Three Round Threshold ECDSA Signing from ECDSA Assumptions

Jack Doerner, Yashvanth Kondi, Eysa Lee, abhi shelat

April 21, 2023

Contents

1 Ideal Threshold ECDSA

Algorithm 1.1. $\mathsf{ECDSAGen}(\mathcal{G})$

1. Uniformly choose a secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q$.
2. Calculate the public key as $\mathsf{pk} \coloneqq \mathsf{sk} \cdot G$.
3. Output $(\mathsf{pk}, \mathsf{sk})$.

Algorithm 1.2. $\mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk} \in \mathbb{Z}_q, m \in \{0,1\}^*)$

1. Uniformly choose an instance key $k \leftarrow \mathbb{Z}_q$.
2. Calculate $R \coloneqq k \cdot G$ and let r_x be the x-coordinate of R, modulo q.
3. Calculate
$$s \coloneqq \frac{\mathsf{SHA2}(m) + \mathsf{sk} \cdot r_\mathsf{x}}{k}$$
4. Output $\sigma \coloneqq (s, r_\mathsf{x})$.

Algorithm 1.3. $\mathsf{ECDSAVerify}(\mathcal{G}, \mathsf{pk} \in \mathbb{G}, m \in \{0,1\}^*, \sigma \in \mathbb{Z}_q^2)$

1. Parse σ as (s, r_{x}).
2. Calculate
$$R' := \frac{\mathsf{SHA2}(m) \cdot G + r_{\mathsf{x}} \cdot \mathsf{pk}}{s}$$
and let r'_{x} be the x-coordinate of R', modulo q.
3. Output 1 if and only if $r'_{\mathsf{x}} = r_{\mathsf{x}}$.

Functionality 1.4. $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$**: Threshold ECDSA Signing**

This functionality is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The setup phase runs once with n parties, and the signing phase may be run many times between (varying) subgroups of parties indexed by $\mathbf{P} \subseteq [n]$ such that $|\mathbf{P}| = t$. All messages are adversarially delayed, and can be replaced with an abort by the adversary.

Setup: On receiving $(\texttt{init}, \mathsf{sid})$ from all parties such that no message of the form $(\texttt{secret-key}, \mathsf{sid}, \cdot)$ exists in memory,

1. Sample the joint secret and public keys, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ECDSAGen}(\mathcal{G})$.
2. Store $(\texttt{secret-key}, \mathsf{sid}, \mathsf{sk})$ in memory, and send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ to all parties.

Signing: On receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$ from any party $\mathcal{P}_i$, parse $\mathbf{P}\|\mathsf{sigid}' := \mathsf{sigid}$ such that $|\mathbf{P}| = t$ and ignore the message if $i \notin \mathbf{P}$ or $\mathbf{P} \not\subseteq [n]$ or sigid is not fresh or if $(\texttt{pk-delievered}, \mathsf{sid}, i)$ does not exist in memory for some $i \in \mathbf{P}$. Otherwise, upon receiving such messages from all parties indexed by $\mathbf{P}$,

3. Sample $\sigma \leftarrow \mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m)$, and send $(\texttt{sig}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to all parties.

Cleans up [DKLs18]

2 Ideal Functionalities

Functionality 2.1. $\mathcal{F}_{\mathsf{zero}}(q, n, t)$**: Pairwise Randomization**

This functionality is parameterized by the party count n, the threshold t, and the integer q. The setup phase runs once with n parties, and the online phase may be run many times between (varying) subgroups of parties indexed by $\mathbf{P} \subseteq [n]$ such that $|\mathbf{P}| = t$. All messages are adversarially delayed, and can be replaced with an abort by the adversary.

Setup: On receiving $(\texttt{init}, \mathsf{sid})$ from all parties such that no message of the form $(\texttt{initialized}, \mathsf{sid})$ exists in memory, store $(\texttt{initialized}, \mathsf{sid})$ and send it to all parties.

Online: On receiving $(\texttt{get-mask}, \mathsf{sid}, \mathsf{sigid}, i)$ from any honest party $\mathcal{P}_i$, parse $\mathbf{P}\|\mathsf{sigid}' := \mathsf{sigid}$ such that $|\mathbf{P}| = t$ and ignore the message if $(\texttt{initialized}, \mathsf{sid})$ does not exist in memory. Otherwise,

1. Let $\mathbf{P_H}$ index the honest parties in $\mathbf{P}$.
2. Retrieve $(\texttt{masks}, \mathsf{sid}, \mathsf{sigid}, \mu)$ from memory. If no such entry already exists, ask the adversary for $\tau \in \mathbb{Z}_q$ and sample $\boldsymbol{\mu} \leftarrow \mathbb{Z}_q^{|\mathbf{P_H}|}$ conditioned on $\sum\limits_{j \in \mathbf{P_H}} \mu_j = -\tau$ and store $(\texttt{masks}, \mathsf{sid}, \mathsf{sigid}, \boldsymbol{\mu})$ in memory.
3. Send $(\texttt{mask}, \mathsf{sid}, \mathsf{sigid}, \mu_i)$ to $\mathcal{P}_i$

The functionality $\mathcal{F}_{\mathsf{zero}}$ can be realized by protocol π_{zero}.

Protocol 2.2. $\pi_{\mathsf{zero}}(q, n, t)$**: Pairwise Randomization**

This protocol is parameterized by the party count n, the threshold t, and the integer q. The setup phase runs once with n parties, and the online phase may be run many times between (varying) subgroups of parties indexed by $\mathbf{P} \subseteq [n]$ such that $|\mathbf{P}| = t$. This protocol makes use of a pseudorandom function $\mathsf{PRF} : \{0,1\}^* \mapsto \mathbb{Z}_q$.

Setup: For each $i, j \in [n]$ such that $i < j$, party $\mathcal{P}_i$ samples $\mathsf{seed}_{ij} \leftarrow \{0,1\}^\kappa$ and sends seed_{ij} to $\mathcal{P}_j$

Online: On input $(\texttt{get-mask}, \mathsf{sid}, \mathsf{sigid}, i)$, parse $\mathbf{P}\|\mathsf{sigid}' := \mathsf{sigid}$ such that $|\mathbf{P}| = t$. Let $\mathbf{P}_0 = \{j : j \in \mathbf{P}, j < i\}$ and $\mathbf{P}_1 = \{j : j \in \mathbf{P}, j > i\}$. Party $\mathcal{P}_i$ outputs $\mu_i = \sum\limits_{j \in \mathbf{P}_0} \mathsf{PRF}(\mathsf{seed}_{ji}, \mathsf{sigid}) - \sum\limits_{j \in \mathbf{P}_1} \mathsf{PRF}(\mathsf{seed}_{ij}, \mathsf{sigid})$

Theorem 2.3. *Protocol* $\pi_{\mathsf{zero}}(q, n, t)$ *UC-realizes* $\mathcal{F}_{\mathsf{zero}}(q, n, t)$ *in the* RO*-hybrid model.*

Functionality 2.4. $\mathcal{F}_{\mathsf{Mul}}(q, n)$**: Pairwise Multiplication**

This functionality is parameterized by the party count n, and the integer q. The setup phase runs once with n parties, between each pair of parties. The online phase can be run between any pair of parties. All messages are adversarially delayed, and can be replaced with an abort by the adversary.

Setup: On receiving $(\texttt{init-alice}, \mathsf{sid}, (i, j))$ from $\mathcal{P}_j$ and $(\texttt{init-bob}, \mathsf{sid}, (i, j))$ from $\mathcal{P}_i$, if no message of the form $(\texttt{initialized}, \mathsf{sid}, (i, j))$ exists in memory, store $(\texttt{initialized}, \mathsf{sid}, (i, j))$ and send it to both parties.

Online: The following interfaces may be invoked arbitrarily many times by any $\mathcal{P}_i, \mathcal{P}_j$ where $(\texttt{initialized}, \mathsf{sid}, (i, j))$ exists in memory.

- On receiving $(\texttt{bob-input}, (\mathsf{sid}, i, j), \mathsf{sigid}_i, \phi_i)$ from $\mathcal{P}_i$, if no entry $(\mathsf{sid}, i, j, \mathsf{sigid}_i, \cdot)$ exists in memory, then store $(\mathsf{sid}, i, j, \mathsf{sigid}_i, \phi_i)$ and send $(\texttt{bob-committed}, (\mathsf{sid}, i, j), \mathsf{sigid}_i)$ to $\mathcal{P}_j$
- On receiving $(\texttt{alice-input}, (\mathsf{sid}, i, j), \mathsf{sigid}_i, (x_j, k_j))$ from $\mathcal{P}_j$, if $(\mathsf{sid}, i, j, \mathsf{sigid}_i, \phi_i)$ exists in memory then sample $t^{ij}_{\mathsf{A},0}, t^{ij}_{\mathsf{A},1}, t^{ij}_{\mathsf{B},0}, t^{ij}_{\mathsf{B},1} \in \mathbb{Z}_q^4$ uniformly subject to:

$$t^{ij}_{\mathsf{A},0} + t^{ij}_{\mathsf{B},0} = \phi_i \cdot x_j \qquad \text{and} \qquad t^{ij}_{\mathsf{A},1} + t^{ij}_{\mathsf{B},1} = \phi_i \cdot k_j$$

  In case $\mathcal{P}_j$ is corrupt, then receive $t^{ij}_{\mathsf{A},0}, t^{ij}_{\mathsf{A},1}$ from the adversary instead of sampling it, and set $t^{ij}_{\mathsf{B},0}, t^{ij}_{\mathsf{B},1}$ accordingly to satisfy the above relationships. Send $(\texttt{alice-output}, (\mathsf{sid}, i, j), (t^{ij}_{\mathsf{A},0}, t^{ij}_{\mathsf{A},1}))$ first to $\mathcal{P}_j$, and then $(\texttt{bob-output}, (\mathsf{sid}, i, j), (t^{ij}_{\mathsf{B},0}, t^{ij}_{\mathsf{B},1}))$ to $\mathcal{P}_i$.

Protocol 2.5. Pairwise Multiplication ($\mathcal{F}_{\mathsf{Mul}}()$)

This protocol is parameterized by the statistical security parameter s, and the group $\mathbb{Z}_q$ over which multiplication is to be performed. Let $\kappa = |q|$ and for convenience let $\xi = \kappa + 2s$, i.e. the number of random choice bits per element in a batch. This protocol makes use of a public *gadget vector* $\mathbf{g} \leftarrow \mathbb{Z}_q^{\xi}$, and it invokes the $\mathcal{F}_{\mathsf{COTe}}$ functionality and the random oracle H. Alice and Bob supply input integers x, k and ϕ respectively. They receive as output vectors of integers $t_{\mathsf{A},0}, t_{\mathsf{A},1}$ and $t_{\mathsf{B},0}, t_{\mathsf{B},1}$.

Init: Alice and Bob transmit $(\texttt{init})$ to $\mathcal{F}_{\mathsf{COTe}}$.

Randomized Multiplication:

1. Bob samples a set of OT choice bits and calculates his pad $\tilde{\mathbf{b}}$

$$\boldsymbol{\beta} \leftarrow \{0,1\}^{\xi} \qquad \text{and} \qquad \tilde{\mathbf{b}} := \left\langle \mathbf{g}, \left\{\boldsymbol{\beta}_j\right\}_{j\in[\xi]} \right\rangle$$

2. Alice samples her check value $\hat{a} \leftarrow \mathbb{Z}_q$ and sets her OT correlation $\boldsymbol{\alpha}$ as

$$\boldsymbol{\alpha} := \{x \| k \| \hat{a}\}_{j\in[\xi]}$$

3. Alice and Bob access the $\mathcal{F}_{\mathsf{COTe}}$ functionality, supplying ξ as the OT-extension batch size. Alice plays the sender, supplying $\boldsymbol{\alpha}$ as her input, and Bob, the receiver, supplies $\boldsymbol{\beta}$. They receive as outputs the arrays $\boldsymbol{\omega}_{\mathsf{A}}$ and $\boldsymbol{\omega}_{\mathsf{B}}$ respectively, which they interpret as

$$\left\{\mathbf{z^x}_{\mathsf{A},j}\middle\|\mathbf{z^k}_{\mathsf{A},j}\middle\|\hat{\mathbf{z}}_{\mathsf{A},j}\right\}_{j\in[\xi]} = \boldsymbol{\omega}_{\mathsf{A}} \qquad \text{and} \qquad \left\{\mathbf{z^x}_{\mathsf{B},j}\middle\|\mathbf{z^k}_{\mathsf{B},j}\middle\|\hat{\mathbf{z}}_{\mathsf{B},j}\right\}_{j\in[\xi]} = \boldsymbol{\omega}_{\mathsf{B}}$$

4. Alice and Bob generate two shared, random values by calling the random oracle. As input they use the shared components of the transcript of the protocol that implements $\mathcal{F}_{\mathsf{COTe}}$, so that that these values have a dependency on the completion of Step 3.

$$\chi \leftarrow H(1\|\mathsf{transcript}) \qquad \text{and} \qquad \hat{\chi} \leftarrow H(2\|\mathsf{transcript})$$

5. Alice computes

$$\mathbf{r} := \left\{\mathbf{z^x}_{\mathsf{A},j} + \chi\cdot\mathbf{z^k}_{\mathsf{A},j} + \hat{\chi}\cdot\hat{\mathbf{z}}_{\mathsf{A},j}\right\}_{j\in[\xi]}$$
$$u := x + \chi\cdot k + \hat{\chi}\cdot\hat{\mathbf{a}}_i$$

   and sends $r = H(3\|\mathbf{r})$ and u to Bob.

   Alice outputs

$$t_{\mathsf{A},0} = \sum_{j\in[\xi]} \mathbf{g}_j\cdot\mathbf{z^x}_{\mathsf{A},j} \qquad \text{and} \qquad t_{\mathsf{A},1} = \sum_{j\in[\xi]} \mathbf{g}_j\cdot\mathbf{z^k}_{\mathsf{A},j}$$

6. Bob computes

$$\mathbf{r}_{\mathsf{B}} = \left\{\boldsymbol{\beta}_j\cdot u - \left(\mathbf{z^x}_{\mathsf{B},j} + \chi\cdot\mathbf{z^k}_{\mathsf{B},j} + \hat{\chi}\cdot\hat{\mathbf{z}}_{\mathsf{B},j}\right)\right\}_{j\in[\xi]}$$

   and aborts if $H(3\|\mathbf{r}_{\mathsf{B}}) \neq r$. Otherwise, he outputs

$$t_{\mathsf{B},0} = \sum_{j\in[\xi]} \mathbf{g}_j\cdot\mathbf{z^x}_{\mathsf{B},j} \qquad \text{and} \qquad t_{\mathsf{B},1} = \sum_{j\in[\xi]} \mathbf{g}_j\cdot\mathbf{z^k}_{\mathsf{B},j}$$

3 Threshold Key Generation

Functionality 3.1. $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$**: Discrete Log Keygen**

This functionality is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. If any party is corrupt, then the adversary $\mathcal{S}$ may instruct the functionality to abort.

Key Generation: On receiving $(\texttt{keygen}, \mathsf{sid})$ from all parties, where sid is fresh and participant-respecting,

1. Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ECDSAGen}(\mathcal{G})$.
2. Store $(\texttt{secret-key}, \mathsf{sid}, \mathsf{sk})$ in memory.
3. Send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ directly to $\mathcal{S}$.
4. Sample a random polynomial p of degree $t-1$ over $\mathbb{Z}_q$, subject to $p(0) = \mathsf{pk}$.
5. On receiving $(\texttt{release}, \mathsf{sid}, i)$ for $i \in [n]$ directly from $\mathcal{S}$, send $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, p(i))$ to $\mathcal{P}_i$.

4 t-Party Threshold ECDSA

Protocol 4.1. $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$**: t-Party ECDSA**

This protocol is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The setup phase runs once with parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, and the signing phase may be run many times between (varying) subsets of parties of size t. The parties in in this protocol interact with the ideal functionalities $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{Mul}}(q)$, and $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$.

Setup: We describe below the instructions followed by $\mathcal{P}_i$. Note that this phase is run exactly once.

1. Send $(\texttt{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$, and receive $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk}_i)$ in response. Store $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk}_i)$ in memory. The common public key is pk. yash: Just standard DKG. The common public key is pk, and party $\mathcal{P}_i$'s share is sk_i. The sk_i values lie on some degree $t-1$ polynomial f such that $f(0) \cdot G = \mathsf{pk}$
2. For each $j \in [n] \setminus \{i\}$, send $(\texttt{init-bob}, (\mathsf{sid}, i, j))$ to $\mathcal{F}_{\mathsf{Mul}}(q)$, and send $(\texttt{init-alice}, (\mathsf{sid}, j, i))$ to $\mathcal{F}_{\mathsf{Mul}}(q)$. yash: Initialize two multipliers with every other party $\mathcal{P}_j$; one where $\mathcal{P}_i$ is Alice (identified as instance (sid, j, i)), and the other where $\mathcal{P}_i$ is Bob (identified as instance (sid, i, j))

Signing: Denote by $\mathbf{P}$ indices of the signing set, such that $|\mathbf{P}| = t$. Denote by $\mathbf{P}^*_{\setminus \mathbf{i}}$ the set $\mathbf{P} \setminus \{i\}$. Let $(\lambda_j)_{j \in \mathbf{P}}$ be the set of Lagrange coefficients that interpolate the y-intercept of a degree $t-1$ polynomial given its evaluations at $\mathbf{P}$. We assume that each $\mathcal{P}_i$ is invoked with a message to be signed m, signing set $\mathbf{P}$, and session identifier $\mathsf{sigid}_{m,\mathbf{P}}$.

We describe below the code of each $\mathcal{P}_i(\mathsf{state}_i, m, \mathbf{P}, \mathsf{sigid}_{m,\mathbf{P}})$:

- If there is already an active session with the parameters $m, \mathbf{P}, \mathsf{sigid}_{m,\mathbf{P}}$, output an abort message and terminate. Otherwise, proceed with signing.

- **Round 1:** Sample $\mathsf{sigid}_i \leftarrow \{0,1\}^\kappa$, $\phi_i \leftarrow \mathbb{Z}_q$, and $k_i \leftarrow \mathbb{Z}_q$. Compute $R_i = k_i \cdot G$ and $C_i = \mathsf{RO}(\mathsf{sigid}_i, R_i)$. For each $j \in \mathbf{P}^*_{\setminus \mathbf{i}}$,
  1. Send $(\texttt{bob-input}, (\mathsf{sid}, i, j), \mathsf{sigid}_i, \phi_i)$ to $\mathcal{F}_{\mathsf{Mul}}(q)$
  2. Send $\mathsf{args}_i = (m, \mathbf{P}, \mathsf{sigid}_{m,\mathbf{P}})$, and $\mathsf{BC}_i = (\mathsf{sigid}_i, C_i)$ to $\mathcal{P}_j$
- **Round 2:**
  1. Wait to receive $\mathsf{args}_j, \mathsf{BC}_j$ from each $\mathcal{P}_j$, and $(\texttt{bob-committed}, (\mathsf{sid}, j, i), \mathsf{sigid}_j)$ from $\mathcal{F}_{\mathsf{Mul}}(q)$, for each $j \in \mathbf{P}^*_{\setminus \mathbf{i}}$.
  2. Compute $\mathsf{digest}_i = \mathsf{RO}(\{\mathsf{args}_j, \mathsf{BC}_j\}_{j \in \mathbf{P}})$
  3. Obtain $\mu_i \in \mathbb{Z}_q$ by querying $(\mathsf{sid}, \mathsf{digest})$ to $\mathcal{F}_{\mathsf{zero}}$
  4. Compute $x_i = \lambda_i \cdot \mathsf{sk}_i + \mu_i$ and $X_i = x_i \cdot G$
  5. For each $j \in \mathbf{P}$,
     (a) Send $(\texttt{alice-input}, (\mathsf{sid}, j, i), \mathsf{sigid}_j, (x_i, k_i))$ to $\mathcal{F}_{\mathsf{Mul}}(q)$, receiving $(\texttt{alice-output}, (\mathsf{sid}, j, i), \mathsf{sigid}_j, (t^{ji}_{\mathsf{A},0}, t^{ji}_{\mathsf{A},1}))$ as local output fom $\mathcal{F}_{\mathsf{Mul}}(q)$.
     (b) Set $\Gamma^0_{j,i} = t^{ji}_{\mathsf{A},0} \cdot G$ and $\Gamma^1_{j,i} = t^{ji}_{\mathsf{A},1} \cdot G$
     (c) Send $\mathsf{digest}_i, X_i, R_i, \Gamma^0_{j,i}, \Gamma^1_{j,i}$ to $\mathcal{P}_j$
- **Round 3:**
  1. For each $j \in \mathbf{P}^*_{\setminus \mathbf{i}}$:
     (a) Wait to receive $\mathsf{digest}_j, X_i, R_j, \Gamma^0_{i,j}, \Gamma^1_{i,j}$ from $\mathcal{P}_j$, and $(\texttt{bob-output}, (\mathsf{sid}, i, j), \mathsf{sigid}_i, (t^{ij}_{\mathsf{B},0}, t^{ij}_{\mathsf{B},1}))$ from $\mathcal{F}_{\mathsf{Mul}}(q)$.
     (b) Verify that $C_j \overset{?}{=} \mathsf{RO}(\mathsf{sigid}_j, R_j)$ and $\mathsf{digest}_i \overset{?}{=} \mathsf{digest}_j$
  2. Compute $R^*_{\setminus i} = \sum_{j \in \mathbf{P}^*_{\setminus \mathbf{i}}} R_j$ and the signing nonce $R = R^*_{\setminus i} + R_i$
  3. Compute $X^*_{\setminus i} = \mathsf{pk} - X_i$, and $T = (\sum_{j \in \mathbf{P}^*_{\setminus \mathbf{i}}} t^{ij}_{\mathsf{B},0}) \cdot G$
  4. Verify that the following three relationships hold:

$$\sum_{j \in \mathbf{P}^*_{\setminus \mathbf{i}}} X_j \overset{?}{=} X^*_{\setminus i} \qquad \sum_{j \in \mathbf{P}^*_{\setminus \mathbf{i}}} \Gamma^0_{i,j} \overset{?}{=} \phi_i \cdot X^*_{\setminus i} - T \qquad \sum_{j \in \mathbf{P}^*_{\setminus \mathbf{i}}} \Gamma^1_{i,j} \overset{?}{=} \phi_i \cdot R^*_{\setminus i} - T$$

5. Compute the signature shares:

$$s_{0,i} = \mathsf{SHA2}(m) \cdot \phi_i + r_{\mathsf{x}} \cdot \left(\mathsf{sk}_i \cdot \phi_i + \sum_{j \in \mathbf{P}^*_{\setminus \mathbf{i}}} \left(t^{ij}_{\mathsf{B},0} + t^{ji}_{\mathsf{A},0} \right) \right)$$

$$s_{1,i} = k_i \cdot \phi_i + \sum_{j \in \mathbf{P}^*_{\setminus \mathbf{i}}} \left(t^{ij}_{\mathsf{B},1} + t^{ji}_{\mathsf{A},1} \right)$$

Send $(R, s_{0,i}, s_{1,i})$ to all parties.

- **Signature Assembly:** Upon collecting $(R, s_{0,j}, s_{1,j})$ from each $\mathcal{P}_j$, compute $s_0 = \sum\limits_{j \in \mathbf{P}} s_{0,j}$ and $s_1 = \sum\limits_{j \in \mathbf{P}} s_{1,j}$. Assemble the putative signature as $(R, s = s_0/s_1)$, and output it if $\mathsf{ECDSAVerify}(\mathsf{pk}, R, s) = 1$.

Theorem 4.2. *Protocol* $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *UC-realizes* $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *in the* $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$*,* RO *hybrid model.*

Proof.

Hybrid $\mathcal{H}_1$. In this hybrid experiment, honest parties abort when there is a session such that ... yash: parties don't agree on $m, P, R, sigid$. This follows easily from collision resistance of RO.

Hybrid $\mathcal{H}_2$. In this hybrid experiment, the instruction for an honest $\mathcal{P}_i$ to abort when $\Gamma^0_{i,j} \neq \phi_i \cdot \mathsf{pk}_j - t^{ij}_{\mathsf{B},0} \cdot G$ is removed. Instead, an honest $\mathcal{P}_i$ aborts if $\exists \mathsf{sigid}_i, j, i$ such that $(\texttt{alice-input}, (\mathsf{sid}, i, j), \mathsf{sigid}_i, (\mathsf{sk}^*_j, k^*_j))$ is sent to $\mathcal{F}_{\mathsf{Mul}}(q)$ by a corrupt $\mathcal{P}_j$ such that $\mathsf{sk}^*_j \cdot G \neq \mathsf{pk}_j$, or $k^*_j \neq R_j$. yash: Follows from a very simple statistical argument.

Hybrid $\mathcal{H}_3$. yash: In this hybrid experiment, instead of decommitting R_i honestly, the experiment first samples R, and programs the RO to equivocate to a corresponding R_i. Indistinguishability follows from entropy of R_i—adv. is unlikely to have queried this point before it's programmed.

Hybrid $\mathcal{H}_4$. yash: In this hybrid experiment, compute an ECDSA signature (R, s) per signing instance. Instead of the honest party broadcasting s_0, s_1, sample s_0 uniformly, and compute s_1 as a function of s, s_0. Distributed identically to the last hybrid.

Hybrid $\mathcal{H}_5$. yash: Syntactic change to use $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ instead.

□

References

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *Proceedings of the 39th IEEE Symposium on Security and Privacy, (S&P)*, pages 980–997, 2018.