# $^{d}/_{v}$-CLSAG: Extension for Concise Linkable Spontaneous Anonymous Group Signatures

sowle[1]

[1]Zano project, val@zano.org

January 2024*

## 1 Introduction

In this paper we present a Schnorr-like linkable ring signature scheme we call $^{d}/_{v}$-CLSAG that is extension for d-CLSAG scheme proposed in [1]. The proposed extension allows the use of different group generators for different layers of the ring members: $\boldsymbol{pk} \coloneqq \boldsymbol{sk} \circ \boldsymbol{G}$, $\boldsymbol{G} = (G_{k_0}, \dots, G_{k_{d-1}}) \in \mathbb{G}^d$, while the original scheme assumes the use of the same generator $G$ across all layers: $\boldsymbol{pk} \coloneqq \boldsymbol{sk} \circ \boldsymbol{G}$, $\boldsymbol{G} = (G, \dots, G) \in \mathbb{G}^d$. To improve the signature size we use key aggregation techniques in the same way, but for distinct group generators $\{G_k\}$ individually. Note, that we don't require the absence of efficiently-computable discrete logarithm relations between $\{G_k\}$. However, it might be possible, that adding such a limitation would allow us to reduce the signature size. This is the subject of future studies.

We provide the security statements for the proposed updated scheme in Theorem 2, Theorem 3, and Theorem 4. The proofs mostly correspond to the original proofs in [1]. We use the same numeration for theorems, definitions, and lemmas as in the original work. For the reader's convenience, all changes are highlighted.

## 2 Application

$^{d}/_{v}$-CLSAG may be used in cases when different group generators for different ring layers are necessary.

For instance, in the Zano project a user can transfer an arbitrary number of assets within a single transaction. The method used for implementing the assets requires using 2 distinct generators, $G$ and $X$, in a 3-layer arrangement $(G, G, X)$ for a normal transaction and in a 5-layer arrangement $(G, G, X, X, G)$ for a Proof-of-Stake mining transaction (see also [3]) for all ring members. Using $^{3}/_{2}$-CLSAG and $^{5}/_{2}$-CLSAG correspondingly in that context solves the problem, while the signature size is increased by only $n$ scalar elements, where $n$ is the ring size.

*Version: 1.0. Last update: 2024-01-26.

# 3 $^{d}/_{v}$-CLSAG construction

**Definition 10** ($^{d}/_{v}$**-CLSAG**). *The tuple (*Setup, KeyGen, Sign, Verify, Link*) as follows is a d-LRS signature scheme.*

- Setup $\rightarrow$ *par. First,* Setup *selects a prime $p$, a group $\mathbb{G}$ with prime order $p$, selects $d$ cryptographic hash functions $\mathcal{H}^s_0, \ldots, \mathcal{H}^s_{d-1}$ (modeled as random oracles) with codomain $\mathbb{F}_p$, selects $v$ group generators $G_0, \ldots, G_{v-1} \in \mathbb{G}$, where $v \leq d$ uniformly at random, selects surjection $g : [0, d-1] \rightarrow [0, v-1]$ that maps indices of elements of $\boldsymbol{G}$ to the corresponding generators, selects a cryptographic hash function $\mathcal{H}^p$ with codomain $\mathbb{G}$. Then,* Setup *outputs the group parameter tuple and the hash functions:*
  $par \coloneqq (p, \mathbb{G}, d, v, g, \{G_k\}_{k=0}^{v-1}, \{\mathcal{H}^s_j\}_{j=0}^{d-1}, \mathcal{H}^p)$.[1]

- KeyGen $\rightarrow$ *(**sk**, **pk**). When required for a new key,* KeyGen *samples a fresh secret key and computes the associated public key:*

$$\boldsymbol{sk} = (z_0, z_1, \ldots, z_{d-1}) \leftarrow (\mathbb{F}_p^*)^d$$
$$\boldsymbol{pk} \coloneqq \boldsymbol{sk} \circ \boldsymbol{G} = (Z_0, Z_1, \ldots, Z_{d-1}) \in \mathbb{G}^d$$

  *where $\boldsymbol{G} = (G_{g(0)}, \ldots, G_{g(d-1)}) \in \mathbb{G}^d$.* KeyGen *outputs (**sk**, **pk**). We say $z_0$ is the linking key, the remaining keys $z_j{}_{j=1}^{d-1}$ are the auxiliary keys, and we denote the linking key with $x$.*

- Sign$(m, Q, \boldsymbol{sk}) \rightarrow \{\perp_{Sign}, \sigma\}$. Sign *takes as input a message $m \in \{0,1\}^*$, a ring $Q = (\boldsymbol{pk}_0, \ldots, \boldsymbol{pk}_{n-1})$ for ring members $\boldsymbol{pk}_i = (Z_{i,0}, \ldots, Z_{i,d-1}) \in \mathbb{G}^d$, and a secret key $\boldsymbol{sk} = (z_0, \ldots, z_{d-1}) \in (\mathbb{F}_p^*)^d$.* Sign *does the following.*

  1. *If $Q \notin \mathbb{G}^{d \times n}$ for some $n$,* Sign *outputs $\perp_{Sign}$ and terminates.*
  2. *Otherwise,* Sign *parses*[2] *$Q$ to obtain each $\boldsymbol{pk}_i$. If the public key associated with the input $\boldsymbol{sk}$ is not a ring member in $Q$, then* Sign *outputs $\perp_{Sign}$ and terminates.*
  3. *Otherwise,* Sign *finds the signing index $l$, such that $\boldsymbol{pk}_l = \boldsymbol{sk} \circ (G_{g(0)}, \ldots, G_{g(d-1)})$.* Sign *samples $\{\alpha_k\}_{k=0}^{v-1} \in (\mathbb{F}_p)^v$ uniformly at random, samples $\{s_{k,i}\}_{k=0, i \neq l}^{v-1} \in (\mathbb{F}_p)^{v(n-1)}$ uniformly at random, and computes the group elements $H_i = \mathcal{H}^p(X_i)$ for each $i$.* Sign *computes the aggregation coefficients $\{\mu_j\}_{j=0}^{d-1}$, the linking tag $\mathfrak{T}$, the auxiliary group elements $\{\mathfrak{D}_j\}_{j=1}^{d-1}$, and the aggregated public keys:*

$$\mathfrak{T} \coloneqq \mathfrak{D}_0\,, \quad \{\mathfrak{D}_j\} \coloneqq \{z_j H_l\} \qquad \mu_j \coloneqq \mathcal{H}^s_j(Q \parallel \mathfrak{T} \parallel \{\mathfrak{D}_j\}_{j=1}^{d-1})$$
$$W_{k,i} \coloneqq \sum_{\forall j : g(j) = k} \mu_j Z_{i,j} \qquad \mathfrak{W}_k \coloneqq \sum_{\forall j : g(j) = k} \mu_j \mathfrak{D}_j$$

  *and the aggregated secret keys:*

$$w_k \coloneqq \sum_{j : g(j) = k} \mu_j z_j$$

  *For $i = l, l+1, \ldots, l-1$, (operating modulo $n$),* Sign *computes:*

[1] Note that domain separation can be used here to take one $\mathcal{H}^s$ and construct each $\mathcal{H}^s_j$ by defining $\mathcal{H}^s_j(x) \coloneqq \mathcal{H}^s(j \parallel x)$.

[2] Note that this parsing always succeeds if Sign does not fail in the previous step.

$$L_{k,l} = \alpha_k G_k \qquad R_{k,l} = \alpha_k H_l \qquad c_{l+1} = \mathcal{H}_0^s(m \parallel Q \parallel \{L_{k,l}\}_{k=0}^{v-1} \parallel \{R_{k,l}\}_{k=0}^{v-1})$$
$$L_{k,i} = s_{k,i} G_k + c_i W_{k,i} \qquad R_{k,i} = s_{k,i} H_i + c_i \mathfrak{W}_k \qquad c_{i+1} = \mathcal{H}_0^s(m \parallel Q \parallel \{L_{k,i}\}_{k=0}^{v-1} \parallel \{R_{k,i}\}_{k=0}^{v-1})$$

*and lastly computes*

$$\{s_{k,l}\} = \{\alpha_k - c_l w_k\}_{k=0}^{v-1}$$

4. Sign *returns the signature* $\sigma = (c_0, \{s_{k,i}\}_{k=0,i=0}^{v-1,n-1}, \mathfrak{T}, \{\mathfrak{D}_j\}_{j=1}^{d-1})$.

– Verify$(m, Q, \sigma) \to \{0,1\}$. Verify *takes as input a message* $m$*, a matrix* $Q = (\boldsymbol{pk}_0, \ldots, \boldsymbol{pk}_{n-1})$*, and a signature* $\sigma$*.*

1. *If* $Q \nsubseteq \mathcal{G}^{d \times n}$ *for some* $n$*, of if* $\sigma \notin \mathcal{F}_p^{n'v'+1} \times \mathcal{G}^d$ *for some* $n', v'$*,* Verify *outputs 0 and terminates. Otherwise, if* $n' \neq n$ *or* $v' \neq v$*,* Verify *outputs 0 and terminates.*
2. Verify *parses*[3] $(\boldsymbol{pk}_0, \ldots, \boldsymbol{pk}_{n-1}) \leftarrow Q$ *for keys* $\boldsymbol{pk}_i \in \mathcal{G}^d$ *for* $i \in [0, n-1]$*, and parses each public key* $(Z_{i,0}, \ldots, Z_{i,d-1}) \leftarrow \boldsymbol{pk}_i$*.* Verify *also parses* $(c_0, \{s_{k,i}\}_{k=0,i=0}^{v-1,n-1}, \mathfrak{T}, \{\mathfrak{D}_j\}_{j=1}^{d-1}) \leftarrow \sigma$*.* Verify *computes each* $H_i = \mathcal{H}^p(X_i)$*, computes the aggregation coefficients, and computes aggregated public keys:*

$$\mathfrak{T} \coloneqq \mathfrak{D}_0\,, \quad \{\mathfrak{D}_j\} \coloneqq \{z_j H_l\} \qquad \mu_j \coloneqq \mathcal{H}_j^s(Q \parallel \mathfrak{T} \parallel \{\mathfrak{D}_j\}_{j=1}^{d-1})$$
$$W_{k,i} \coloneqq \sum_{\forall j : g(j) = k} \mu_j Z_{i,j} \qquad \mathfrak{W}_k \coloneqq \sum_{\forall j : g(j) = k} \mu_j \mathfrak{D}_j$$

3. Verify *sets* $c'_0 \coloneqq c_0$ *and, for* $i = 1, 2, \ldots, n-1$*, computes the following.*

$$\{L_{k,i}\} \coloneqq \{s_{k,i} G_k + c'_i W_{k,i}\}$$
$$\{R_{k,i}\} \coloneqq \{s_{k,i} H_i + c'_i \mathfrak{W}_k\}$$
$$c'_{i+1} \coloneqq \mathcal{H}_0^s(m \parallel Q \parallel \{L_{k,i}\}_{k=0}^{v-1} \parallel \{R_{k,i}\}_{k=0}^{v-1})$$

4. *If* $c'_n = c_0$*,* Verify *outputs* 1*, and otherwise outputs* 0*.*

– Link$((m, Q, \sigma), (m', Q', \sigma')) \to \{0,1\}$. Link *takes as input two message-ring-signature triples.*

1. *If* Verify$(m, Q, \sigma) = 0$ *or* Verify$(m', Q', \sigma') = 0$*,* Link *outputs* 0 *and terminates.*
2. *Otherwise,* Link *parses*[4] *the signatures to obtain the individual linking tags* $(\mathfrak{T}, \{\mathfrak{D}_j\}_j), (\mathfrak{T}', \{\mathfrak{D}'_j\}_j) \leftarrow \sigma, \sigma'$*.* Link *outputs* 1 *if* $\mathfrak{W} = \mathfrak{W}'$ *and* 0 *otherwise.*

This implementation has *full-key-oriented* linkability with linkability tags $\mathfrak{W}$: two signatures will link if they not only are signed using the same linking and auxiliary keys, but also the same ring. We can replace the Link algorithm with *single-key-oriented* linkability*:*

– Link$((m, Q, \sigma), (m', Q', \sigma')) \to \{0,1\}$. Link *takes as input two message-ring-signature triples.*

1. *If* Verify$(m, Q, \sigma) = 0$ *or* Verify$(m', Q', \sigma') = 0$*,* Link *outputs* 0 *and terminates.*
2. *Otherwise,* Link *parses*[5] *the signatures to obtain the individual linking tags* $(\mathfrak{T}, \{\mathfrak{D}_j\}_j), (\mathfrak{T}', \{\mathfrak{D}'_j\}_j) \leftarrow \sigma, \sigma'$*.* Link *outputs* 1 *if* $\mathfrak{T} = \mathfrak{T}'$ *and* 0 *otherwise.*

[3] This parsing is always successful if the previous step does not terminate Verify.
[4] As before with Verify, this parsing is always successful if the previous step does not terminate Link.
[5] As before with Verify, this parsing is always successful if the previous step does not terminate Link.

# 4 Proofs of Security

**Lemma 2.** *For any $Q \subseteq \mathcal{PK}$, for any private key $sk = (\{z_j\}_j) \in Q$, the map $sk \to \sum_j \mu_j z_j$ where $\mu_j$ are computed as in Definition 10 is a collision-resistant function.*

**Theorem 2** (**Hardness of Discrete Logarithms of Linear Combinations Implies Unforgeability**). *If a $(t, \epsilon, q)$-solver of the unforgeability exists for the scheme of Definition 10 that makes $\kappa'$ corruption oracle queries, then there exists a $(2(t+t_0)+t_1), \epsilon(\frac{\epsilon}{q} - \frac{1}{2^\eta}) - \mu, \lfloor \frac{q}{d} \rfloor$-solver of the $2d\kappa'$-one-more discrete logarithm of linear combinations problem in $\mathtt{G}$ for some negligible $\mu$ and some constants $t_0, t_1$.*

Proof: Assume $\mathtt{A}$ is a $(t, \epsilon, q)$-solver of the non-slanderability game of Definition 8. We wrap $\mathtt{A}$ in an algorithm $\mathtt{B}$. The algorithm $\mathtt{B}$ executes $\mathtt{A}$ in a black box, handling oracle queries for $\mathtt{A}$. Then, $\mathtt{B}$ regurgitates the output of $\mathtt{A}$ together with an index $idx$. This way, $\mathtt{B}$ is sutable for use in the forking lemma. We wrap $\mathcal{F}^B$ in a master algorithm $\mathtt{M}$ that is a $(2(t+t_0)+t_1), \epsilon(\frac{\epsilon}{q} - \frac{1}{2^\eta}) - \mu, \lfloor \frac{q}{d} \rfloor$-solver of the $\kappa$-one-more discrete logarithm of linear combinations problem in $\mathtt{G}$, where $\eta$ is as defined in Lemma 1.

If $\mathtt{A}$ produces a successful forgery, each verification query of the form

$$c_{l+1} = \mathcal{H}^s(m \parallel Q \parallel \{L_{k,l}\}_{k=0}^{v-1} \parallel \{R_{k,l}\}_{k=0}^{v-1})$$

occurs in the transcript between $\mathtt{A}$ and the random oracle $\mathcal{H}^s$. Indeed, the signature triple produced by $\mathtt{A}$ passes verification, so each challenge $c_{l+1}$, whether made with oracle queries in the transcript or not, must be matched by random oracle queries made by the verifier. The prover cannot guess the output of such a query before making it except with negligible probability. Hence, if $\mathtt{A}$ outputs a valid signature, all verification challenges are computed by an actual oracle query. See [2] for a formal proof of this fact. Since all verification challenges are found through genuine oracle queries, which are well-ordered, there exists a first $\mathcal{H}^s$ query made by $\mathtt{A}$ for computing verification challenges, say $c = \mathcal{H}^s(m \parallel Q \parallel \{L_k^*\} \parallel \{R_k^*\})$. This was not necessarily the first query made to $\mathcal{H}^s$ overall, though; say it was the $a^{th}$ query. Although the ring index for this query may not have been decided when this query was first issued by $\mathtt{A}$, be the end of the transcript the ring index has been decided.

We construct $\mathtt{B}$ in the following way. We grant $\mathtt{B}$ access to the same oracles as $\mathtt{A}$. Any oracle queries made by $\mathtt{A}$ are passed along by $\mathtt{B}$ to the oracles. The responses are recorded and then passed back to $\mathtt{A}$. The algorithm $\mathtt{B}$ works by finding two indices to augment the output of $\mathtt{A}$. First, $\mathtt{B}$ finds the $\mathcal{H}^s$ query index $a$ corresponding to the first verification challenge computed by $\mathtt{A}$ used in verifying the purported forgery. Second, $\mathtt{B}$ inspects the transcript of $\mathtt{A}$ to find the anonymity set index $l$ in the transcript such that $c = c_{l+1}$ and $\{R_k^*\} = \{R_{k,l}\}$ and $\{L_k^*\} = \{L_{k,l}\}$. Now $\mathtt{B}$ outputs $idx = (a, l)$ along with whatever $\mathtt{A}$ outputs. Clearly, $\mathtt{B}$ makes the same number of corruptions oracle queries as $\mathtt{A}$.

Note $\mathtt{B}$ succeeds whenever $\mathtt{A}$ does and runs in time at most $t$ just like $\mathtt{A}$, except for some additional time $t_0$ to search the transcript for $idx$. Since $\mathtt{B}$ is suitable for use in the forking lemma, we can use $\mathcal{F}^B$ to construct $\mathtt{M}$.

The algorithm $\mathcal{F}^b$ is granted oracle access to the same oracles as $\mathtt{B}$ except $\mathcal{H}^s$ and $\mathtt{SO}$. The algorithm $\mathcal{F}^B$ simulates $\mathtt{SO}$ queries made by $\mathtt{B}$ by simple back-patching of $\mathcal{H}^s$ and simulates the other queries made to $\mathcal{H}^s$ queries made by $\mathtt{B}$ using the random tapes **h, h*** as described in Section 3.1. All other oracle queries made by $\mathtt{B}$ are passed along by $\mathcal{F}^B$ to the actual oracles and handed back to $\mathtt{B}$.

Note that $\mathcal{F}^B$ runs in time $2(t + t_0)$ and (with probability at least $\epsilon(\frac{\epsilon}{q} - \frac{1}{2^\eta})$) outputs a pair of valid signature triples $(m, Q, \sigma), (m', Q', \sigma')$. The messages and anonymity sets are selected before the fork point in the transcripts, so $m = m'$ and $Q = Q'$. Moreover, $\mathcal{F}^B$ makes at most $2\kappa'$ corruption queries. The challenges for the two transcripts are distinct since the forking algorithm outputs the failure symbol $\perp$ and terminates if the challenges for $c_{l+1}$ are the same in each transcript.

$$c_{l+1} \leftarrow \mathcal{H}_0^s(m \parallel Q \parallel \{L_{k,l}\}_{k=0}^{v-1} \parallel \{R_{k,l}\}_{k=0}^{v-1}) \rightarrow c'_{l+1}$$

We wrap $\mathcal{F}^B$ in an algorithm $\mathtt{M}$ that plays the $\kappa$-one-more discrete logarithm game of Definition 1 for $\kappa = 2 \cdot d \cdot \kappa'$. The algorithm $\mathtt{M}$ has corruption oracle access and runs $\mathcal{F}^B$ in a black box, passing corruption oracle queries made by $\mathcal{F}^B$ along. The algorithm $\mathtt{M}$ finds the following system of equations in the transcripts by inspecting the verification challenge queries:

$$\{L_{k,l}\} = \{s_{k,l}G_k + c_l W_{k,l}\} = \{s'_{k,l}G_k + c'_l W_{k,l}\}$$
$$\{R_{k,l}\} = \{s_{k,l}H_l + c_l \mathfrak{W}_k\} = \{s'_{k,l}H_l + c'_l \mathfrak{W}_k\}$$

This $\mathtt{M}$ has enough information to compute

$$\{W_{k,l}\} = \left\{\frac{s_{k,l} - s'_{k,l}}{c'_l - c_l} G_k\right\}, \quad \{\mathfrak{W}_k\} = \left\{\frac{s_{k,l} - s'_{k,l}}{c'_l - c_l} H_l\right\}$$

and therefore the private keys $w_k = \dfrac{s_{k,l} - s'_{k,l}}{c'_l - c_l}$.

Formally, $\mathtt{M}$ operates as follows.

(1) $\mathtt{M}$ inputs the set of discrete logarithm challenge public keys $S = \left\{\widetilde{pk}_i\right\}_{i=0}^{q-1}$.

(2) $\mathtt{M}$ partitions the challenge keys into lists of $d$ keys

$$\begin{aligned} pk_0 &= (Z_{0,0}, \ldots, Z_{0,d-1}) &&\coloneqq (\widetilde{pk}_0, \ldots, \widetilde{pk}_{d-1}) \\ pk_1 &= (Z_{1,0}, \ldots, Z_{1,d-1}) &&\coloneqq (\widetilde{pk}_d, \ldots, \widetilde{pk}_{2d-1}) \\ &\vdots \end{aligned}$$

obtaining $S \coloneqq \{pk_i\}_{i=0}^{\lfloor \frac{q}{d} \rfloor}$.

(3) $\mathtt{M}$ picks two random tapes **h, h'** to simulate oracle query responses for $\mathcal{F}^B$.

(4) $\mathtt{M}$ executes $\mathcal{F}^B$ in a black box, using $S$ as input. Upon receiving a corruption query from $\mathcal{F}^B$ on some $pk_i$, $\mathtt{M}$ makes a $\mathtt{CO}$ query on each $Z_{j,i}$ passing $sk_i$ back to $\mathcal{F}^B$. Each corruption query made by $\mathcal{F}^B$ consists of $d$ corruption queries made to $\mathtt{CO}$ by $\mathtt{M}$.

(5) If $\mathcal{F}^B$ fails, or if $\mathcal{F}^B$ succeeds with all zero coefficients $\mu_j$, then $\mathtt{M}$ samples random $\{w_k\} \in (\mathtt{F}_p)^v$, samples a random subset of challenge keys $\{pk_j^*\}_j \subseteq S$, samples random coefficients $\{h_j\}_j$, outputs $\{w_k\}, \{pk_j^*\}_j, \{h_j\}_j$, and terminates.

(6) Otherwise, $\mathtt{M}$ obtains two signature triples with the same message and ring, $(m, Q, \sigma), (m, Q, \sigma')$ at at least one non-zero aggregation coefficient. $\mathtt{M}$ computes the challenge discrete logarithms $w_k = \frac{s_{k,l} - s'_{k,l}}{c'_l - c_l}$. $\mathtt{M}$ outputs $w_k, \{\mu_j\}_j$, and $\{Z_{l,j}\}_j$.

Denote with $t_1$ the time it takes for $\mathtt{M}$ to inspect the transcript, perform field operations, and process corruption queries for $\mathcal{F}^B$. Then the algorithm $\mathtt{M}$ runs in time at most $2(t + t_0) + t_1$.

To complete the proof, consider the overall success probability and timing of $\mathtt{M}$. Since $\mathtt{A}$ is a $(t, \epsilon, q)$-solver of the unforgeability game and these are successful signatures, there must be at least one query made to $\mathtt{SO}$ corresponding to an uncorrupted challenge key linking to these signatures. In particular, $w_k \cdot G_k = W_{k,l} = \sum_{j:g(j)=k} \mu_j Z_{l,j}$ for some $(\{Z_{l,j}\}_j) \in Q$. The algorithm $\mathtt{M}$ succeeds at forking $\mathtt{B}$ and at least one coefficient $\mu_j$ is non-zero; we denote the probability of obtaining any zero coefficients as $\mu$. We note that $\mu$ is negligible under the random oracle model. Thus, $\mathtt{M}$ runs in time at most $2(t + t_0) + t_1$, has success probability exceeding $\epsilon(\frac{\epsilon}{q} - \frac{1}{2^\eta}) - \mu$. □

The proof of Theorem 2 demonstrates that the validity of a triple implies that the aggregated private keys $w_k$ are the discrete logarithms of the aggregated linking tags $\mathfrak{W}_k$ with respect to $H_l$ and are also the discrete logarithms of the aggregated keys $W_{k,l}$ with respect to $G_k$. In this way, the linking tag of a valid signature must be the linking tag corresponding to at least one ring member, except possibly with negligible probability.

**Corollary 1** (**No Alien Linking Tags**). *If there exists a PPT algorithm $\mathtt{A}$ that produces a valid signature triple $(m, Q, \sigma)$ with the scheme in Definition 10, then there exists a ring member in $Q$ whose aggregated keys $W_{k,l}$ have the same discrete logarithms $w_k$ with respect to $G_k$ as $\mathfrak{W}_k$ have with respect to $H_l$, and these $w_k$ are known to $\mathtt{A}$ (except possibly with negligible probability).*

**Theorem 3.** *The scheme in Definition 10 is linkable under Definition 5 and Definition 6.*

*Proof.* We show that valid, non-oracle signature triples from the scheme in Definition 10 satisfying the corrupted key conditions in the game of Definition 5 always link. Hence, any algorithm fails at that game except with negligible probability.

Assume that $\mathtt{A}$, while playing the game of ACST linkability from Definition 5, produces a pair of valid, non-oracle signature triples $(m, Q, \sigma), (m^*, Q^*, \sigma^*)$ such that at most one key in $Q \cup Q^*$ is corrupted or outside of $S$. This algorithm can be forked and rewound as above to compute the aggregated private key used in computing each signature, say $\boldsymbol{w} = \{w_{g(j)}\}, \boldsymbol{w^*} = \{w^*_{g(j)}\},\ j = 0 \ldots d-1$. At most one key in $Q \cup Q^*$ is corrupted or outside of $S$. Since $\mathtt{A}$ has knowledge of $\boldsymbol{w}$, then $\boldsymbol{w}$ is corrupted or outside of $S$, and likewise $\boldsymbol{w^*}$ is corrupted or outside of $S$. Since at most one key in $Q \cup Q^*$ can be corrupted or outside of $S$, we conclude $\boldsymbol{w} = \boldsymbol{w^*}$.

Since key aggregation is preimage-resistant by its construction using hash functions and $\boldsymbol{w} \circ \boldsymbol{G}$ is the aggregated public key for some public key $\boldsymbol{pk}_l = (\{Z_{l,j}\}_j) \in Q \cap Q^*$, $\boldsymbol{w}$ must be aggregated from a private key $(\{z_{l,j}\}_j)$ using the aggregation function. In both the case of single-key-oriented linkability and full-key-oriented linkability, the linkability tags are therefore exactly equal. Hence, with probability 1, the pair of triples $(m, Q, \sigma), (m^*, Q^*, \sigma^*)$ are linked, and $\mathtt{A}$ fails at ACST linkability except with negligible probability.

Similarly, an algorithm that outputs $q + 1$ unlinked signatures can be rewound to compute $2(q+1)$ signatures from which $q+1$ aggregated keys can be computed. Moreover, if these signatures are unlinked, then the $q + 1$ aggregated keys are distinct, violating $q$-pigeonhole linkability. □

**Theorem 4.** *If there exists a $(t, \epsilon, q)$-solver of the linkable anonymity game of Definition 9 under the construction of Definition 10, then there exists a $(t + t', \epsilon/2, q)$-solver of the RO-DDH[6] game of Definition 2 for some $t'$.*

*Proof.* Let $\mathtt{A}$ be such a solver of the linkable anonymity game. We will construct an algorithm $\mathtt{B}$ that executes $\mathtt{A}$ in a black box and is a solver of the RO-DDH game, acting as the challenger for $\mathtt{A}$; the algorithm will pass on $\mathcal{H}^p$ random oracle queries to its own challenger, flip coins for $\{\mathcal{H}^s_j\}$ random oracle queries, and simulate signing oracle queries by backpatching. We assume that $\mathtt{B}$ keeps internal tables to maintain consistency between the random oracle queries needed to simulate signing oracle queries.

The algorithm $\mathtt{B}$ operates as follows:

- $\mathtt{B}$ receives a set of tuples $\{(R_i, R'_i, R''_i)\}_{i=0}^{q-1}$ from its challenger, and chooses a bit $b' \in \{0, 1\}$ uniformly at random. Note that $\mathtt{B}$ does not know if its tuples are RO-DDH triples or not, as its challenger chose a secret bit $b \in \{0, 1\}$ uniformly at random to determine this.
- For all $i \in [0; q)$, $\mathtt{B}$ defines $Z_{i,0} \coloneqq R_i$ and records the $\mathcal{H}^p$ oracle mapping $\mathcal{H}^p(Z_{i,0}) = R'_i$. It chooses $\{z_{i,j}\}_{j=1}^{d-1}$ from $\mathtt{F}_p$ uniformly at random, and builds a set of public keys $S \coloneqq \{(Z_{i,0}, z_{i,1}G_{g(1)}, \ldots, z_{i,d-1}G_{g(d-1)})\}_{i=0}^{q-1}$. $\mathtt{B}$ provides the set $S$ to $\mathtt{A}$.
- $\mathtt{A}$ returns indices $0 \leq i_0, i_1 < q$ to $\mathtt{B}$.
- $\mathtt{B}$ receives signing oracle queries of the form $\mathtt{SO}(m, Q, pk)$, where $0 \leq l < q$ is the index of $pk \in Q, pk \in S$, and $|Q| = n$. There are two cases, which determine how $\mathtt{B}$ simulates the oracle response, flipping coins for $\{\mathcal{H}^s_j\}$ oracle queries:
  1. If it is the case that $\{pk_{i_0}, pk_{i_1}\} \not\subset Q$ or $pk \notin \{pk_{i_0}, pk_{i_1}\}$, then $\mathtt{B}$ proceeds with its signing oracle simulation using the key $pk$.
  2. Otherwise, there exists a bit $c \in \{0, 1\}$ such that $pk = pk_{i_c}$. In this case, $\mathtt{B}$ sets $c' \coloneqq c \oplus b'$ and proceeds with its signing oracle simulation using the key $pk_{i_{c'}}$. This is, if $b' = 0$, then $\mathtt{B}$ simulates a signature using the requested key from the player-provided index set. If instead $b' = 1$, then $\mathtt{B}$ simulates a signature using the other key.

  In either case, $\mathtt{B}$ parses the public key set $Q$ provided by $\mathtt{A}$. For any key $pk_i \coloneqq (Z'_{i,0}, Z'_{i,1}, \ldots, Z'_{i,d-1}) \in Q \setminus S$, it makes oracle queries to its challenger to obtain $\mathcal{H}^p(Z'_{i,0})$. Then $\mathtt{B}$ simulates the signature:

  1. Define a map $\pi : [0, n) \to [0, q) \cup \{\perp\}$ that maps indices of elements of $Q$ to the corresponding elements of $S$ (or returns the distinguished failure symbol $\perp$ for indices not mapping to elements of $S$), and let $0 \leq l < n$ be the index of $pk \in Q$.
  2. Choose $c_l, \{s_{k,i}\}_{k=0,i=0}^{v-1,n-1} \in \mathtt{F}_p$ uniformly at random.
  3. Since $pk \in S$ by construction, $\pi(l) \neq \perp$. Set $\mathfrak{T} \coloneqq \mathfrak{D}_0 \coloneqq R''_{\pi(l)}$ and $\{\mathfrak{D}_j\}_{j=1}^{d-1}$ such that each $\mathfrak{D}_j \coloneqq z_{\pi(l),j}\mathcal{H}^p(Z_{\pi(l),0})$.

[6] In this theorem and its proof we employ a slightly modified version of the RO-DDH (as defined in Definition 2), where the fixed generator $G$ is replaced with the fixed generator $G_{g(0)}$.

4. Define the following:

$$\mu_j \leftarrow \mathcal{H}^s_j(Q, \{\mathfrak{D}_j\}^{d-1}_{j=0}) \qquad \text{for } j \in [0, d)$$

$$W_{k,i} \coloneqq \begin{cases} \sum_{j:g(j)=k} \mu_j Z_{\pi(i),j} & (\pi(i) \neq \perp) \\ \sum_{j:g(j)=k} \mu_j Z'_{i,j} & (\pi(i) = \perp) \end{cases}$$

$$\mathfrak{W}_k \coloneqq \sum_{j:g(j)=k} \mu_j \mathfrak{D}_j$$

5. For each $i = l, l+1, \ldots, n-1, 0, \ldots, l-1$ (that is, indexing modulo n), define the following:

$$L_{k,i} \coloneqq s_{k,i} G_k + c_i W_{k,i}$$

$$R_{k,i} \coloneqq \begin{cases} s_{k,i} \mathcal{H}^p(Z_{\pi(i),0}) + c_i \mathfrak{W}_k & (\pi(i) \neq \perp) \\ s_{k,i} \mathcal{H}^p(Z'_{i,0}) + c_i \mathfrak{W}_k & (\pi(i) = \perp) \end{cases}$$

$$c_{i+1} \leftarrow \mathcal{H}^s_0(m, Q, \{L_{k,i}\}^{v-1}_{k=0}, \{R_{k,i}\}^{v-1}_{k=0})$$

6. B returns to A the tuple $(c_0, \{s_{k,i}\}^{v-1,n-1}_{k=0,i=0}, \{\mathfrak{D}_j\})$.

– A returns a bit $b^*$ to B.

– If $b^* = b'$, then B returns 0 to its challenger. Otherwise, it returns 1.

It is the case that B wins the RO-DDH game precisely when it correctly guesses the bit $b$ chosen by its challenger. Hence $\mathbb{P}[\text{B wins}] = \frac{1}{2}\mathbb{P}[\text{B} \rightarrow 0 | b = 0] + \frac{1}{2}\mathbb{P}[\text{B} \rightarrow 1 | b = 1]$.

if $b = 1$, then the RO-DDH challenger provided random points $\{R''_i\}$ that B used in its simulated signatures, so A can do no better than random chance at determining $b'$. Since $\text{B} \rightarrow 1$ exactly when A loses the linkable anonymity game, we have $\mathbb{P}[\text{B} \rightarrow 1 | b = 1] = \frac{1}{2}$.

On the other hand, if $b = 0$, then the RO-DDH challenger provided structured tuples that B used in its simulated signatures, and A wins the linkable anonymity game with non-negligible advantage $\epsilon$ over random chance. Since $\text{B} \rightarrow 0$ exactly when A wins the linkable anonymity game, we have $\mathbb{P}[\text{B} \rightarrow 0 | b = 0] = \frac{1}{2} + \epsilon$.

This means B wins the RO-DDH game with probability $\mathbb{P}[\text{B wins}] = \frac{1}{2} + \frac{\epsilon}{2}$ and has non-negligible advantage $\frac{\epsilon}{2}$. Further, B finishes with an added time $t'$ used in simulating oracle queries and performing lookups. Hence, B is a $(t + t', \epsilon/2, q)$-solver of the RO-DDH game. □

# References


[1] Brandon Goodell, Sarang Noether, and Arthur Blue. *Concise Linkable Ring Signatures and Forgery Against Adversarial Keys*. https://eprint.iacr.org/2019/654.pdf. 2019.

[2] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. *Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups*. https://eprint.iacr.org/2004/027. 2004.

[3] sowle and koe. *Zarcanum: A Proof-of-Stake Scheme for Confidential Transactions with Hidden Amounts*. https://eprint.iacr.org/2021/1478.pdf. 2021.