Structure-Preserving Signatures on Equivalence Classes (SPS-EQ)

Adwaiya Srivastav

SPS EQ Project

April 15, 2025

Outline

Introduction to SPS-EQ

- Primitive introduced by G. Fuchsbauer, C. Hanser, and D. Slamanig in 2015.

Introduction to SPS-EQ

- Primitive introduced by G. Fuchsbauer, C. Hanser, and D. Slamanig in 2015.
- Based on bilinear pairings

Introduction to SPS-EQ

- Primitive introduced by G. Fuchsbauer, C. Hanser, and D. Slamanig in 2015.
- Based on bilinear pairings
- Messages are equivalence classes of group elements

Introduction to SPS-EQ

- Primitive introduced by G. Fuchsbauer, C. Hanser, and D. Slamanig in 2015.
- Based on bilinear pairings
- Messages are equivalence classes of group elements
- Signatures remain valid across equivalent messages

Introduction to SPS-EQ

- Primitive introduced by G. Fuchsbauer, C. Hanser, and D. Slamanig in 2015.
- Based on bilinear pairings
- Messages are equivalence classes of group elements
- Signatures remain valid across equivalent messages
- Extends randomizable signatures where a tuple (M, σ) using a scalar μ and pk can be converted to (M, σ').

Introduction to SPS-EQ

- Primitive introduced by G. Fuchsbauer, C. Hanser, and D. Slamanig in 2015.
- Based on bilinear pairings
- Messages are equivalence classes of group elements
- Signatures remain valid across equivalent messages
- Extends randomizable signatures where a tuple (M, σ) using a scalar μ and pk can be converted to (M, σ').
- Here, a tuple (M, σ) can be converted to (M', σ') using a scalar μ and pk.

Mathematical Background: Bilinear Pairings

For elliptic curve groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ with generators $P \in \mathbb{G}_1$ and $\hat{P} \in \mathbb{G}_2$, a bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that satisfies the following properties:

- Bilinear: $e(aP, b\hat{P}) = e(P, \hat{P})^{ab}$
- Non-degenerate: $e(P, \hat{P}) \neq 1$
- Computable: There exists an efficient algorithm to compute $e(S, T)$ for any $S \in \mathbb{G}_1$ and $T \in \mathbb{G}_2$

Mathematical Background: Bilinear Pairings

For elliptic curve groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ with generators $P \in \mathbb{G}_1$ and $\hat{P} \in \mathbb{G}_2$, a bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that satisfies the following properties:

- Bilinear: $e(aP, b\hat{P}) = e(P, \hat{P})^{ab}$
- Non-degenerate: $e(P, \hat{P}) \neq 1$
- Computable: There exists an efficient algorithm to compute $e(S, T)$ for any $S \in \mathbb{G}_1$ and $T \in \mathbb{G}_2$

Types:

- Type-1: $\mathbb{G}_1 = \mathbb{G}_2$
- Type-2: $\mathbb{G}_1 \neq \mathbb{G}_2$ but there exists an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$
- Type-3: No restrictions on $\mathbb{G}_1$ and $\mathbb{G}_2$ **(Used in this work)**

Mathematical Background: Projective Space

Lets define a relation $\mathcal{R}$ on $(\mathbb{G}_1^*)^\ell$ as follows:

$$\mathcal{R} = \{(M, M') \mid (M, M') \in (\mathbb{G}_1^*)^{2\cdot\ell} \text{ and } \exists \mu \in \mathbb{Z}_p^* \text{ s.t. } M' = \mu \cdot M\}$$

Proposition: $\mathcal{R}$ is an equivalence relation$\leftrightarrow |\mathbb{G}_1| = p$ prime.
Proof:

- $\Rightarrow$ For transitivity we require if $A \mathcal{R} B$ and $B \mathcal{R} C$ then $A \mathcal{R} C$. Thus, $B = \nu \cdot C$ and $A = \mu \cdot B = (\mu \cdot \nu) \cdot C$. Thus, $\mu \cdot \nu \in \mathbb{Z}_n^*$ for arbitrary $\mu, \nu \in \mathbb{Z}_n^*$. This means there are no zero divisors in $\mathbb{Z}_n^*$ implying n is prime.
- $\Leftarrow$ Reflexivity: For any $M \in (\mathbb{G}_1^*)^\ell$, we have $M = 1 \cdot M$, so $M \mathcal{R} M$.
  Symmetry: If $M \mathcal{R} M'$, then $M' = \mu \cdot M$ for some $\mu \in \mathbb{Z}_p^*$. Since $\mu \in \mathbb{Z}_p^*$, μ^{-1} exists, giving $M = \mu^{-1} \cdot M'$, thus $M' \mathcal{R} M$.
  Transitivity: If $M \mathcal{R} M'$ and $M' \mathcal{R} M''$, then $M' = \mu \cdot M$ and $M'' = \nu \cdot M'$ for some $\mu, \nu \in \mathbb{Z}_p^*$. Thus $M'' = \nu \cdot (\mu \cdot M) = (\nu\mu) \cdot M$. Since $\mathbb{Z}_p^*$ has no zero divisors, $\nu\mu \in \mathbb{Z}_p^*$, so $M \mathcal{R} M''$.

Mathematical Background: Projective Space

- **Definition 0.1:** For a message $M \in (\mathbb{G}_1^*)^\ell$, we define its mutual ratios as:

$$r_M = \left(\frac{M_1}{x_2}, \frac{M_1}{x_3}, \ldots, \frac{M_1}{x_\ell}, \frac{M_2}{x_3}, \ldots, \frac{M_2}{x_\ell}, \ldots, \frac{M_\ell}{x_\ell}\right)$$

where x_i is the discrete log of M_i in $\mathbb{G}_1$.

Mathematical Background: Projective Space

- **Definition 0.1:** For a message $M \in (\mathbb{G}_1^*)^\ell$, we define its mutual ratios as:

$$r_M = \left(\frac{M_1}{x_2}, \frac{M_1}{x_3}, \ldots, \frac{M_1}{x_\ell}, \frac{M_2}{x_3}, \ldots, \frac{M_2}{x_\ell}, \ldots, \frac{M_\ell}{x_\ell} \right)$$

where x_i is the discrete log of M_i in $\mathbb{G}_1$.

Mathematical Background: Projective Space

- **Definition 0.1:** For a message $M \in (\mathbb{G}_1^*)^\ell$, we define its mutual ratios as:

$$r_M = \left(\frac{M_1}{x_2}, \frac{M_1}{x_3}, \ldots, \frac{M_1}{x_\ell}, \frac{M_2}{x_3}, \ldots, \frac{M_2}{x_\ell}, \ldots, \frac{M_\ell}{x_\ell} \right)$$

  where x_i is the discrete log of M_i in $\mathbb{G}_1$.

- **Proposition 0.1:** Mutual ratios remain invariant under scalar multiplication.

Mathematical Background: Projective Space

- **Definition 0.1:** For a message $M \in (\mathbb{G}_1^*)^\ell$, we define its mutual ratios as:

$$r_M = \left(\frac{M_1}{x_2}, \frac{M_1}{x_3}, \ldots, \frac{M_1}{x_\ell}, \frac{M_2}{x_3}, \ldots, \frac{M_2}{x_\ell}, \ldots, \frac{M_\ell}{x_\ell} \right)$$

  where x_i is the discrete log of M_i in $\mathbb{G}_1$.

- **Proposition 0.1:** Mutual ratios remain invariant under scalar multiplication. *Proof:*

$$\begin{aligned} r_{\mu \cdot M} &= \left(\frac{\mu \cdot M_1}{\mu x_2}, \frac{\mu \cdot M_1}{\mu x_3}, \ldots, \frac{\mu \cdot M_1}{\mu x_\ell}, \frac{\mu \cdot M_2}{\mu x_3}, \ldots, \frac{\mu \cdot M_2}{\mu x_\ell}, \ldots, \frac{\mu \cdot M_\ell}{\mu x_\ell} \right) \\ &= \left(\frac{M_1}{x_2}, \frac{M_1}{x_3}, \ldots, \frac{M_1}{x_\ell}, \frac{M_2}{x_3}, \ldots, \frac{M_2}{x_\ell}, \ldots, \frac{M_\ell}{x_\ell} \right) \\ &= r_M \end{aligned}$$

Mathematical Background: Projective Space

- **Proposition 0.2:** $M \mathcal{R} M' \leftrightarrow r_M = r_{M'}$.

Mathematical Background: Projective Space

- **Proposition 0.2:** $M \mathcal{R} M' \leftrightarrow r_M = r_{M'}$.
  - $\Rightarrow$ If $M \mathcal{R} M'$, then $M' = \mu \cdot M$ for some $\mu \in \mathbb{Z}_p^*$. Thus by above proposition, $r_M = r_{M'}$.
  - $\Leftarrow$ If $r_M = r_{M'}$, then $M_i = \mu \cdot M'_i$ for some $\mu \in \mathbb{Z}_p^*$. Thus, $M \mathcal{R} M'$.

- **Proposition 0.2:** $M \mathcal{R} M' \leftrightarrow r_M = r_{M'}$.
  - $\Rightarrow$ If $M \mathcal{R} M'$, then $M' = \mu \cdot M$ for some $\mu \in \mathbb{Z}_p^*$. Thus by above proposition, $r_M = r_{M'}$.
  - $\Leftarrow$ If $r_M = r_{M'}$, then $M_i = \mu \cdot M'_i$ for some $\mu \in \mathbb{Z}_p^*$. Thus, $M \mathcal{R} M'$.

  Mutual ratios can be used to check if two messages are equivalent by only knowing the discrete log of one of them.

Mathematical Background: Projective Space

- **Proposition 0.2:** $M \mathcal{R} M' \leftrightarrow r_M = r_{M'}$.
  - $\Rightarrow$ If $M \mathcal{R} M'$, then $M' = \mu \cdot M$ for some $\mu \in \mathbb{Z}_p^*$. Thus by above proposition, $r_M = r_{M'}$.
  - $\Leftarrow$ If $r_M = r_{M'}$, then $M_i = \mu \cdot M_i'$ for some $\mu \in \mathbb{Z}_p^*$. Thus, $M \mathcal{R} M'$.

  Mutual ratios can be used to check if two messages are equivalent by only knowing the discrete log of one of them.

  Consider two messages $M, M' \in (\mathbb{G}_1^*)^\ell$ and $x_1, x_2, \ldots, x_\ell \in \mathbb{Z}_p$ s.t. $M_i = x_i \cdot P$. Define,

$$r_{(M',M)} = \left(\frac{M_1'}{x_2}, \frac{M_1'}{x_3}, \ldots, \frac{M_1'}{x_\ell}, \frac{M_2'}{x_3}, \ldots, \frac{M_2'}{x_\ell}, \ldots, \frac{M_\ell'}{x_\ell} \right).$$

Mathematical Background: Projective Space

- **Proposition 0.2:** $M \mathcal{R} M' \leftrightarrow r_M = r_{M'}$.
  - $\Rightarrow$ If $M \mathcal{R} M'$, then $M' = \mu \cdot M$ for some $\mu \in \mathbb{Z}_p^*$. Thus by above proposition, $r_M = r_{M'}$.
  - $\Leftarrow$ If $r_M = r_{M'}$, then $M_i = \mu \cdot M_i'$ for some $\mu \in \mathbb{Z}_p^*$. Thus, $M \mathcal{R} M'$.

  Mutual ratios can be used to check if two messages are equivalent by only knowing the discrete log of one of them.

  Consider two messages $M, M' \in (\mathbb{G}_1^*)^\ell$ and $x_1, x_2, \ldots, x_\ell \in \mathbb{Z}_p$ s.t. $M_i = x_i \cdot P$. Define,

$$r_{(M',M)} = \left(\frac{M_1'}{x_2}, \frac{M_1'}{x_3}, \ldots, \frac{M_1'}{x_\ell}, \frac{M_2'}{x_3}, \ldots, \frac{M_2'}{x_\ell}, \ldots, \frac{M_\ell'}{x_\ell} \right).$$

Claim: $M \mathcal{R} M' \leftrightarrow r_M = \lambda \cdot r_{(M',M)}$.

Mathematical Background: Projective Space

- **Proposition 0.2:** $M \mathcal{R} M' \leftrightarrow r_M = r_{M'}$.
  - $\Rightarrow$ If $M \mathcal{R} M'$, then $M' = \mu \cdot M$ for some $\mu \in \mathbb{Z}_p^*$. Thus by above proposition, $r_M = r_{M'}$.
  - $\Leftarrow$ If $r_M = r_{M'}$, then $M_i = \mu \cdot M'_i$ for some $\mu \in \mathbb{Z}_p^*$. Thus, $M \mathcal{R} M'$.

  Mutual ratios can be used to check if two messages are equivalent by only knowing the discrete log of one of them.

Consider two messages $M, M' \in (\mathbb{G}_1^*)^\ell$ and $x_1, x_2, \ldots, x_\ell \in \mathbb{Z}_p$ s.t. $M_i = x_i \cdot P$. Define,

$$r_{(M',M)} = \left(\frac{M'_1}{x_2}, \frac{M'_1}{x_3}, \ldots, \frac{M'_1}{x_\ell}, \frac{M'_2}{x_3}, \ldots, \frac{M'_2}{x_\ell}, \ldots, \frac{M'_\ell}{x_\ell} \right).$$

Claim: $M \mathcal{R} M' \leftrightarrow r_M = \lambda \cdot r_{(M',M)}$.

Proof:

- $\Rightarrow$ If $M \mathcal{R} M'$, then $M = \mu \cdot M'$ for some $\mu \in \mathbb{Z}_p^*$. Thus, $r_M = \mu \cdot r_{(M',M)}$.
- $\Leftarrow$ If $r_M = \lambda \cdot r_{(M',M)}$, then $M_i = \lambda \cdot M'_i$ for some $\lambda \in \mathbb{Z}_p^*$. Thus, $M \mathcal{R} M'$.

SPS-EQ

Definition(SPS-EQ): A structure-preserving signature scheme for equivalence relation $\mathcal{R}$ over $\mathbb{G}_1$ is a tuple SPS EQ of following PPT algorithms:

SPS-EQ

Definition(SPS-EQ): A structure-preserving signature scheme for equivalence relation $\mathcal{R}$ over $\mathbb{G}_1$ is a tuple SPS EQ of following PPT algorithms:

- $\mathbf{BGGgen}_{\mathcal{R}}(1^{\kappa})$: is a bilinear group generator algorithm that outputs a bilinear group BG of prime order p where p is a κ bit prime

SPS-EQ

Definition(SPS-EQ): A structure-preserving signature scheme for equivalence relation $\mathcal{R}$ over $\mathbb{G}_1$ is a tuple SPS EQ of following PPT algorithms:

- **BGGgen**$_{\mathcal{R}}(1^{\kappa})$: is a bilinear group generator algorithm that outputs a bilinear group BG of prime order *p* where *p* is a κ bit prime
- **KeyGen**$_{\mathcal{R}}(\mathrm{BG}, 1^{\ell})$: is a probabilistic algorithm which on input a bilinear group BG and a vector of length $\ell > 1$, outputs a key pair (sk, pk)

SPS-EQ

Definition(SPS-EQ): A structure-preserving signature scheme for equivalence relation $\mathcal{R}$ over $\mathbb{G}_1$ is a tuple SPS EQ of following PPT algorithms:

- $\textbf{BGGgen}_{\mathcal{R}}(1^{\kappa})$: is a bilinear group generator algorithm that outputs a bilinear group BG of prime order p where p is a κ bit prime
- $\textbf{KeyGen}_{\mathcal{R}}(\mathrm{BG}, 1^{\ell})$: is a probabilistic algorithm which on input a bilinear group BG and a vector of length $\ell > 1$, outputs a key pair (sk, pk)
- $\textbf{Sign}_{\mathcal{R}}(\vec{M}, sk)$: is a probabilistic algorithm that which on input a representative $\vec{M} \in (\mathbb{G}_1^*)^{\ell}$ of an equivalence class $[\vec{M}]_{\mathcal{R}}$, a secret key sk, outputs a signature σ on $\vec{M}$

SPS-EQ

Definition(SPS-EQ): A structure-preserving signature scheme for equivalence relation $\mathcal{R}$ over $\mathbb{G}_1$ is a tuple SPS EQ of following PPT algorithms:

- $\textbf{BGGgen}_{\mathcal{R}}(1^\kappa)$: is a bilinear group generator algorithm that outputs a bilinear group BG of prime order p where p is a κ bit prime
- $\textbf{KeyGen}_{\mathcal{R}}(\mathsf{BG}, 1^\ell)$: is a probabilistic algorithm which on input a bilinear group BG and a vector of length $\ell > 1$, outputs a key pair (sk, pk)
- $\textbf{Sign}_{\mathcal{R}}(\vec{M}, sk)$: is a probabilistic algorithm that which on input a representative $\vec{M} \in (\mathbb{G}_1^*)^\ell$ of an equivalence class $[\vec{M}]_{\mathcal{R}}$, a secret key sk, outputs a signature σ on $\vec{M}$
- $\textbf{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk)$: is a probabilistic algorithm which on input a representative $\vec{M} \in (\mathbb{G}_1^*)^\ell$ of an equivalence class $[\vec{M}]_{\mathcal{R}}$, a signature σ on $\vec{M}$, a scalar μ, and a public key pk, outputs an updated signature σ' on $\vec{M}' = \mu \cdot \vec{M}$

Abstract Scheme (continued)

- **Verify**$_{\mathcal{R}}(\vec{M}, \sigma, pk)$: is a deterministic algorithm which on input a representative $\vec{M} \in (\mathbb{G}_1^*)^\ell$, and a signature σ, outputs 1 if σ is valid for $\vec{M}$ under pk and 0 otherwise.

Abstract Scheme (continued)

- **Verify**$_{\mathcal{R}}(\vec{M}, \sigma, pk)$: is a deterministic algorithm which on input a representative $\vec{M} \in (\mathbb{G}_1^*)^\ell$, and a signature σ, outputs 1 if σ is valid for $\vec{M}$ under pk and 0 otherwise.
- **VKey**$_{\mathcal{R}}(sk, pk)$: is a deterministic algorithm which on input a secret key sk and a public key pk, checks their consistency and outputs 1 if sk is valid for pk and 0 otherwise.

Notions: Correctness

Definition 1 (Correctness): An SPS-EQ scheme SPS-EQ over $\mathbb{G}_1$ is correct if for all security parameters $\kappa \in \mathbb{N}$, for all $\ell > 1$, and all bilinear groups $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \in [BGGen_R(1^\kappa)]$, all key pairs $(sk, pk) \in [\mathsf{KeyGen}(\mathsf{BG}, 1^\ell)]$ and all messages $\vec{M} \in (\mathbb{G}_1^*)^\ell$ and scalars $\mu \in \mathbb{Z}_p^*$, the following holds:

Notions: Correctness

Definition 1 (Correctness): An SPS-EQ scheme SPS-EQ over $\mathbb{G}_1$ is correct if for all security parameters $\kappa \in \mathbb{N}$, for all $\ell > 1$, and all bilinear groups $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \in [BGGen_R(1^\kappa)]$, all key pairs $(sk, pk) \in [\mathsf{KeyGen}(\mathsf{BG}, 1^\ell)]$ and all messages $\vec{M} \in (\mathbb{G}_1^*)^\ell$ and scalars $\mu \in \mathbb{Z}_p^*$, the following holds:

- $\mathsf{VKey}_{\mathcal{R}}(sk, pk) = 1$

Notions: Correctness

Definition 1 (Correctness): An SPS-EQ scheme SPS-EQ over $\mathbb{G}_1$ is correct if for all security parameters $\kappa \in \mathbb{N}$, for all $\ell > 1$, and all bilinear groups $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \in [BGGen_R(1^\kappa)]$, all key pairs $(sk, pk) \in [\mathsf{KeyGen}(\mathsf{BG}, 1^\ell)]$ and all messages $\vec{M} \in (\mathbb{G}_1^*)^\ell$ and scalars $\mu \in \mathbb{Z}_p^*$, the following holds:

- $\mathsf{VKey}_{\mathcal{R}}(sk, pk) = 1$
- $\mathsf{Verify}_{\mathcal{R}}(pk, \vec{M}, \mathsf{Sign}_{\mathcal{R}}(sk, \vec{M})) = 1$

Notions: Correctness

Definition 1 (Correctness): An SPS-EQ scheme SPS-EQ over $\mathbb{G}_1$ is correct if for all security parameters $\kappa \in \mathbb{N}$, for all $\ell > 1$, and all bilinear groups $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \in [BGGen_R(1^\kappa)]$, all key pairs $(sk, pk) \in [\mathsf{KeyGen}(\mathsf{BG}, 1^\ell)]$ and all messages $\vec{M} \in (\mathbb{G}_1^*)^\ell$ and scalars $\mu \in \mathbb{Z}_p^*$, the following holds:

- $\mathsf{VKey}_{\mathcal{R}}(sk, pk) = 1$
- $\mathsf{Verify}_{\mathcal{R}}(pk, \vec{M}, \mathsf{Sign}_{\mathcal{R}}(sk, \vec{M})) = 1$
- $\mathsf{Verify}_{\mathcal{R}}(\vec{M}', \mathsf{ChgRep}_{\mathcal{R}}(\vec{M}, \mathsf{Sign}_{\mathcal{R}}(sk, \vec{M}), \mu, pk)) = 1$

Notions: EUF-CMA Security

Definition 2 (EUF-CMA Security): An SPS-EQ scheme over $\mathbb{G}_1$ is existentially un- forgeable under *adaptive chosen-message attacks* if for all $\ell > 1$, all PPT algorithms $\mathcal{A}$ with oracle access to $\mathsf{Sign}_{\mathcal{R}}(\cdot, sk)$, $\exists$ negligible function $\epsilon(\cdot)$:

Notions: EUF-CMA Security

Definition 2 (EUF-CMA Security): An SPS-EQ scheme over $\mathbb{G}_1$ is existentially un- forgeable under *adaptive chosen-message attacks* if for all $\ell > 1$, all PPT algorithms $\mathcal{A}$ with oracle access to $\mathsf{Sign}_{\mathcal{R}}(\cdot, sk)$, $\exists$ negligible function $\epsilon(\cdot)$:

$$\Pr\left[\begin{array}{c} \mathsf{BG} \xleftarrow{\$} \mathsf{BGGen}_{\mathcal{R}}(1^{\kappa}), (sk, pk) \xleftarrow{\$} \mathsf{KeyGen}_{\mathcal{R}}(\mathsf{BG}, 1^{\ell}), \\ (\vec{M}^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}_{\mathcal{R}}(\cdot, sk)}(pk) : \forall \vec{M} \in Q : [\vec{M}^*]_{\mathcal{R}} \neq [\vec{M}]_{\mathcal{R}} \\ \wedge\ \mathsf{Verify}_{\mathcal{R}}(\vec{M}^*, \sigma^*, pk) = 1 \end{array}\right] \leq \epsilon(\kappa)$$

where Q is the set of queries that $\mathcal{A}$ makes to the signing oracle.

Notions: Class Hiding

Definition 3 (Class Hiding): Let $\ell > 1$ and $\mathbb{G}_i^*$ be a base group of a bilinear group. The message space $(\mathbb{G}_i^*)^\ell$ is *class-hiding* if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that

Notions: Class Hiding

Definition 3 (Class Hiding): Let $\ell > 1$ and $\mathbb{G}_i^*$ be a base group of a bilinear group. The message space $(\mathbb{G}_i^*)^\ell$ is *class-hiding* if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that

$$\Pr\left[\begin{array}{c} b \xleftarrow{\$} \{0,1\}, \mathsf{BG} \xleftarrow{\$} \mathsf{BGGen}_{\mathcal{R}}(1^\kappa), \vec{M} \xleftarrow{\$} (\mathbb{G}_i^*)^\ell, \\ \vec{M}^{(0)} \xleftarrow{\$} (\mathbb{G}_i^*)^\ell, \vec{M}^{(1)} \xleftarrow{\$} [\vec{M}]_{\mathcal{R}}, b^* \xleftarrow{\$} \mathcal{A}(\mathsf{BG}, \vec{M}, \vec{M}^{(b)}) : \\ b^* = b \end{array}\right] - \frac{1}{2} \leq \epsilon(\kappa)$$

Notions: Class Hiding

Definition 3 (Class Hiding): Let $\ell > 1$ and $\mathbb{G}_i^*$ be a base group of a bilinear group. The message space $(\mathbb{G}_i^*)^\ell$ is *class-hiding* if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that

$$\Pr\left[\begin{array}{c} b \xleftarrow{\$} \{0,1\}, \mathsf{BG} \xleftarrow{\$} \mathsf{BGGen}_{\mathcal{R}}(1^\kappa), \vec{M} \xleftarrow{\$} (\mathbb{G}_i^*)^\ell, \\ \vec{M}^{(0)} \xleftarrow{\$} (\mathbb{G}_i^*)^\ell, \vec{M}^{(1)} \xleftarrow{\$} [\vec{M}]_{\mathcal{R}}, b^* \xleftarrow{\$} \mathcal{A}(\mathsf{BG}, \vec{M}, \vec{M}^{(b)}) : \\ b^* = b \end{array}\right] - \frac{1}{2} \leq \epsilon(\kappa)$$

Proposition 1: Let $\ell > 1$ and $\mathbb{G}$ be a group of prime order p. Then $(\mathbb{G}^*)^\ell$ is a class- hiding message space if and only if the DDH assumption holds in $\mathbb{G}$.

Notions: Signature Adaptation

Definition 4 (Signature Adaptation): Let $\ell > 1$. An SPS-EQ scheme on $(\mathbb{G}_i^*)^\ell$ perfectly adapts signatures if for all tuples $(sk, pk, \vec{M}, \sigma, \mu)$ with:

- $\mathsf{VKey}_\mathcal{R}(sk, pk) = 1$
- $\vec{M} \in (\mathbb{G}_1^*)^\ell$
- $\mathsf{Verify}_\mathcal{R}(\vec{M}, \sigma, pk) = 1$
- $\mu \in \mathbb{Z}_p^*$

the distributions of $\mathsf{ChgRep}_\mathcal{R}(\vec{M}, \sigma, \mu, pk)$ and $\mathsf{Sign}_\mathcal{R}(\mu \cdot \vec{M}, sk)$ are identical.

Notions: Signature Adaptation under Malicious Keys

Definition 5 (Signature Adaptation under Malicious Keys): Let $\ell > 1$. An SPS-EQ scheme on $(\mathbb{G}_i^*)^\ell$ perfectly adapts signatures under malicious keys if for all tuples $(pk, \vec{M}, \sigma, \mu)$ with:

- $\vec{M} \in (\mathbb{G}_i^*)^\ell$
- $\mathsf{Verify}_{\mathcal{R}}(\vec{M}, \sigma, pk) = 1$
- $\mu \in \mathbb{Z}_p^*$

we have the output $\mathsf{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk)$ is a uniformly random element in the space of signatures, conditioned on $\mathsf{Verify}_{\mathcal{R}}(\mu \cdot \vec{M}, \sigma, pk) = 1$.

SPS-EQ: Bilinear Group Generation

$\mathbf{BGGgen}_{\mathcal{R}}(1^{\kappa})$:

- Outputs $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P})$

SPS-EQ: Bilinear Group Generation

BGGgen$_{\mathcal{R}}(1^\kappa)$:

- Outputs $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P})$
- $\mathbb{G}_1, \mathbb{G}_2$ are bilinear groups of prime order p with generator $P \in \mathbb{G}_1$, $\hat{P} \in \mathbb{G}_2$

SPS-EQ: Bilinear Group Generation

$\textbf{BGGgen}_{\mathcal{R}}(1^{\kappa})$:

- Outputs $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P})$
- $\mathbb{G}_1, \mathbb{G}_2$ are bilinear groups of prime order p with generator $P \in \mathbb{G}_1$, $\hat{P} \in \mathbb{G}_2$
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a Type-3 bilinear pairing

SPS-EQ: Key Generation

KeyGen$(\mathsf{BG}, 1^\ell)$:

- Generate secret key $sk = (x_1, \ldots, x_\ell) \in (\mathbb{Z}_p^*)^\ell$

SPS-EQ: Key Generation

KeyGen$(\mathsf{BG}, 1^\ell)$:

- Generate secret key $sk = (x_1, \ldots, x_\ell) \in (\mathbb{Z}_p^*)^\ell$
- Compute public key $pk = (X_1, \ldots, X_\ell) = (x_1\hat{P}, \ldots, x_\ell\hat{P}) \in \mathbb{G}_2^\ell$

SPS-EQ: Key Generation

KeyGen$(\mathsf{BG}, 1^\ell)$:

- Generate secret key $sk = (x_1, \ldots, x_\ell) \in (\mathbb{Z}_p^*)^\ell$
- Compute public key $pk = (X_1, \ldots, X_\ell) = (x_1\hat{P}, \ldots, x_\ell\hat{P}) \in \mathbb{G}_2^\ell$
- Return (sk, pk)

SPS-EQ: Key Generation

KeyGen$(\mathsf{BG}, 1^\ell)$:

- Generate secret key $sk = (x_1, \ldots, x_\ell) \in (\mathbb{Z}_p^*)^\ell$
- Compute public key $pk = (X_1, \ldots, X_\ell) = (x_1\hat{P}, \ldots, x_\ell\hat{P}) \in \mathbb{G}_2^\ell$
- Return (sk, pk)

Intuition:

- $(x_1, \ldots, x_\ell)$ are the discrete logs of $(X_1, \ldots, X_\ell)$ with respect to $\hat{P}$
- By DLOG assumption, it is infeasible to compute $(x_1, \ldots, x_\ell)$ from $(X_1, \ldots, X_\ell)$
- Each X_i is a public commitment to the corresponding secret value x_i

SPS-EQ: Signing

Sign$(\vec{M}, sk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell) \in (\mathbb{G}_1^*)^\ell$ and secret key $sk = (x_1, \ldots, x_\ell)$

SPS-EQ: Signing

Sign($\vec{M}, sk$):

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell) \in (\mathbb{G}_1^*)^\ell$ and secret key $sk = (x_1, \ldots, x_\ell)$
- Choose random $y \in \mathbb{Z}_p^*$

SPS-EQ: Signing

Sign$(\vec{M}, sk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell) \in (\mathbb{G}_1^*)^\ell$ and secret key $sk = (x_1, \ldots, x_\ell)$
- Choose random $y \in \mathbb{Z}_p^*$
- Compute $Z = y \cdot \langle \vec{M}, \vec{x} \rangle$, where $\langle \vec{M}, \vec{x} \rangle = \sum_{i=1}^{\ell} x_i \cdot M_i$

SPS-EQ: Signing

Sign$(\vec{M}, sk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell) \in (\mathbb{G}_1^*)^\ell$ and secret key $sk = (x_1, \ldots, x_\ell)$
- Choose random $y \in \mathbb{Z}_p^*$
- Compute $Z = y \cdot \langle \vec{M}, \vec{x} \rangle$, where $\langle \vec{M}, \vec{x} \rangle = \sum_{i=1}^{\ell} x_i \cdot M_i$
- Compute $Y = y^{-1} \cdot P$ and $\hat{Y} = y^{-1} \cdot \hat{P}$

SPS-EQ: Signing

Sign$(\vec{M}, sk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell) \in (\mathbb{G}_1^*)^\ell$ and secret key $sk = (x_1, \ldots, x_\ell)$
- Choose random $y \in \mathbb{Z}_p^*$
- Compute $Z = y \cdot \langle \vec{M}, \vec{x} \rangle$, where $\langle \vec{M}, \vec{x} \rangle = \sum_{i=1}^{\ell} x_i \cdot M_i$
- Compute $Y = y^{-1} \cdot P$ and $\hat{Y} = y^{-1} \cdot \hat{P}$
- Return signature $\sigma = (Z, Y, \hat{Y})$

SPS-EQ: Signing

Sign$(\vec{M}, sk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell) \in (\mathbb{G}_1^*)^\ell$ and secret key $sk = (x_1, \ldots, x_\ell)$
- Choose random $y \in \mathbb{Z}_p^*$
- Compute $Z = y \cdot \langle \vec{M}, \vec{x} \rangle$, where $\langle \vec{M}, \vec{x} \rangle = \sum_{i=1}^{\ell} x_i \cdot M_i$
- Compute $Y = y^{-1} \cdot P$ and $\hat{Y} = y^{-1} \cdot \hat{P}$
- Return signature $\sigma = (Z, Y, \hat{Y})$

Intuition:

- The inner product $\langle \vec{M}, \vec{x} \rangle$ binds the message to the secret key
- Randomization factor y prevents signature forgery
- Z captures the inner product, while Y and $\hat{Y}$ enable verification
- When message is scaled by μ, the inner product scales linearly
- The structure ensures signatures can be transformed for equivalent messages

SPS-EQ: Verification

Verify$(\vec{M}, \sigma, pk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell)$, signature $\sigma = (Z, Y, \hat{Y})$, public key $pk = (X_1, \ldots, X_\ell)$

SPS-EQ: Verification

Verify$(\vec{M}, \sigma, pk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell)$, signature $\sigma = (Z, Y, \hat{Y})$, public key $pk = (X_1, \ldots, X_\ell)$
- If for some $i \in [l]$, $M_i \notin \mathbb{G}_1^*$, or $X_i \notin \mathbb{G}_2^*$, or $Z \notin \mathbb{G}_1$ or $Y \notin \mathbb{G}_1^*$, or $\hat{Y} \notin \mathbb{G}_2^*$, output 0 and halt

SPS-EQ: Verification

Verify$(\vec{M}, \sigma, pk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell)$, signature $\sigma = (Z, Y, \hat{Y})$, public key $pk = (X_1, \ldots, X_\ell)$
- If for some $i \in [l]$, $M_i \notin \mathbb{G}_1^*$, or $X_i \notin \mathbb{G}_2^*$, or $Z \notin \mathbb{G}_1$ or $Y \notin \mathbb{G}_1^*$, or $\hat{Y} \notin \mathbb{G}_2^*$, output 0 and halt
- Check 1: $\prod_{i=1}^{\ell} e(M_i, X_i) = e(Z, \hat{Y})$

SPS-EQ: Verification

Verify$(\vec{M}, \sigma, pk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell)$, signature $\sigma = (Z, Y, \hat{Y})$, public key $pk = (X_1, \ldots, X_\ell)$
- If for some $i \in [l]$, $M_i \notin \mathbb{G}_1^*$, or $X_i \notin \mathbb{G}_2^*$, or $Z \notin \mathbb{G}_1$ or $Y \notin \mathbb{G}_1^*$, or $\hat{Y} \notin \mathbb{G}_2^*$, output 0 and halt
- Check 1: $\prod_{i=1}^{\ell} e(M_i, X_i) = e(Z, \hat{Y})$
- Check 2: $e(Y, \hat{P}) = e(P, \hat{Y})$

SPS-EQ: Verification

Verify$(\vec{M}, \sigma, pk)$:

- Input: Message $\vec{M} = (M_1, \ldots, M_\ell)$, signature $\sigma = (Z, Y, \hat{Y})$, public key $pk = (X_1, \ldots, X_\ell)$
- If for some $i \in [l]$, $M_i \notin \mathbb{G}_1^*$, or $X_i \notin \mathbb{G}_2^*$, or $Z \notin \mathbb{G}_1$ or $Y \notin \mathbb{G}_1^*$, or $\hat{Y} \notin \mathbb{G}_2^*$, output 0 and halt
- Check 1: $\prod_{i=1}^{\ell} e(M_i, X_i) = e(Z, \hat{Y})$
- Check 2: $e(Y, \hat{P}) = e(P, \hat{Y})$
- Return 1 if both checks pass, 0 otherwise

Intuition:

- Check 1 verifies that Z correctly encodes the message-secret key relationship
- The left side $\prod_{i=1}^{\ell} e(M_i, X_i) = \prod_{i=1}^{\ell} e(M_i, x_i\hat{P}) = e(\sum x_i M_i, \hat{P}) = e(\langle \vec{M}, \vec{x} \rangle, \hat{P})$
- The right side $e(Z, \hat{Y}) = e(y \cdot \langle \vec{M}, \vec{x} \rangle, y^{-1}\hat{P}) = e(\langle \vec{M}, \vec{x} \rangle, \hat{P})$
- Check 2 confirms Y and $\hat{Y}$ are properly formed with the same y value

SPS-EQ: Change of Representative

ChgRep$(\vec{M}, \sigma, \mu, pk)$:

- Input: Representative $\vec{M}$ of equivalence class $[\vec{M}]_{\mathcal{R}}$, signature $\sigma = (Z, Y, \hat{Y})$, scalar $\mu \in \mathbb{Z}_p^*$, public key pk

SPS-EQ: Change of Representative

ChgRep$(\vec{M}, \sigma, \mu, pk)$:

- Input: Representative $\vec{M}$ of equivalence class $[\vec{M}]_{\mathcal{R}}$, signature $\sigma = (Z, Y, \hat{Y})$, scalar $\mu \in \mathbb{Z}_p^*$, public key pk
- Return $\perp$ if Verify$(\vec{M}, \sigma, pk) = 0$

SPS-EQ: Change of Representative

ChgRep$(\vec{M}, \sigma, \mu, pk)$:

- Input: Representative $\vec{M}$ of equivalence class $[\vec{M}]_{\mathcal{R}}$, signature $\sigma = (Z, Y, \hat{Y})$, scalar $\mu \in \mathbb{Z}_p^*$, public key pk
- Return $\perp$ if Verify$(\vec{M}, \sigma, pk) = 0$
- Sample $\psi \xleftarrow{\$} \mathbb{Z}_p^*$

SPS-EQ: Change of Representative

ChgRep$(\vec{M}, \sigma, \mu, pk)$:

- Input: Representative $\vec{M}$ of equivalence class $[\vec{M}]_{\mathcal{R}}$, signature $\sigma = (Z, Y, \hat{Y})$, scalar $\mu \in \mathbb{Z}_p^*$, public key pk
- Return $\perp$ if Verify$(\vec{M}, \sigma, pk) = 0$
- Sample $\psi \xleftarrow{\$} \mathbb{Z}_p^*$
- Return new signature $\sigma' = (\psi \cdot \mu \cdot Z, \psi^{-1} \cdot Y, \psi^{-1} \cdot \hat{Y})$. This is a valid signature on $\mu \cdot \vec{M}$

SPS-EQ: Change of Representative

ChgRep$(\vec{M}, \sigma, \mu, pk)$:

- Input: Representative $\vec{M}$ of equivalence class $[\vec{M}]_{\mathcal{R}}$, signature $\sigma = (Z, Y, \hat{Y})$, scalar $\mu \in \mathbb{Z}_p^*$, public key pk
- Return $\perp$ if Verify$(\vec{M}, \sigma, pk) = 0$
- Sample $\psi \xleftarrow{\$} \mathbb{Z}_p^*$
- Return new signature $\sigma' = (\psi \cdot \mu \cdot Z, \psi^{-1} \cdot Y, \psi^{-1} \cdot \hat{Y})$. This is a valid signature on $\mu \cdot \vec{M}$

Intuition:

- When we scale $\vec{M}$ by μ, the inner product $\langle \mu \cdot \vec{M}, \vec{x} \rangle = \mu \cdot \langle \vec{M}, \vec{x} \rangle$ also scales by μ
- For a valid signature on $\mu \cdot \vec{M}$, we need to adjust Z proportionally by μ
- The random ψ provides re-randomization, making the new signature indistinguishable from a fresh one
- Y' and $\hat{Y}'$ are adjusted by ψ^{-1} to maintain the verification equation consistency

SPS-EQ: Key Validation

VKey(sk, pk):

- Input: Secret key $sk = (x_1, \ldots, x_\ell) \in (\mathbb{Z}_p^*)^\ell$, public key $pk = (X_1, \ldots, X_\ell) \in \mathbb{G}_2^\ell$

SPS-EQ: Key Validation

VKey(sk, pk):

- Input: Secret key $sk = (x_1, \ldots, x_\ell) \in (\mathbb{Z}_p^*)^\ell$, public key $pk = (X_1, \ldots, X_\ell) \in \mathbb{G}_2^\ell$
- Output 1 if for all $i \in [1, \ell]$, $x_i \in \mathbb{Z}_p^*$ and $X_i \in \mathbb{G}_2^*$ and $X_i = x_i \hat{P}$

SPS-EQ: Key Validation

VKey(sk, pk):

- Input: Secret key $sk = (x_1, \ldots, x_\ell) \in (\mathbb{Z}_p^*)^\ell$, public key $pk = (X_1, \ldots, X_\ell) \in \mathbb{G}_2^\ell$
- Output 1 if for all $i \in [1, \ell]$, $x_i \in \mathbb{Z}_p^*$ and $X_i \in \mathbb{G}_2^*$ and $X_i = x_i \hat{P}$
- Output 0 otherwise

Security Guarantees

- **Theorem 1**: The SPS-EQ scheme is correct (satisfies Definition 1)

.

Security Guarantees

- **Theorem 1**: The SPS-EQ scheme is correct (satisfies Definition 1)
- **Theorem 2**: This construction is EUF-CMA (Definition 2) secure in the Generic Group Model for Type-3 pairings

.

Security Guarantees

- **Theorem 1**: The SPS-EQ scheme is correct (satisfies Definition 1)
- **Theorem 2**: This construction is EUF-CMA (Definition 2) secure in the Generic Group Model for Type-3 pairings
  - In ASIACRYPT 2024, Bauer, Fuchsbauer, and Regen proved that the SPS-EQ scheme is EUF-CMA secure in the Algebraic Group Model for Type-3 pairings [BFR'24].

.

Security Guarantees

- **Theorem 1**: The SPS-EQ scheme is correct (satisfies Definition 1)
- **Theorem 2**: This construction is EUF-CMA (Definition 2) secure in the Generic Group Model for Type-3 pairings
  - In ASIACRYPT 2024, Bauer, Fuchsbauer, and Regen proved that the SPS-EQ scheme is EUF-CMA secure in the Algebraic Group Model for Type-3 pairings [BFR'24].
- **Lemma 1**: This construction has perfect adaptation of signatures and perfect adaptation of signatures under malicious keys (Definition 5).

Incorporating SPS-EQ in our scheme

- Each user requires a unique equivalence class for signature transformation
- World population: ≈ 8 billion equivalence classes needed

Counting equivalence classes:

Incorporating SPS-EQ in our scheme

- Each user requires a unique equivalence class for signature transformation
- World population: ≈ 8 billion equivalence classes needed

Counting equivalence classes:

- $\vec{M} = (M_1, \ldots, M_\ell) \sim_{\mathcal{R}} (P, M_2/x_1, \ldots, M_\ell/x_1)$ where x_1 satisfies $M_1 = x_1 \cdot P$

Incorporating SPS-EQ in our scheme

- Each user requires a unique equivalence class for signature transformation
- World population: ≈ 8 billion equivalence classes needed

Counting equivalence classes:

- $\vec{M} = (M_1, \ldots, M_\ell) \sim_{\mathcal{R}} (P, M_2/x_1, \ldots, M_\ell/x_1)$ where x_1 satisfies $M_1 = x_1 \cdot P$
- Each M_i can have $p - 1$ possible values

Incorporating SPS-EQ in our scheme

- Each user requires a unique equivalence class for signature transformation
- World population: ≈ 8 billion equivalence classes needed

Counting equivalence classes:

- $\vec{M} = (M_1, \ldots, M_\ell) \sim_{\mathcal{R}} (P, M_2/x_1, \ldots, M_\ell/x_1)$ where x_1 satisfies $M_1 = x_1 \cdot P$
- Each M_i can have $p - 1$ possible values
- Number of equivalence classes: $(p-1)^{\ell-1}$

Incorporating SPS-EQ in our scheme

- Each user requires a unique equivalence class for signature transformation
- World population: ≈ 8 billion equivalence classes needed

Counting equivalence classes:

- $\vec{M} = (M_1, \ldots, M_\ell) \sim_{\mathcal{R}} (P, M_2/x_1, \ldots, M_\ell/x_1)$ where x_1 satisfies $M_1 = x_1 \cdot P$
- Each M_i can have $p - 1$ possible values
- Number of equivalence classes: $(p-1)^{\ell-1}$
- For security parameter $\kappa = 128$, $p \geq 2^\kappa$

Incorporating SPS-EQ in our scheme

- Each user requires a unique equivalence class for signature transformation
- World population: ≈ 8 billion equivalence classes needed

Counting equivalence classes:

- $\vec{M} = (M_1, \ldots, M_\ell) \sim_{\mathcal{R}} (P, M_2/x_1, \ldots, M_\ell/x_1)$ where x_1 satisfies $M_1 = x_1 \cdot P$
- Each M_i can have $p - 1$ possible values
- Number of equivalence classes: $(p-1)^{\ell-1}$
- For security parameter $\kappa = 128$, $p \geq 2^\kappa$
- With $\ell = 2$, we have 2^{128} equivalence classes

Incorporating SPS-EQ in our scheme

- Each user requires a unique equivalence class for signature transformation
- World population: ≈ 8 billion equivalence classes needed

Counting equivalence classes:

- $\vec{M} = (M_1, \ldots, M_\ell) \sim_{\mathcal{R}} (P, M_2/x_1, \ldots, M_\ell/x_1)$ where x_1 satisfies $M_1 = x_1 \cdot P$
- Each M_i can have $p - 1$ possible values
- Number of equivalence classes: $(p-1)^{\ell-1}$
- For security parameter $\kappa = 128$, $p \geq 2^\kappa$
- With $\ell = 2$, we have 2^{128} equivalence classes
- Scheme dictates $\ell > 1$

Incorporating SPS-EQ in our scheme

- Each user requires a unique equivalence class for signature transformation
- World population: ≈ 8 billion equivalence classes needed

Counting equivalence classes:

- $\vec{M} = (M_1, \ldots, M_\ell) \sim_{\mathcal{R}} (P, M_2/x_1, \ldots, M_\ell/x_1)$ where x_1 satisfies $M_1 = x_1 \cdot P$
- Each M_i can have $p - 1$ possible values
- Number of equivalence classes: $(p-1)^{\ell-1}$
- For security parameter $\kappa = 128$, $p \geq 2^\kappa$
- With $\ell = 2$, we have 2^{128} equivalence classes
- Scheme dictates $\ell > 1$
- $\Rightarrow$ Vector length is minimum for our application

Incorporating SPS-EQ in our scheme (Cont.)

Method 1: Without using user's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Service provider maintains a mapping of users to their equivalence classes

Incorporating SPS-EQ in our scheme (Cont.)

Method 1: Without using user's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Service provider maintains a mapping of users to their equivalence classes
- Generates $\vec{M} = [P, aP] \in (\mathbb{G}_1^*)^2$ such that no existing user is mapped to aP

Incorporating SPS-EQ in our scheme (Cont.)

Method 1: Without using user's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Service provider maintains a mapping of users to their equivalence classes
- Generates $\vec{M} = [P, aP] \in (\mathbb{G}_1^*)^2$ such that no existing user is mapped to aP
- It signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user

Incorporating SPS-EQ in our scheme (Cont.)

Method 1: Without using user's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Service provider maintains a mapping of users to their equivalence classes
- Generates $\vec{M} = [P, aP] \in (\mathbb{G}_1^*)^2$ such that no existing user is mapped to aP
- It signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- When user i wants to sign message m, it generates a random $\mu \xleftarrow{\$} \mathbb{Z}_p^*$

Incorporating SPS-EQ in our scheme (Cont.)

Method 1: Without using user's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Service provider maintains a mapping of users to their equivalence classes
- Generates $\vec{M} = [P, aP] \in (\mathbb{G}_1^*)^2$ such that no existing user is mapped to aP
- It signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- When user i wants to sign message m, it generates a random $\mu \xleftarrow{\$} \mathbb{Z}_p^*$
- It computes $\sigma' = \mathsf{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk_{\mathsf{SP}})$ and $M' = \mu \cdot \vec{M}$ and sends (σ', M') to another user.

Incorporating SPS-EQ in our scheme (Cont.)

Method 1: Without using user's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Service provider maintains a mapping of users to their equivalence classes
- Generates $\vec{M} = [P, aP] \in (\mathbb{G}_1^*)^2$ such that no existing user is mapped to aP
- It signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- When user i wants to sign message m, it generates a random $\mu \xleftarrow{\$} \mathbb{Z}_p^*$
- It computes $\sigma' = \mathsf{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk_{\mathsf{SP}})$ and $M' = \mu \cdot \vec{M}$ and sends (σ', M') to another user.
- User verifies (σ', M') with pk_{SP}.

Incorporating SPS-EQ in our scheme (Cont.)

Method 1: Without using user's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Service provider maintains a mapping of users to their equivalence classes
- Generates $\vec{M} = [P, aP] \in (\mathbb{G}_1^*)^2$ such that no existing user is mapped to aP
- It signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- When user i wants to sign message m, it generates a random $\mu \xleftarrow{\$} \mathbb{Z}_p^*$
- It computes $\sigma' = \mathsf{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk_{\mathsf{SP}})$ and $M' = \mu \cdot \vec{M}$ and sends (σ', M') to another user.
- User verifies (σ', M') with pk_{SP}.
- SP to revoke user i publishes $(\vec{x'}, r_{\vec{M'}})$ of M' (Definition 0.1) where $\vec{x}$ are discrete logs of $\vec{M}$. To check if a user is banned, another user simply checks if there exists a set of published $(\vec{x'}, r_{\vec{M'}})$ such that for their received M, $\exists \lambda \in \mathbb{Z}_p^*$ s.t. $r_{M'} = \lambda \cdot r_{(M,M')}$.

Incorporating SPS-EQ in our scheme (Cont.)

Method 1: Without using user's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Service provider maintains a mapping of users to their equivalence classes
- Generates $\vec{M} = [P, aP] \in (\mathbb{G}_1^*)^2$ such that no existing user is mapped to aP
- It signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- When user i wants to sign message m, it generates a random $\mu \xleftarrow{\$} \mathbb{Z}_p^*$
- It computes $\sigma' = \mathsf{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk_{\mathsf{SP}})$ and $M' = \mu \cdot \vec{M}$ and sends (σ', M') to another user.
- User verifies (σ', M') with pk_{SP}.
- SP to revoke user i publishes $(\vec{x'}, r_{\vec{M'}})$ of M' (Definition 0.1) where $\vec{x}$ are discrete logs of $\vec{M}$. To check if a user is banned, another user simply checks if there exists a set of published $(\vec{x'}, r_{\vec{M'}})$ such that for their received M, $\exists \lambda \in \mathbb{Z}_p^*$ s.t. $r_{M'} = \lambda \cdot r_{(M,M')}$.

Problem: Signature is not linked to public key

Incorporating SPS-EQ in our scheme (Cont.)

Method 2: Using user U's public key. Server has $(sk_{\mathrm{SP}}, pk_{\mathrm{SP}})$.

- Let $g_{pk_{\mathrm{U}}}$ represent the mapping of U's public key pk_{U} to $\mathbb{G}_1^*$. User sends $(pk_{\mathrm{U}}, g_{pk_{\mathrm{U}}})$ to SP

Incorporating SPS-EQ in our scheme (Cont.)

Method 2: Using user U's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Let $g_{pk_{\mathsf{U}}}$ represent the mapping of U's public key pk_{U} to $\mathbb{G}_1^*$. User sends $(pk_{\mathsf{U}}, g_{pk_{\mathsf{U}}})$ to SP
- SP generates $\vec{M} = [aP, g_{pk_{\mathsf{U}}}]$ where a is chosen appropriately in $\mathbb{Z}_p^*$ to get an unused equivalence class $[aP, g_{pk_{\mathsf{U}}}] \in (\mathbb{G}_1^*)^2$. SP signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user

Incorporating SPS-EQ in our scheme (Cont.)

Method 2: Using user U's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Let $g_{pk_{\mathsf{U}}}$ represent the mapping of U's public key pk_{U} to $\mathbb{G}_1^*$. User sends $(pk_{\mathsf{U}}, g_{pk_{\mathsf{U}}})$ to SP
- SP generates $\vec{M} = [aP, g_{pk_{\mathsf{U}}}]$ where a is chosen appropriately in $\mathbb{Z}_p^*$ to get an unused equivalence class $[aP, g_{pk_{\mathsf{U}}}] \in (\mathbb{G}_1^*)^2$. SP signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- User (U) generates new public key pk' and $g_{pk'}$. It sets $\mu = x_{pk'}/x_{pk}$ where $x_{pk}, x_{pk'}$ are the discrete logs of $g_{pk}, g_{pk'}$ in $\mathbb{G}_1^*$

Incorporating SPS-EQ in our scheme (Cont.)

Method 2: Using user U's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Let $g_{pk_{\mathsf{U}}}$ represent the mapping of U's public key pk_{U} to $\mathbb{G}_1^*$. User sends $(pk_{\mathsf{U}}, g_{pk_{\mathsf{U}}})$ to SP
- SP generates $\vec{M} = [aP, g_{pk_{\mathsf{U}}}]$ where a is chosen appropriately in $\mathbb{Z}_p^*$ to get an unused equivalence class $[aP, g_{pk_{\mathsf{U}}}] \in (\mathbb{G}_1^*)^2$. SP signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- User (U) generates new public key pk' and $g_{pk'}$. It sets $\mu = x_{pk'}/x_{pk}$ where $x_{pk}, x_{pk'}$ are the discrete logs of $g_{pk}, g_{pk'}$ in $\mathbb{G}_1^*$
- User computes $\sigma' = \mathrm{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk_{\mathsf{U}})$ which is a signature on

$$\begin{aligned} M' &= \mu \cdot \vec{M} = [(\mu a)P, \mu \cdot g_{pk}] \\ &= [(\mu a)P, ({}^{x_{pk'}}/{}_{x_{pk}}) \cdot x_{pk} \cdot P] = [(\mu a)P, x_{pk'} \cdot P] = [(\mu a)P, g_{pk'}] \end{aligned}$$

  and sends (σ', M') to another user.

Incorporating SPS-EQ in our scheme (Cont.)

Method 2: Using user U's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Let $g_{pk_{\mathsf{U}}}$ represent the mapping of U's public key pk_{U} to $\mathbb{G}_1^*$. User sends $(pk_{\mathsf{U}}, g_{pk_{\mathsf{U}}})$ to SP
- SP generates $\vec{M} = [aP, g_{pk_{\mathsf{U}}}]$ where a is chosen appropriately in $\mathbb{Z}_p^*$ to get an unused equivalence class $[aP, g_{pk_{\mathsf{U}}}] \in (\mathbb{G}_1^*)^2$. SP signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- User (U) generates new public key pk' and $g_{pk'}$. It sets $\mu = x_{pk'}/x_{pk}$ where $x_{pk}, x_{pk'}$ are the discrete logs of $g_{pk}, g_{pk'}$ in $\mathbb{G}_1^*$
- User computes $\sigma' = \mathsf{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk_{\mathsf{U}})$ which is a signature on
$$\begin{aligned} M' &= \mu \cdot \vec{M} = [(\mu a)P, \mu \cdot g_{pk}] \\ &= [(\mu a)P, ({}^{x_{pk'}}/{}_{x_{pk}}) \cdot x_{pk} \cdot P] = [(\mu a)P, x_{pk'} \cdot P] = [(\mu a)P, g_{pk'}] \end{aligned}$$
and sends (σ', M') to another user.
- User verifies (σ', M') with pk_{SP} and can further check if $g_{pk'}$ is a valid mapping of pk' to $\mathbb{G}_1^*$

Incorporating SPS-EQ in our scheme (Cont.)

Method 2: Using user U's public key. Server has $(sk_{\mathsf{SP}}, pk_{\mathsf{SP}})$.

- Let $g_{pk_{\mathsf{U}}}$ represent the mapping of U's public key pk_{U} to $\mathbb{G}_1^*$. User sends $(pk_{\mathsf{U}}, g_{pk_{\mathsf{U}}})$ to SP
- SP generates $\vec{M} = [aP, g_{pk_{\mathsf{U}}}]$ where a is chosen appropriately in $\mathbb{Z}_p^*$ to get an unused equivalence class $[aP, g_{pk_{\mathsf{U}}}] \in (\mathbb{G}_1^*)^2$. SP signs $\vec{M}$ with sk_{SP} and sends $(\sigma, \vec{M})$ to the user
- User (U) generates new public key pk' and $g_{pk'}$. It sets $\mu = x_{pk'}/x_{pk}$ where $x_{pk}, x_{pk'}$ are the discrete logs of $g_{pk}, g_{pk'}$ in $\mathbb{G}_1^*$
- User computes $\sigma' = \mathsf{ChgRep}_{\mathcal{R}}(\vec{M}, \sigma, \mu, pk_{\mathsf{U}})$ which is a signature on
$$\begin{aligned} M' &= \mu \cdot \vec{M} = [(\mu a)P, \mu \cdot g_{pk}] \\ &= [(\mu a)P, ({}^{x_{pk'}}/_{x_{pk}}) \cdot x_{pk} \cdot P] = [(\mu a)P, x_{pk'} \cdot P] = [(\mu a)P, g_{pk'}] \end{aligned}$$
and sends (σ', M') to another user.
- User verifies (σ', M') with pk_{SP} and can further check if $g_{pk'}$ is a valid mapping of pk' to $\mathbb{G}_1^*$
- SP can revoke user i same as before.

Incorporating SPS-EQ in our scheme (Cont.)

Problems and Challenges:

- User needs knowledge of discrete logarithm of g_{pk_1} in $\mathbb{G}_1$
- Requires a secure mapping function from public keys to group elements
- Revocation checking requires server to have discrete logs of banned users
- Revocation checking requires comparing against all banned mutual ratios
- Another user can transform receive (M, σ) to another (M', σ') in the same equivalence class

Implementation in Python

- Implemented the SPS-EQ scheme using py_ecc library for elliptic curve operations
- Use the bn128 ($y^2 = x^3 + 3$) elliptic curve for the scheme where $\mathbb{G}_1$ and $\mathbb{G}_2$ are of prime order p
- $\mathbb{G}_1$ is over base field $\mathbb{F}_p$ and $\mathbb{G}_2$ is over extension field $\mathbb{F}_{p^2}$. $\mathbb{G}_T$ is a subgroup of $\mathbb{F}_{p^{12}}$
- Operations in $\mathbb{G}_1$ are faster, hence why messages are in $\mathbb{G}_1$
- Implementation below is a watered down version for presentation's sake.

Key Functions in Python (1/2)

```
# multiply, add are elliptic curve operations
# curve_order is the order of the elliptic curve. For bn128, its prime and same for G1, G2.
# pairing is the elliptic curve pairing, e(., .).
# For some reason, py_ecc defines pairings as e(Q,P) where Q is in G2 and P is in G1.
from py_ecc.bn128 import G1 as P, G2 as P_hat, multiply, add, curve_order as p, pairing as e
import random

def inner_product(vec_a, vec_b):
    zero = P.zero()
    for i in range(len(vec_a)):
        zero = add(zero, multiply(vec_a[i], vec_b[i]))
    return zero

def keygen(l):
    sk = [random.randint(1, p - 1) for _ in range(l)]
    pk = [multiply(P_hat, sk[i]) for i in range(l)]
    return sk, pk

def sign(M, sk):
    y = random.randint(1, p - 1)  # Random element in Z_p*
    Z = multiply(inner_product(M, sk), y)
    Y = multiply(P, pow(y, -1, p))
    Y_hat = multiply(P_hat, pow(y, -1, p))

    return (Z, Y, Y_hat)
```

Key Functions in Python (2/2)

```
# FQ12 is the target group G_T for bn128
from py_ecc.bn128 import FQ12

def verify(M, sig, pk):
    Z, Y, Y_hat = sig

    # Check 1
    e_1 = FQ12.one()
    for i in range(len(M)):
        e_1 = e_1 * pairing(M[i], pk[i])
    e_2 = pairing(Z, Y_hat)

    if e_1 != e_2:
        return 0

    # Check 2
    e_3 = pairing(Y, P_hat)
    e_4 = pairing(P, Y_hat)

    return 1 if e_3 == e_4 else 0

def chgRep(M, sigma, mu, pk):

    Z, Y, Y_hat = sigma
    psi = random.randint(1, curve_order - 1)  # Randomization factor
    scalar = (psi * mu) % curve_order

    return (
        multiply(Z, scalar),
        multiply(Y, pow(psi, -1, curve_order)),
        multiply(Y_hat, pow(psi, -1, curve_order))
    )
```

References

G. Fuchsbauer, C. Hanser, and D. Slamanig
Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials
Journal of Cryptology, 2019
DOI: 10.1007/s00145-018-9281-4

py_ecc: Elliptic curve crypto in Python
https://github.com/ethereum/py_ecc