# Zano: tokens with hidden amounts (DRAFT)

sowle

July 26, 2022

**Abstract**

In this paper, we describe two possible options to implement tokens (a.k.a. colored coins) in Zano with unlimited decoy mixing capability and hidden amounts as an extension to the Ring Confidential Transactions scheme.

## 1 Notation

Let $\mathbb{G}$ denote the main subgroup of the Ed25519 curve and let $\mathbb{Z}_p$ denote a ring of integers modulo $p$.
Let $l$ be the order of $\mathbb{G}$: $l = \#\mathbb{G} = 2^{252} + 27742317777372353535851937790883648493$.
For any set $X$, $x \overset{\$}{\leftarrow} X$ means uniform sampling of $x$ at random from $X$.
$H_s$ is a scalar hash-function: $H_s : \{0,1\}^* \to \mathbb{Z}_l$
$H_p$ is a deterministic hash function: $H_p : \{0,1\}^* \to \mathbb{G}$.

## 2 Variant A

### 2.1 Basic idea

In a normal RingCT transaction each output's amount $a$ is committed to in a commitment

$$A = aH + fG$$

where $H, G \in \mathbb{G}$ are group elements with no efficiently-computable discrete logarithm relation and $f \in \mathbb{Z}_l$ is a random blinding factor.

The initial emission of a token in Zano will be made through a transaction which pays (or burns) some amount of native coins and puts a public *token descriptor* with the following information associated with the given token:

- unique name, symbol, text description, etc.;
- emission parameters (like total or unlimited supply);
- owner pubkey;
- etc.

The idea is to use a different amount-bonded generator $H_t = H_p(token\_descriptor\_t)$ for each token $t$ instead of the generator $H$. So the amount $a$ for the token $t$ is committed to in:

$$A_t = aH_p(token\_descriptor\_t) + fG$$

### 2.2 Hiding real token generator and range proofs

Let $X \in \mathbb{G}$ be another generator, i.e. has no efficiently-computable discrete logarithm relation with both $H$ and $G$. Then a typical transaction output would be represented by a tuple (hereinafter we omit output stealth address and encoded amount for simplicity):

$$(\sigma^{rp}, T, A = aT + fG)$$

where $T = H_t + rX = H_p(\mathit{token\_descriptor\_t}) + rX$ is masked generator, $r = H_s(...)$ is a pseudo-random mask, known only to the output's owner and the sender, and $\sigma^{rp}$ is a range proof for the fact, that $a < 2^{64}$.

We disclose the masked generator $T$ so blockchain observers can verify the range proof. We cannot disclose token-specific generator $H_t$ for the same purpose, because otherwise it would be possible for observers to learn the type of a corresponding token from it.

For example, consider a RingCT transaction with two inputs and four outputs ($m = 2, k = 4$). Each of input refers to a ring of three outputs ($n = 3$), and the second one is actually being spent.

| # | Ring | Pseudo output commitments | Ring signature add. assertions | Outputs |
|---|---|---|---|---|
| 0 | $(\sigma_0^{rp}, T_0, A_0 = ?)$ | | $A_0 - A_0^p = ?$ <br> $T_0 - T_0^p = ?$ | $E_0 = e_0(\underbrace{H_4 + r'_0X}_{}) + y_0G$ <br> $(\sigma_0'^{rp}, T'_0, E_0)$ |
| | $(\sigma_1^{rp}, T_1, A_1 = a_1(\underbrace{H_1 + r_1X}_{T_1}) + f_1G)$ | $A_0^p = a_1T_1 + f'_1G$ <br> $T_0^p = T_1 + r'_1X$ | $A_1 - A_0^p = (f_1 - f'_1)G$ <br> $T_1 - T_0^p = (r_1 - r'_1)X$ | $E_1 = e_1(\underbrace{H_4 + r'_1X}_{}) + y_1G$ <br> $(\sigma_1'^{rp}, T'_1, E_1)$ |
| | $(\sigma_2^{rp}, T_2, A_2 = ?)$ | | $A_2 - A_0^p = ?$ <br> $T_2 - T_0^p = ?$ | $E_2 = e_2(\underbrace{H_1 + r'_2X}_{}) + y_2G$ <br> $(\sigma_2'^{rp}, T'_2, E_2)$ |
| 1 | $(\sigma_3^{rp}, T_3, A_3 = ?)$ | | $A_3 - A_1^p = ?$ <br> $T_3 - T_1^p = ?$ | $E_3 = e_3(\underbrace{H_4 + r'_3X}_{}) + y_3G$ <br> $(\sigma_3'^{rp}, T'_3, E_3)$ |
| | $(\sigma_4^{rp}, T_4, A_4 = a_4(\underbrace{H_4 + r_4X}_{T_4}) + f_4G)$ | $A_1^p = a_4T_4 + f'_4G$ <br> $T_1^p = T_4 + r'_4X$ | $A_4 - A_1^p = (f_4 - f'_4)G$ <br> $T_4 - T_1^p = (r_4 - r'_4)X$ | |
| | $(\sigma_5^{rp}, T_5, A_5 = ?)$ | | $A_5 - A_1^p = ?$ <br> $T_5 - T_1^p = ?$ | |

In this example input 0 corresponds to the native coin, and so output $E_2$ (note using $H_1$). Hence $a_1 = e_2 + \mathit{fee}$[1]. Input 1 corresponds to a token, associated with generator $H_4$, and outputs 0, 1, and 3 use the same token-associated generator $H_4$. Hence, $a_4 = e_0 + e_1 + e_3$. Thanks to homomorphism we can combine both equations into one using corresponding commitments:

$$\sum_{i=0}^{m-1} A_i^p - \sum_{j=0}^{k-1} E_j - \text{fee} \cdot H_1 = rX + yG \tag{1}$$

where $m$ is the number of inputs, and $k$ is the number of outputs. Observers make sure that the equation above holds for some secret $r, y$ using a vector Schnorr proof. Note, that $r'_j$ and $y_j$ are calculated using a shared secret ($r'_j = H_s(...), y_j = H_s(...)$) so output's receiver is able to calculate them.

## 2.3 Enforcing output commitments

Amount-bonded generators $T'_j$ in outputs have to be restricted by additional proof to prevent malicious use. Consider the following approach.

For each output $E_j$ use a ring signature for a ring of public keys $\mathcal{R} = \{T_i^p - T'_j\},\ i = 0 \ldots m-1,\ j = 0 \ldots k-1$ to prove knowing a DL secret $x$ with respect to $X$ for one of a public key in the ring:

$$T_i^p - T'_j = xX$$

(where $m$ is the number of inputs and $k$ is the number of outputs).

In total, we need $k$ ring signatures, each having a ring of size $m$, so the estimated additional size is $km + 1$ scalars (the simplest case of non-aggregated ring signature).

Using different generators $T'_j$ for each output commitment requires additional steps to aggregate range proofs (otherwise we would need to use single range proof for each output which is very consuming). For each output commitment $E_j = e_jT'_j + y_jG$ we provide additional commitment to the same amount $E'_j = e_jU + y'_jG$ (where $U \in \mathbb{G}$ is another fixed generator with no efficiently-computable discrete logarithm relation with others), and a vector Schnorr proof of knowing DL $e_j, y''$ in

$$E_j - E'_j = e_j(T'_j - U) + y''G$$

which in total would require 1 group element $E'_j$ and 2 scalars per output, and one scalar for a shared Fiat-Shamir challenge and one aggregated range proof for all outputs.

This approach does not link any outputs to inputs, but relatively space- and computational expensive.

[1] Here we assume that the transaction fee can only be paid in native tokens, and its plain unencoded value explicitly stated in transaction data.

# 3 Variant B

## 3.1 Basic idea

Let $t = H_s(token\_descriptor) \in \mathbb{Z}_l$ be the token's unique public scalar identifier. Consider a corresponding commitment for amount $a$ having the following form:

$$A = aH + tX + fG$$

where $H, X, G \in \mathbb{G}$ are group elements with no efficiently-computable discrete logarithm relation and $f \in \mathbb{Z}_l$ is a random blinding factor. Then a typical transaction output would be represented by a tuple:

$$(\sigma^{rp}, A = aH + tX + fG)$$

Where $\sigma^{rp}$ is a double-blinded range proof with respect to amount $a$ (for instance, using double-blinded extension of Bulletproofs+, suggested in Zarcanum whitepaper).

Consider the same RingCT transaction as above with two inputs and four outputs ($m = 2, k = 4$). Each of input refers to a ring of three outputs, and the second of each is actually being spent.

| # | Ring | Pseudo output commitment | Ring signature add. assertion | Outputs |
|---|---|---|---|---|
|  | $(\sigma_0^{rp}, A_0 = ?)$ |  | $A - A_0^p = ?$ | $(\sigma_0'^{rp}, E_0 = e_0H + t_4X + y_0G)$ |
| 0 | $(\sigma_1^{rp}, A_1 = a_1H + t_1X + f_1G)$ | $A_0^p = a_1H + t_1X + f_1'G$ | $A - A_0^p = (f_1 - f_1')G$ |  |
|  | $(\sigma_2^{rp}, A_2 = ?)$ |  | $A - A_0^p = ?$ | $(\sigma_1'^{rp}, E_1 = e_1H + t_4X + y_1G)$ |
|  | $(\sigma_3^{rp}, A_3 = ?)$ |  | $A - A_1^p = ?$ | $(\sigma_2'^{rp}, E_2 = e_2H + t_1X + y_2G)$ |
| 1 | $(\sigma_4^{rp}, A_4 = a_4H + t_4X + f_4G)$ | $A_1^p = a_4H + t_4X + f_4'G$ | $A - A_1^p = (f_4 - f_4')G$ |  |
|  | $(\sigma_5^{rp}, A_5 = ?)$ |  | $A - A_1^p = ?$ | $(\sigma_3'^{rp}, E_3 = e_3H + t_4X + y_3G)$ |

In this example, as before, input 0 corresponds to the native coin, and so output 2 (note using $t_1$). Input 1 corresponds to a token, associated with scalar $t_4$, and outputs 0, 1, and 3 use the same token-associated scalar $t_4$. Hence we need to prove that the following equations hold:[2]

$$a_1 = e_2 + fee$$
$$a_4 = e_0 + e_1 + e_3$$

This can be done by proving knowledge of DL secrets $r_0, r_1, y_0', y_1'$ with respect to $X, G$ for each different token:

$$A_0^p - E_2 - fee \cdot H = r_0X + y_0'G$$
$$A_1^p - E_0 - E_1 - E_3 = r_1X + y_1'G$$

or

$$\sum_{inputs(t)} A_i^p - \sum_{outputs(t)} E_j = r_iX + y_i'G \qquad (2)$$

where $inputs(t)$ and $outputs(t)$ are sets of inputs and outputs associated with the given token $t$.

As each output $j$ is explicitly connected to some input $i$ with the proof, this approach has an obvious disadvantage of linking inputs and outputs of the same token, which decreases anonymity.

## 3.2 Enforcing input-output commitments linkage

To ensure that all $A_i^p$ and $E_j$ in Eq. 2 correspond to the same token, the sender needs to prove that all of them have the same $tX$ component. To do that, the sender calculates a Schnorr-like proof of knowing DL secrets $x_h, x_g$ of $A_i^p - E_j$ with respect to $H, G$:

$$A_i^p - E_j = x_hH + x_gG$$

for each pair of $A_i^p$, $E_j$ chain in Eq. 2. Having $m$ inputs, $k$ outputs and $n_T$ tokens in a transaction, we would need to use $m + k - n_t$ such proofs. Each such proof would require 2 scalars, and Fiat-Shamir challenge can be shared.

[2]As above, we assume that the transaction fee can only be paid in native tokens, and its plain unencoded value explicitly stated in transaction data.

## 4 Possible attack vector and mitigation

Variant A is sensitive to cryptographic properties of the deterministic hash function $H_p$. Namely, we need to ensure that there's no efficiently-computable way to solve the following problem A:

*Problem A: Let $H_t = H_p(T_0)$. Given $H_t$ and $T_0$ find $x, y$ such that:*

$$H_p(x) = yH_p(T_0)$$

Otherwise one would be able to generate arbitrary amount of tokens while still keeping Eq. 1 hold.

The complexity of brute force attacks (including MITM) can possibly be increased by using computational-expensive hash function $H_e$ to calculate token-specific generator $H_t$:

$$H_t = H_p(H_e(token\_\ descriptor\_\ t))$$

because normally the calculation of $H_t$ is rare operation and its result can be cached.

## 5 Size comparison

Let's try to estimate size of the all signatures and proofs for a transaction with $m$ inputs (each having $n$ elements in its ring) and $k$ outputs.

| Parameter | Variant A | Variant B |
|---|---|---|
| Inputs (key images) | $m\ \mathcal{G}$ | $m\ \mathcal{G}$ |
| Pseudo output commitments | $2m\ \mathcal{G}$ | $m\ \mathcal{G}$ |
| Ring signature (CLSAG) | $(n+1)m\ \mathcal{S}$ [3] | $(n+1)m\ \mathcal{S}$ |
| Outputs' range proofs (BP+) | $(\mathcal{G} + 2\mathcal{S})k + \mathcal{S} + [(2 \cdot \lceil \log_2(64) + \log_2(1) \rceil + 3)\mathcal{G} + 3\mathcal{S}]$ | $(2 \cdot \lceil \log_2(64) + \log_2(k) \rceil + 3)\mathcal{G} + 4\mathcal{S}$ [4] |
| Outputs' additional proofs | $(km+1)\ \mathcal{S}$ [5] | $(2(m+k-n_t)+1)\ \mathcal{S}$ [6] |
| Outputs data (except proofs) | $(3\ \mathcal{G} + 1\ \mathcal{S})k$ | $(2\ \mathcal{G} + 1\ \mathcal{S})k$ |
| Total | $(3m+4k+15)\ \mathcal{G}$ <br> $((n+k+1)m+3k+5)\ \mathcal{S}$ | $(2m+2k+2\lceil \log_2(k) \rceil + 15)\ \mathcal{G}$ <br> $((n+3)m+3k-2n_t+5)\ \mathcal{S}$ |

Table 1: Size comparison. $\mathcal{G}$ denotes group elements and $\mathcal{S}$ denotes field elements.

[3]CLSAG compresses all additional layers.

[4]One aggregated Bulletproofs+ proof with double-blinded commitments

[5]A ring signature for each output.

[6]2-generators Schnorr proofs for each output with shared Fiat-Shamir challenge.