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathbf{m})$.
- Send $(\mathsf{leakage}, \mathsf{sid}, \mathsf{ssid}, (e, s))$ to $\mathcal{S}$.
- Send $(\mathsf{signature}, \mathsf{sid}, \mathsf{ssid}, (\mathbf{H}, X), (A, e, s))$ to $\mathcal{C}$ as adversarially-delayed private output.

Weak Partially Blind Sign: On receiving $(\mathsf{wb\_Sign}, \mathsf{sid}, \mathsf{ssid}, \mathbf{m} \in \mathbb{Z}_q^\ell, \mathcal{T})$ from a client $\mathcal{C}$ with a fresh sid, if a record of the form $(\mathsf{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ does not exist in memory or $|\mathcal{T}| < t$, this call is ignored. Otherwise, wait to receive $(\mathsf{wb\_Sign}, \mathsf{sid}, \mathsf{ssid}, i, \phi_i)$ from $\mathcal{C}$, and pass $(\mathsf{wb\_Sign}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, i, \phi_i)$ to party P_i if $i \in \mathcal{T}$ and ϕ_i is the description of a predicate on $\mathbf{m}$ s.t. $\phi_i(\mathbf{m}) = 1$. Upon receiving $(\mathsf{accept}, \mathsf{sid}, \mathsf{ssid})$ from all parties indexed by $\mathcal{T}$, $\mathcal{F}_{\mathsf{BBS+}}(t)$ executes as follows,

- Obtain $(A, e, s) \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathbf{m})$.
- Send $(\mathsf{leakage}, \mathsf{sid}, \mathsf{ssid}, (B, e))$ to $\mathcal{S}$ where
$$B = G_1 + s \cdot H_1 + \sum_{i \in [\ell]} m_i \cdot H_{i+1}.$$
- Send $(\mathsf{signature}, \mathsf{sid}, \mathsf{ssid}, (\mathbf{H}, X), (A, e, s))$ to $\mathcal{C}$ as adversarially-delayed private output.

Figure 1: The functionality $\mathcal{F}_{\mathsf{BBS+}}(t)$ for weak partially blind signing.

key $(\mathbf{H}_k, X_k)$ for honest user P_k (without knowing the corresponding secret key) so that the resulting public key is $(\mathbf{H}, X)$. After that, $\mathcal{S}$ samples a random $\tilde{x_k}$ and generates the encryption $\mathsf{CL.Enc}(\tilde{x_k})$ to substitute for the encryption of the unknown secret key share x_k where x_k is implicitly implied in X_k's discrete-logarithm and $\tilde{x_k}$ is independent of x_k. In addition, $\mathcal{S}$ can extract the secret information (e.g., zero-share seeds $\{(\beta_{i,j}, \beta_{j,i})\}$ and secret shares of partial decryption key $\{(s_{i,j}, s_{j,i})\}$) of every corrupted party.

In the signing phase, $\mathcal{S}$ always extract (γ_j, r_{γ_j}) for all corrupted parties via $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}rand}}}$ and thus deduce their "honest" values $(\beta_{\mathcal{T},j}, d_{\mathcal{T},j}, B_j)$ along with $\{s_{i,j}\}$ and $\{\beta_{i,j}\}$. Here "honest" values refer to the expected value when corrupted parties act honestly. When the client $\mathcal{C}$ is honest, $\mathcal{S}$ acts as $\mathcal{C}$ to compute the offset deviated from the "honest" values and check whether the signing succeeds. When the client $\mathcal{C}$ is corrupted, $\mathcal{S}$ can receive (A, e, s) from $\mathcal{F}_{\mathsf{BBS+}}(t)$ on behalf of $\mathcal{C}$. Using the deduced "honest" values of corrupted parties, $\mathcal{S}$ can correctly sample $(z_i, B_i, pd_{i,y})$ for all honest parties and manipulate them to send these values so that $\mathcal{C}$ can conduct the signature as (A, e, s) if they honestly use their secret values.

The security of our protocol can be reduced to CL's CPA security. The real-world distribution can be generated from X_k and $\mathsf{CL.Enc}(x_k)$, while the ideal-world distribution can be generated from X_k and $\mathsf{CL.Enc}(\tilde{x_k})$. The difference between the pairs $(X_k, \mathsf{CL.Enc}(x_k))$ and $(X_k, \mathsf{CL.Enc}(\tilde{x_k}))$ is computationally bounded by CL's CPA security.

The formal proof is given below. For better readability, we rewrite the simulator and the reduction to make it more Intuitive. Meanwhile, we add more games so that distinguishing games from the real world to the ideal world is smoother. We also use * to indicate the simulation for the added/significantly modified parts. In addition, we provide some insights into how the transformation in Section 3.2 of our original paper facilitates our protocol to work in the t-out-of-n setting in section 3. These insights are a more detailed explanation of Section 3.2 in the original submission.

Proof. Let $\mathcal{A}$ be the adversary that corrupts up to $t-1$ parties at the beginning of the protocol, and we use $\mathbf{P}^*$ to represent the index set of corrupted parties index. Let $\overline{\mathbf{P}^*} := [n] \setminus \mathbf{P}^*$ be the index set of honest parties, $k \in \overline{\mathbf{P}^*}$ be a fixed index and $\overline{\mathbf{P}^*}' := \overline{\mathbf{P}^*} \setminus \{k\}$. We use "honest people" to refer to all honest parties and honest clients. We first construct a simulator $\mathcal{S}^{\mathcal{A}}$ with access to $\mathcal{A}$ to simulate $\mathcal{A}$'s view and real-world behaviors. Note that when sampling an element from the cyclic group $\langle h \rangle$, an exponent r is initially drawn from the distribution $\mathcal{D}_{\mathcal{H}}$, and the resulting element is set to h^r. This sampling method is statistically equivalent to a uniform sample from $\langle h \rangle$.
Keygen. $\mathcal{S}^{\mathcal{A}}$ would immediately instruct $\mathcal{F}_{\mathsf{BBS+}}(t)$ to abort if any check fails:

1. Upon receiving the key generation command $(\mathsf{Kgen}, \mathsf{sid}, i)$ for some $i \in [n]$, $\mathcal{S}^{\mathcal{A}}$ directly sends

Keygen of SET-BBS+, involving n parties $P_1, \cdots, P_n$ in the t-out-of-n setting.

Party P_i, where $i \in [n]$ | **All** $\{P_j\}_{j \in [n] \setminus \{i\}}$

CL public key, X-part of verification key, and commitment of H-part of verification key

$d_i \leftarrow \mathcal{D}_{\mathcal{H}}, D_i := h^{d_i}$ $\xrightarrow{(\text{gen-pub}, \text{sid}, D_i, d_i, \mathcal{H})}$ $\mathcal{F}_{\mathsf{GenPub}}$ $\xrightarrow{(\text{gen-key}, \{D_i\}_{i \in [n]}, D, \mathcal{H})}$ CL public key T

$x_i \overset{\$}{\leftarrow} \mathbb{Z}_q, X_i := x_i \cdot G_2$ $\xrightarrow{(\text{gen-pub}, \text{sid}, X_i, x_i, \mathbb{G}_2)}$ $\mathcal{F}_{\mathsf{GenPub}}$ $\xrightarrow{(\text{gen-key}, \{X_i\}_{i \in [n]}, X, \mathbb{G}_2)}$ X-part of verification key

$\mathbf{H}_i \overset{\$}{\leftarrow} \mathbb{G}_1^{\ell+1}$ $\xrightarrow{(\text{commit}, \text{sid}, [n], \mathbf{H}_i)}$ $\mathcal{F}_{\mathsf{COM}}$ $\xrightarrow{(\text{committed}, \text{sid}, [n], i)}$ **H**-part of verification key

Decommitment of H **and CL encryption of** x

$r_{x_i} \leftarrow \mathcal{D}_{\mathcal{H}}$

$(u_{x_i}, v_{x_i}) \leftarrow \mathsf{CL.Enc}(D, x_i, r_{x_i})$ $\xrightarrow{(\text{prove}, \text{sid}, [n], (u_{x_i}, v_{x_i}, X_i), (x_i, r_{x_i}))}$ $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}enc} \wedge \mathsf{dl}}}$ $\xrightarrow{(\text{proof}, \text{sid}, [n], i, (u_{x_i}, v_{x_i}, X_i), b_i)}$ if $b_i = 0$, **abort**

$(u_x, v_x) \leftarrow \mathsf{CL.Add}(\{u_{x_j}, v_{x_j}\}_{j \in [n]})$

$\xrightarrow{(\text{decommit}, \text{sid})}$ $\mathcal{F}_{\mathsf{COM}}$ $\xrightarrow{(\text{decommitted}, \text{sid}, [n], i, \mathbf{H}_i)}$

$\mathbf{H} := \sum_{j \in [n]} \mathbf{H}_j$

Store $(\mathbf{H}, X, (u_x, v_x))$

***Secret shares of** d$_i$ **and zero shares**

$\xrightarrow{(\text{dealing}, \text{sid}, d_i)}$ $\mathcal{F}_{\mathsf{PVSS}}(t)$ $\xrightarrow{(\text{share}, \text{sid}, [n], i, \mathcal{M}_i)}$ if $\mathcal{M}_i = \perp$, **abort**

parse $(\{A_{j,k}\}_{k=0}^{t-1}, s_{j,i}) \leftarrow \mathcal{M}_i$

if $A_{j,0} \neq D_j$, **abort**

Store $\{s_{j,i}\}_{j \in [n]}$

Sample $\{\beta'_{i,j}\}_{j \in [n] \setminus \{i\}} \leftarrow \mathcal{D}_{\mathcal{H}}^{n-1}$ $\xrightarrow{(\text{zeroshare}, \text{sid}, j, \beta'_{i,j})}$ if $j > i$, $\beta_{j,i} := \beta'_{j,i} - \beta'_{i,j}$

Else, $\beta_{j,i} := \beta'_{i,j} - \beta'_{j,i}$

Store $\{\beta_{i,j}\}_{j \in [n] \setminus \{i\}}$

Figure 2: The key generation phase of SET-BBS+.

Sign of SET-BBS+ on message $\mathbf{m} \in \mathbb{Z}_q^\ell$ and session id $(\mathrm{sid}, \mathrm{ssid})$, involving $|\mathcal{T}| \geq t$ parties indexed by $\mathcal{T}$ and the client $\mathcal{C}$.

Party P_i, where $i \in \mathcal{T}$ — **Party** P_j for $j \in \mathcal{T}$

Round one: commit to nonces (e_i, s_i) **and encrypt** $\gamma_i x + \rho_i$

$e_i, s_i \xleftarrow{\$} \mathbb{Z}_q$ $\quad \xrightarrow{(\mathsf{commit}, \mathrm{sid}\|\mathrm{ssid}, \mathcal{T}, (e_i, s_i))} \mathcal{F}_{\mathsf{COM}} \xrightarrow{(\mathsf{committed}, \mathrm{sid}\|\mathrm{ssid}, \mathcal{T}, i)}$

$\gamma_i, \rho_i \xleftarrow{\$} \mathbb{Z}_q, r_{y_i} \leftarrow \mathcal{D}_{\mathcal{H}}$

$(u_{\gamma_i x}, v_{\gamma_i x}) \leftarrow \mathsf{CL.Scal}(D, (u_x, v_x), \gamma_i, r_{y_i})$

$(u_{y_i}, v_{y_i}) := (u_{\gamma_i x}, v_{\gamma_i x} f^{\rho_i})$ $\quad \xrightarrow{(\mathsf{prove}, \ldots, \mathcal{T}, (u_x, u_{y_i}), (\gamma_i, r_{y_i}))} \mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}rand}}} \xrightarrow{(\ldots, \mathcal{T}, i, (u_x, u_{y_i}), b_i)}$ $\quad$ if $b_i = 0$, **abort**

$$u_y = \prod_{j \in \mathcal{T}} u_{y_j}$$

Round two: reveal (e_i, s_i)

$\xrightarrow{(\mathsf{decommit}, \mathrm{sid}\|\mathrm{ssid})} \mathcal{F}_{\mathsf{COM}} \xrightarrow{(\cdots, \mathcal{T}, i, (e_i, s_i))}$

$$e := \sum_{j \in \mathcal{T}} e_j, s := \sum_{j \in \mathcal{T}} s_j$$

Party P_i, where $i \in \mathcal{T}$ — **The client** $\mathcal{C}$

Open $\gamma_i e - \rho_i, B_i$, **partially decrypt** $\gamma x + \rho$ **to** $\mathcal{C}$

$$*\beta_{\mathcal{T},i} := \sum_{j \in \mathcal{T}, j<i} \beta_{i,j} - \sum_{j \in \mathcal{T}, j>i} \beta_{i,j}$$

$$*d_{\mathcal{T},i} := \Delta L_{\mathcal{T},i} \sum_{j \in [n]} \Delta s_{j,i}$$

$$z_i := \gamma_i e - \rho_i$$

$$B_i := \gamma_i \cdot (G_1 + s \cdot H_1 + \sum_{j \in [\ell]} m_j \cdot H_{j+1})$$

$*pd_{i,y} := \dfrac{v_{y_i}^{\Delta^3}}{u_y^{d_{\mathcal{T},i} + \beta_{\mathcal{T},i}}}$ $\quad \xrightarrow{(\ldots, (\mathbf{H}, X), (e, s), z_i, B_i, pd_{i,y})}$ $\quad$ if $(\mathbf{H}, X, e, s)$ inconsistent, failure

$*$Update $\beta_{i,j} \leftarrow \mathcal{RO}(\beta_{i,j})$

$$*y := \mathsf{Solve}_{\mathsf{DL}}(\prod_{j \in \mathcal{T}} pd_{j,y})\Delta^{-3} \mod q$$

if $y = \perp$, failure

$$A := \frac{\sum_{i \in \mathcal{T}} B_i}{y + \sum_{i \in \mathcal{T}} z_i}$$

if $\mathsf{Verify}(A, e, s) = 0$, failure

otherwise, output (A, e, s)

Figure 3: The signing phase of SET-BBS+.

$(\mathsf{Kgen}, \mathrm{sid})$ to $\mathcal{F}_{\mathsf{BBS+}}(t)$ and receives back a public key $\mathsf{pk} = (\mathbf{H}, X)$. Meanwhile, $\mathcal{S}^{\mathcal{A}}$ also instructs corrupted parties the key generation command.

2. $\mathcal{S}^{\mathcal{A}}$ samples the CL public key D from $\langle h \rangle$.[1]

3. $\mathcal{S}^{\mathcal{A}}$ runs honest parties' code to generate $(x_i, X_i, d_i, D_i, \mathbf{H}_i)$ for every $i \in \overline{\mathbf{P}^*}'$.

4. $\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{F}_{\mathsf{GenPub}}$ and $\mathcal{F}_{\mathsf{COM}}$ for publicly constructing $(\mathbf{H}, X)$ and D. First, $\mathcal{S}^{\mathcal{A}}$ checks the validity of (D_j, d_j) and (X_j, x_j) for every corrupted party

[1]Actually, readers may assume $\mathcal{S}^{\mathcal{A}}$ also samples the CL secret/decryption key d, as done in our original proof. However, neither of our proofs uses d to decrypt; we only use D to encrypt, which allows us to reduce the security of our protocol to CPA security. Thus, we let $\mathcal{S}^{\mathcal{A}}$ only store D here to avoid misunderstanding.

Weak partially blind signing on $\mathbf{m} \in \mathbb{Z}_q^{\ell}$ and session id $(\mathrm{sid}, \mathrm{ssid})$, involving $|\mathcal{T}| \geq t$ parties indexed by $\mathcal{T}$ and the client $\mathcal{C}$.

Party P_i, where $i \in \mathcal{T}$ | **Party** P_j for $j \in \mathcal{T}$

Round one: commit to nonces (e_i, s_i) **and encrypt** $\gamma_i x + \rho_i$

$e_i, s_i \xleftarrow{\$} \mathbb{Z}_q$ $\quad \xrightarrow{(\text{commit}, \text{sid}\|\text{ssid}, \mathcal{T}, (e_i, s_i))} \mathcal{F}_{\text{COM}} \xrightarrow{(\text{committed}, \text{sid}\|\text{ssid}, \mathcal{T}, i)}$

$\gamma_i, \rho_i \xleftarrow{\$} \mathbb{Z}_q, r_{y_i} \leftarrow \mathcal{D}_{\mathcal{H}}$

$(u_{\gamma_i x}, v_{\gamma_i x}) \leftarrow \mathsf{CL.Scal}(D, (u_x, v_x), \gamma_i, r_{y_i})$

$(u_{y_i}, v_{y_i}) := (u_{\gamma_i x}, v_{\gamma_i x} f^{\rho_i})$ $\quad \xrightarrow{(\text{prove}, \ldots, \mathcal{T}, (u_x, u_{y_i}), (\gamma_i, r_{y_i}))} \mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{cl-rand}}} \xrightarrow{(\ldots, \mathcal{T}, i, (u_x, u_{y_i}), b_i)}$ $\quad$ if $b_i = 0$, **abort**

$u_y = \prod_{j \in \mathcal{T}} u_{y_j}$

Round two: reveal (e_i, s_i)

$\xrightarrow{(\text{decommit}, \text{sid}\|\text{ssid})} \mathcal{F}_{\text{COM}} \xrightarrow{(\cdots, \mathcal{T}, i, (e_i, s_i))}$

$e := \sum_{j \in \mathcal{T}} e_j, s := \sum_{j \in \mathcal{T}} s_j$

Party P_i, where $i \in \mathcal{T}$ | **The client** $\mathcal{C}$

Open $\gamma_i e - \rho_i, B_i$, **partially decrypt** $\gamma x + \rho$ **to** $\mathcal{C}$

$\beta_{\mathcal{T},i} := \sum_{j \in \mathcal{T}, j<i} \beta_{i,j} - \sum_{j \in \mathcal{T}, j>i} \beta_{i,j}$

$d_{\mathcal{T},i} := \Delta L_{\mathcal{T},i} \sum_{j \in [n]} \Delta s_{j,i}$

$z_i := \gamma_i e - \rho_i$

$*B_i := \gamma_i \cdot (B_0 + s \cdot H_1)$

$pd_{i,y} := \dfrac{v_{y_i}^{\Delta^3}}{u_y^{d_{\mathcal{T},i} + \beta_{\mathcal{T},i}}}$ $\quad \xrightarrow{(\ldots, (\mathbf{H}, X), (e, s), z_i, B_i, pd_{i,y})}$ $\quad$ if $(\mathbf{H}, X, e, s)$ inconsistent, failure

Update $\beta_{i,j} \leftarrow \mathcal{RO}(\beta_{i,j})$ $\quad$ $y := \mathsf{Solve}_{\mathsf{DL}}(\prod_{j \in \mathcal{T}} pd_{j,y})\Delta^{-3} \mod q$

if $y = \perp$, failure

$A := \dfrac{\sum_{i \in \mathcal{T}} B_i}{y + \sum_{i \in \mathcal{T}} z_i}$

$*s' := s_0 + s$

$*$if $\mathsf{Verify}(A, e, s') = 0$, failure

$*$otherwise, output (A, e, s')

Figure 4: The weak partially blind signing phase of SET-BBS+. Before the above process begins, $\mathcal{C}$ must broadcast B_0 to all signing parties and send the proof message $(\text{proof}, \text{sid} \parallel \text{ssid}, \{i\}, \mathcal{C}, (B_0, \mathbf{H}), b)$ with $b = 1$ to party P_i (if $b = 0$, then party P_i will simply ignore this message) through $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{dl}\wedge\phi_i}}$ for every $i \in \mathcal{T}$.

P_j. Then, $\mathcal{S}^{\mathcal{A}}$ computes $X_k := X - \sum_{i \in [n] \setminus \{k\}} X_i$, $D_k = D / \prod_{i \in [n] \setminus \{k\}} D_i$ and $\mathbf{H}_k = \mathbf{H} - \sum_{i \in [n] \setminus \{k\}} \mathbf{H}_i$. Lastly, $\mathcal{S}^{\mathcal{A}}$ distributes $\{X_i, D_i, \mathbf{H}_i\}_{i \in [n]}$ to all corrupted parties.

5. $\mathcal{S}^{\mathcal{A}}$ runs honest parties' code to generate ciphertexts (u_{x_i}, v_{x_i}) for every $i \in \overline{\mathbf{P}^*}'$ and generates a ciphertext (u_{x_k}, v_{x_k}) for a uniformly sampled plaintext $\tilde{x}_k \leftarrow \mathbb{Z}_q$.

6. $\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{cl-enc}\wedge\text{dl}}}$ for publicly constructing (u_x, v_x). Firstly, $\mathcal{S}^{\mathcal{A}}$ checks the validity of $((u_{x_j}, v_{x_j}, X_j), (x_j, r_{x_j}))$ for every corrupted party

P_j. Meanwhile, $\mathcal{S}^{\mathcal{A}}$ distributes $\{(u_{x_i}, v_{x_i})\}_{i \in \overline{\mathbf{P}^*}}$ to all corrupted parties. Lastly, $\mathcal{S}^{\mathcal{A}}$ conducts (u_x, v_x) as $\mathsf{CL.Add}(\{u_{x_i}, v_{x_i}\}_{i\in[n]})$.

7. *$\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{F}_{\mathsf{PVSS}}(t)$ to distribute $(\{A_{i,p}\}_{p=0}^{t-1}, s_{i,j})$ from party P_i to party P_j for every $(i,j) \in \overline{\mathbf{P}^*} \times \mathbf{P}^*$. For $i \in \overline{\mathbf{P}^*}'$, $(\{A_{i,p}\}_{p=0}^{t-1}, s_{i,j})$ are generated as described in $\mathcal{F}_{\mathsf{PVSS}}(t)$ with input d_i. For k, $\{s_{k,j}\}_{j\in\mathbf{P}^*}$ are uniformly sampled and $A_{k,0} := D_k$, $\{A_{k,p}\}_{p=1}^{t-1}$ are then calculated using Lagrange Interpolation as non-constant term coefficients of the degree-$(t-1)$ polynomial $F(\cdot)$ fixed by $F(0) = D_k^{\Delta^2}$ and $F(j) = h^{\Delta s_{j,i}}$.[2] $\mathcal{S}^{\mathcal{A}}$ stores $\{s_{i,j}\}_{(i,j)\in\overline{\mathbf{P}^*}\times\mathbf{P}^*}$.

8. *$\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{F}_{\mathsf{PVSS}}(t)$ also to check the validity of $(\{A_{j,p}\}_{p=0}^{t-1}, s_{j,i})$ for every $(i,j) \in \overline{\mathbf{P}^*} \times \mathbf{P}^*$ and extract $s_{j,i}$ for every $(i,j) \in [n] \times \mathbf{P}^*$. $\mathcal{S}^{\mathcal{A}}$ stores $\{s_{i,j}\}_{(i,j)\in\mathbf{P}^*\times\mathbf{P}^*}$.

9. *$\mathcal{S}^{\mathcal{A}}$ plays the role of honest party P_i to send randomly-generated $\beta'_{i,j}$ to and receive $\beta'_{j,i}$ from corrupted party P_j for every $(i,j) \in \overline{\mathbf{P}^*} \times \mathbf{P}^*$. After that, $\mathcal{S}^{\mathcal{A}}$ computes and extracts $\{\beta_{j,i}\}_{(i,j)\in[n]\times\mathbf{P}^*}$. $\mathcal{S}^{\mathcal{A}}$ stores $\{\beta_{i,j}\}_{(i,j)\in[n]\times\mathbf{P}^*}$.

Given the signing/weak partially blind signing parties as $\mathcal{T}$, we use $\mathcal{T}^* := \mathcal{T} \cap \mathbf{P}^*$ to denote the set of corrupted parties indexed within $\mathcal{T}$, and $\overline{\mathcal{T}^*} := \mathcal{T} \cap \overline{\mathbf{P}^*}$ to denote the set of honest parties indexed within $\mathcal{T}$. We fix the index k' of an arbitrary honest signing party and define $\overline{\mathcal{T}^*}' := \overline{\mathcal{T}^*} \setminus \{k'\}$.
Signing. $\mathcal{S}^{\mathcal{A}}$ would immediately instruct $\mathcal{F}_{\mathsf{BBS+}}(t)$ to abort if any check fails:

10. Upon receiving the signing command $(\mathsf{Sign}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathbf{m} \in \mathbb{Z}_q^l, \mathcal{T})$ for some $\mathcal{T} \subseteq [n]$ with $|\mathcal{T}| \geq t$, $\mathcal{S}^{\mathcal{A}}$ directly sends $(\mathsf{Sign}, \mathsf{sid}, \mathsf{ssid}, \mathbf{m}, \mathcal{T})$ to $\mathcal{F}_{\mathsf{BBS+}}(t)$ and receives back the leakage (e,s). Meanwhile, $\mathcal{S}^{\mathcal{A}}$ also instructs corrupted parties the signing command.

11. $\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{F}_{\mathsf{COM}}$ to send "commit" messages of honest parties to corrupted parties. Meanwhile, $\mathcal{S}^{\mathcal{A}}$ extracts $\{(e_j, s_j)\}_{j\in\mathcal{T}^*}$.

12. $\mathcal{S}^{\mathcal{A}}$ runs honest parties' code to generate ciphertexts (u_{y_i}, v_{y_i}) for every $i \in \overline{\mathcal{T}^*}'$ and sample $u_{y_{k'}}$ from $\langle h \rangle$.

13. $\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}rand}}}$ for publicly constructing u_y. Firstly, $\mathcal{S}^{\mathcal{A}}$ checks the validity of $((u_x, v_{y_j}), (\gamma_j, r_{y_j}))$ for every corrupted signing party P_j. Meanwhile, $\mathcal{S}^{\mathcal{A}}$ distributes u_{y_i} for every honest signing party P_i to all corrupted parties. Lastly, $\mathcal{S}^{\mathcal{A}}$ conducts $u_y := \prod_{i\in\mathcal{T}} u_{y_i}$.

[2]We let the exponent be a multiple of Δ so that Δ times the Lagrange coefficient is still an integer.

14. $\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{F}_{\mathsf{COM}}$ to for publicly constructing (e,s). Firstly, $\mathcal{S}^{\mathcal{A}}$ uniformly samples $\{(e_i, s_i)\}_{i\in\overline{\mathcal{T}^*}}$ conditioned on $\sum_{i\in\mathcal{T}} e_i = e$ and $\sum_{i\in\mathcal{T}} s_i = s$. Then, $\mathcal{S}^{\mathcal{A}}$ decommits and releases $\{(e_i, s_i)\}_{i\in\mathcal{T}}$ to all corrupted parties.

15. *For every $j \in \mathcal{T}^*$, $\mathcal{S}^{\mathcal{A}}$ deduces $\beta_{\mathcal{T},j}$, $d_{\mathcal{T},j}$ and B_j using extracted/stored information $\{s_{i,j}\}_{i\in[n]}$, $\{\beta_{i,j}\}_{i\in\mathcal{T}}$ and (γ_j, r_{γ_j}). Then, $\mathcal{S}^{\mathcal{A}}$ defaultly sets $\rho_j = 0$ and deduces z_j, v_{y_j} and $pd_{j,y}$.

16. *This step is executed when $\mathcal{C}$ is corrupted. $\mathcal{S}^{\mathcal{A}}$ first receives the signature (A,e,s) from $\mathcal{F}_{\mathsf{BBS+}}(t)$ as the corrupted $\mathcal{C}$. For $i \in \overline{\mathcal{T}^*}$, $\mathcal{S}^{\mathcal{A}}$ runs honest parties' code to generates (z_i, B_i) but sets $pd_{i,y} := \frac{v_{y_i}^{\Delta^3}}{u_y^{\alpha_i}}$ for a random $\alpha_i \leftarrow \mathcal{D}_{\mathcal{H}}$. For k', $\mathcal{S}^{\mathcal{A}}$ samples $y \xleftarrow{\$} \mathbb{Z}_q$ and sets
    - $z_{k'} \xleftarrow{\$} \mathbb{Z}_q$;
    - $B_{k'} = (y + \sum_{i\in\mathcal{T}} z_i) \cdot A - \sum_{i\in\mathcal{T}\setminus\{k'\}} B_i$;
    - $pd_{k',y} := \frac{f^{\Delta^3 y}}{\prod_{i\in\mathcal{T}\setminus\{k'\}} pd_{i,y}}$.

    Lastly, $\mathcal{S}^{\mathcal{A}}$ plays the role of honest signing parties to send $\{(z_i, B_i, pd_{i,y})\}_{i\in\overline{\mathcal{T}^*}}$ to $\mathcal{C}$.

17. *This step is executed when $\mathcal{C}$ is honest. $\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{C}$ to receive $\{(z'_j, B'_j, pd'_{j,y})\}_{j\in\overline{\mathcal{T}^*}}$ from corrupted parties as their replied information. $\mathcal{S}^{\mathcal{A}}$ first extracts the sum ρ' of random offsets $\{\rho_j\}_{j\in\mathcal{T}^*}$, then computes the offset deviated from the honest actions:
    - $\rho' := \sum_{j\in\mathcal{T}^*} z_j - \sum_{j\in\mathcal{T}^*} z'_j$;
    - $\Delta_B := \sum_{j\in\mathcal{T}^*} B_j - \sum_{j\in\mathcal{T}^*} B'_j$;
    - $\Delta_v := \mathsf{Solve}_{\mathsf{DL}}(\frac{\prod_{j\in\mathcal{T}^*} pd'_{j,y}}{f^{\Delta^3\rho'} \prod_{j\in\mathcal{T}^*} pd_{j,y}})\Delta^{-3} \mod q$.

    $\mathcal{S}^{\mathcal{A}}$ instructs $\mathcal{F}_{\mathsf{BBS+}}(t)$ to output a failure to $\mathcal{C}$ if one of the following conditions appears:
    - $\mathsf{Solve}_{\mathsf{DL}}(\cdot)$ outputs $\perp$ when computing Δ_v.
    - Only one of Δ_B and Δ_v is non-zero.
    - Both Δ_B and Δ_v are non-zero, but $\mathsf{Verify}((\mathbf{H}, X), \mathbf{m}, (\frac{\Delta_B}{\Delta_v}, e, s)) = 0$, that is, $(\frac{\Delta_B}{\Delta_v}, e, s)$ is not a valid signature.

    Otherwise, $\mathcal{S}^{\mathcal{A}}$ instructs $\mathcal{F}_{\mathsf{BBS+}}(t)$ to output the signature of current signing command to $\mathcal{C}$.

18. *$\mathcal{S}^{\mathcal{A}}$ updates the stored information $\{\beta_{i,j}\}_{(i,j)\in\mathcal{T}^*\times\mathcal{T}}$ using $\mathcal{RO}(\cdot)$.

Weak Partially blind Signing. $\mathcal{S}^{\mathcal{A}}$ would immediately instruct $\mathcal{F}_{\mathsf{BBS+}}(t)$ to abort if any check fails:

19. If $\mathcal{C}$ is corrupted, $\mathcal{S}^{\mathcal{A}}$ can direcly receive the signature (A,e,s')[3] from the signature and extract $(s_0,\mathbf{m})$ from $\mathcal{C}$ by acting $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{dl}\wedge\phi}}$. After distributing B_0 and the corresponding proofs and predicates to signing parties, $\mathcal{S}^{\mathcal{A}}$ runs $\mathcal{S}^{\mathcal{A}}$'s code from steps 11 to 18 with the following modifications:

    - The random nonce s is computed as $s := s' - s_0$.
    - The deduced B_i for every $i \in \mathbf{P}^*$ is computed as

    $$\gamma_i \cdot (G_1 + s' \cdot H_1 + \sum_{j\in[\ell]} m_j \cdot H_{j+1}).$$

    instead of

    $$\gamma_i \cdot (G_1 + s \cdot H_1 + \sum_{j\in[\ell]} m_j \cdot H_{j+1}).$$

20. If $\mathcal{C}$ is honest, $\mathcal{S}^{\mathcal{A}}$ can direcly receive the signature (B,e) form $\mathcal{F}_{\mathsf{BBS+}}(t)$. $\mathcal{S}^{\mathcal{A}}$ uniformly selects $s \xleftarrow{\$} \mathbb{Z}_q$ and computes $B_0 := B - s \cdot H_1$, then $\mathcal{S}^{\mathcal{A}}$ plays the role of $\mathcal{C}$ and $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{dl}\wedge\phi}}$ to distribute B_0 and the corresponding proofs and predicates to signing parties, after that, $\mathcal{S}^{\mathcal{A}}$ runs $\mathcal{S}^{\mathcal{A}}$'s code from steps 11 to 18 with the following modifications:

    - $\mathcal{S}^{\mathcal{A}}$ doesn't receive $\mathbf{m}$ and the random nonce s is randomly selects s by $\mathcal{S}^{\mathcal{A}}$.
    - The deduced B_i for every $i \in \mathbf{P}^*$ is computed as

    $$\gamma_i \cdot (B_0 + s \cdot H_1)$$

    instead of

    $$\gamma_i \cdot (G_1 + s \cdot H_1 + \sum_{j\in[\ell]} m_j \cdot H_{j+1}).$$

    - When both Δ_B and Δ_v are non-zero, the condition

    $$\mathsf{Verify}((\mathbf{H},X),\mathbf{m},(\frac{\Delta_B}{\Delta_v},e,s)) = 0$$

    is replaced by

    $$\mathsf{e}(\frac{\Delta_B}{\Delta_v}, X + e \cdot G_2) \neq \mathsf{e}(B, G_2). \quad (1)$$

Note that the equation 1 is essentially equivalent as:

$$\mathsf{Verify}((\mathbf{H},X),\mathbf{m},(\frac{\Delta_B}{\Delta_v},e,s')) = 0,$$

where s' is the corresponding nonce that is hidden from $\mathcal{S}^{\mathcal{A}}$ in weak partially blind signing.

[3]Here, the notation s' is the random nonce s described in the functionality description. We use another notation to show the difference between signing and weak partially blind signing.

We use hybrid games to bound the difference between the real-world distribution and the ideal-world distribution. Each hybrid's distribution should be the joint distribution of $\mathcal{A}$'s view[4] and honest people's output. Hybrid $\mathcal{H}_0$ is denoted as the real-world distribution.

Hybrid $\mathcal{H}_1$. In $\mathcal{H}_1$, we define an initial simulator $\mathcal{S}$ that replaces all the honest and ideal functionalities but simulates their roles by running their codes. Note that $\mathcal{S}$ also captures any value sent from corrupted parties to honest people and ideal functionalities. Meanwhile, $\mathcal{S}$ also knows all secret values used by honest parties. The difference between $\mathcal{H}_1$ and H_0 is purely syntactic; $\mathcal{H}_1 = \mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. In $\mathcal{H}_2$, part of $\mathcal{S}$'s code in the key generation phase is modified. Specifically, $\mathcal{S}$ previously samples BBS+ public-secret key pair $((\mathbf{H},X),x)$ and CL public-secret key (D,d), and manipulates honest parties and functionalities $\mathcal{F}_{\mathsf{GenPub}}$ and $\mathcal{F}_{\mathsf{COM}}$ to make the final outcoming BBS+ and CL public keys be as $(\mathbf{H},X)$ and D (similar to steps 3 and 4 of $\mathcal{S}^{\mathcal{A}}$). The function provided by $\mathcal{F}_{\mathsf{GenPub}}$ and $\mathcal{F}_{\mathsf{COM}}$ makes the distribution of $\mathcal{H}_2$ and $\mathcal{H}_1$ identical, that is, $\mathcal{H}_2 = \mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. In $\mathcal{H}_3$, part of $\mathcal{S}$'s code in the signing and partially blind signing phase is modified. Specifically, $\mathcal{S}$ previously samples the random nonce (e,s) and manipulates honest parties and functionalities $\mathcal{F}_{\mathsf{COM}}$ to make the final outcoming random nonce as (e,s), similar to steps 11 and 14 and their modifications in weak partially blind signing of $\mathcal{S}^{\mathcal{A}}$). Note that s isn't randomly selected in the **modified steps** 11 and 14 specified in step 19, but the overall distribution is uniform since $s = s' - s_0$ and s' is uniformly sampled by the description of BBS+ signing algorithm. The function provided by $\mathcal{F}_{\mathsf{COM}}$ makes the distributions of $\mathcal{H}_3$ and $\mathcal{H}_2$ identical, that is, $\mathcal{H}_3 = \mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. In $\mathcal{H}_4$, we alter the code of $\mathcal{S}$ when an honest $\mathcal{C}$ conducts the signature in the signing phase. Specifically, $\mathcal{S}$ runs steps 12, 15 and 17 to determine whether or not to output a failure. Note that step 12 doesn't impact any of $\mathcal{A}$'s view since $v_{y'_k}$ is entirely hidden in $\mathcal{A}$'s view and the distribution of $u_{k'}$ is statically close to the one in $\mathcal{H}_3$ due to $\mathcal{D}_H$'s property. If the signing phase doesn't fail, $\mathcal{S}$ computes (A,e,s) directly from BBS+ secret key x and (e,s) and outputs the signature on behalf of $\mathcal{C}$. Let $\gamma := \sum_{i\in\mathcal{T}} \gamma_i B$, where γ_i is extracted using ideal functionalities, A should satisfy

$$A = \frac{\gamma B}{\gamma(x+e)}.$$

However, the offsets maliciously made by $\mathcal{A}$ will make an honest $\mathcal{C}$ to conduct the A-part of a signature as

$$\frac{\gamma B + \Delta_B}{\gamma(x+e) + \Delta_v}.$$

Since A is fixed by $(\mathbf{m},e,s)$, the verification passes if and only if (Δ_B,Δ_v) are both zero or $A = \frac{\gamma B + \Delta_B}{\gamma(x+e)+\Delta_v}$, which is equivalent

[4]We can assume that the hybrid programs invoke $\mathcal{A}$ to output its view.

to $A = \frac{\Delta_B}{\Delta_v}$. Therefore, step 17's strategy fully captures whether the signing phase successes. This means the distributions of $\mathcal{H}_4$ and $\mathcal{H}_3$ are statistically close, that is, $\mathcal{H}_4 \approx_s \mathcal{H}_3$.

Hybrid $\mathcal{H}_5$. In $\mathcal{H}_5$, we alter the code of $\mathcal{S}$ when an honest $\mathcal{C}$ conducts the signature in the weak partially blind signing phase. Specifically, $\mathcal{S}$ runs **modified steps** 12, 15 and 17 specified in step 20 to determine whether or not to output a failure. If the signing phase doesn't fail, $\mathcal{S}$ computes (A, e, s') directly from BBS+ secret key x and (e, s') and outputs the signature on behalf of $\mathcal{C}$. The randomness s in the original BBS+ signing has the same impact as s' in computing B. Thus, the method for computing B remains consistent from signing to weak partially blind signing if we treat s' as the s in signing. Therefore, we can apply the same security argument of $\mathcal{H}_4$ to have $\mathcal{H}_5 \approx_s \mathcal{H}_4$.

Hybrid $\mathcal{H}_6$. In $\mathcal{H}_6$, we alter the code of $\mathcal{S}$ when a corrupted $\mathcal{C}$ conducts the signature in the signing phase. Specifically, $\mathcal{S}$ runs steps 12, 15 and 16 to determine the honest signing parties' sending information $\{(z_i, B_i, pd_{i,y})\}_{i \in \overline{\mathcal{T}^*}}$. Note that step 12 doesn't impact any of $\mathcal{A}$'s view since $v_{y'_k}$ is entirely hidden in $\mathcal{A}$'s view and the impact of $v_{y'_k}$ will be balanced in the partial decryption as shown in the later discussion. In the following discussion, we assume an event E happens, which is, for every $(i, j) \in \overline{\mathcal{T}^*}' \times \overline{\mathcal{T}^*}$, $\beta_{i,j}$ hasn't been queried by $\mathcal{A}$ as the $\mathcal{RO}(\cdot)$'s input. Then, the distribution of $\{(z_i, B_i, pd_{i,y})\}_{i \in \overline{\mathcal{T}^*}}$ is identical to that of $\mathcal{H}_5$, since $\{(z_i, B_i, pd_{i,y})\}_{i \in \overline{\mathcal{T}^*}'}$ are generated in the same way as in $\mathcal{H}_5$, $z_{k'}$ is uniformly distributed since $\rho_{k'}$ is uniformly sampled, and the generation of $\{B_{k'}, pd_{k',y}\}$ allows $\mathcal{A}$ to decrypt a uniformly sampled y and conduct (A, e, s)[5]. Thus, the distributions of $\mathcal{H}_6$ and $\mathcal{H}_5$ are statistically close (due to the distribution of $u_{y'_k}$) when E happens. Let's now consider the probability E doesn't happen. For every $\beta_{i,j}$ with $(i, j) \in \overline{\mathcal{T}^*}' \times \overline{\mathcal{T}^*}$, it's fully hidden and uniformly distributed in $\mathcal{A}$'s view, so the probability of querying $\beta_{i,j}$ by $\mathcal{A}$ should be negligible since $\mathcal{A}$ can at least make polynomial times of queries. The size of $\{(i, j) | (i, j) \in \overline{\mathcal{T}^*}' \times \overline{\mathcal{T}^*}\}$ is also bounded in polynomial, so the probability of at least one $\beta_{i,j}$ has been queried by $\mathcal{A}$ is still negligible. Therefore, the difference between $\mathcal{H}_6$ and $\mathcal{H}_5$ is not greater than the difference under the condition that E happens plus the probability that E doesn't happen, which results in $\mathcal{H}_6 \approx_c \mathcal{H}_5$.

Hybrid $\mathcal{H}_7$. In $\mathcal{H}_7$, we alter the code of $\mathcal{S}$ when a corrupted $\mathcal{C}$ conducts the signature in the weak partially blind signing phase. Specifically, $\mathcal{S}$ runs **modified steps** 12, 15 and 16 specified in step 19 to determine the honest signing parties' sending information $\{(z_i, B_i, pd_{i,y})\}_{i \in \overline{\mathcal{T}^*}}$. Message m and the random nonce s_0 can be extracted from $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{dl} \wedge \phi}}$. Thus, we can apply the same security argument of $\mathcal{H}_6$ to have $\mathcal{H}_7 \approx_c \mathcal{H}_6$.

Hybrid $\mathcal{H}_8$. In $\mathcal{H}_8$, $\mathcal{S}$ now plays the role of $\mathcal{F}_{\mathsf{BBS+}}(t)$ to sample $((\mathbf{H}, X), x)$ and instruct the conduction of signatures. In addition, $\mathcal{S}$ samples the CL public key D but doesn't store the CL secret key d. This doesn't matter since $\mathcal{S}$ already doesn't need d to perform decryption in $\mathcal{H}_7$. The difference between $\mathcal{H}_8$ and $\mathcal{H}_7$ is purely syntactic; $\mathcal{H}_8 = \mathcal{H}_7$.

Hybrid $\mathcal{H}_9$. In $\mathcal{H}_9$, we directly use $\mathcal{S}^{\mathcal{A}}$ as the simulator. The only difference between $\mathcal{H}_9$ and $\mathcal{H}_8$ is the consistency between the discrete-logorathm x_k of X_k and the plaintext of (u_{x_k}, v_{x_k}). In $\mathcal{H}_8$, the plaintext equals x_k. In $\mathcal{H}_9$, the plaintext is $\tilde{x_k}$ that is uniformly sampled and independent from x_k. Distinguishing such difference can be reduced to CL's CPA security, where a middle adversary $\mathcal{B}$ plays the role of $\mathcal{F}_{\mathsf{BBS+}}(t)$ to sample $((\mathbf{H}, X), x)$, receives the CL public key D from the CPA challenger and sends $(x_k, \tilde{x_k})$ to the challenger. Then, $\mathcal{B}$ receives ciphertext (u_{x_k}, v_{x_k}). Later, the executions of $\mathcal{S}$ and $\mathcal{S}^{\mathcal{A}}$ are identical, and $\mathcal{B}$ follows their executions to perform further instructions. In the end, $\mathcal{B}$ should output the joint distribution of $\mathcal{A}$'view and the conducting signatures/failures to the distinguisher for distinguishing $\mathcal{H}_8$ and $\mathcal{H}_9$, and $\mathcal{B}$ answers the plaintext of (u_{x_k}, v_{x_k}) is x_k if and only if distinguisher deduces the distribution is $\mathcal{H}_8$. Conducted from the above reduction, the distinguishing advantage of $\mathcal{H}_8$ and $\mathcal{H}_9$ is computationally bounded by the advantage of CL's CPA game. Thus, $\mathcal{H}_8 \approx_c \mathcal{H}_9$.

From the above analysis, we can have $\mathcal{H}_0 \approx_c \mathcal{H}_9$. Therefore, our protocol securely realizes $\mathcal{F}_{\mathsf{BBS+}}(t)$ in the $(\mathcal{F}_{\mathsf{COM}}, \mathcal{F}_{\mathsf{GenPub}}, \mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}enc} \wedge \mathsf{dl}}}, \mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{cl\text{-}rand}}}, \mathcal{F}_{\mathsf{PVSS}}(t), \mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathsf{dl} \wedge \phi}})$-hybrid model.

□

2 Functionality Difference

Compared with the BBS+ functionality specified in [1], the **Weak Partially Blind Signing** function in our functionality $\mathcal{F}_{\mathsf{BBS+}}(t)$ additionally introduces

$$B = G_1 + s' \cdot H_1 + \sum_{i \in [\ell]} m_i \cdot H_{i+1}$$

as the leakage information for the resulting signature (A, e, s'). Note that we use the notation s' instead of s here for clearer presentation.

We claim that, even if B is leaked, the function description still satisfies the definition of weak partial blindness, which requires the message $\mathbf{m}$ to be hidden from all signing parties. According to BBS+'s Sign algorithm, s' is uniformly distributed in $\mathbb{Z}_q$. Indeed, such B forms a Pedersen commitment of $\mathbf{m}$ if we treat s' as a mask, since s' does not leak to the signing parties and $\mathcal{S}$. Therefore, $\mathcal{B}$ can information-theoretically hide $\mathbf{m}$.

Moreover, B is essentially $s \cdot H_1 + B_0$ in our protocol. The only difference between B and B_0 as a Pedersen commitment is that there is an offset s in their random mask, which is public to all signing parties, and thus, $\mathbf{m}$ is still hidden even both B and B_0 are known to signing parties.

[5]Although $\mathcal{A}$ may choose different $\{\rho_j\}_{j \in \mathcal{T}^*}$, the value of $\prod_{j \in \mathcal{T}^*}(v_{y_j} f^{-\rho_j})$ is always consistent when conducting the signature.

3 Conversion Insights

In the following content, we assume $|\mathcal{T}| \geq t$ and $i \in \mathcal{T}$ when using i as a party index for convenience.

For any secret d_j, it is shared as $\{s_{j,i}\}_{i\in[n]}$ through $\mathcal{F}_{\mathsf{PVSS}}(t)$ in the form of a degree-$(t-1)$ polynomial. Thus, we can recover d_j from $\{s_{j,i}\}_{i\in\mathcal{T}}$ using Lagrange Interpolation as:

$$d_j = \frac{1}{\Delta}\sum_{i\in\mathcal{T}} L_{\mathcal{T},i} s_{j,i}.$$

Here, $\Delta := n!$, which is the factorial of n. The decryption key d can be recovered as:

$$d = \sum_{j\in[n]} d_j = \sum_{j\in[n]} \frac{1}{\Delta}\sum_{i\in\mathcal{T}} L_{\mathcal{T},i} s_{j,i} = \sum_{i\in\mathcal{T}} (\frac{L_{\mathcal{T},i}}{\Delta}\sum_{j\in[n]} s_{j,i}).$$

Therefore, each signing party P_i can locally compute $\frac{L_{\mathcal{T},i}}{\Delta}\sum_{j\in[n]} s_{j,i}$ as the additive share of d. Usually, $L_{\mathcal{T},i}$ is not an integer, but its Δ times is an integer. To avoid computational errors, party P_i for $i \in \mathcal{T}$ calculates

$$d_{\mathcal{T},i} := \Delta^3 \cdot \frac{L_{\mathcal{T},i}}{\Delta}\sum_{j\in[n]} s_{j,i} = \Delta L_{\mathcal{T},i}\sum_{j\in[n]} \Delta s_{j,i}$$

and forms the additive share to $\Delta^3 d$. The impact of multiplying Δ^3 on decryption can be easily recovered (we'll discuss it later). Meanwhile, party P_i also computes $\beta_{\mathcal{T},i} := \sum_{j\in\mathcal{T},j<i}\beta_{i,j} - \sum_{j\in\mathcal{T},j>i}\beta_{i,j}$. Note that $\beta_{i,j} = \beta_{j,i}$, so $\{\beta_{\mathcal{T},i}\}_{i\in\mathcal{T}}$ indeed form the zero shares:

$$\begin{aligned}
&\sum_{i\in\mathcal{T}} \beta_{\mathcal{T},i}\\
=&\sum_{i\in\mathcal{T}}(\sum_{j\in\mathcal{T},j<i}\beta_{i,j} - \sum_{j\in\mathcal{T},j>i}\beta_{i,j})\\
=&\sum_{i\in\mathcal{T}}\sum_{j\in\mathcal{T},j<i}\beta_{i,j} - \sum_{i\in\mathcal{T}}\sum_{j\in\mathcal{T},j>i}\beta_{i,j}\\
=&\sum_{i\in\mathcal{T}}\sum_{j\in\mathcal{T},j<i}\beta_{i,j} - \sum_{j\in\mathcal{T}}\sum_{i\in\mathcal{T},i<j}\beta_{j,i}\\
=&\sum_{i\in\mathcal{T}}\sum_{j\in\mathcal{T},j<i}(\beta_{i,j} - \beta_{i,j})\\
=&0.
\end{aligned}$$

By the definition of zero shares, $\{d_{\mathcal{T},i} + \beta_{\mathcal{T},i}\}_{i\in\mathcal{T}}$ still form the additive shares of $\Delta^3 d$. Signing parties use $\{d_{\mathcal{T},i} + \beta_{\mathcal{T},i}\}_{i\in\mathcal{T}}$ as partial decryption keys to partially decrypt the ciphertext. It's worth mentioning that there's a typo in Section 3.2 of our original paper where $d_{\mathcal{T},i}$ is wrongly computed as $\beta_{\mathcal{T},i} + \Delta L_{\mathcal{T},i}\sum_{j\in[n]}\Delta s_{j,i}$, so parties use $\{d_{\mathcal{T},i} + 2\beta_{\mathcal{T},i}\}_{i\in\mathcal{T}}$ as partial decryption keys. Although this doesn't impact correctness or security since $\{2\beta_{\mathcal{T},i}\}_{i\in\mathcal{T}}$ also form the zero shares, we still point it out here.

Next, we can compute:

$$\begin{aligned}
&\prod_{i\in\mathcal{T}} pd_{i,y}\\
=&\prod_{i\in\mathcal{T}}\frac{v_{y_i}^{\Delta^3}}{u_y^{d_{\mathcal{T},i}+\beta_{\mathcal{T},i}}}\\
=&\frac{(\prod_{i\in\mathcal{T}} v_{y_i})^{\Delta^3}}{\prod_{i\in\mathcal{T}} u_y^{d_{\mathcal{T},i}+\beta_{\mathcal{T},i}}}\\
=&\frac{v_y^{\Delta^3}}{u_y^{\Delta^3 d}}\\
=&(\frac{v_y}{u_y^d})^{\Delta^3}\\
=&f^{\Delta^3 y}.
\end{aligned}$$

So the client finally gets the plaintext y by computing

$$\mathsf{Solve}_{\mathsf{DL}}(\prod_{j\in\mathcal{T}} pd_{j,y})\Delta^{-3} = \frac{\Delta^3 y}{\Delta^3} = y \mod q.$$

Note that in $\mathbb{Z}_q$, results of divisions are always integers.

Some readers might have found that if parties directly use $d_{\mathcal{T},i}$ as partial decryption key instead of $d_{\mathcal{T},i} + \beta_{\mathcal{T},i}$, the correctness still holds. However, using $d_{\mathcal{T},i}$ as the partial decryption key may be insecure, as our security is based on CL encryption's CPA security, meaning that the secret key d is unknown in the reduction. Now assume we are constructing the reduction, and WLOG assumes there are $t-1$ corrupted parties. Usually, we can also assume that we know all parties' secret d_i and their corresponding secret shares $s_{i,j}$ for every $j \in [n]$, except for a fixed honest party P_k. For P_k, we only know his secret share $s_{k,j}$ for all corrupted parties j. Otherwise, the Lagrange Interpolation should enable us to calculate d_k and compute d, which contradicts the CPA security. Meanwhile, when there are 2 or more honest parties involved in the signing phase, the secret share $s_{k,j}$ of d_k for some honest party P_j must be provided to calculate $d_{\mathcal{T},j}$ and pass $pd_{j,y}$ to the client, which similarly enable the calculation of d_k and thus not allowed. So, directly using $d_{\mathcal{T},i}$ as the partial decryption key is not secure in our understanding.

But why is using $d_{\mathcal{T},i} + \beta_{\mathcal{T},i}$ as the partial decryption key secure? The reason is that $\beta_{i,j}$ where parties P_i and P_j are honest is invisible and uniformly random in adversaries' view. During reduction, when there are k honest parties for example, we can uniformly sample $d_{\mathcal{T},i} + \beta_{\mathcal{T},i}$ for $k-1$ honest parties without providing unknown secret shares and deduce $pd_{i,y}$ from y for the remaining honest party, which completely avoids the pre-discussed condition.

References

[1] Jack Doerner, Yashvanth Kondi, Eysa Lee, Abhi Shelat, and LaKyah Tyner. Threshold bbs+ signatures for distributed anonymous credential issuance. In *2023 IEEE*

Symposium on Security and Privacy (SP), pages 773–789. IEEE, 2023.