The Landscape of Pointcheval-Sanders Signatures: Mapping to Polynomial-Based Signatures and Beyond

Kristian L. McDonald[1]

Clearmatics
London, UK

Abstract

Pointcheval-Sanders (PS) signatures [PS16] are well-studied in the literature and have found use within e.g. threshold credential schemes [SAB+19] and redactable anonymous credential schemes [San19]. The present work leverages a mapping between PS signatures and a related class of polynomial-based signatures to construct multiple new signature/credential schemes. Specifically, new protocols for multi-message signatures, sequential aggregate signatures, signatures for message commitments, redactable signatures, and unlinkable redactable signatures are presented. A redactable anonymous credential scheme is also constructed. All original protocols employ constant-sized secret keys rather than linear-sized (in the number of messages/attributes). Security properties of the new protocols are analysed and a general discussion of security properties for both PS signatures and the new schemes is provided.

[1]klmcd protonmail com

Contents

1 Introduction

Digital signatures [DH76] are a widely deployed cryptographic primitive and many variant schemes exist in the literature. The birth of pairing-based cryptography [Jou00, BF01], in particular, has seen the emergence of many new protocols (e.g. [BLS01, BBS04, CL04, Gro15, Gha16, LMPY16]). Among these new constructs, multi-message Camenisch-Lysyanskaya (CL) signatures are well-studied due to their utility within larger protocols [CL04]. For example, CL signatures have been leveraged within aggregate signature schemes [LLY13], group signature schemes [BCN$^+$10], direct anonymous attestation schemes [CPS10, BFG$^+$13, DLST14], and an e-cash protocol [CPST15].

Drawing inspiration from CL signatures, Pointcheval and Sanders (PS) constructed a related pairing-based multi-message digital signature scheme that achieved similar functionalities but availed some efficiencies [PS16]. Pointcheval and Sanders' insight was that many protocols employing CL signatures required type-3 pairings, yet CL signatures were designed for type-1 pairings [PS16].[1] This mismatch permitted efficiency gains by leveraging the properties of type-3 pairings, which do not admit an efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ [GPS08]. Hence, PS generated efficiencies by separating the signature space ($\mathbb{G}_1$) from the public key space ($\mathbb{G}_2$), since elements of the latter could not be used to forge elements of the former. Multi-message PS signatures require two $\mathbb{G}_1$ elements for q messages (i.e., independent of q), compared to $1 + 2q$ $\mathbb{G}_1$ elements for CL signatures [CL04]. Thus, PS signatures are a variant of CL signatures with purpose-built design features that leverage the properties of type-3 pairings. Accordingly, PS signatures can serve as plug-in replacements for CL signatures in applications employing type-3 pairings, since they offer similar functionalities but achieve some efficiencies [PS16]. One drawback of both CL and PS signatures, however, is that security is proven in the generic group model [Nec94, Sho97], which is a stronger assumption than the standard model [Fis00, KM07]. Nonetheless, consistent with other works [BB08, AGHO11], this approach permits more efficient constructs.

More generally, signature schemes come in many distinct flavours, each offering users and verifiers different functionalities and guarantees. For example, multi-message digital signatures allow a single signer to generate a single signature for a set of messages [CL04, PS16]. This differs from sequentially aggregated signatures, which permit a set of signers to sequentially add signatures over independent messages, generating a single signature for multiple messages under multiple signers [LMRS04]. Similar to blind signatures [Cha83, PS00], signatures for message commitments avail a degree of privacy by allowing signature-users to obtain a signature over a commitment to the underlying messages/attributes [ASM06, BFPV11]. Conversely, redactable signatures provide users with optionality to selectively disclose *only a subset* of the messages when revealing a signature/credential to a verifier [HHH$^+$08, NTKK09, BBD$^+$10]. Consequently redactable signatures provide additional privacy guarantees against malicious verifiers [HHH$^+$08]. Taking this notion further, unlinkable redactable signatures allow users to obtain multiple distinct signatures for the same set of (selectively disclosed) messages. This optionality inhibits the ability of malicious entities

[1]Recall that a bilinear pairing is a mapping involving three groups, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $G_1 = G_2$ ($\mathbb{G}_1 \neq \mathbb{G}_2$) for a type-1 (type-3) pairing [GPS08].

to gather additional information across multiple reveals of a signature [CL01], and is well-suited for use in redactable anonymous credential schemes [CL01, CDHK15, FHS19] (for early work on anonymous credentials, see Chaum's paper [Cha85]). Pointcheval-Sanders signatures can be generalised to realise variants of each of the above categories of signature schemes [PS16, San19]. Given this utility, PS signatures have found use in, e.g., a threshold credential issuance protocol [SAB$^+$19] and an anonymous credential scheme [San19].

The present work investigates a class of digital signature protocols that offer similar functionalities to PS and CL signatures. More precisely, a mapping between PS signatures and a class of related polynomial-based signatures is described. Leveraging this mapping, multiple new signature schemes are identified and constructed. New protocols for multi-message signatures [CL04, PS16], sequential aggregate signatures [LMRS04], signatures for message commitments [ASM06], redactable signatures [BBD$^+$10, HHH$^+$08, NTKK09], and unlinkable redactable signatures [CL01] are presented. A new redactable anonymous credential scheme [CL01, CDHK15, FHS19] is also constructed. The new signature schemes achieve the same functionality as their PS-based counterparts but employ shorter secret keys. Security properties of both the new protocols and PS signatures are discussed in detail. More explicitly, the present paper contains the following contributions:

- New security proofs and discussion of security properties for multi-message PS signatures, including generalised PS security assumptions.
- A new polynomial-based multi-message digital signature protocol.
- Two new multi-message sequential aggregate signature schemes. The first scheme generalises the sequential aggregate signature scheme introduced by PS [PS16]; the second leverages the new polynomial-based signatures.
- New polynomial-based protocols for (*i*) signatures over message commitments, (*ii*) redactable signatures and (*iii*) unlinkable redactable signatures.
- A polynomial-based redactable anonymous credential scheme.
- Security assumptions and detailed security proofs for all new constructs.
- Discussion of relationships among the various signature schemes.

Signature protocols introduced in this work were also leveraged in recently-proposed *switched threshold signature schemes*, which also used a secret sharing protocol with polynomially-related shares [McD20].

The layout of this paper is as follows. Section 2 introduces relevant preliminary concepts and terminology. Single-message PS signatures are described in Section 3 - the discussion here is elementary but provides context for subsequent sections. Sections 4 through 8 each introduce original signature schemes and provide relevant security proofs. A new anonymous credential scheme with selective disclosure is presented in Section 9. The paper concludes in Section 10. Each section aims to be largely self-contained so readers may freely jump to particular sections of interest. The main exceptions being that (*i*) some security proofs use reductions involving earlier-described constructs; (*ii*) the presentation of unlinkable redactable signatures (Section 8) draws on definitions relating to redactable signatures (Section 7); and (*iii*) discussion of relationships among signature schemes necessarily draws on content from multiple sections.

2 Preliminaries

The signature schemes described in this work use bilinear groups with a type-3 pairing function. Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ denote cyclic groups of prime order p, which are related by an efficiently computable bilinear map (i.e. pairing) $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, with the following properties:

- For generators $g_{1,2} \in \mathbb{G}_{1,2}$, the mapping e gives $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$, and $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.
- For all $g_{1,2} \in \mathbb{G}_{1,2}$, and $a, b \in \mathbb{F}_p$, one has $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.

For type-3 pairings, one has $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ exists [GPS08]. In general, type-3 pairings are compatible with the "XDH assumption" [Sco02, BBS04, GR04], meaning the Decisional Diffie-Hellman (DDH) assumption may hold in $\mathbb{G}_1$ or $\mathbb{G}_2$. The DDH assumption posits that, given $(g, g^\alpha, g^\beta, g^\gamma) \in \mathbb{G}^4$, for $\alpha, \beta, \gamma \in \mathbb{F}_p$ and some group $\mathbb{G}$, no adversary can efficiently distinguish between $\gamma = \alpha \cdot \beta$ and $\gamma \xleftarrow{R} \mathbb{F}_p$. The discrete log ($\mathsf{DL}$) assumption posits that, given $(g, g^\beta) \in \mathbb{G}^2$, no adversary can efficiently output β. The notation $a \in [n]$ is used below to denote $a \in \{1, 2, \ldots, n\}$. Also, $\mathsf{poly}(\lambda)$ refers to a polynomial function (here of the security parameter λ).

A digital signature scheme is described by the following four algorithms:

- $\mathsf{DS.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters pp.
- $\mathsf{DS.Keygen}(\mathsf{pp})$: Takes the public parameters pp as input and outputs a key pair $(\mathsf{sk}, \mathsf{pk})$, comprised of a secret key sk and a public key pk.
- $\mathsf{DS.Sign}(\mathsf{pp}, \mathsf{sk}, m)$: On input a secret key sk and message m, outputs a signature σ.
- $\mathsf{DS.Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: On input a message m, a signature σ and a public key pk, outputs accept (reject) if σ is a valid (invalid) signature on m under pk.

Hereafter, the dependence on the public parameters pp is suppressed if an algorithm takes a secret key or public key as input (i.e. the key is assumed to include a description of the public parameters). Security properties are defined with respect to the generic bilinear group model and the public parameters pp describe a generic bilinear group with type-3 pairing.

Pointcheval-Sanders signatures utilise bilinear groups with a type-3 pairing function. In their original work [PS16], PS defined protocols in which the verifier performed work in $\mathbb{G}_2$ (i.e. $\mathbb{G}_2$ is the public key group), whereas the signer worked in $\mathbb{G}_1$ (i.e. $\mathbb{G}_1$ is the signature group). For type-3 pairings, $\mathbb{G}_2$ computations are generally more expensive than $\mathbb{G}_1$ computations. Thus, PS' definitions minimise the signer's work at the cost of increased verifier complexity [PS16]. However, if minimising verifier complexity is a priority for a particular use case, e.g. on a blockchain, one can switch the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$, placing the public key in $\mathbb{G}_1$, and generating signatures in $\mathbb{G}_2$. This switch increases the signer's work (signatures are now computed in $\mathbb{G}_2$) but reduces verifier work (verifiers now work in $\mathbb{G}_1$). Following PS, most of new schemes presented in this work are defined with signature elements in $\mathbb{G}_1$ and public key elements in $\mathbb{G}_2$. In all cases, however, one can switch the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ to obtain identical signature schemes with reduced verifier complexity.

3 Single-Message Signature Schemes

Pointcheval and Sanders proposed a single-message signature scheme defined by the following algorithms [PS16]:

- $\mathsf{PS.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where e is a type-3 pairing.

- $\mathsf{PS.Keygen}(\mathsf{pp})$: Randomly selects a generator $\tilde{g} \overset{R}{\leftarrow} \mathbb{G}_2^*$, and $(x, y_1) \overset{R}{\leftarrow} (\mathbb{F}_p^*)^2$, computes $(\tilde{X}, \tilde{Y}_1) = (\tilde{g}^x, \tilde{g}^{y_1})$, and sets $\mathsf{sk} = (x, y_1)$ and $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1)$.

- $\mathsf{PS.Sign}(\mathsf{sk}, m)$: Parses sk as (x, y_1), randomly selects a generator $h \overset{R}{\leftarrow} \mathbb{G}_1^*$, and outputs a signature $\sigma = (h, h^{x+my_1})$ for message m.

- $\mathsf{PS.Verify}(\mathsf{pk}, m, \sigma)$: Parses σ as (σ_1, σ_2), checks that $\sigma_1 \neq 1_{\mathbb{G}_1}$, and executes the verification check:

$$e(\sigma_1, \tilde{X} \cdot \tilde{Y}_1^m) = e(\sigma_2, \tilde{g}). \tag{1}$$

  If both checks pass, outputs accept, otherwise reject.

Notice that the signer works in $\mathbb{G}_1$, while the verifier works in $\mathbb{G}_2$ (to construct $\tilde{X} \cdot \tilde{Y}_1^m$). Verification costs two pairing calculations, independent of whether signatures appear in $\mathbb{G}_1$ or $\mathbb{G}_2$.

As discussed below, PS generalised this single-message signature scheme to obtain a multi-message signature scheme [PS16]. Before discussing this generalisation, consider a signature scheme in which multiple messages are encoded using a polynomial in the exponent, e.g., $\sum_i m_i y^i$, for a random field element y (as detailed in Section 4). In the "single-message limit" of this scheme, namely $\sum_i m_i y^i \to m_1 y^1 \equiv my$, the algorithms for this "polynomial-based" single-message signature scheme are defined as follows:

- $\mathsf{PolySig.Setup}(1^\lambda)$: Given a security parameter λ, outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where e is a type-3 pairing.

- $\mathsf{PolySig.Keygen}(\mathsf{pp})$: Randomly selects a generator $\tilde{g} \overset{R}{\leftarrow} \mathbb{G}_2^*$, and $(x, y) \overset{R}{\leftarrow} (\mathbb{F}_p^*)^2$. Computes $(\tilde{X}, \tilde{Y}) = (\tilde{g}^x, \tilde{g}^y)$, and sets $\mathsf{sk} = (x, y)$ and $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y})$.
- $\mathsf{PolySig.Sign}(\mathsf{sk}, m)$: Parses sk as (x, y), randomly selects a group generator $h \overset{R}{\leftarrow} \mathbb{G}_1^*$, and outputs the signature $\sigma = (h, h^{x+my})$ for message m.
- $\mathsf{PolySig.Verify}(\mathsf{pk}, m, \sigma)$: Parses σ as (σ_1, σ_2), checks that $\sigma_1 \neq 1_{\mathbb{G}_1}$ and tests whether $e(\sigma_1, \tilde{X} \cdot \tilde{Y}^m) = e(\sigma_2, \tilde{g})$. If both checks pass, outputs accept, otherwise reject.

Clearly, for a single message, signature scheme PS and this "polynomial" signature scheme are identical. Thus, there exists a (trivial) mapping ϕ between the two schemes:

$$\phi : \mathsf{PS} \to \mathsf{PolySig}, \quad \text{with} \quad \phi^{-1} : \mathsf{PolySig} \to \mathsf{PS}, \tag{2}$$

such that $\phi(y_1) = \phi(\phi^{-1}(y)) = y$. Equivalently, one can say that an interchange symmetry exists between the two schemes, $\mathsf{PS} \leftrightarrow \mathsf{PolySig}$, defined by the interchange $y_1 \leftrightarrow y$. The equivalence of these schemes is unsurprising; for a single message m, PS may be interpreted as an encoding of m using a degree one polynomial in the variable y_1 (with x considered constant). This equivalence between PS and $\mathsf{PolySig}$ renders the distinct labels redundant. The label SM is henceforth used to collectively refer these (identical) single-message signature schemes. For completeness, scheme SM is defined by the following algorithms:

- $\mathsf{SM.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, for a type-3 pairing e.
- $\mathsf{SM.Keygen}(\mathsf{pp})$: Randomly selects a generator $\tilde{g} \overset{R}{\leftarrow} \mathbb{G}_2$. and $(x, y) \overset{R}{\leftarrow} (\mathbb{F}_p^*)^2$. Computes $(\tilde{X}, \tilde{Y}) = (\tilde{g}^x, \tilde{g}^y)$, and sets $\mathsf{sk} = (x, y)$ and $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y})$.
- $\mathsf{SM.Sign}(\mathsf{sk}, m)$: Parses sk as (x, y), randomly selects a generator $h \overset{R}{\leftarrow} \mathbb{G}_1^*$, and outputs the signature $\sigma = (h, h^{x+my})$.
- $\mathsf{SM.Verify}(\mathsf{pk}, m, \sigma)$: Parses σ as (σ_1, σ_2), checks that $\sigma_1 \neq 1_{\mathbb{G}_1}$ and:

  $$e(\sigma_1, \tilde{X} \cdot \tilde{Y}^m) = e(\sigma_2, \tilde{g}). \tag{3}$$

  If both checks pass, output accept, otherwise reject.

Though obvious, explicating this basic equivalence between the single-message schemes PS and $\mathsf{PolySig}$ provides context for the multi-message generalisations that follow.

3.1 Security Notions

The standard signature security notion of *existential unforgeability under chosen message attacks* (EUF-CMA) [GMR87] requires that no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ can win the following game, except with negligible probability:

- EUF-CMA SM Security Game
  - Challenger $\mathcal{C}$ executes $\mathsf{SM.Setup}$ and $\mathsf{SM.Keygen}$, and provides public parameters pp and public key pk_* to adversary $\mathcal{A}$.
  - $\mathcal{A}$ may adaptively request signatures on at most Q messages, $\{m_\kappa\}_{\kappa\in[Q]}$. For each query, challenger $\mathcal{C}$ returns $\sigma_\kappa \leftarrow \mathsf{SM.Sign}(\mathsf{sk}, m_\kappa)$ (i.e. equivalently, the challenger provides access to a signing oracle $\mathcal{O}_{\mathsf{SM}}$).
  - Eventually, $\mathcal{A}$ outputs a pair (m_*, σ_*), and wins the game if $m_* \notin \{m_\kappa\}_{\kappa\in[Q]}$, and $\mathsf{SM.Verify}(\mathsf{pk}, m_*, \sigma_*) = \mathsf{accept}$.

Inspired by the LRSW assumption [LRSW00], Pointcheval and Sanders defined the following security assumptions [PS16]:

Definition 3.1. *PS Assumption One (1-PS).* Let $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ define a type-3 bilinear group, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$. For random field elements $x, y \xleftarrow{R} \mathbb{F}_p^*$, let $\tilde{X} = \tilde{g}^x$, $\tilde{Y} = \tilde{g}^y$. Define oracle $\mathcal{O}_{\mathsf{PS}}(m)$, which takes as input $m \in \mathbb{F}_p$, chooses generator $h \xleftarrow{R} \mathbb{G}_1^*$, and outputs a pair $P = (h, h^{x+my})$. Given $(\tilde{g}, \tilde{X}, \tilde{Y})$ and unlimited access to $\mathcal{O}_{\mathsf{PS}}$, no adversary can efficiently generate a pair $P_* = (h_*, h_*^{x+m_*y})$, for $h_* \neq 1_{\mathbb{G}_1}$, where $m_* \in \mathbb{F}_p$ was not previously submitted to the oracle.

Definition 3.2. *PS Assumption Two (PS2).* Let $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ define a type-3 bilinear group, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$. For random field elements $x, y \xleftarrow{R} \mathbb{F}_p^*$, let $X = g^x$, $Y = g^y$, $\tilde{X} = \tilde{g}^x$, $\tilde{Y} = \tilde{g}^y$. Define oracle $\mathcal{O}_{\mathsf{PS2}}(m)$, which takes as input $m \in \mathbb{F}_p$, chooses generator $h \xleftarrow{R} \mathbb{G}_1^*$, and outputs a pair $P = (h, h^{x+my})$. Given $(g, \tilde{g}, Y, \tilde{X}, \tilde{Y})$ and unlimited access to oracle $\mathcal{O}_{\mathsf{PS2}}$, no adversary can efficiently generate a pair $P_* = (h_*, h_*^{x+m_*y})$, for $h_* \neq 1_{\mathbb{G}_1}$, where $m_* \in \mathbb{F}_p$ was not previously submitted to the oracle.

The 1-PS assumption is clearly equivalent to the EUF-CMA SM security game. Moreover, assumption PS2 implies the 1-PS assumption. Pointcheval and Sanders define the security of the single-message signature scheme SM with respect to the 1-PS assumption; i.e. no adversary can efficiently forge an SM signature without breaking the 1-PS assumption. More precisely, assumption PS2 (and thus 1-PS) was shown to hold in the generic bilinear group model, such that, after Q oracle queries and $Q_{\mathbb{G}}$ bilinear group oracle queries, no PPT adversary can generate a valid pair $P_* = (h_*, h_*^{x+m_*y})$ with probability greater than [PS16][2]

$$3(5 + 2Q + Q_{\mathbb{G}})(4 + 2Q + Q_{\mathbb{G}})/2p \;\leq\; \frac{6(Q + Q_{\mathbb{G}})^2}{p} + O\left(\frac{Q + Q_{\mathbb{G}}}{p}\right). \tag{4}$$

To derive this result, note that signing-oracle queries generate $2Q$ polynomials (as exponents of outputted group elements) and group oracle queries generate $Q_{\mathbb{G}}$ polynomials. Combined with the public key elements, one has a total of $(5+2Q+Q_{\mathbb{G}})$ distinct polynomials. Therefore the adversary receives $(5+2Q+Q_{\mathbb{G}})(4+2Q+Q_{\mathbb{G}})/2$ distinct pairs of polynomials, with degree at most three. Hence, the probability that the adversary receives an accidental collision in

[2]This counting differs slightly to PS [PS16] as here the generators are included.

these polynomials is given by Eq. (4). The adversary is also unable to symbolically construct a forgery in the generic bilinear group model (see Theorem 4 in Ref. [PS16]). Thus, SM is EUF-CMA secure under the 1-PS assumption.

Assumptions PS2 and 1-PS are interactive. In subsequent work, Pointcheval and Sanders modified their signature scheme and demonstrated EUF-CMA security with respect to a non-interactive q-type assumption [PS18]. The present work investigates generalisations of the original PS signature scheme and accordingly retains PS' interactive assumptions [PS16] (and generalisations thereof). Future work could generalise the present results to permit non-interactive assumptions, along the lines of Ref. [PS18].

4 Multi-Message Signature Schemes

For a single message, the PS signature scheme and the "polynomial" signature scheme are identical and can therefore be labelled collectively as SM. However, users often require signatures for multiple messages rather than a single message. In these cases one may ask 'How should SM be generalised?' As discussed below, scheme SM affords multiple multi-message generalisations, including one using the PS techniques, and another using polynomial encodings. Accordingly, henceforth the label "PS" shall refer to one particular multi-message generalisation of signature scheme SM, while "PolySig" will refer to a distinct multi-message generalisation. The multi-message signature schemes PS and PolySig are defined in what follows. Security properties of both schemes are then analysed in turn.

4.1 Multi-Message PS Signatures

Consider a set of q messages $(m_1, \dots, m_q) \in \mathbb{F}_p^q$. The multi-message signature scheme PS comprises the following algorithms [PS16]:

- $\mathsf{PS.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, for a type-3 pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
- $\mathsf{PS.Keygen}(\mathsf{pp}, q)$: On input the public parameters pp and an integer q, randomly selects a group generator $\tilde{g} \xleftarrow{R} \mathbb{G}_2^*$, and $q+1$ field elements $(x, y_1, \dots, y_q) \xleftarrow{R} (\mathbb{F}_p^*)^{q+1}$, which are assigned to the secret key:

$$\mathsf{sk} = (x, y_1, \dots, y_q). \tag{5}$$

  Computes the group elements:

$$(\tilde{X}, \tilde{Y}_1, \dots, \tilde{Y}_q) = (\tilde{g}^x, \tilde{g}^{y_1}, \dots, \tilde{g}^{y_q}), \tag{6}$$

  and sets $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \dots, \tilde{Y}_q)$.
- $\mathsf{PS.Sign}(\mathsf{sk}, \{m_i\}_{i\in[q]})$: Parses sk as $(x, y_1, \dots, y_q)$, randomly selects a group generator $h \xleftarrow{R} \mathbb{G}_1^*$, and outputs the signature $\sigma = (h, h^{x+\sum_i m_i y_i})$ for messages $\{m_i\}_{i\in[q]}$.

- $\mathsf{PS.Verify}(\mathsf{pk}, \{m_i\}_{i\in[q]}, \sigma)$: Parses σ as (σ_1, σ_2), checks that $\sigma_1 \neq 1_{\mathbb{G}_1}$ and tests whether $e(\sigma_1, \tilde{X} \cdot \prod_{i\in[q]} \tilde{Y}_i^{m_i}) = e(\sigma_2, \tilde{g})$. If both checks are satisfied, outputs accept, otherwise reject.

In this work, protocol PS is interpreted as a multi-message generalisation of the single-message signature scheme SM.

4.2 Multi-Message Polynomial-Based Signatures

Signature scheme PolySig is an alternative multi-message generalisation of SM that uses a polynomial to encode a set of q messages $(m_1, \ldots, m_q) \in \mathbb{F}_p^q$. The corresponding algorithms are defined as follows:

- $\mathsf{PolySig.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a type-3 pairing.
- $\mathsf{PolySig.Keygen}(\mathsf{pp})$: Randomly selects a generator $\tilde{g} \overset{R}{\leftarrow} \mathbb{G}_2$, and two field elements (x, y), which are assigned to the secret key:

$$\mathsf{sk} = (x, y) \overset{R}{\leftarrow} \mathbb{F}_p^2. \tag{7}$$

  Computes the group elements:

$$(\tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_q) = (\tilde{g}^x, \tilde{g}^{y^1}, \ldots, \tilde{g}^{y^q}), \tag{8}$$

  and sets $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_q)$.
- $\mathsf{PolySig.Sign}(\mathsf{sk}, \{m_i\}_{i\in[q]})$: Parses sk as (x, y), randomly selects a generator $h \overset{R}{\leftarrow} \mathbb{G}_1^*$, and outputs a signature $\sigma = (h, h^{x+\sum_i m_i y^i})$ for messages $\{m_i\}_{i\in[q]}$.
- $\mathsf{PolySig.Verify}(\mathsf{pk}, \{m_i\}_{i\in[q]}, \sigma)$: Parses σ as (σ_1, σ_2), checks that $\sigma_1 \neq 1_{\mathbb{G}_1}$ and executes the check:

$$e(\sigma_1, \tilde{X} \cdot \prod_{i\in[q]} \tilde{Y}_i^{m_i}) = e(\sigma_2, \tilde{g}). \tag{9}$$

  If both checks pass, outputs accept, otherwise reject.

4.3 Security Analysis

4.3.1 Security Assumptions

The following q-type generalisations of the 1-PS assumption are used below.

Definition 4.1. *q-PS Assumption.* Let $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ define a type-3 bilinear group, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$. For $q+1$ random field elements $x, y_i \overset{R}{\leftarrow} \mathbb{F}_p^*$, with $i \in [q]$, let $\tilde{X} = \tilde{g}^x$, and $\tilde{Y}_i = \tilde{g}^{y_i}$. Define oracle $\mathcal{O}_{\mathsf{PS}}(\{m_i\}_{i\in[q]})$, which takes as input messages $\{m_i\}_{i\in[q]} \equiv (m_1, \ldots, m_q) \in (\mathbb{F}_p)^q$, chooses a generator $h \overset{R}{\leftarrow} \mathbb{G}_1^*$, and outputs a pair $P = (h, h^{x+\sum_{i\in[q]} m_i y_i})$. Given $(\tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]})$, and unlimited access to oracle $\mathcal{O}_{\mathsf{PS}}$, no probabilistic polynomial time adversary can efficiently generate a pair $P_* = (h_*, h_*^{x+\sum_{i\in[q]} m_i^* y_i})$, for $h_* \neq 1_{\mathbb{G}_1}$, where $(m_1^*, \ldots, m_q^*) \in (\mathbb{F}_p)^q$ was not previously submitted to the oracle.

Definition 4.2. *q-PolyS Assumption.* Let $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ define a type-3 bilinear group, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$. For random field elements $x, y \overset{R}{\leftarrow} \mathbb{F}_p^*$, let $\tilde{X} = \tilde{g}^x$, and $\tilde{Y}_i = \tilde{g}^{y^i}$, for $i \in [q]$. Define oracle $\mathcal{O}_{\mathsf{PolySig}}(\{m_i\}_{i\in[q]})$, which takes as input messages $\{m_i\}_{i\in[q]} \equiv (m_1, \ldots, m_q) \in (\mathbb{F}_p)^q$, chooses a generator $h \overset{R}{\leftarrow} \mathbb{G}_1^*$, and outputs a pair $P = (h, h^{x+\sum_{i\in[q]} m_i y^i})$. Given $(\tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]})$, and unlimited access to oracle $\mathcal{O}_{\mathsf{PolySig}}$, no probabilistic polynomial time adversary can efficiently generate a pair $P_* = (h_*, h_*^{x+\sum_{i\in[q]} m_i^* y^i})$, for $h_* \neq 1_{\mathbb{G}_1}$, where $(m_1^*, \ldots, m_q^*) \in (\mathbb{F}_p)^q$ was not previously submitted to the oracle.

These assumptions may also be formalised as security games. Let $\mathcal{P} \in \{\mathsf{PS}, \mathsf{PolySig}\}$ denote a multi-message signature scheme. The EUF-CMA PS security game and the EUF-CMA Poly-Sig security game are (collectively) defined as follows:

- <u>EUF-CMA $\mathcal{P}$ Security Game</u>
  - Challenger $\mathcal{C}$ executes $\mathcal{P}.\mathsf{Setup}$ and $\mathcal{P}.\mathsf{Keygen}$, and provides the public parameters pp, public key pk_*, and signing oracle $\mathcal{O}_{\mathcal{P}}$ to the PPT adversary $\mathcal{A}$.
  - $\mathcal{A}$ may adaptively request signatures on at most Q_O message sets $\{m_{i,\kappa}\}_{i\in[q]}$. For each query $\kappa \in [Q_O]$, the oracle returns $(\sigma_{1,\kappa}, \sigma_{2,\kappa}) \leftarrow \mathcal{P}.\mathsf{Sign}(\mathsf{sk}_*, \{m_i\}_{i\in[q]})$.
  - Eventually, $\mathcal{A}$ outputs messages $\{m_i^*\}_{i\in[q]}$, and a signature $\sigma_* = (\sigma_1^*, \sigma_2^*)$. The adversary wins the game if:

$$\mathcal{P}.\mathsf{Verify}(\mathsf{pk}, \{m_i^*\}_{i\in[q]}, \sigma_*) = \mathsf{accept}, \tag{10}$$

    and $\{m_i^*\}_{i\in[q]} \neq \{m_{i,\kappa}\}_{i\in[q]}$, for all $\kappa \in [Q_O]$.

The requirement that $\{m_i^*\}_{i\in[q]} \neq \{m_{i,\kappa}\}_{i\in[q]}$ implies that either there exists $m_j^* \in \{m_i^*\}_{i\in[q]}$ such that $m_j^* \notin \{m_{i,\kappa}\}_{i\in[q]}$ for all $\kappa \in [Q_O]$, or that all elements $m_j^* \in \{m_i^*\}_{i\in[q]}$ were previously queried but $\{m_i^*\}_{i\in[q]}$ was not queried as a single message set. The second condition implies that an adversary may win by, e.g., outputting an accepting forgery for messages:

$$(m_1^*, \ldots, m_q^*) = (m_1, \ldots, m_j, m'_{j+1}, \ldots, m'_q). \tag{11}$$

where the signing oracle was previously queried for messages $(m_1, \ldots, m_q)$ and $(m'_1, \ldots, m'_q)$.

Let PS_q denote the execution of PS for q messages. Protocol PS_q is EUF-CMA PS secure if the q-PS assumption holds. Similarly, Theorem 4.4 (below) proves that $\mathsf{PolySig}_q$ is EUF-CMA Poly-Sig secure if the q-PolyS assumption holds. Theorem 4.5 then demonstrates that the q-PolyS assumption holds in the generic bilinear group model, except with negligible probability. Though not explicitly provided here, a similar proof may be readily constructed to demonstrate that the q-PS assumption holds in the generic bilinear group model.

4.3.2 PS Signature Security

In what follows, the EUF-CMA security of PS is explicitly proven [PS16]. Additional original theorems relating to the security of PS are also proven and a general discussion of the security properties of PS is provided.

The following theorem shows that, if a PPT adversary can break the q-PS assumption, one may leverage this adversary to efficiently break the 1-PS assumption. This theorem is slightly different from, yet equivalent to, Theorem 6 in Ref. [PS16].

Theorem 4.1. *The multi-message signature scheme* PS *is EUF-CMA secure if the q-PS assumption holds. Moreover, an adversary that can efficiently break the q-PS assumption may be leveraged to break the EUF-CMA security of the single-message signature scheme* SM.

Proof. One may readily adapt the proof for Theorem 4.4 below to show that PS_q is EUF-CMA PS secure iff the q-PS assumption holds. To show that breaking the q-PS assumption for $q > 1$ implies the ability to break the 1-PS assumption, assume that adversary $\mathcal{A}_q$ can break the q-PS assumption. Let $\mathcal{A}$ be an adversary against the EUF-CMA SM security game, such that challenger $\mathcal{C}$ supplies $\mathcal{A}$ with elements $(\tilde{g}, \tilde{X}, \tilde{Y})$, for public parameters pp, and access to oracle $\mathcal{O}_{\mathsf{SM}}$. Upon receipt of these elements, $\mathcal{A}$ acts as a q-PS challenger to $\mathcal{A}_q$ by selecting $a_i, b_i \overset{R}{\leftarrow} \mathbb{F}_p^*$, for $i \in [q]$, setting $\tilde{Y}_i = (\tilde{Y})^{b_i} \cdot \tilde{g}^{a_i}$, and providing $(\mathsf{pp}, \tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]})$ to $\mathcal{A}_q$. When $\mathcal{A}_q$ makes the κ-th query to the PS_q signing oracle, for messages $(m_{1,\kappa}, \ldots, m_{q,\kappa})$, adversary $\mathcal{A}$ constructs $M_\kappa = \sum_i b_i m_{i,\kappa}$ and submits it to oracle $\mathcal{O}_{\mathsf{SM}}$, which outputs the pair $P_\kappa = (h_\kappa, h_\kappa^{x+M_\kappa y}) = (h_\kappa, h_\kappa^{x+\sum_{i\in[q]} m_{i,\kappa} y\, b_i})$. Adversary $\mathcal{A}$ then constructs

$$P'_\kappa = (h_\kappa, h_\kappa^{\sum_{i\in[q]} a_i\, m_{i,\kappa}} \cdot h_\kappa^{x+M_\kappa y}), \tag{12}$$

which is a valid PS_q signature on messages $(m_{1,\kappa}, \ldots, m_{q,\kappa})$, with respect to the parameters $(\mathsf{pp}, \tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]})$, and returns P'_κ to $\mathcal{A}_q$.

By assumption, adversary $\mathcal{A}_q$ eventually outputs a set $\{m_i^*\}_{i\in[q]}$ and a valid signature $P_* = (h, h^{x+\sum_{i\in[q]}(y\, b_i + a_i) m_i^*})$. Adversary $\mathcal{A}$ aborts if $\sum_{i\in[q]} b_i m_i^* = \sum_{i\in[q]} b_i m_{i,\kappa}$, for any previous query labelled by κ, and begins again. Otherwise $\mathcal{A}$ constructs

$$P'_* = (h, h^{-\sum_{i\in[q]} a_i\, m_i^*} \cdot h^{x+\sum_{i\in[q]}(y\, b_i + a_i) m_i^*}), \tag{13}$$

which is a valid SM signature for message $M_* = \sum_i m_i^* b_i$. Thus, given that $M_* \neq M_\kappa$ for any previously submitted query, $\mathcal{A}$ outputs the pair (M_*, P'_*) to challenger $\mathcal{C}$ and wins the challenge.

Even if $\mathcal{A}$ aborts during execution, $\mathcal{A}$ eventually succeeds in efficiently outputting a valid SM pair (M_*, P'_*), unless adversary $\mathcal{A}_q$ leveraged information about the quantities b_i to construct a set $\{m_i^*\}_{i\in[q]}$ that satisfies $M_* = M_\kappa$ for some κ. To demonstrate that this is not the case, note that $\mathcal{A}$ can always redefine $\tilde{Y}'_i$ as:

$$\tilde{Y}'_i = \tilde{Y}_i\, \tilde{g}^{a_i} = (\tilde{Y}_i\, \tilde{Y}^{-d_i}) \cdot (\tilde{g}^{a_i}\, \tilde{Y}^{d_i}), \tag{14}$$

which can be interpreted as using different values $b'_i = b_i - d_i$ and $a'_i = a_i + yd_i$, implying that the public key reveals no information about the values b_i. Thus, the adversary's view is independent of b_i, giving a probability of at most Q/p that $\mathcal{A}$ aborts (when $\mathcal{A}_q$ makes Q signing queries). Hence, in either case, $\mathcal{A}$ can efficiently output a valid pair (M_*, P'_*) and break the 1-PS assumption if adversary $\mathcal{A}_q$ can win the EUF-CMA SM security game with non-negligible probability. ■

This proof demonstrates that any adversary that can forge PS_q signatures may be leveraged to forge SM signatures [PS16]. Thus, as SM is EUF-CMA secure [PS16], such an adversary should not exist. Note that the above proof does not demonstrate that an adversary which can break SM may be leveraged to break PS_q for $q > 1$. This inverse reduction is trivially possible if the signing algorithm accepts $q-1$ null messages. However, even if the PS_q signing algorithm does not accept null messages, the following theorem shows that the inverse reduction is possible if protocol PS_q does not perform any additional checks on the inputted messages.

Theorem 4.2. *If the algorithms* $\mathsf{PS.Sign}$ *and/or* $\mathsf{PS.Verify}$ *do not perform checks on inputted messages* $\{m_i\}_{i\in[q]}$ *(i.e. arbitrary* $m_i \in \mathbb{F}_p$ *is allowed, for all* $i \in [q]$*), a PPT adversary that can efficiently break the* 1*-PS assumption may be leveraged to efficiently break the EUF-CMA PS security of the multi-message signature scheme* PS*.*

Proof. Assume that the PPT adversary $\mathcal{A}$ can forge SM signatures with non-negligible probability and therefore break the 1-PS assumption. Let the EUF-CMA PS security game challenger $\mathcal{C}$ execute $\mathsf{PS.Setup}$ and $\mathsf{PS.Keygen}$, and output the public key $\mathsf{pk}_* = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_q)$ to PPT adversary $\mathcal{A}_q$. Upon receipt of pk_*, adversary $\mathcal{A}_q$ sets:

$$\tilde{Y} = \prod_{i\in[q]} \tilde{Y}_i = \tilde{g}^{\sum_{i\in[q]} y_i}, \tag{15}$$

and passes the public key $(\tilde{g}, \tilde{X}, \tilde{Y})$, corresponding to secret key $\mathsf{sk} = (x, y)$ with $y \leftarrow \sum_i y_i$, to adversary $\mathcal{A}$. When $\mathcal{A}$ requests a signature on message m, adversary $\mathcal{A}_q$ sets $m_i \equiv m$ for all $i \in [q]$, and submits the messages $\{m_i\}_{i\in[q]}$ to oracle $\mathcal{O}_{\mathsf{PS}}$. The oracle outputs a pair:

$$P = (h, h^{x+\sum_i m_i y_i}) = (h, h^{x+m\sum_i y_i}). \tag{16}$$

The last expression shows that P is also a valid signature for the single message m under the secret key $\mathsf{sk} = (x, y)$. Adversary $\mathcal{A}_q$ may therefore return P to $\mathcal{A}$.

Eventually $\mathcal{A}$ outputs a message m_* and valid signature pair $P_* = (h, h^{x+m_* y})$, where no signature request was previously made on m_*. Adversary $\mathcal{A}_q$ outputs the messages $\{m_i^*\}_{i\in[q]}$

with $m_i^* \leftarrow m_*$ for all $i \in [q]$, and the pair $P_* = (h, h^{x+m_* y}) = (h, h^{x+\sum_i m_i^* y_i})$ to $\mathcal{C}$. As P_* is also a valid PS signature for messages $\{m_i^*\}_{i\in[q]}$, algorithm $\mathcal{A}_q$ wins the challenge.

■

Implementing a simple check in, e.g., algorithm $\mathsf{PS.Sign}$ to ensure the message set $\{m_i\}_{i\in[q]}$ contains distinct messages prevents the attack described in Theorem 4.2. Such a check is likely in many use cases, such as when the signatures comprise credentials and the messages represent distinct user-attributes. More generally, however, if the signing algorithm for the multi-message signature scheme PS_q implements a simple check requiring the message set to contain distinct messages, it is unclear whether an efficient adversary with a non-negligible probability of breaking SM can be leveraged to break PS_q for $q > 1$. Certainly this inverse reduction was not proven in Ref. [PS16]. Thus, based on existing proofs, it is unclear if security of protocol $\mathsf{PS}_{q>1}$ is dependent on the 1-PS assumption or the q-PS assumption. Discerning this difference would avail a comprehensive understanding of the security properties of PS_q. Two possibilities present themselves, each with different security implications:

1. It is possible to construct an efficient reduction, using an adversary $\mathcal{A}$ that can forge a SM signature, to construct an adversary $\mathcal{A}_q$ that can forge a verifying PS_q signature. In this case, security of PS_q is dependent on the 1-PS assumption.
2. It is not possible to construct an efficient reduction, using an adversary $\mathcal{A}$ that can forge a SM signature, to construct an adversary $\mathcal{A}_q$ that can forge a verifying PS_q signature. In this case, security of PS_q is dependent on the q-PS assumption.

If Case One holds, a break of the single-message signature scheme SM would enable one to break the security of PS_q for arbitrary $q > 1$. Consequently security of the multi-message signature scheme PS_q would be demonstrably dependent on the 1-PS assumption, for any number of messages q. In this case, a single security assumption is required to define the security properties of PS_q for arbitrary $q > 0$. If Case One is true, a proof that the 1-PS assumption was invalid would demonstrably break the security of PS_q for arbitrary $q > 0$.

If Case Two holds, the ability to forge a PS_q signature, for $q > 1$, would be sufficient to break the security of SM (and therefore break the 1-PS assumption). However, unlike Case One, the reverse statement would not hold - construction of an adversary that can efficiently break SM would not imply the existence of an adversary that may break PS_q for arbitrary $q > 1$. In this case the security of PS_q would be dependent on the q-PS assumption, and a break of the q-PS assumption for arbitrary $q > 1$ would imply that the 1-PS assumption could be broken. However, a break of the 1-PS assumption would not imply a break of the q-PS assumption and, more generally, the question of whether a break of the q-PS assumption would imply a break of the q'-PS assumption for $q' > q$ would not be resolved.

To summarise these points, at present the following provable claims can made regarding the security properties of the q-message signature scheme PS_q:

- The q-message signature scheme PS_q is secure provided the q-PS assumption holds.

- If there exists a PPT adversary $\mathcal{A}_q$ that can break the q-PS assumption for $q > 1$ by forging a PS_q signature, one may leverage this adversary to break the q'-PS assumption, for any non-zero $q' < q$, and thus forge a $\mathsf{PS}_{q'}$ signature (see Theorem 4.3 below).

Note that the second claim above is stronger than that implied by Theorem 4.1 (i.e. PS' Theorem 6 [PS16]), which proved only that, if adversary $\mathcal{A}_q$ can break the q-PS assumption, one may leverage $\mathcal{A}_q$ to break the 1-PS assumption. However, Theorem 4.3 below proves that $\mathcal{A}_q$ can be leveraged to break the q'-PS assumption for $q' = q - 1$. Furthermore, using Theorem 4.3 inductively, one proves that $\mathcal{A}_q$ may be leveraged to break the q''-PS assumption for any non-zero $q'' < q$, proving the validity of the second claim above.

In addition to the two provable claims regarding security of PS_q signatures listed above, the following open question is noted:

- Open Question: If there exists a PPT adversary $\mathcal{A}_q$ that can break the q-PS assumption for $q \geq 1$ by forging a PS_q signature, can one leverage this adversary to construct an adversary $\mathcal{A}_{q'}$ which can break the q'-PS assumption for $q' = q + 1$, and thus forge a $\mathsf{PS}_{q'}$ signature, when the $\mathsf{PS}_{q'}$ adversary $\mathcal{A}_{q'}$ cannot submit null messages?

The answer to this question is clearly yes if the $\mathsf{PS}_{q'}$ adversary is allowed to submit null messages. However, if a check is implemented to preclude null messages, the answer appears unclear. Absent an answer, one may conservatively state that the EUF-CMA security of PS_q is dependent on the q-PS assumption. A proof that the above Open Question can be answered affirmatively would demonstrate that the security of PS_q is dependent on the 1-PS assumption (i.e. a single assumption is required, for all q). A negative answer to the Open Question would confirm the conservative view that security of q-PS is dependent on the q-PS assumption (i.e. a different assumption is required for each value of q). Again, if null messages are allowed, there is no distinction.

Theorem 4.3. *An adversary that can efficiently break the q-PS assumption for $q > 1$, may be leveraged to break the q'-PS assumption for $q' = q - 1$.*

Proof. To prove that breaking the q-PS assumption for $q > 1$ implies the ability to break the q'-PS assumption, assume that adversary $\mathcal{A}_q$ can break the q-PS assumption. Let $\mathcal{A}'$ be an adversary against the q'-PS assumption, with $q' = q - 1$, such that challenger $\mathcal{C}'$ supplies $\mathcal{A}'$ with elements $(\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{q'})$, for public parameters pp, and access to oracle $\mathcal{O}_{\mathsf{PS}_{q'}}$. Upon receipt of these elements, $\mathcal{A}'$ acts as a q-PS challenger to $\mathcal{A}_q$ by selecting $a, b \overset{R}{\leftarrow} \mathbb{F}_p^*$, setting $\tilde{Y}_q = (\tilde{Y}_{q'})^b \cdot \tilde{g}^a$, and providing $(\mathsf{pp}, \tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i \in [q']}, \tilde{Y}_q)$ to $\mathcal{A}_q$. When $\mathcal{A}_q$ makes the κ-th query to the PS_q signing oracle, for messages $(m_{1,\kappa}, \ldots, m_{q,\kappa})$ with $\kappa \in [Q]$, adversary $\mathcal{A}'$ constructs $M_{q',\kappa} = m_{q',\kappa} + bm_{q,\kappa}$, and submits the set:

$$(m_{1,\kappa}, \ldots, m_{q'-1,\kappa}, M_{q',\kappa}), \tag{17}$$

to oracle $\mathcal{O}_{\mathsf{PS}_{q'}}$, which outputs the pair:

$$P_\kappa = (h_\kappa, h_\kappa^{x + M_{q',\kappa} y_{q'} + \sum_{i \in [q'-1]} m_{i,\kappa} y_i}). \tag{18}$$

Adversary $\mathcal{A}'$ computes:

$$h_\kappa^{x+M_{q',\kappa}y_{q'}+\sum_{i\in[q'-1]} m_{i,\kappa}y_i} \cdot h_\kappa^{am_{q,\kappa}}, \tag{19}$$

and outputs the pair:

$$P'_\kappa = (h_\kappa, h_\kappa^{x+M_{q',\kappa}y_{q'}+\sum_{i\in[q'-1]} m_{i,\kappa}y_i} \cdot h_\kappa^{a\,m_{q,\kappa}}) \tag{20}$$

to $\mathcal{A}_q$, which is a valid PS_q signature for messages $(m_{1,\kappa}, \ldots, m_{q,\kappa})$ under $(\mathsf{pp}, \tilde{g}, \tilde{X}, \{\tilde{Y}'_i\}_{i\in[q']}, \tilde{Y}'_q)$.

By assumption, adversary $\mathcal{A}_q$ eventually outputs a set $\{m^*_i\}_{i\in[q]}$ and a valid signature

$$P_* = (h, h^{x+m^*_q(by_{q'}+a)+\sum_{i\in[q']} m^*_i y_i}). \tag{21}$$

If $m^*_{q'} + bm^*_q = m_{q',\kappa} + bm_{q,\kappa}$, for any previous query, $\mathcal{A}'$ aborts and begins again. Otherwise message $M^*_{q'} \equiv m^*_{q'} + bm^*_q$ was not previously queried, and $\mathcal{A}'$ constructs

$$P'_* = (h, h^{x+m^*_q(by_{q'}+a)+\sum_{i\in[q']} m^*_i y_i} \cdot h^{-m^*_q a}), \tag{22}$$

which is a valid $\mathsf{PS}_{q'}$ signature for messages $\{m^*_i\}_{i\in[q'-1]} \cup \{M^*_{q'}\}$. Given that $M^*_{q'}$ was not previously submitted, $\mathcal{A}'$ outputs P'_* and messages $\{m^*_i\}_{i\in[q'-1]} \cup \{M^*_{q'}\}$ to challenger $\mathcal{C}'$.

Even if $\mathcal{A}'$ aborts during execution, it eventually succeeds in efficiently outputting a valid signature P'_* for messages $\{m^*_i\}_{i\in[q'-1]} \cup \{M^*_{q'}\}$, unless $\mathcal{A}_q$ used information about b to construct the messages $\{m^*_i\}_{i\in[q]}$, such that $M^*_{q'} \in \{m_{1,\kappa}, \ldots, m_{q'-1,\kappa}, M_{q',\kappa}\}_{\kappa\in[Q]}$. To demonstrate that this is not the case, note that adversary $\mathcal{A}'$ can always redefine $\tilde{Y}'_q$ as:

$$\tilde{Y}'_q = \tilde{Y}_q\, \tilde{g}^a = (\tilde{Y}_q\, \tilde{Y}_{q'}^{-d}) \cdot (\tilde{g}^a\, \tilde{Y}_{q'}^d), \tag{23}$$

which can be interpreted as using different values $b' = b - d$ and $a' = a + dy_{q'}$, implying that the public key reveals no information about b. Consequently the adversary's view is independent of b, giving a probability of at most $Q \times q'/p$ that $\mathcal{A}'$ aborts. Hence, in either case, $\mathcal{A}'$ can efficiently output messages $(m^*_1, \ldots, m^*_{q'-1}, M^*_{q'})$ and corresponding valid signature P'_*, breaking the q'-PS assumption, if adversary $\mathcal{A}_q$ can efficiently break the q-PS assumption for $q = q' + 1$. ■

4.3.3 Poly-Sig Security

The EUF-CMA security proof for the multi-message signature scheme $\mathsf{PolySig}_q$ is performed in two steps. First, it is demonstrated that EUF-CMA Poly-Sig security is equivalent to the q-PolyS assumption. Then it is proven that the q-PolyS assumption holds in the generic bilinear group model (except with negligible probability).

Theorem 4.4. *Multi-message signature scheme* $\mathsf{PolySig}_q$ *is EUF-CMA secure if and only if the q-PolyS assumption holds.*

Proof. Let $\mathcal{A}_{\mathrm{UF}}$ be an adversary in the EUF-CMA Poly-Sig security game, and $\mathcal{A}_{q\mathrm{Poly}}$ an adversary against a q-PolyS assumption challenge. Assume that $\mathcal{A}_{\mathrm{UF}}$ can win the EUF-CMA Poly-Sig security game with non-negligible probability. Upon receipt of the q-PolyS challenge public key:

$$\mathsf{pk}_* = (\tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]}), \tag{24}$$

adversary $\mathcal{A}_{q\mathrm{Poly}}$ passes pk_* to $\mathcal{A}_{\mathrm{UF}}$, and responds to $\mathcal{A}_{\mathrm{UF}}$'s oracle queries using the q-PolyS oracle. With non-negligible probability, $\mathcal{A}_{\mathrm{UF}}$ eventually outputs a verifying $\mathsf{PolySig}_q$ signature for a set of messages, winning the EUF-CMA Poly-Sig security game. Adversary $\mathcal{A}_{q\mathrm{Poly}}$ passes this output to the q-PolyS challenger, successfully breaking the q-PolyS assumption with the same non-negligible probability.

Conversely, assume that $\mathcal{A}_{q\mathrm{Poly}}$ can break the q-PolyS assumption with non-negligible probability. Upon receipt of the EUF-CMA Poly-Sig security game public key pk_*, adversary $\mathcal{A}_{\mathrm{UF}}$ acts as a q-PolyS challenger to $\mathcal{A}_{q\mathrm{Poly}}$ and passes the public key to $\mathcal{A}_{q\mathrm{Poly}}$. Adversary $\mathcal{A}_{\mathrm{UF}}$ answers $\mathcal{A}_{q\mathrm{Poly}}$'s oracle queries using the EUF-CMA Poly-Sig security game oracle. With non-negligible probability, $\mathcal{A}_{q\mathrm{Poly}}$ eventually outputs a verifying $\mathsf{PolySig}$ signature for a set of messages, breaking the q-PolyS assumption. Adversary $\mathcal{A}_{\mathrm{UF}}$ passes this output to the EUF-CMA Poly-Sig security game challenger, and successfully wins the game with the same non-negligible probability.

■

The following theorem shows that the q-PolyS assumption holds in the generic bilinear group model, completing the EUF-CMA security proof for $\mathsf{PolySig}_q$.

Theorem 4.5. *The q-PolyS assumption holds in the generic bilinear group model.*

Proof. Let the PPT adversary $\mathcal{A}$ be provided with the q-PolyS public parameters pp, corresponding public key $\mathsf{pk}_* = (\tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]})$, a signing oracle $\mathcal{O}_{\mathsf{PolySig}}$, and a generic bilinear group oracle $\mathcal{O}_{\mathbb{G}}$. The signing oracle outputs a $\mathsf{PolySig}$ signature $\sigma_\kappa = (\sigma_{1,\kappa}, \sigma_{2,\kappa})$, on input messages $\{m_{i,\kappa}\}_{i\in[q]}$, where $\kappa \in [Q_O]$ labels the Q_O signing oracle queries. To generate new group elements in the generic bilinear group model, the adversary uses generic group operations such as group multiplications and pairing operations. Hence, there exist scalars

$$\begin{aligned}&(\{\gamma_\kappa\}_{\kappa\in[Q_O]}, \{\delta_\kappa\}_{\kappa\in[Q_O]}),\\ &(\{\gamma'_\kappa\}_{\kappa\in[Q_O]}, \{\delta'_\kappa\}_{\kappa\in[Q_O]}),\end{aligned} \tag{25}$$

such that the elements in the adversary's forgery (σ_1^*, σ_2^*) may be written as:[3]

$$\begin{aligned}\sigma_1^* &= \prod_{\kappa\in[Q_O]} \sigma_{1,\kappa}^{\gamma_\kappa} \cdot \prod_{\kappa\in[Q_O]} \sigma_{2,\kappa}^{\delta_\kappa},\\ \sigma_2^* &= \prod_{\kappa\in[Q_O]} \sigma_{1,\kappa}^{\gamma'_\kappa} \cdot \prod_{\kappa\in[Q_O]} \sigma_{2,\kappa}^{\delta'_\kappa}.\end{aligned} \tag{26}$$

[3]Here, the adversary's forgery is constructed using all/any previously seen $\mathbb{G}_1$ elements. Note that the proof carries through if the adversary is also given access to a generator $g \in \mathbb{G}_1^*$ (as occurs in protocols described below); one should then include powers of the generator in the forgery but the proof carries through much the same.

Writing $\sigma_{1,\kappa} = g^{v_\kappa}$, for random element v_κ drawn during signature construction, the adversary's forgery utilises polynomials (in the exponent) in the unknown field elements x, y and $\{v_\kappa\}_{\kappa\in[Q_O]}$, namely:

$$\begin{aligned}
\sigma_1^* &= \prod_{\kappa\in[Q_O]} g^{v_\kappa\gamma_\kappa} \cdot \prod_{\kappa\in[Q_O]} g^{v_\kappa\delta_\kappa(x+\sum_{i\in[q]} m_{i,\kappa}\, y^i)}, \\
\sigma_2^* &= \prod_{\kappa\in[Q_O]} g^{v_\kappa\gamma'_\kappa} \cdot \prod_{\kappa\in[Q_O]} g^{v_\kappa\delta'_\kappa(x+\sum_{i\in[q]} m_{i,\kappa}\, y^i)}.
\end{aligned} \tag{27}$$

If the forgery is valid, it must satisfy the verification check in Eq. (9), giving:

$$\begin{aligned}
&\left(\sum_{\kappa\in[Q_O]} v_\kappa\gamma_\kappa + \sum_{\kappa\in[Q_O]} v_\kappa(x+\sum_{i\in[q]} m_{i,\kappa}y^i)\delta_\kappa\right)\left(x+\sum_{i\in[q]} m_i^*\, y^i\right) \\
= &\sum_{\kappa\in[Q_O]} v_\kappa\gamma'_\kappa + \sum_{\kappa\in[Q_O]} v_\kappa(x+\sum_{i\in[q]} m_{i,\kappa}y^i)\delta'_\kappa.
\end{aligned} \tag{28}$$

Inspecting the $\mathcal{O}(x^2)$ terms and the $\mathcal{O}(v_\kappa)$ terms, in turn, shows that a valid forgery requires $\delta_\kappa = \gamma'_\kappa = 0$, leaving:

$$\left(\sum_{\kappa\in[Q_O]} v_\kappa\gamma_\kappa\right)\left(x+\sum_{i\in[q]} m_i^*\, y^i\right) = \sum_{\kappa\in[Q_O]} v_\kappa(x+\sum_{i\in[q]} m_{i,\kappa}\, y^i)\delta'_\kappa, \tag{29}$$

Treating this expression as a polynomial in the unknown constants v_κ, one has:

$$(x+\sum_{i\in[q]} y^i m_i^*)\times\gamma_\kappa = (x+\sum_{i\in[q]} y^i m_{i,\kappa})\times\delta'_\kappa. \tag{30}$$

For the adversary to produce an accepting forgery, for each oracle query labelled by $\kappa\in[Q_O]$, there must be at least one message $m_{i,\kappa}$, for some $i\in[q]$, such that $m_i^*\neq m_{i,\kappa}$. Absent constructing a non-trivial linear combination of the unknown variables (including powers), a PPT adversary cannot solve the above expression in the generic group model for all κ unless $\gamma_\kappa = \delta'_\kappa = 0$, for all $\kappa\in[Q_O]$, which does not constitute a forgery. Hence, explicit construction of a forgery is not possible.

Next we bound the probability that two polynomials in the adversary's set accidentally evaluate to the same value [BBG05]. The public key contains $q+2$ elements, all with monomial exponents of degree $\leq q$ in the secret elements x, y. A query to the signing oracle outputs two group elements, (σ_1, σ_2), with exponents of maximal degree $\deg(\sigma_1)\leq 1$, and $\deg(\sigma_2)\leq(q+1)$, in the secret elements and the random elements v_κ (drawn during signature generation). The largest-degree polynomial (in the exponent of a group element) returned by a call to the bilinear group oracle occurs for a pairing call. The maximal degree of a polynomial in the exponent of a $\mathbb{G}_2$ element is q, and for a $\mathbb{G}_1$ element it's $(q+1)$. Thus, the degree of the exponent for an element outputted by the group oracle is bounded as $\leq q(q+1)$. In total, an adversary that makes Q_O signing oracle queries and $Q_{\mathbb{G}}$ group oracle queries generates $(2Q_O + Q_{\mathbb{G}} + q + 1)$ polynomials, all with degree $\leq q(q+1)$. By the Schwartz-Zippel lemma [DL78, Zip79, Sch80], the probability of collision among these polynomials is at most:

$$\frac{q(q+1)}{2p}(2Q_O + Q_{\mathbb{G}} + q + 2)(2Q_O + Q_{\mathbb{G}} + q + 1), \tag{31}$$

which is negligible in security parameter $\lambda = \log_2(p)$, for a PPT adversary, with $q \leq \mathsf{poly}(\lambda)$. Thus, the q-PolyS assumption holds in the generic bilinear group model, except with negligible probability. ■

The strategy employed to relate the security of PS_q to that of SM in Theorem 4.2 does not immediately translate over for $\mathsf{PolySig}_q$, as the latter protocol employs related exponents within the elements $\tilde{Y}_i$. Analogous to the open question for scheme PS_q above, open security questions regarding $\mathsf{PolySig}_q$ relate to whether a break of the q-PolyS assumption allows one to also break the q'-Poly assumption for $q' = q \pm 1$. Again, security of $\mathsf{PolySig}_q$ holds if the q-PolyS assumption is valid but it remains to prove whether a break of security for one value of q impacts the security for all/other values of q.[4]

4.4 Relationships Among Multi-Message Signature Schemes

The multi-message signature schemes PS and $\mathsf{PolySig}$ are related by an interchange symmetry with action:

$$y_i \leftrightarrow y^i \qquad \text{and} \qquad \mathsf{sk}_{\mathsf{PS}} \leftrightarrow \mathsf{sk}_{\mathsf{PolySig}}, \tag{32}$$

which converts one scheme into the other, $\mathsf{PS} \leftrightarrow \mathsf{PolySig}$. Despite the existence of this interchange symmetry, the two schemes are not identical. Algorithm $\mathsf{PS.Keygen}$ selects a *set* of random values $y_i \xleftarrow{R} \mathbb{F}_p$, to form $\mathsf{sk}_{\mathsf{PS}}$, and generates the public parameters $\tilde{Y}_i \leftarrow \tilde{g}^{y_i}$. However, $\mathsf{PolySig.Keygen}$ selects a *single* random element $y \xleftarrow{R} \mathbb{F}_p$, to form the secret key $\mathsf{sk}_{\mathsf{PolySig}}$, then *computes* the values $\{y^i\}$ to generate the public parameters $\tilde{Y}_i \leftarrow \tilde{g}^{y^i}$. This difference may render one scheme more efficient than the other in particular use cases.

When contrasting the single-message signature schemes in the preceding section, the interchange symmetry $y_1 \leftrightarrow y$ could be interpreted as an identification; namely by setting $y_1 = y$, one demonstrates that the two single-message signature schemes are *identical*. This equivalence made it consistent to define a single scheme SM. However, for the multi-message schemes PS and $\mathsf{PolySig}$, the interchange operation $y_i \leftrightarrow y^i$ is *merely an interchange symmetry*, not an identification. In particular, if one were to set $y_i = y^i$ for a particular value of $i \in [q]$, in general, one would have $y_j \neq y^j$ for $j \in [q]\backslash\{i\}$. Thus, the multi-message schemes PS and $\mathsf{PolySig}$ are related by the interchange operation $y_i \leftrightarrow y^i$, but are

[4]If, e.g., an adversary can send arbitrary messages to the oracle, they could request a signature on empty messages and the ability to break the q-PolyS assumption would trivially imply the ability to break the $(q-1)$-Poly assumption. Imposing basic restrictions/checks on messages submitted to the oracle [San19] precludes this possibility.

not identical. Properties of the schemes, such as their efficiency, may therefore differ (and in fact do).

Note that the single-message scheme SM may be interpreted as the common single-message limit of both PS and $\mathsf{PolySig}$. That is, in the limit that all but one of the messages goes to zero, protocol SM emerges is the common single-message limit of both PS and $\mathsf{PolySig}$:

$$\mathsf{SM} = \lim_{m_j \to 0,\, \forall j \neq i} \mathsf{PS} \qquad \text{and} \qquad \mathsf{SM} = \lim_{m_j \to 0,\, \forall j \neq i'} \mathsf{PolySig}, \tag{33}$$

where $m_i = m_{i'} \neq 0$. The (single) identification $y_i \leftrightarrow y^{i'}$ shows that the single-message limits of both PS and $\mathsf{PolySig}$ are equivalent. Thus, protocol SM may be interpreted as the common single-message limit of both PS and $\mathsf{PolySig}$.

Finally, note that other related generalisations of SM are possible. Let $\mathcal{S}_{\mathsf{PS}}$ denote the $(q+1)$-dimensional secret space for protocol PS, such that $(x, y_1, y_2, \ldots, y_q) \in \mathcal{S}_{\mathsf{PS}}$. Similarly let $\mathcal{S}_{\mathsf{PolySig}}$ denote the $(q+1)$-dimensional space spanned by sets $(x, y^1, y^2, \ldots, y^q) \in \mathcal{S}_{\mathsf{PolySig}}$, built from a $\mathsf{PolySig}$ secret key (x, y). One has $\mathcal{S}_{\mathsf{PolySig}} \subset \mathcal{S}_{\mathsf{PS}}$, so all (expanded) secret keys for protocol $\mathsf{PolySig}$ are also valid PS secret keys. At the protocol level, however, the algorithms of these two schemes are different. More generally, one can define alternative generalisations of PS' scheme, which define a secret key space as a subspace of $\mathcal{S}_{\mathsf{PolySig}}$ by ordering subsets of secret elements in some alternative way. For example, a secret key could be comprised of even or odd powers of the secret y, or built via some other relationship. Consequently there exist many ways to generalise SM and obtain related multi-message signature schemes.

5 Sequential Aggregate Signatures

Sequential aggregate signatures (SAS) are a generalisation of multi-message signatures in which a single aggregate signature is constructed in place of a set of signatures on distinct messages by distinct signers [LMRS04]. In an SAS, the aggregation is performed sequentially, by each signer in turn, as they sign their message, rather than by a single entity after all messages are signed. In general terms, an SAS is defined by four algorithms:

- $\mathsf{SAS.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters pp.
- $\mathsf{SAS.Keygen}(\mathsf{pp})$: On input the public parameters pp, outputs a key pair $(\mathsf{sk}, \mathsf{pk})$, comprised of a secret key sk and a public key pk.
- $\mathsf{SAS.Sign}(m, \mathsf{sk}, \{m_\alpha\}_{\alpha \in [n]}, \{\mathsf{pk}_\alpha\}_{\alpha \in [n]})$: On input a message m, a secret key sk, and an aggregate signature σ for a set of messages $\{m_\alpha\}_{\alpha \in [n]}$, under public keys $\{\mathsf{pk}_\alpha\}_{\alpha \in [n]}$, outputs a new aggregate signature σ' on messages $\{m_\alpha\}_{\alpha \in [n]} \cup \{m\}$ provided $\mathsf{pk} \notin \{\mathsf{pk}_\alpha\}_{\alpha \in [n]}$.
- $\mathsf{SAS.Verify}(\sigma, \{m_\alpha\}_{\alpha \in [n]}, \{\mathsf{pk}_\alpha\}_{\alpha \in [n]})$: On input an aggregate signature σ, a set of messages $\{m_\alpha\}_{\alpha \in [n]}$, and a set $\{\mathsf{pk}_\alpha\}_{\alpha \in [n]}$ of distinct public keys, outputs accept (reject) if σ a valid (invalid) signature on the messages $\{m_\alpha\}_{\alpha \in [n]}$ under the keys $\{\mathsf{pk}_\alpha\}_{\alpha \in [n]}$.

Pointcheval and Sanders constructed a single-message SAS and (i.e. each aggregation includes a single new message) and interpreted the resulting scheme as a modification or generalisation of their multi-message signature scheme (protocol PS above) [PS16]. One may also interpret the single-message SAS as a generalisation of the single-message signature scheme SM, which is itself the common single-message limit of PS and PolySig. In support of this viewpoint, the single-message SAS is here extended in two distinct ways, realising either a multi-message SAS incorporating features of the PS scheme (here called SAS_PS) or a multi-message SAS leveraging the polynomial-based scheme PolySig (SAS_Poly). From this perspective, the single-message SAS may be interpreted as the common single-message SAS limit of two more-general multi-message SAS schemes. The reduction of both SAS_PS and SAS_Poly to a common limit is completely analogous to the aforementioned reduction of both PS and PolySig to SM. Thus, from a theoretical perspective, it appears appropriate to interpret PS' single-message SAS as a generalisation of SM, such that *both* SM and the single-message SAS may be extended to realise multi-message variants, using either the PS approach or a polynomial-based approach. These correspondences and generalisations are described in what follows.

5.1 Single-Message Sequential Aggregate Signatures

Pointcheval and Sanders [PS16] presented an SAS protocol, labelled here as SAS_SM, in which a set of signing entities may sequentially sign single-messages and append their signature to an existing aggregated signature. Scheme SAS_SM is defined by the following algorithms:

- $\mathsf{SAS\_SM.Setup}(1^\lambda)$: On input a security parameter λ, randomly selects $x \xleftarrow{R} \mathbb{F}_p^*$, and outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ and $(g, X, \tilde{g}, \tilde{X})$, where $(X, \tilde{X}) = (g^x, \tilde{g}^x)$, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a type-3 pairing.
- $\mathsf{SAS\_SM.Keygen}(\mathsf{pp})$: On input the public parameters pp, randomly selects a field element $y_\alpha \xleftarrow{R} \mathbb{F}_p$, computes $\tilde{Y}_\alpha = \tilde{g}^{y_\alpha}$, and sets $\mathsf{sk}_\alpha = y_\alpha$, and $\mathsf{pk}_\alpha = \tilde{Y}_\alpha$. Here α labels the α-th signer and KeyGen may be called repeatedly to generate distinct keys for multiple signers.
- $\mathsf{SAS\_SM.Sign}(\mathsf{sk}, \sigma, \{m_1, \ldots, m_n\}, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}, m)$: Performs the following checks:
  - If $n = 0$, sets $\sigma \leftarrow (g, X)$.
  - If $n > 0$, aborts if $\mathsf{SAS\_SM.Verify}(\mathsf{pp}, \{m_1, \ldots, m_n\}, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}, \sigma) = \mathsf{reject}$.
  - If $m = 0$, aborts.
  - If $\mathsf{pk} = \mathsf{pk}_\alpha$ for some $\alpha \in [n]$, aborts.

  Else, parses sk as y, parses σ as (σ_1, σ_2), and randomly selects $t \xleftarrow{R} \mathbb{F}_p^*$. Computes:

  $$\sigma' = (\sigma_1', \sigma_2') \leftarrow \left(\sigma_1^t, (\sigma_2 \cdot \sigma_1^{ym})^t\right), \qquad (34)$$

  and outputs σ'.

- $\mathsf{SAS\_SM.Verify}(\{m_1,\ldots,m_n\},\{\mathsf{pk}_1,\ldots,\mathsf{pk}_n\},\sigma)$: Parses σ as (σ_1,σ_2), parses pk_α as $\tilde{Y}_\alpha$ for $\alpha\in[n]$, checks that $\sigma_1\neq 1_{\mathbb{G}_1}$, and tests whether:
$$e(\sigma_1, \tilde{X}\cdot\prod_{\alpha\in[n]}(\tilde{Y}_\alpha)^{m_\alpha}) = e(\sigma_2,\tilde{g}). \tag{35}$$
If both checks are satisfied, outputs accept, otherwise reject.

To illustrate the functioning of these algorithms, consider an initial execution of the signing algorithm $\mathsf{SAS\_SM.Sign}(\mathsf{sk}_1,\sigma,\emptyset,\emptyset,m_1)$, such that $n_1=0$ (i.e. there are no previously signed messages). The algorithm sets $\sigma=(\sigma_1,\sigma_2)\leftarrow(g,X)$, selects $t_1\overset{R}{\leftarrow}\mathbb{F}_p^*$, and computes $\sigma'=(\sigma_1',\sigma_2')\leftarrow(g^{t_1},g^{t_1(x+y_1m_1)})$. Notice that this signature has the form of a valid signature under SM, with the identification $h\leftarrow g^{t_1}$. Now consider the execution of $\mathsf{SAS\_SM.Sign}$ by another signer who wishes to sequentially construct an aggregate signature, using the signature σ', to include a signature on message m_2. The second signer executes $\mathsf{SAS\_SM.Sign}(\mathsf{sk}_2,\sigma',\{m_1\},\{\mathsf{pk}_1\},m_2)$, which selects $t_2\overset{R}{\leftarrow}\mathbb{F}_p$, and computes
$$\sigma''=(\sigma_1'',\sigma_2'')\leftarrow(g^{t_1t_2},g^{t_1t_2(x+y_1m_1+y_2m_2)}). \tag{36}$$
The above process can be repeated iteratively for n messages contributed by distinct signers, giving:
$$\sigma=(\sigma_1,\sigma_2)\leftarrow\left(g^{(\prod_\alpha t_\alpha)},g^{(\prod_\alpha t_\alpha)(x+\sum_\alpha y_\alpha m_\alpha)}\right), \tag{37}$$
which is a valid sequential aggregate signature for the messages $\{m_\alpha\}_{\alpha\in[n]}$ under public keys $\{\mathsf{pk}_\alpha\}_{\alpha\in[n]}$.

The signature in Eq. (37) appears very similar to a multi-message signature for n messages output by $\mathsf{PS.Sign}$. Specifically, the aggregate signature outputted by $\mathsf{SAS\_SM.Sign}$ has the same form as a $\mathsf{PS.Sign}$ signature, if one identifies $h\leftarrow g^{\prod_\alpha t_\alpha}$. This similarity motivated the very natural interpretation of PS [PS16], whereby $\mathsf{SAS\_SM}$ was identified as a modification/generalisation of the multi-message scheme PS. Note, however, that although the aggregate signature (37) contains multiple messages, each signer has only contributed *a single message* and, in particular, the set of secret keys $\{y_\alpha\}_{\alpha\in[n]}$ contains *only one key per signer* - no individual signer has contributed multiple messages nor used multiple keys; i.e., no individual signer uses secret values y_α and $y_{\alpha'}$, for $\alpha\neq\alpha'$, as these values were distributed to distinct signers.

5.2 Sequential Aggregate Signature Security

The EUF-CMA security model for single-message SAS schemes (SM-SAS) requires that no PPT adversary can forge a verifying aggregate signature on a set of messages (selected by the adversary) for a set of signers whose secret keys are not all known to the adversary (i.e. the adversary may choose all but one of the signing keys used in the forgery). Security is defined in the certified public keys scenario. That is, no PPT adversary can win the following game, except with negligible probability:

- EUF-CMA SM-SAS Security Game
  - Challenger $\mathcal{C}$ initialises an empty keylist $\mathcal{K} = \emptyset$, and executes the algorithms $\mathsf{SAS\_SM.Setup}$ and $\mathsf{SAS\_SM.Keygen}$, generating parameters $(\mathsf{pp}, g, X, \tilde{g}, \tilde{X})$, and keys $(\mathsf{sk}_*, \mathsf{pk}_*)$. The parameters and public key pk_* is given to the PPT adversary $\mathcal{A}_{\mathsf{SAS\_SM}}$.
  - Adversary $\mathcal{A}_{\mathsf{SAS\_SM}}$ may request to add public keys pk_α to the keylist $\mathcal{K}$. Added keys must be certified by supplying a proof of knowledge of the corresponding secret key.
  - The challenger provides a signing oracle and $\mathcal{A}_{\mathsf{SAS\_SM}}$ may adaptively request aggregate signatures on at most Q messages $(m_1, \ldots, m_Q)$, under public key pk_*. For the κ-th query under key pk_*, $\mathcal{A}_{\mathsf{SAS\_SM}}$ provides a message m_κ and an aggregate signature σ_κ on messages $(m_{1,\kappa}, \ldots, m_{n_\kappa,\kappa})$, under public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_a,\kappa})$, where each public key was already added to the keylist (n_κ denotes the number of already-signed messages in the aggregate signature σ_κ).
  - The oracle verifies that σ_κ is a valid signature on messages $(m_{1,\kappa}, \ldots, m_{n_\kappa,\kappa})$ under public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_a,\kappa})$, and aborts if not. Otherwise returns:

$$\sigma_{\kappa+1} \leftarrow \mathsf{SAS\_SM.Sign}(\mathsf{pp}, \mathsf{sk}_*, \sigma_\kappa, (m_{1,\kappa}, \ldots, m_{n_\kappa,\kappa}), (\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa}), m_\kappa)$$

  - Eventually, $\mathcal{A}_{\mathsf{SAS\_SM}}$ outputs an aggregate signature σ^* on messages $(m_1^*, \ldots, m_{n_*}^*)$, under public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_*})$. The adversary wins the game if the aggregate signature verifies and there exists $\alpha_* \in [n_*]$, such that $\mathsf{pk}_{\alpha_*} = \mathsf{pk}_*$ and $m_{\alpha_*}^*$ was not previously submitted to the oracle. Furthermore, all other public keys must appear on the keylist, namely $\mathsf{pk}_\alpha \in \mathcal{K}$, $\forall \alpha \in [n_*]$ with $\alpha \neq \alpha_*$.

The point here is that the adversary must generate an aggregate signature for a set of messages containing a previously unsigned message $m_{\alpha_*}^*$, which verifies with respect to a set of public keys containing the supplied public key pk_*, with message $m_{\alpha_*}^*$ signed by the secret key corresponding to pk_*. The adversary may freely choose all messages in the final message-set and all additional public keys in the public key set, though the additional keys must be submitted to the keylist before outputting the final forgery.

Security of $\mathsf{SAS\_SM}$ may be proven via a reduction to SM [PS16]. Namely, if there exists a PPT adversary $\mathcal{A}_{\mathsf{SAS\_SM}}$ that can output a verifying $\mathsf{SAS\_SM}$ forgery to win the EUF-CMA SM-SAS security game with non-negligible probability, one may construct an adversary $\mathcal{A}$ that can forge a verifying SM signature. For convenience, the proof is reproduced here.

Theorem 5.1. *The single-message sequential aggregate signature scheme* $\mathsf{SAS\_SM}$ *is EUF-CMA secure, in the certified public key setting, if the* 1*-PS assumption holds. More precisely, a PPT adversary that can efficiently break* $\mathsf{SAS\_SM}$*, by winning the EUF-CMA SM-SAS security game, may be leveraged to break the EUF-CMA security of the single-message signature scheme* SM*.*

Proof. Assume that the PPT adversary $\mathcal{A}_{\mathsf{SAS\_SM}}$ can win the EUF-CMA SM-SAS game with non-negligible probability. Challenger $\mathcal{C}$ for the EUF-CMA SM game executes $\mathsf{SM.Setup}$ and

$\mathsf{SM.Keygen}$, and passes the public key $(\tilde{g}, \tilde{X}, \tilde{Y})$ and public parameters pp to PPT adversary $\mathcal{A}$. Adversary $\mathcal{A}$ requests a signature on $m = 0$, receives the signature $\tau = (\tau_1, \tau_2)$, and assigns $g \leftarrow \tau_1$ and $X \leftarrow \tau_2$. Next, adversary $\mathcal{A}$ initializes an empty keylist $\mathcal{K} = \emptyset$, and acts as a challenger for adversary $\mathcal{A}_{\mathsf{SAS\_SM}}$, passing the public parameters $(\mathsf{pp}, g, X, \tilde{g}, \tilde{X})$ to $\mathcal{A}_{\mathsf{SAS\_SM}}$, along with the (designated) public key $\mathsf{pk}_* \leftarrow \tilde{Y}$. Whenever $\mathcal{A}_{\mathsf{SAS\_SM}}$ seeks to add a public key pk_α to the keylist, it certifies the key by proving knowledge of the corresponding secret key sk_α. Key certification permits extraction of sk_α, and $\mathcal{A}$ maintains its own list of key pairs $(\mathsf{sk}_\alpha, \mathsf{pk}_\alpha)$ for all public keys $\mathsf{pk}_\alpha \in \mathcal{K}$ submitted by $\mathcal{A}_{\mathsf{SAS\_SM}}$. When $\mathcal{A}_{\mathsf{SAS\_SM}}$ requests a signature aggregating message m, under public key pk_*, to an aggregated signature σ_κ for messages $(m_{1,\kappa}, \ldots, m_{n_\kappa,\kappa})$, under public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa})$, $\mathcal{A}$ verifies the signature σ_κ, then requests a signature on m from $\mathcal{C}$, which outputs (σ_1, σ_2). All public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa})$ were previously certified, so $\mathcal{A}$ knows the corresponding secret keys. Hence, $\mathcal{A}$ may draw $t \xleftarrow{R} \mathbb{F}_p^*$ and return the aggregated signature:

$$(\sigma_1^t, \sigma_2^t \cdot \sigma_1^{t(\sum_{\alpha\in[n_\kappa]} m_{\alpha,\kappa}\, y_{\alpha,\kappa})}), \tag{38}$$

which is a valid $\mathsf{SAS\_SM}$ signature. Note that the above simulated signature and actual $\mathsf{SAS\_SM}$ signatures both have the form (τ, τ'), where $\tau \in \mathbb{G}_1^*$ is a uniform element and $\tau' \in \mathbb{G}_1^*$ is the unique element satisfying:

$$e(\tau, \tilde{X} \cdot \prod_{\alpha\in[n_\kappa]} (\mathsf{pk}_{\alpha,\kappa})^{m_{\alpha,\kappa}} \cdot (\mathsf{pk}_*)^m) \;=\; e(\tau', \tilde{g}). \tag{39}$$

Thus, the simulated and actual aggregated signatures have the same distribution, so the simulation is perfect.

By assumption, eventually $\mathcal{A}_{\mathsf{SAS\_SM}}$ outputs a set of messages $(m_1^*, \ldots, m_{n_*}^*)$, a set of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_*})$ and a verifying forged aggregate signature (σ_1^*, σ_2^*), such that $\mathsf{pk}_{\alpha_*} \equiv \mathsf{pk}_* \in \{\mathsf{pk}_\alpha\}_{\alpha\in[n_*]}$, and a signature was not previously requested on the message $m_{\alpha_*}^* \in \{m_\alpha^*\}_{\alpha\in[n_*]}$. Algorithm $\mathcal{A}$ has previously extracted all corresponding secret keys (except for pk_*) and may therefore compute:

$$\hat{\sigma}_2^* \leftarrow \sigma_2^* \cdot \prod_{\alpha\neq\alpha_*} (\sigma_1^*)^{-y_\alpha m_\alpha^*}. \tag{40}$$

Finally, $\mathcal{A}$ outputs the pair $(\sigma_1^*, \hat{\sigma}_2^*)$ to $\mathcal{C}$, which is a verifying pair under SM, for message $m_{\alpha_*}^*$ under the challenge key pk_*. ■

An adversary that can forge a verifying $\mathsf{SAS\_SM}$ signature may thus be leveraged to forge a verifying SM signature, and thereby break the 1-PS assumption. Security of single-message SAS scheme $\mathsf{SAS\_SM}$ against PPT adversaries was therefore related to the 1-PS assumption [PS16].

5.3 Multi-Message PS Sequential Aggregate Signatures

Despite the apparent similarity between signatures outputted by $\mathsf{SAS\_SM}$ and PS, the scheme $\mathsf{SAS\_SM}$ remains a single-message aggregate signature scheme - each sequential aggregate signatures incorporates a *single new message*. It is natural to ask how SAS_SM may be generalised to allow multiple signers to contribute *multiple messages*. Two generalisations of SAS_SM are presented below. The first generalisation incorporates the multi-message techniques used in PS (labelled SAS_PS). The second generalisation leverages PolySig and is described in Section 5.4.

The multi-message sequential aggregate signature scheme SAS_PS is defined by the following algorithms:

- $\mathsf{SAS\_PS.Setup}(1^\lambda)$: On input a security parameter λ, randomly selects $x \overset{R}{\leftarrow} \mathbb{F}_p^*$, and outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ and $(g, X, \tilde{g}, \tilde{X})$, where $(X,\ \tilde{X}) = (g^x,\ \tilde{g}^x)$, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a type-3 pairing.
- $\mathsf{SAS\_PS.Keygen}(\mathsf{pp}, q)$: Takes as input the public parameters pp, and integer q, and randomly selects q field elements:

$$(y_{\alpha,1}, \dots, y_{\alpha,q}) \overset{R}{\leftarrow} (\mathbb{F}_p^*)^q, \tag{41}$$

  for each signer (labelled here as α). Computes the corresponding group elements:

$$(\tilde{Y}_{\alpha,1}, \dots, \tilde{Y}_{\alpha,q}) = (\tilde{g}^{y_{\alpha,1}}, \dots, \tilde{g}^{y_{\alpha,q}}), \tag{42}$$

  and sets $\mathsf{sk}_\alpha = (y_{\alpha,1}, \dots, y_{\alpha,q})$ and $\mathsf{pk}_\alpha = (\tilde{Y}_{\alpha,1}, \dots, \tilde{Y}_{\alpha,q})$.
- $\mathsf{SAS\_PS.Sign}(\mathsf{sk}, \sigma, \{\{m_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n]}, \{\mathsf{pk}_\alpha\}_{\alpha\in[n]}, \{m_i\}_{i\in[q]})$: Performs the following checks:
  - If $n = 0$, sets $\sigma \leftarrow (g, X)$.
  - If $n > 0$, aborts if $\mathsf{SAS\_PS.Verify}(\mathsf{pp}, \{\{m_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n]}, \{\mathsf{pk}_\alpha\}_{\alpha\in[n]}, \sigma) = \mathsf{reject}$.
  - If $\{m_i\}_{i\in[q]} = \emptyset$, aborts.
  - If $\mathsf{pk} = \mathsf{pk}_\alpha$ for some $\alpha \in [n]$, aborts.

  Else, parses sk as $(y_1, \dots, y_q)$, and σ as (σ_1, σ_2), and randomly selects $t \overset{R}{\leftarrow} \mathbb{F}_p^*$. Finally, computes:

$$\sigma' = (\sigma_1', \sigma_2') \leftarrow \left(\sigma_1^t, \left\{\sigma_2 \cdot \sigma_1^{\sum_i y_{\alpha,i} m_{\alpha,i}}\right\}^t\right), \tag{43}$$

  and outputs the signature σ'.
- $\mathsf{SAS\_PS.Verify}(\{\ \{m_{\alpha,i}\}_{i\in[q]}\ \}_{\alpha\in[n]}, \{\mathsf{pk}_\alpha\}_{\alpha\in[n]}, \sigma)$: Parses σ as (σ_1, σ_2), parses pk_α as $(\tilde{Y}_{\alpha,1}, \dots, \tilde{Y}_{\alpha,q})$ for $\alpha \in [n]$, checks that $\sigma_1 \neq 1_{\mathbb{G}_1}$, and tests whether

$$e(\sigma_1, \tilde{X} \cdot \prod_{\alpha\in[n]} \prod_{i\in[q]} (\tilde{Y}_{\alpha,i})^{m_{\alpha,i}}) = e(\sigma_2, \tilde{g}). \tag{44}$$

  If both checks are satisfied, outputs accept, otherwise reject.

Intuitively, $\mathsf{SAS\_PS}$ constructs an SAS for sets of multiple messages, $\{m_{\alpha,i}\}_{i\in[q]}$, contributed by multiple signers (labelled by α). To illustrate the functioning of this new SAS scheme, consider an initial execution of the signing algorithm on messages $\{m_{1,i}\}_{i\in[q]}$:

$$\mathsf{SAS\_PS.Sign}(\mathsf{sk}_1, \sigma, \emptyset, \emptyset, \{m_{1,i}\}_{i\in[q]}), \tag{45}$$

such that $n_1 = 0$ (i.e. no previously signed messages). The algorithm sets $\sigma = (\sigma_1, \sigma_2) \leftarrow (g, X)$, parses σ as (σ_1, σ_2), selects $t_1 \overset{R}{\leftarrow} \mathbb{F}_p^*$, and computes:

$$\sigma' = (\sigma_1', \sigma_2') \leftarrow (g^{t_1}, g^{t_1(x+\sum_i y_{1,i}\, m_{1,i})}), \tag{46}$$

which, incidentally, has the form of a valid multi-message signature under PS_q, with the identification $h \leftarrow g^{t_1}$. Next, another signer ($\alpha = 2$) executes $\mathsf{SAS\_PS.Sign}$ to sequentially construct an aggregate signature that includes the messages $\{m_{2,i}\}_{i\in[q]}$. The second signer executes:

$$\mathsf{SAS\_PS.Sign}(\mathsf{sk}_2, \sigma', \{\{m_{1,i}\}_{i\in[q]}\}, \{\mathsf{pk}_1\}, \{m_{2,i}\}_{i\in[q]}), \tag{47}$$

such that $n_2 = 1$. The algorithm selects $t_2 \overset{R}{\leftarrow} \mathbb{F}_p^*$, and computes

$$\sigma' = (\sigma_1', \sigma_2') \;\; \leftarrow \;\; \left(g^{t_1 t_2}, g^{t_1 t_2\{x+\sum_i (y_{1,i}\, m_{1,i})+\sum_j (y_{2,j}\, m_{2,j})\}}\right), \tag{48}$$

with $i, j \in [q]$. This process can be repeated iteratively for n sets of messages with distinct signers, giving:

$$\sigma' = (\sigma_1', \sigma_2') \leftarrow \left(g^{(\prod_\alpha t_\alpha)}, g^{(\prod_\alpha t_\alpha)\{x+\sum_{\alpha,i}(y_{\alpha,i}\, m_{\alpha,i})\}}\right), \tag{49}$$

which is a valid $\mathsf{SAS\_PS}$ signature for a set of multi-message sets, $\{\{m_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n]}$, under the public keys $\{\mathsf{pk}_\alpha\}_{\alpha\in[n]}$. Thus, as anticipated, $\mathsf{SAS\_PS}$ provides a multi-message generalisation of $\mathsf{SAS\_SM}$, such that each of the $\alpha \in [n]$ signers that contributes to the aggregated signature signs a set of q messages, rather than a single message.

5.3.1 Multi-Message SAS Security Game

One may collectively define a single EUF-CMA security game for protocols $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$. Let $\mathcal{P} \in \{\mathsf{PS}, \mathsf{Poly}\}$. The following EUF-CMA security game is employed in the security analysis for $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$ below.

- EUF-CMA $\mathcal{P}$-SAS Security Game
  - Challenger $\mathcal{C}$ initialises an empty keylist $\mathcal{K} = \emptyset$, and executes the algorithms $\mathsf{SAS\_}\mathcal{P}\mathsf{.Setup}$ and $\mathsf{SAS\_}\mathcal{P}\mathsf{.Keygen}$ to generate parameters $(\mathsf{pp}, g, X, \tilde{g}, \tilde{X})$, and keys that include the pair $(\mathsf{sk}_*, \mathsf{pk}_*)$. The challenger shares $(\mathsf{pp}, g, X, \tilde{g}, \tilde{X})$, the public parameters and key pk_* with the PPT adversary $\mathcal{A}_{\mathsf{SAS\_}\mathcal{P}}$.
  - The adversary may request to add public keys pk_α to the keylist $\mathcal{K}$, where each key pk_α contains q elements. Added keys are certified by providing a proof of knowledge of the corresponding secret key.

– The challenger provides a signing oracle and $\mathcal{A}_{\mathsf{SAS\_}\mathcal{P}}$ may adaptively request aggregate signatures on at most Q sets of messages:

$$(\{m_{i,1}\}_{i\in[q]}, \{m_{i,2}\}_{i\in[q]}, \ldots, \{m_{i,Q}\}_{i\in[q]}),$$

under public key pk_*. For each query $\kappa \in [Q]$, $\mathcal{A}_{\mathsf{SAS\_}\mathcal{P}}$ provides an aggregate signature σ_κ on message sets

$$(\{m_{1,i,\kappa}\}_{i\in[q]}, \{m_{2,i,\kappa}\}_{i\in[q]}, \ldots, \{m_{n_\kappa,i,\kappa}\}_{i\in[q]}), \tag{50}$$

under public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa})$, where each public key was already added to $\mathcal{K}$. Here $\{m_{\alpha,i,\kappa}\}_{i\in[q]}$ denotes the α-th set of messages, among a collection comprising n_κ sets of messages (i.e. $\alpha \in [n_\kappa]$), submitted as part of the κ-th query to the signing oracle, where each set contains q messages. The aggregate signature σ_κ (purportedly) includes a signature for each such set $\{m_{\alpha,i,\kappa}\}_{i\in[q]}$ under a corresponding public key $\mathsf{pk}_{\alpha,\kappa}$.

– Challenger $\mathcal{C}$ verifies that σ_κ is a valid signature on messages (50) under public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa})$, and aborts if not. Otherwise, returns:

$$\sigma_{\kappa+1} \leftarrow \mathsf{SAS\_}\mathcal{P}.\mathsf{Sign}(\mathsf{sk}_*, \sigma_\kappa, \{\{m_{\alpha,i,\kappa}\}_{i\in[q]}\}_{\alpha\in[n_\kappa]}), \{\mathsf{pk}_{\alpha,\kappa}\}_{\alpha\in[n_\kappa]}, \{m_{i,\kappa}\}_{i\in[q]}).$$

– Eventually, $\mathcal{A}_{\mathsf{SAS\_}\mathcal{P}}$ outputs an aggregate signature σ_* on message sets:

$$(\{m^*_{1,i}\}_{i\in[q]}, \ldots, \{m^*_{n_*,i}\}_{i\in[q]}),$$

under public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_*})$. The adversary wins the game if the aggregate signature σ_* verifies under $\mathsf{SAS\_}\mathcal{P}.\mathsf{Verify}$, and there exists a value $\alpha_* \in [n_*]$, such that $\mathsf{pk}_{\alpha_*} = \mathsf{pk}_*$ and the set $\{m^*_{\alpha_*,i}\}_{i\in[q]}$ was not previously submitted to the signing oracle for aggregation under key pk_*. Furthermore, all other public keys must appear on the keylist, namely $\mathsf{pk}_\alpha \in \mathcal{K}$, $\forall\alpha \neq \alpha_*$.

Here the adversary must generate an aggregate signature which verifies with respect to a set of public keys containing the supplied public key pk_*, employed to sign a message-set $\{m^*_{\alpha_*,i}\}_{i\in[q]}$, that must contain a previously unsigned message. Beyond these constraints, the adversary may freely choose the additional messages and public keys in the outputted message-set and public key sets, respectively. However, additional keys must be submitted to the keylist for certification before outputting the final forgery.

5.3.2 Security of the Multi-Message PS-Based SAS

Theorem 5.1 reduced security of $\mathsf{SAS\_SM}$ against PPT adversaries to the 1-PS assumption. Similarly, security of scheme $\mathsf{SAS\_PS}$ against PPT adversaries may be shown to rely on the q-PS assumption (defined in Section 4.3.1) by constructing a reduction between a PPT adversary capable of forging $\mathsf{SAS\_PS}$ signatures and an adversary that may forge PS_q signatures.

Theorem 5.2. *The multi-message sequential aggregate signature scheme* $\mathsf{SAS\_PS}$ *is EUF-CMA secure, in the certified public key setting, if the q-PS assumption holds. More precisely, a PPT adversary that can efficiently break* $\mathsf{SAS\_PS}$*, by winning the EUF-CMA PS-SAS game, may be leveraged to break the EUF-CMA security of the multi-message signature scheme* PS_q.

Proof. Assume that the PPT adversary $\mathcal{A}_{\mathsf{SAS\_PS}}$ can win the EUF-CMA PS-SAS game with non-negligible probability. Challenger $\mathcal{C}$ for the EUF-CMA PS_q game executes $\mathsf{PS.Setup}$ and $\mathsf{PS.Keygen}$, and passes the public key $(\tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]})$ and public parameters pp to PPT adversary $\mathcal{A}$. Adversary $\mathcal{A}$ requests a signature on $\{m_i = 0\}_{i\in[q]}$, receives the signature $\tau = (\tau_1, \tau_2)$, and assigns $g \leftarrow \tau_1$ and $X \leftarrow \tau_2$. Adversary $\mathcal{A}$ then initializes an empty keylist $\mathcal{K} = \emptyset$, and acts as a challenger for adversary $\mathcal{A}_{\mathsf{SAS\_PS}}$, passing $(\mathsf{pp}, g, X, \tilde{g}, \tilde{X})$ to $\mathcal{A}_{\mathsf{SAS\_PS}}$, along with the (designated) public key $\mathsf{pk}_* \leftarrow \{\tilde{Y}_i\}_{i\in[q]}$. Whenever $\mathcal{A}_{\mathsf{SAS\_PS}}$ wishes to add a public key pk_α to the keylist, it must prove knowledge of the corresponding secret key sk_α. Key certification permits extraction of sk_α, and $\mathcal{A}$ maintains its own list of key pairs $(\mathsf{sk}_\alpha, \mathsf{pk}_\alpha)$, for all public keys $\mathsf{pk}_\alpha \in \mathcal{K}$ submitted by $\mathcal{A}_{\mathsf{SAS\_PS}}$. When $\mathcal{A}_{\mathsf{SAS\_PS}}$ requests to add a signature for message-set $\{m_i\}_{i\in[q]}$, under public key pk_*, to the aggregate signature σ_κ for messages-sets (the subscript κ labels the κ-th query):

$$(\{m_{(1,i,\kappa)}\}_{i\in[q]}, \{m_{(2,i,\kappa)}\}_{i\in[q]}, \ldots, \{m_{(n_\kappa,i,\kappa)}\}_{i\in[q]}),$$

under public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa})$, $\mathcal{A}$ verifies the signature σ_κ, and requests a signature on the message-set $\{m_i\}_{i\in[q]}$ from $\mathcal{C}$, which outputs (σ_1, σ_2). All public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa})$ were previously certified, so $\mathcal{A}$ knows the corresponding secret keys. Hence, $\mathcal{A}$ may draw $t \xleftarrow{R} \mathbb{F}_p^*$, and return the aggregated signature:

$$\sigma_{\kappa+1} \leftarrow \left(\sigma_1^t, \sigma_2^t \cdot \sigma_1^{t(\sum_{\alpha,i} m_{(\alpha,i,\kappa)}\, y_{(\alpha,i,\kappa)})}\right), \tag{51}$$

which is a valid $\mathsf{SAS\_PS}$ signature on messages:

$$(\{m_{(1,i,\kappa)}\}_{i\in[q]}, \{m_{(2,i,\kappa)}\}_{i\in[q]}, \ldots, \{m_{(n_\kappa,i,\kappa)}\}_{i\in[q]}, \{m_i\}_{i\in[q]}),$$

under public keys:

$$(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa}, \mathsf{pk}_*). \tag{52}$$

Here $\mathsf{sk}_{\alpha,\kappa} = (y_{(\alpha,1,\kappa)}, y_{(\alpha,2,\kappa)}, \ldots, y_{(\alpha,q,\kappa)})$ is the secret key for public key $\mathsf{pk}_{\alpha,\kappa}$, for the α-th signer, included in the κ-th query. The above simulated signature and actual $\mathsf{SAS\_PS}$ signatures both have the form (τ, τ'), where $\tau \in \mathbb{G}_1^*$ is a uniform element and $\tau' \in \mathbb{G}_1^*$ is the unique element satisfying:

$$e(\tau, \tilde{X} \cdot \prod_{\alpha\in[n_a]} \prod_{i\in[q]} (\mathsf{pk}_{\alpha,i,\kappa})^{m_{(\alpha,i,\kappa)}} \cdot \prod_{i\in[q]} (\mathsf{pk}_{*,i})^{m_i}) \;=\; e(\tau', \tilde{g}). \tag{53}$$

Thus, the simulated and actual aggregated signatures have the same distribution, so the simulation is perfect (here the subscript i on $\mathsf{pk}_{*,i}$ and $\mathsf{pk}_{\alpha,i,\kappa}$ refers to the i-th component of the public key, e.g. $\mathsf{pk}_{*,i} = \tilde{Y}_i$).

By assumption, $\mathcal{A}_{\mathsf{SAS\_PS}}$ eventually outputs a collection of message-sets:

$$(\{m^*_{1,i}\}_{i\in[q]}, \{m^*_{2,i}\}_{i\in[q]}, \ldots, \{m^*_{n_*,i}\}_{i\in[q]}), \tag{54}$$

as well as a set of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_*})$, and a verifying forged aggregate signature (σ^*_1, σ^*_2), such that $\mathsf{pk}_{\alpha_*} \equiv \mathsf{pk}_* \in \{\mathsf{pk}_\alpha\}_{\alpha\in[q_*]}$. Moreover, a signature was not previously requested on one of the message-sets $\{m^*_{\alpha_*,i}\}_{i\in[q]} \in \{\{m^*_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n_*]}$. Algorithm $\mathcal{A}$ has previously extracted all corresponding secret keys (except for pk_*) and may therefore compute:

$$\hat{\sigma}^*_2 \leftarrow \sigma^*_2 \cdot \prod_{\alpha\neq\alpha_*} \prod_{i\in[q]} (\sigma^*_1)^{-y_{\alpha,i}\, m^*_{\alpha,i}}. \tag{55}$$

Finally, $\mathcal{A}$ outputs the pair $(\sigma^*_1, \hat{\sigma}^*_2)$ to $\mathcal{C}$, which is a verifying pair under PS_q, for message-set $\{m^*_{\alpha_*,i}\}_{i\in[q]}$ under the challenge key pk_*. ■

The above proof demonstrates that an efficient adversary which can forge a verifying SAS_PS signature can be leveraged to forge a verifying PS_q signature, and thereby break the q-PS assumption. Scheme SAS_PS is therefore secure against PPT adversaries if the q-PS assumption holds, generalising the SM-SAS results of Ref. [PS16] to the multi-message PS-SAS scheme.

5.4 Multi-Message Poly-Based Sequential Aggregate Signatures

One may also construct a multi-message generalisation of SAS_SM using the polynomial-based techniques employed in PolySig. The resulting multi-message SAS, named SAS_Poly, contains the following algorithms:

- $\mathsf{SAS\_Poly.Setup}(1^\lambda)$: On input a security parameter λ, randomly selects $x \overset{R}{\leftarrow} \mathbb{F}^*_p$, and outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ and $(g, X, \tilde{g}, \tilde{X})$, where $(X,\, \tilde{X}) = (g^x,\, \tilde{g}^x)$, with generators $(g, \tilde{g}) \in \mathbb{G}^*_1 \times \mathbb{G}^*_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a type-3 pairing.
- $\mathsf{SAS\_Poly.Keygen}(\mathsf{pp}, q)$: Takes as input the public parameters pp, integer q, and randomly selects a field element, $y_\alpha \overset{R}{\leftarrow} \mathbb{F}^*_p$, where α labels a signer. Computes the following group elements:

$$(\tilde{Y}_{\alpha,1}, \tilde{Y}_{\alpha,2}, \ldots, \tilde{Y}_{\alpha,q}) = (\tilde{g}^{y^1_\alpha}, \tilde{g}^{y^2_\alpha}, \ldots, \tilde{g}^{y^q_\alpha}), \tag{56}$$

  and sets $\mathsf{sk}_\alpha = y_\alpha$ and $\mathsf{pk}_\alpha = (\tilde{Y}_{\alpha,1}, \ldots, \tilde{Y}_{\alpha,q})$.
- $\mathsf{SAS\_Poly.Sign}(\mathsf{sk}, \sigma, \{\{m_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n]}, \{\mathsf{pk}_\alpha\}_{\alpha\in[n]}, \{m_i\}_{i\in[q]})$: Performs the following checks:
  - If $n = 0$, sets $\sigma \leftarrow (g, X)$.
  - If $n > 0$, aborts if $\mathsf{SAS\_Poly.Verify}(\mathsf{pp}, \{\{m_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n]}, \{\mathsf{pk}_\alpha\}_{\alpha\in[n]}, \sigma) = \mathsf{reject}$.
  - If $\{m_i\}_{i\in[q]} = \emptyset$, aborts.

– If $\mathsf{pk} = \mathsf{pk}_\alpha$ for some $\alpha \in [n]$, aborts.

Else, parses sk as y, and σ as (σ_1, σ_2), and randomly selects $t \xleftarrow{R} \mathbb{F}_p^*$. Outputs the signature

$$\sigma' = (\sigma_1', \sigma_2') \leftarrow \left(\sigma_1^t, (\sigma_2 \cdot \sigma_1^{\sum_i m_i y^i})^t\right). \tag{57}$$

- $\mathsf{SAS\_Poly.Verify}(\{\{m_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n]}, \{\mathsf{pk}_\alpha\}_{\alpha\in[n]}, \sigma)$: Parses σ as (σ_1, σ_2), parses pk_α as $(\tilde{Y}_{\alpha,1}, \ldots, \tilde{Y}_{\alpha,q})$ for $\alpha \in [n]$, checks that $\sigma_1 \neq 1_{\mathbb{G}_1}$, and tests whether

$$e(\sigma_1, \tilde{X} \cdot \prod_{\alpha\in[n]} \prod_{i\in[q]} (\tilde{Y}_{\alpha,i})^{m_{\alpha,i}}) = e(\sigma_2, \tilde{g}). \tag{58}$$

If both checks are satisfied, outputs accept, otherwise reject.

To demonstrate the functioning of this new SAS scheme, consider an example execution of the signing algorithm on initial messages $\{m_{1,i}\}_{i\in[q]}$:

$$\mathsf{SAS\_Poly.Sign}(\mathsf{sk}_1, \sigma, \emptyset, \emptyset, \{m_{1,i}\}_{i\in[q]}), \tag{59}$$

such that $n_1 = 0$ (i.e. no previously signed message-sets). The algorithm sets $\sigma = (\sigma_1, \sigma_2) \leftarrow (g, X)$, parses σ as (σ_1, σ_2), selects $t_1 \xleftarrow{R} \mathbb{F}_p^*$, and computes:

$$\sigma' = (\sigma_1', \sigma_2') \leftarrow (g^{t_1}, g^{t_1(x+\sum_i m_{1,i}\, y_1^i)}), \tag{60}$$

which has the form of a valid multi-message signature under $\mathsf{PolySig}$, with the identification $h \leftarrow g^{t_1}$. Now consider the execution of $\mathsf{SAS\_Poly.Sign}$ by another signer ($\alpha = 2$), who wishes to sequentially construct an aggregate signature that includes the messages $\{m_{2,i}\}_{i\in[q]}$. The second signer executes:

$$\mathsf{SAS\_Poly.Sign}(\mathsf{sk}_2, \sigma', \{\{m_{1,i}\}_{i\in[q]}\}, \{\mathsf{pk}_1\}, \{m_{2,i}\}_{i\in[q]}), \tag{61}$$

where $n_2 = 1$. This algorithm selects $t_2 \xleftarrow{R} \mathbb{F}_p^*$, and computes

$$\sigma' = (\sigma_1', \sigma_2') \quad \leftarrow \quad \left(g^{t_1 t_2}, g^{t_1 t_2\{x+\sum_i (m_{1,i}\, y_1^i) + \sum_j (m_{2,j}\, y_2^j)\}}\right), \tag{62}$$

with $i, j \in [q]$ and $\alpha \in [n]$. Repeating this process iteratively for n sets of messages with distinct signers, gives:

$$\sigma' = (\sigma_1', \sigma_2') \leftarrow \left(g^{(\prod_\alpha t_\alpha)}, g^{(\prod_\alpha t_\alpha)\{x+\sum_{\alpha,i}(m_{\alpha,i}\, y_\alpha^i)\}}\right), \tag{63}$$

which is a valid $\mathsf{SAS\_Poly}$ signature for a set of multi-message sets, $\{\{m_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n]}$, under the public keys $\{\mathsf{pk}_\alpha\}_{\alpha\in[n]}$. Thus, $\mathsf{SAS\_Poly}$ provides an alternative multi-message generalisation of $\mathsf{SAS\_SM}$.

5.4.1 Security of the Multi-Message Poly-SAS

Security of $\mathsf{SAS\_PS}$ was defined relative to the q-PS assumption using a reduction between a PPT $\mathsf{SAS\_PS}$ adversary and a PPT PS_q adversary. Similarly, security of scheme $\mathsf{SAS\_Poly}$ against PPT adversaries may be defined relative to the q-PolyS assumption (defined in Section 4.3.1)) by constructing a reduction between a PPT adversary capable of forging $\mathsf{SAS\_Poly}$ signatures and an adversary that may forge $\mathsf{PolySig}_q$ signatures. The following theorem uses the EUF-CMA Poly-SAS security game defined in Section 5.3.1 (with $\mathcal{P} = \mathsf{Poly}$).

Theorem 5.3. *The multi-message sequential aggregate signature scheme* $\mathsf{SAS\_Poly}$ *is EUF-CMA secure, in the certified public key setting, if the q-PolyS assumption holds. More precisely, a PPT adversary that can efficiently break* $\mathsf{SAS\_Poly}$*, by winning the EUF-CMA Poly-SAS game, may be leveraged to break the EUF-CMA security of the multi-message signature scheme* $\mathsf{PolySig}_q$*.*

Proof. Assume that the PPT adversary $\mathcal{A}_{\mathsf{SAS\_Poly}}$ can win the EUF-CMA Poly-SAS game with non-negligible probability. Challenger $\mathcal{C}$ for the EUF-CMA $\mathsf{PolySig}_q$ game executes $\mathsf{PolySig.Setup}$ and $\mathsf{PolySig.Keygen}$, and passes the public key $(\tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]})$ and public parameters pp to PPT adversary $\mathcal{A}$. Adversary $\mathcal{A}$ requests a signature on $\{m_i = 0\}_{i\in[q]}$, receives the signature $\tau = (\tau_1, \tau_2)$, and assigns $g \leftarrow \tau_1$ and $X \leftarrow \tau_2$. Next, $\mathcal{A}$ initializes an empty keylist $\mathcal{K} = \emptyset$, and acts as a challenger for adversary $\mathcal{A}_{\mathsf{SAS\_Poly}}$, passing the public parameters $(\mathsf{pp}, g, X, \tilde{g}, \tilde{X})$ to $\mathcal{A}_{\mathsf{SAS\_Poly}}$, along with the (designated) public key $\mathsf{pk}_* \leftarrow \{\tilde{Y}_i\}_{i\in[q]}$. Whenever $\mathcal{A}_{\mathsf{SAS\_Poly}}$ wishes to add a public key pk_α to the keylist, it must prove knowledge of the corresponding secret key sk_α. Key certification permits extraction of sk_α, and $\mathcal{A}$ maintains its own list of key pairs $(\mathsf{sk}_\alpha, \mathsf{pk}_\alpha)$ for all public keys $\mathsf{pk}_\alpha \in \mathcal{K}$ submitted by $\mathcal{A}_{\mathsf{SAS\_Poly}}$. When $\mathcal{A}_{\mathsf{SAS\_Poly}}$ requests a signature aggregating message-set $\{m_i\}_{i\in[q]}$, under the secret key corresponding to public key pk_*, to the aggregate signature σ_κ for messages-sets:

$$(\{m_{(1,i,\kappa)}\}_{i\in[q]}, \{m_{(2,i,\kappa)}\}_{i\in[q]}, \ldots, \{m_{(n_\kappa,i,\kappa)}\}_{i\in[q]}),$$

under public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{\kappa,n_\kappa})$, $\mathcal{A}$ verifies the signature σ_κ, and requests a signature on the message-set $\{m_i\}_{i\in[q]}$ from $\mathcal{C}$, which outputs (σ_1, σ_2). Here the subscript κ labels the κ-th query. All public keys $(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa})$ were previously certified, so $\mathcal{A}$ knows the corresponding secret keys. Hence, $\mathcal{A}$ may draw $t \xleftarrow{R} \mathbb{F}_p^*$, and return the aggregated signature:

$$\sigma_{\kappa+1} \leftarrow \left(\sigma_1^t, \sigma_2^t \cdot \sigma_1^{t(\sum_{\alpha,i} m_{(\alpha,i,\kappa)}\, y^i_{(\alpha,\kappa)})}\right), \tag{64}$$

which is a valid $\mathsf{SAS\_Poly}$ signature on messages:

$$(\{m_{(1,i,\kappa)}\}_{i\in[q]}, \{m_{(2,i,\kappa)}\}_{i\in[q]}, \ldots, \{m_{(n_\kappa,i,\kappa)}\}_{i\in[q]}, \{m_i\}_{i\in[q]}),$$

under public keys:

$$(\mathsf{pk}_{1,\kappa}, \ldots, \mathsf{pk}_{n_\kappa,\kappa}, \mathsf{pk}_*). \tag{65}$$

Here $\mathsf{sk}_{\alpha,\kappa} = y_{(\alpha,\kappa)}$ denotes the single-element secret key corresponding to the public key $\mathsf{pk}_{\alpha,\kappa}$ for the α-th signer, included in the κ-th query. The above simulated signature and actual $\mathsf{SAS\_Poly}$ signatures both have the form (τ, τ'), where $\tau \in \mathbb{G}_1^*$ is a uniform element and $\tau' \in \mathbb{G}_1^*$ is the unique element satisfying:

$$e(\tau, \tilde{X} \cdot \prod_{\alpha\in[n_\kappa]} \prod_{i\in[q]} (\mathsf{pk}_{\alpha,i,\kappa})^{m_{(\alpha,i,\kappa)}} \cdot \prod_{i\in[q]} (\mathsf{pk}_{*,i})^{m_i}) \quad = \quad e(\tau', \tilde{g}). \tag{66}$$

Thus, the simulated and actual aggregated signatures have the same distribution, so the simulation is perfect. Note that the subscript i refers to the i-th component of the private key (e.g. $\mathsf{pk}_{*,i} = \tilde{Y}_i$).

By assumption, $\mathcal{A}_{\mathsf{SAS\_Poly}}$ eventually outputs a collection of message-sets:

$$(\{m_{1,i}^*\}_{i\in[q]}, \{m_{2,i}^*\}_{i\in[q]}, \ldots, \{m_{n_*,i}^*\}_{i\in[q]}), \tag{67}$$

as well as a set of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_*})$, and a verifying forged aggregate signature (σ_1^*, σ_2^*), such that $\mathsf{pk}_{\alpha_*} \equiv \mathsf{pk}_* \in \{\mathsf{pk}_\alpha\}_{\alpha\in[q_*]}$. Moreover, a signature was not previously requested for (at least) one of the message-sets $\{m_{\alpha_*,i}^*\}_{i\in[q]} \in \{\{m_{\alpha,i}^*\}_{i\in[q]}\}_{\alpha\in[n_*]}$. Algorithm $\mathcal{A}$ has previously extracted all corresponding secret keys (except for pk_*) and may therefore compute:

$$\hat{\sigma}_2^* \leftarrow \sigma_2^* \cdot \prod_{\alpha\neq\alpha_*} \prod_{i\in[q]} (\sigma_1^*)^{-y_\alpha^i m_{\alpha,i}^*}. \tag{68}$$

Finally, $\mathcal{A}$ outputs the pair $(\sigma_1^*, \hat{\sigma}_2^*)$ to $\mathcal{C}$, which is a verifying pair under $\mathsf{PolySig}_q$, for message-set $\{m_{\alpha_*,i}^*\}_{i\in[q]}$ under the challenge key pk_*. ■

The above proof demonstrates that an efficient adversary that can forge a verifying $\mathsf{SAS\_Poly}$ signature may be leveraged to forge a verifying $\mathsf{PolySig}_q$ signature, and thereby break the q-PolyS assumption. Security of scheme $\mathsf{SAS\_Poly}$ against PPT adversaries is therefore assured if the q-PolyS assumption holds.

5.5 Relationships Among SAS Schemes

$\mathsf{SAS\_Poly}$ is an alternative multi-message generalisation of the single-message scheme $\mathsf{SAS\_SM}$. Nonetheless $\mathsf{SAS\_Poly}$ is very similar to $\mathsf{SAS\_PS}$, except that the secret key for an individual signer using $\mathsf{SAS\_Poly}$ is comprised of a single element ($\mathsf{sk}_\alpha^{\mathsf{SAS\_Poly}} = y_\alpha$) instead of a set of q elements ($\mathsf{sk}_\alpha^{\mathsf{SAS\_PS}} = (y_{\alpha,1}, \ldots, y_{\alpha,q})$). For n distinct signers of q messages, one therefore requires n distinct secret elements when using $\mathsf{SAS\_Poly}$ (namely $\{y_\alpha\}_{\alpha\in[n]}$), compared to $n \times q$ secret elements when using $\mathsf{SAS\_PS}$ (i.e. $\{\{y_{\alpha,i}\}_{i\in[q]}\}_{\alpha\in[n]}$). This reduced number of secret elements can trigger further efficiencies when leveraged within e.g. threshold credential issuance schemes.

Similar to the interchange relationship between PS and $\mathsf{PolySig}$, one may define an interchange operation between $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$. Specifically, consider a valid signature outputted by $\mathsf{SAS\_PS}$ (i.e. $\sigma_{\mathsf{SAS\_PS}}$), and another outputted by $\mathsf{SAS\_Poly}$ ($\sigma_{\mathsf{SAS\_Poly}}$), each for a fixed value of the integers q and n. Perform the interchange operation:

$$y_{\alpha,i} \leftrightarrow y^{i}_{\alpha}, \tag{69}$$

which induces the following transformations:

$$\sigma_{\mathsf{SAS\_PS}} \leftrightarrow \sigma_{\mathsf{SAS\_Poly}} \qquad \text{and} \qquad \mathsf{pk}^{\mathsf{SAS\_PS}}_{\alpha} \leftrightarrow \mathsf{pk}^{\mathsf{SAS\_Poly}}_{\alpha}. \tag{70}$$

The transformed signatures successfully verify under the transformed public keys, demonstrating a symmetry relationship between $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$.

Note also that $\mathsf{SAS\_SM}$ may be interpreted as the common single-message limit of both $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$, namely:

$$\mathsf{SAS\_SM} \;=\; \lim_{m_{\alpha,j}\to 0,\,\forall j\neq i} \mathsf{SAS\_PS}, \tag{71}$$

and:

$$\mathsf{SAS\_SM} \;=\; \lim_{m_{\alpha,j}\to 0,\,\forall j\neq i} \mathsf{SAS\_Poly}, \tag{72}$$

where $m_{\alpha,i} \neq 0$, in both cases. Identifying $y_{\alpha,i} \leftrightarrow y^{i}_{\alpha}$, after taking the limit, shows that the single-message limits of both $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$ are equivalent. Taking these limits transforms the multi-message SAS schemes $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$ (i.e. each signer signs multiple messages), to the single-message SAS scheme $\mathsf{SAS\_SM}$ (i.e. each signer signs a single message). Thus, $\mathsf{SAS\_SM}$ may be understood as the common single-message limit of both $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$.

The multi-signer multi-message schemes $\mathsf{SAS\_PS}$ and $\mathsf{SAS\_Poly}$ are also related to the single-signer multi-message schemes PS_q and $\mathsf{PolySig}_q$, respectively. Taking the single-signer limit of $\mathsf{SAS\_PS}$, one obtains:

$$\mathsf{PS}_q \;=\; \lim_{y_{\alpha,i}\to 0,\,\forall \alpha\neq\alpha',i\in[q]} \mathsf{SAS\_PS}, \tag{73}$$

where $y_{\alpha',i} \neq 0$, for a single fixed value of $\alpha = \alpha'$ (i.e. the single remaining signer is labelled as α'), demonstrating that PS_q is the single-signer limit of $\mathsf{SAS\_PS}$. Thus, $\mathsf{SAS\_PS}$ combines aspects of both the multi-signer single-message SAS scheme $\mathsf{SAS\_SM}$, and the single-signer multi-message scheme PS_q. $\mathsf{SAS\_PS}$ is therefore understood as a generalisation of both $\mathsf{SAS\_SM}$ and PS_q. Similarly, $\mathsf{PolySig}_q$ is the single-signer limit of $\mathsf{SAS\_Poly}$:

$$\mathsf{PolySig}_q \;=\; \lim_{y_{\alpha}\to 0,\,\forall \alpha\neq\alpha'} \mathsf{SAS\_Poly}, \tag{74}$$

where $y_{\alpha'} \neq 0$ for a fixed signer labelled by $\alpha = \alpha'$. Hence, $\mathsf{SAS\_Poly}$ is a generalisation of both $\mathsf{SAS\_SM}$ and $\mathsf{PolySig}_q$.

6 Signing Message Commitments

Pointcheval and Sanders also presented a scheme for generating signatures on committed messages [PS16]. Signatures over message commitments leverage notions first introduced by Chaum, relating to blind signatures [Cha83]. Such schemes are useful if the message-set contains secret information as occurs, e.g., within anonymous credential schemes [San19]. The basic idea is that users supply a commitment to the messages (such as a Pedersen commitment [Ped92]) and, if convinced that the commitment is valid, the signer provides a signature on the commitment [ASM06]. Users may then *unblind* the signature by performing an operation on the signed message commitment to produce a valid signature over the underlying messages, without revealing the messages to the signer (e.g. [BFPV13]). In what follows, the PS scheme for signed commitments is briefly presented. Leveraging the mapping between PS-based signatures and polynomial-based signatures, a new polynomial-based protocol for signed message commitments is then constructed.

6.1 PS Signatures for Committed Messages

The PS scheme for generating signatures on committed messages (here called $\mathsf{SCM\_PS}$) comprises the following algorithms:

- $\mathsf{SCM\_PS.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters $\mathsf{pp} \leftarrow (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a type-3 pairing.
- $\mathsf{SCM\_PS.Keygen}(\mathsf{pp}, q)$: On input the public parameters pp and an integer q, selects random generators $(g, \tilde{g}) \xleftarrow{R} \mathbb{G}_1^* \times \mathbb{G}_2^*$, and selects the random field elements:
$$(x, y_1, \ldots, y_q) \xleftarrow{R} (\mathbb{F}_p^*)^{q+1}. \tag{75}$$
Computes:
$$\begin{aligned}(X, Y_1, \ldots, Y_q) &= (g^x, g^{y_1}, \ldots, g^{y_q}),\\ (\tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_q) &= (\tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_q}),\end{aligned} \tag{76}$$
sets $\mathsf{sk} = X$, and sets:
$$\mathsf{pk} = (g, Y_1, \ldots, Y_q, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_q). \tag{77}$$
- $\mathsf{SCM\_PS.Protocol}$: To obtain a signature on messages $\{m_i\}_{i\in[q]}$, a user selects a random blinding factor $b \xleftarrow{R} \mathbb{F}_p^*$, computes the commitment $C \leftarrow g^b \prod_{i\in[q]} Y_i^{m_i}$, and sends C to the signer. The user and signer engage in a proof of knowledge protocol for the commitment openings. If the signer is convinced by the user, the signer selects $u \xleftarrow{R} \mathbb{F}_p^*$, and returns $\sigma' \leftarrow (g^u, (XC)^u)$. The user generates an unblinded signature as:
$$\sigma \;\leftarrow\; (\sigma'_1, \sigma'_2(\sigma'_1)^{-b}) \;=\; (g^u, (XCg^{-b})^u) \;=\; (g^u, g^{u(x+\sum_{i\in[q]} y_i m_i)}), \tag{78}$$
which is a valid PS signature for the messages $\{m_i\}_{i\in[q]}$.

To convince a verifier that the user's attributes were certified by the credential issuer, without revealing the underlying attributes, PS propose that the user and verifier engage in a zero-knowledge proof of knowledge (such as Schnorr's interactive protocol [Sch90]). Specifically, the user aggregates a signature on messages $\{m_i\}_{i\in[q]}$, with a random dummy message $r \xleftarrow{R} \mathbb{F}_p^*$, under a "secret key" $y_r = 1 \in \mathbb{F}_p$. The resulting signature is valid for the set $\{m_i\}_{i\in[q]} \cup \{r\}$ and does not reveal information about the messages $\{m_i\}_{i\in[q]}$. More precisely, the user selects $t, r \xleftarrow{R} \mathbb{F}_p^*$, and computes:

$$\sigma' \;\leftarrow\; (\sigma_1^t, \sigma_2^t \cdot \sigma_1^{rt}) \;=\; (g^{ut}, g^{ut(x+r+\sum_{i\in[q]} y_i m_i)}). \tag{79}$$

The user sends σ' to the verifier and constructs a zero-knowledge proof of knowledge π for $\{m_i\}_{i\in[q]} \cup \{r\}$, which satisfy:

$$e(\sigma_1', \tilde{g}^r \cdot \tilde{X} \cdot \prod_{i\in[q]} \tilde{Y}_i^{m_i}) \;=\; e(\sigma_2', \tilde{g}). \tag{80}$$

If the proof of knowledge π is valid, the verifier accepts the signature.

The public key for a single-message implementation of $\mathsf{SCM\_PS}$ has the form $\mathsf{pk} = (g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$. Thus, the single-message implementation of $\mathsf{SCM\_PS}$ is secure if the PS2 assumption holds [PS16]. With regard to the multi-message implementation of $\mathsf{SCM\_PS}_q$ (i.e. for $q > 1$), PS note that one may reduce an adversary that can break the EUF-CMA property of $\mathsf{SCM\_PS}_{q>1}$ to an adversary that can forge $\mathsf{SCM\_PS}_{q=1}$ signatures. Similar to the points raised in Section 4, proving that a successful adversary against $\mathsf{SCM\_PS}_q$ can be leveraged to break $\mathsf{SCM\_PS}$ (i.e. $q = 1$) does not ensure that the inverse is true - i.e. that a break of PS2 implies a break of $\mathsf{SCM\_PS}_q$ for $q > 1$. More generally, scheme $\mathsf{SCM\_PS}_q$ is secure if the following q-type generalisation of the PS2 assumption holds.

Definition 6.1. *q-PS Assumption Two (q-PS2).* Let $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ define a type-3 bilinear group, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$. For $q + 1$ random field elements $x, y_1, \ldots, y_q \xleftarrow{R} \mathbb{F}_p^*$, let $X = g^x$, $Y_i = g^{y_i}$, $\tilde{X} = \tilde{g}^x$, $\tilde{Y}_i = \tilde{g}^{y_i}$. Define oracle $\mathcal{O}(\{m_i\}_{i\in[q]})$, which takes as input a set of messages $\{m_i\}_{i\in[q]}$, such that $m_i \in \mathbb{F}_p$ for all $i \in [q]$, chooses $h \xleftarrow{R} \mathbb{G}_1^*$, and outputs a pair $P = \left(h, h^{x+\sum_{i\in[q]} m_i y_i}\right)$. Given $\left(g, \tilde{g}, \{Y_i\}_{i\in[q]}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]}\right)$, and unlimited access to oracle $\mathcal{O}$, no adversary can efficiently generate a pair $P_* = \left(h_*, h_*^{x+\sum_{i\in[q]} m_i^* y_i}\right)$, for $h_* \neq 1_{\mathbb{G}_1}$, where the set $(m_1^*, \ldots, m_q^*) \in (\mathbb{F}_p)^q$ was not previously submitted to the oracle.

By construction, scheme $\mathsf{SCM\_PS}_q$ is secure if the q-PS2 assumption holds. Following the methods employed in Theorem 6.1 below, it is straight-forward to prove that the q-PS2 assumption holds in the generic bilinear group model, except with negligible probability.

6.2 Polynomial-Based Signatures for Committed Messages

The scheme $\mathsf{SCM\_PS}$ for signatures over committed messages utilises the PS signature scheme. One may construct a variant signature scheme for committed messages by mapping the PS

signatures to polynomial-based signatures. The resulting scheme (here called $\mathsf{SCM\_Poly}$) involves the following algorithms:

- $\mathsf{SCM\_Poly.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters $\mathsf{pp} \leftarrow (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a type-3 pairing.
- $\mathsf{SCM\_Poly.Keygen}(\mathsf{pp}, q)$: On input the public parameters pp and an integer q, selects random generators $(g, \tilde{g}) \xleftarrow{R} \mathbb{G}_1^* \times \mathbb{G}_2^*$, and selects the random field elements:
$$(x, y) \xleftarrow{R} (\mathbb{F}_p^*)^2. \tag{81}$$
Computes:
$$\begin{aligned}(X, Y_1, Y_2, \ldots, Y_q) &\leftarrow (g^x, g^y, g^{y^2}, \ldots, g^{y^q}),\\ (\tilde{X}, \tilde{Y_1}, \tilde{Y_2}, \ldots, \tilde{Y_q}) &\leftarrow (\tilde{g}^x, \tilde{g}^y, \tilde{g}^{y^2}, \ldots, \tilde{g}^{y^q}),\end{aligned} \tag{82}$$
sets $\mathsf{sk} \leftarrow X$, and sets:
$$\mathsf{pk} \leftarrow (g, Y_1, \ldots, Y_q, \tilde{g}, \tilde{X}, \tilde{Y_1}, \ldots, \tilde{Y_q}). \tag{83}$$
- $\mathsf{SCM\_Poly.Protocol}$: To obtain a signature on messages $\{m_i\}_{i \in [q]}$, a user selects a random blinding factor $b \xleftarrow{R} \mathbb{F}_p$, computes the commitment $C \leftarrow g^b \prod_{i \in [q]} Y_i^{m_i}$, and sends it to the signer. The signer and user engage in a proof of knowledge protocol for the commitment openings. If the signer is convinced by the user, the signer selects $u \xleftarrow{R} \mathbb{F}_p^*$ and returns $\sigma' \leftarrow (g^u, (XC)^u)$. The user may unblind the signature as:
$$\sigma \;\leftarrow\; (\sigma_1', \sigma_2'(\sigma_1')^{-b}) \;=\; (g^u, (XCg^{-b})^u) \;=\; (g^u, g^{u(x + \sum_{i \in [q]} m_i y^i)}), \tag{84}$$
which is a valid $\mathsf{PolySig}$ signature for the messages $\{m_i\}_{i \in [q]}$.

Following the preceding section, to convince a verifier that the user's attributes were certified, the user and verifier may engage in a zero-knowledge proof of knowledge. Specifically, the user aggregates a signature on messages $\{m_i\}_{i \in [q]}$ with a random dummy message $r \xleftarrow{R} \mathbb{F}_p^*$. The resulting signature is valid for the set $\{m_i\}_{i \in [q]} \cup \{r\}$ and does not reveal information about the messages $\{m_i\}_{i \in [q]}$. More precisely, the user randomly selects $t, r \xleftarrow{R} \mathbb{F}_p^*$, computes
$$\sigma' \;\leftarrow\; (\sigma_1^t, \sigma_2^t \cdot \sigma_1^{rt}) \;=\; (g^{ut}, g^{ut(x + r + \sum_{i \in [q]} m_i y^i)}), \tag{85}$$
and sends σ' to the verifier. The user constructs a zero-knowledge proof of knowledge π for $\{m_i\}_{i \in [q]} \cup \{r\}$, such that:
$$e(\sigma_1', \tilde{g}^r \cdot \tilde{X} \cdot \prod_{i \in [q]} \tilde{Y}_i^{m_i}) \;=\; e(\sigma_2', \tilde{g}). \tag{86}$$
If the verifier is convinced by the proof of knowledge π, they accept the credential as legitimate. This scheme will be leveraged to construct an unlinkable anonymous credential scheme below.

Security of $\mathsf{SCM\_Poly}_q$ is defined relative to an alternative q-type generalisation of the PS2 assumption.

Definition 6.2. *q-PolyS Assumption Two (q-PolyS2).* Let $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ define a type-3 bilinear group, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$. For random field elements $x, y \xleftarrow{R} \mathbb{F}_p^*$, let $X = g^x$, $Y_i = g^{y^i}$, $\tilde{X} = \tilde{g}^x$, $\tilde{Y}_i = \tilde{g}^{y^i}$. Define oracle $\mathcal{O}(\{m_i\}_{i\in[q]})$, which takes as input a set of messages $\{m_i\}_{i\in[q]}$, such that $m_i \in \mathbb{F}_p$ for all $i \in [q]$, chooses $h \xleftarrow{R} \mathbb{G}_1^*$, and outputs a pair $P = \left(h, h^{x+\sum_{i\in[q]} m_i y^i}\right)$. Given $\left(g, \tilde{g}, \{Y_i\}_{i\in[q]}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]}\right)$, and unlimited access to oracle $\mathcal{O}$, no adversary can efficiently generate a pair $P_* = \left(h_*, h_*^{x+\sum_{i\in[q]} m_i^* y^i}\right)$, for $h_* \neq 1_{\mathbb{G}_1}$, where $(m_1^*, \ldots, m_q^*) \in (\mathbb{F}_p)^q$ was not previously submitted to the oracle.

The q-PolyS2 assumption implies the q-PolyS assumption. It is straightforward to show that an adversary can only construct a valid forgery if they can break the q-PolyS2 assumption. The following theorem shows that the q-PolyS2 assumption holds in the generic bilinear group model.

Theorem 6.1. *The q-PolyS2 assumption holds in the generic bilinear group model.*

Proof. Let the PPT adversary $\mathcal{A}$ be provided with q-PolyS2 public parameters pp, corresponding public key $\mathsf{pk}_* = (g, \tilde{g}, \{Y_i\}_{i\in[q]}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[q]})$, a signing oracle $\mathcal{O}_{\mathsf{PolySig}}$, and a generic bilinear group oracle $\mathcal{O}_{\mathbb{G}}$. The signing oracle outputs a $\mathsf{PolySig}$ signature $\sigma_\kappa = (\sigma_{1,\kappa}, \sigma_{2,\kappa})$, on input messages $\{m_{i,\kappa}\}_{i\in[q]}$, where $\kappa \in [Q_O]$ labels the Q_O signing oracle queries. To generate new group elements in the generic bilinear group model, the adversary uses generic group operations such as group multiplications and pairing operations. Hence, there exist scalars

$$\begin{aligned} &(\mu, \{\nu_i\}_{i\in[q]}, \{\gamma_\kappa\}_{\kappa\in[Q_O]}, \{\delta_\kappa\}_{\kappa\in[Q_O]}), \\ &(\mu', \{\nu'_i\}_{i\in[q]}, \{\gamma'_\kappa\}_{\kappa\in[Q_O]}, \{\delta'_\kappa\}_{\kappa\in[Q_O]}), \end{aligned} \tag{87}$$

such that the elements in the adversary's forgery (σ_1^*, σ_2^*) may be written as:

$$\begin{aligned} \sigma_1^* &= g^{\mu} \cdot \prod_{i\in[q]} Y_i^{\nu_i} \cdot \prod_{\kappa\in[Q_O]} \sigma_{1,\kappa}^{\gamma_\kappa} \cdot \prod_{\kappa\in[Q_O]} \sigma_{2,\kappa}^{\delta_\kappa}, \\ \sigma_2^* &= g^{\mu'} \cdot \prod_{i\in[q]} Y_i^{\nu'_i} \cdot \prod_{\kappa\in[Q_O]} \sigma_{1,\kappa}^{\gamma'_\kappa} \cdot \prod_{\kappa\in[Q_O]} \sigma_{2,\kappa}^{\delta'_\kappa}. \end{aligned} \tag{88}$$

Writing $\sigma_{1,\kappa} = g^{v_\kappa}$, for random element v_κ drawn during signature construction, the adversary's forgery utilises polynomials (in the exponent) in the unknown field elements x, y, and $\{v_\kappa\}_{\kappa\in[Q_O]}$, namely:

$$\begin{aligned} \sigma_1^* &= g^{\mu} \cdot \prod_{i\in[q]} g^{\nu_i y^i} \cdot \prod_{\kappa\in[Q_O]} g^{v_\kappa \gamma_\kappa} \cdot \prod_{\kappa\in[Q_O]} g^{v_\kappa \delta_\kappa (x+\sum_{i\in[q]} m_{i,\kappa} y^i)}, \\ \sigma_2^* &= g^{\mu'} \cdot \prod_{i\in[q]} g^{\nu'_i y^i} \cdot \prod_{\kappa\in[Q_O]} g^{v_\kappa \gamma'_\kappa} \cdot \prod_{\kappa\in[Q_O]} g^{v_\kappa \delta'_\kappa (x+\sum_{i\in[q]} m_{i,\kappa} y^i)}. \end{aligned} \tag{89}$$

If the forgery is valid, it must satisfy the verification check in Eq. (9), giving:

$$\begin{aligned}\left(\mu + \sum_{i\in[q]} \nu_i\, y^i + \sum_{\kappa\in[Q_O]} v_\kappa \gamma_\kappa + \sum_{\kappa\in[Q_O]} v_\kappa (x + \sum_{i\in[q]} m_{i,\kappa} y^i)\delta_\kappa\right)\left(x + \sum_{i\in[q]} m_i^*\, y^i\right) \\ = \mu' + \sum_{i\in[q]} \nu_i'\, y^i + \sum_{\kappa\in[Q_O]} v_\kappa \gamma_\kappa' + \sum_{\kappa\in[Q_O]} v_\kappa (x + \sum_{i\in[q]} m_{i,\kappa} y^i)\delta_\kappa'. \qquad (90)\end{aligned}$$

Inspecting the $\mathcal{O}(x^2)$ terms, the $\mathcal{O}(xy^i)$ terms, the $\mathcal{O}(v_\kappa)$ terms, the $\mathcal{O}(1)$ term, the $\mathcal{O}(x)$ terms, and the $\mathcal{O}(y^i)$ terms,in turn, shows that a valid forgery requires $\delta_\kappa = \nu_i = \gamma_\kappa' = \mu' = \mu = \nu_i' = 0$, leaving:

$$\left(\sum_{\kappa\in[Q_O]} v_\kappa \gamma_\kappa\right)\left(x + \sum_{i\in[q]} m_i^*\, y^i\right) = \sum_{\kappa\in[Q_O]} v_\kappa (x + \sum_{i\in[q]} m_{i,\kappa}\, y^i)\delta_\kappa', \qquad (91)$$

Treating this expression as a polynomial in the unknown constants v_κ, one has:

$$(x + \sum_{i\in[q]} y^i m_i^*) \times \gamma_\kappa = (x + \sum_{i\in[q]} y^i m_{i,\kappa}) \times \delta_\kappa'. \qquad (92)$$

This expression is the same as Eq. (30) in Theorem 4.5. Hence, by the same argument, it is not feasible for the adversary to explicitly construct a forgery,

Now we bound the probability that two polynomials two of the adversary's polynomials (in the exponent) accidentally collide [BBG05]. The public key contains $2q+3$ elements, all with monomial exponents of degree $\leq q$ in the secret elements x, y. A query to the signing oracle outputs two group elements, (σ_1, σ_2), with exponents of maximal degree $\deg(\sigma_1) \leq 1$, and $\deg(\sigma_2) \leq (q+1)$, in the secret elements and the random elements v_κ (drawn during signature generation). The largest-degree polynomial returned by a call to the bilinear group oracle occurs for a pairing call. The maximal degree of a polynomial in the exponent of a $\mathbb{G}_2$ element is q, and for a $\mathbb{G}_1$ element it's $(q+1)$. Thus, the maximal degree of the exponent for an element outputted by the group oracle is $\leq q(q+1)$. An adversary that makes Q_O signing oracle queries and $Q_\mathbb{G}$ group oracle queries generates $(2Q_O + Q_\mathbb{G} + 2q + 1)$ polynomials in total, all with degree $\leq q(q+1)$. By the Schwartz-Zippel lemma [DL78, Zip79, Sch80], the probability of collision among these polynomials is at most:

$$\frac{q(q+1)}{2p}(2Q_O + Q_\mathbb{G} + 2q + 3)(2Q_O + Q_\mathbb{G} + 2q + 2), \qquad (93)$$

which is negligible in security parameter $\lambda = \log_2(p)$, for a PPT adversary, with $q \leq \mathsf{poly}(\lambda)$. Thus, the q-PolyS2 assumption holds in the generic bilinear group model, except with negligible probability. ■

Finally note that the mapping between PS_q and $\mathsf{PolySig}_q$ induces a mapping between schemes $\mathsf{SCM\_PS}_q$ and $\mathsf{SCM\_Poly}_q$. Moreover, one has:

$$\mathsf{SCM\_PS}_{q=1} = \mathsf{SCM\_Poly}_{q=1}, \qquad (94)$$

so protocols $\mathsf{SCM\_PS}_{q>1}$ and $\mathsf{SCM\_Poly}_{q>1}$ may be regarded as distinct multi-message generalisations of a common single-message commitment signature scheme:

$$\mathsf{SCM}_{q=1} \;=\; \lim_{q\to 1} \mathsf{SCM\_PS}_q \;=\; \lim_{q\to 1} \mathsf{SCM\_Poly}_q. \tag{95}$$

7 Redactable Signatures

The multi-message signature schemes described in the preceding sections required users to treat all messages "democratically." That is, users seeking to disclose one or more messages during verification were forced to disclose *all signed messages*. Similarly, the signature schemes for committed messages required users to hide all the messages. However, in some use cases individuals may prefer to *selectively disclose* a subset of signed messages, thereby revealing appropriate information without suffering undue additional disclosures. Signature protocols with this functionality are known as *redactable signature schemes* [BBD$^+$10, HHH$^+$08, NTKK09]. Sanders proposed a generalisation of PS signatures which permits users to selectively disclose a subset of signed messages [San19]. The resulting redactable signature scheme can serve as a building block for both unlinkable redactable signature schemes and redactable anonymous credential systems. Redactable signature schemes utilise an additional group element called an *accumulator* [San19]. The user hides non-disclosed messages inside the accumulator and sends both the accumulator and the signature to the verifier. Verification of the signature is implemented without requiring the user to reveal the contents of the accumulator. However, merely introducing an accumulator is insufficient to obtain a secure redactable signature scheme; in the simplest implementations, an adversary may hide malicious elements in the accumulator such that signatures on invalid messages may nonetheless verify [San19]. Introducing an additional signature element (here referred to as an *enforcer*), one may incorporate an extra verification check to ensure the accumulator has the appropriate structure and prevent adversaries from hiding malicious information [San19]. In essence, the second verification check uses the enforcer to ensure the accumulator was constructed correctly.

Sanders' redactable signature scheme leverages PS signatures. Accordingly, one may use the mapping between PS-based signatures and polynomial-based signatures to realise a new redactable signature scheme. In what follows, the PS-based redactable signature scheme is described and subsequently generalised to a polynomial-based redactable signature scheme (in Section 7.2). Note that Ref. [San19] swapped the role of $\mathbb{G}_1$ and $\mathbb{G}_2$, relative to PS [PS16], such that signatures were constructed in $\mathbb{G}_2$ rather than $\mathbb{G}_1$. To facilitate comparisons, the same convention is adopted in what follows.

7.1 Redactable PS Signatures

The redactable PS signature scheme (here labelled $\mathsf{RS\_PS}$) allows users to obtain a signature on a set of messages and selectively disclose *only a subset* of the signed messages during verification. Users may disclose different subsets of messages to distinct verifiers. Scheme

$\mathsf{RS\_PS}$ is defined by the following algorithms [San19]:

- $\mathsf{RS\_PS.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a type-3 pairing.

- $\mathsf{RS\_PS.Keygen}(\mathsf{pp}, q)$: On input the public parameters pp and an integer q, randomly selects group generators $(g, \tilde{g}) \overset{R}{\leftarrow} \mathbb{G}_1^* \times \mathbb{G}_2^*$, and $q + 1$ field elements:
$$(x, y_1, \ldots, y_q) \overset{R}{\leftarrow} (\mathbb{F}_p^*)^{q+1}. \tag{96}$$
Computes the following group elements:
$$\Big(X, \{Y_i\}, \{\tilde{Y}_i\}, \{W_{i,j}\}_{i \neq j}\Big) = \left(g^x, \{g^{y_i}\}, \{\tilde{g}^{y_i}\}, \{g^{y_i y_j}\}_{i \neq j}\right), \quad \text{for} \quad i, j \in [q], \tag{97}$$
and sets:
$$\begin{aligned} \mathsf{sk} &= (x, y_1, \ldots, y_q), \\ \mathsf{pk} &= (g, \tilde{g}, X, \{Y_i\}, \{\tilde{Y}_i\}, \{W_{i,j}\}_{i \neq j}). \end{aligned} \tag{98}$$

- $\mathsf{RS\_PS.Sign}(\mathsf{sk}, \{m_i\}_{i \in [q]})$: Checks that $m_i \neq 0$, $\forall i$, randomly selects a group generator $\tilde{\sigma}_1 \overset{R}{\leftarrow} \mathbb{G}_2^*$, computes $\tilde{\sigma}_2 \leftarrow \tilde{\sigma}_1^{x + \sum_i y_i m_i}$ and $\sigma_2 = g^{\sum_i y_i}$, sets $\sigma_1 = 1_{\mathbb{G}_1}$, and outputs the signature $\sigma = (\sigma_1, \sigma_2, \tilde{\sigma}_1, \tilde{\sigma}_2)$ for messages $\{m_i\}$.

- $\mathsf{RS\_PS.Derive}(\mathsf{pk}, \sigma, \{m_i\}_{i \in [q]}, \mathcal{I})$: On input a subset $\mathcal{I} \subseteq \{1, \ldots, q\}$, and signature σ on messages $\{m_i\}_{i \in [q]}$ under public key pk, parses σ as $(\sigma_1, \sigma_2, \tilde{\sigma}_1, \tilde{\sigma}_2)$, and checks whether $\bar{\mathcal{I}} = \emptyset$, where $\bar{\mathcal{I}}$ is the compliment of $\mathcal{I}$ (namely $\bar{\mathcal{I}} = \{1, \ldots, q\} \backslash \mathcal{I}$). If $\bar{\mathcal{I}} \neq \emptyset$, generates the accumulator $\sigma_1 \leftarrow \prod_{j \in \bar{\mathcal{I}}} Y_j^{m_j}$, and enforcer:
$$\sigma_2 \leftarrow \prod_{i \in \mathcal{I}} \prod_{j \in \bar{\mathcal{I}}} W_{i,j}^{m_j}. \tag{99}$$
Otherwise, for $\bar{\mathcal{I}} = \emptyset$, sets $(\sigma_1, \sigma_2) = (1_{\mathbb{G}_1}, 1_{\mathbb{G}_2})$. In either case, outputs the derived signature:
$$\sigma_{\mathcal{I}} = (\sigma_1, \sigma_2, \tilde{\sigma}_1, \tilde{\sigma}_2), \tag{100}$$
for messages $\{m_i\}_{i \in \mathcal{I}}$.

- $\mathsf{RS\_PS.Verify}(\mathsf{pk}, \{m_i\}_{i \in \mathcal{I}}, \sigma)$: Parses σ as $(\sigma_1, \sigma_2, \tilde{\sigma}_1, \tilde{\sigma}_2)$, checks that $m_i \neq 0$, $\forall i \in \mathcal{I}$, and executes the checks:
$$\begin{aligned} &1. \quad e(X \cdot \sigma_1 \cdot \prod_{i \in \mathcal{I}} Y_i^{m_i}, \tilde{\sigma}_1) = e(g, \tilde{\sigma}_2) \\ &2. \quad e(\sigma_1, \prod_{i \in \mathcal{I}} \tilde{Y}_i) = e(\sigma_2, \tilde{g}). \end{aligned} \tag{101}$$
If both checks pass, outputs accept, otherwise reject.

Algorithm $\mathsf{RS\_PS.Sign}$ generates an aggregate signature on the full set of messages $\{m_i\}_{i\in[q]}$, whereas $\mathsf{RS\_PS.Derive}$ takes an aggregate signature σ as input and outputs a derived signature $\sigma_{\mathcal{I}}$, for a subset of messages labelled by $\mathcal{I} \subseteq [q]$. Users provide the verifier with an aggregate signature $\tilde{\sigma}_2 \in \mathbb{G}_2$ for a full set of messages $\{m_i\}_{i\in[q]}$, but only reveal a subset of messages $\{m_i\}_{i\in\mathcal{I}}$, whilst hiding the remaining messages $\{m_j\}_{i\in\bar{\mathcal{I}}}$ inside the accumulator σ_1. The first verification check tests whether the aggregate signature is a valid signature for the full set of messages (i.e. both the revealed and redacted messages, $\{m_i\}_{i\in\mathcal{I}}$ and $\{m_j\}_{i\in\bar{\mathcal{I}}}$, respectively). However, this check does not preclude forgeries as an adversary may hide elements inside the accumulator σ_1 and yet construct a forgery that verifies. The second check in Eq. (101) tests whether σ_1 has the correct form to preclude such forgeries. Sanders' key insight was that, if σ_1 has the correct form, the exponent of the target group element $e(\sigma_1, \prod_{i\in\mathcal{I}} \tilde{Y}_i)$ will not contain any quadratic powers of the form y_i^2, for some $i \in [q]$. Conversely if σ_1 is a verifying forgery, the exponent of element $e(\sigma_1, \prod_{i\in\mathcal{I}} \tilde{Y}_i)$ necessarily contains one or more quadratic terms y_i^2 [San19]. The second check requires users to construct the enforcer element σ_2, using the public elements $W_{i,j}$, which contain no quadratic powers of the secret elements y_i in the exponent. Consequently the second check only passes if the accumulator σ_1 has the correct form. Note, however, that the accumulator check permits additional redundancies in the accumulator. These redundancies are not considered forgeries as they may be leveraged to blind the signature and achieve unlinkability [San19].

Unforgeability of $\mathsf{RS\_PS}$ was proven by demonstrating that an adversary cannot construct a verifying forgery on a previously unsigned message in the generic bilinear group model, except with negligible probability [San19]. This proof strategy is equivalent to defining the following q-PSR assumption and demonstrating that this assumption holds in the generic bilinear group model, except with negligible probability.

Definition 7.1. *q-PS Redactable (q-PSR) Assumption.* Let $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ define a type-3 bilinear group, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$. For random field elements $x, y_1, \ldots, y_q \xleftarrow{R} \mathbb{F}_p^*$, let $X = g^x$, $Y_i = g^{y_i}$, $\tilde{Y}_i = \tilde{g}^{y_i}$, $W_{i,j} = g^{y_i y_j}$, for $i, j \in [q]$ and $i \neq j$. Define oracle $\mathcal{O}_{\mathsf{RS\_PS}}(\{m_i\}_{i\in[q]})$, which takes as input messages $m_i \in \mathbb{F}_p^*$, and outputs an $\mathsf{RS\_PS}$ signature. Given $(g, \tilde{g}, X, \{Y_i\}, \{\tilde{Y}_i\}, \{W_{i,j}\}_{i\neq j})$, and unlimited access to oracle $\mathcal{O}_{\mathsf{RS\_PS}}$, no adversary can efficiently generate a valid $\mathsf{RS\_PS}$ signature for a message set $\{m_i^*\}_{i\in\mathcal{I}}$, for some $\mathcal{I} \subseteq [q]$, such that at least one of the messages $m_j^* \in \{m_i^*\}_{i\in\mathcal{I}}$ was not previously submitted to the oracle.

The security proof for protocol $\mathsf{RS\_PS}$ is not reproduced here [San19]. Below, in Section 7.3, an equivalence between the (related) q-PolyR assumption and the EUF-CMA RS-Poly security game is proven. An analogous proof can be constructed for $\mathsf{RS\_PS}$, permitting one to leverage Sanders' proof of EUF-CMA security for protocol $\mathsf{RS\_PS}$ [San19], to demonstrate that the q-PSR assumption holds in the generic bilinear group model, except with negligible probability (i.e. security of $\mathsf{RS\_PS}$ may equivalently be defined using the q-PSR assumption). The proof is omitted here but is a straight-forward variant of results in Section 7.3.

7.2 Polynomial-Based Redactable Signatures

The scheme $\mathsf{RS\_PS}$ derives from the multi-message signature scheme PS. Specifically, the signature generated by $\mathsf{RS\_PS.Sign}$, $\tilde{\sigma}_2 \leftarrow \tilde{\sigma}_1^{x+\sum_i y_i m_i}$, has the same form as a signature generated by $\mathsf{PS.Sign}$. Leveraging the interchange relationship between PS and $\mathsf{PolySig}$, a variant redactable signature scheme based on $\mathsf{PolySig}$ may be constructed. This *polynomial-based redactable signature scheme* (i.e. $\mathsf{RS\_Poly}$) is defined by the following algorithms:

- $\mathsf{RS\_Poly.Setup}(1^\lambda)$: On input a security parameter λ, outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a type-3 pairing.

- $\mathsf{RS\_Poly.Keygen}(\mathsf{pp}, q)$: On input the public parameters pp and an integer q, randomly selects the group generators $(g, \tilde{g}) \overset{R}{\leftarrow} \mathbb{G}_1^* \times \mathbb{G}_2^*$, and three field elements, $(x, y, z) \overset{R}{\leftarrow} (\mathbb{F}_p^*)^3$. Computes the following group elements:

$$(X,\ \{Y_i\},\ \{\tilde{Z}_i\},\ \{\tilde{W}_{i,j}\}_{i\neq j}) = (g^x,\ \{g^{y^i}\},\ \{\tilde{g}^{z^i}\},\ \{\tilde{g}^{z^i y^j}\}_{i\neq j}), \quad \text{for} \quad i, j \in [q], \tag{102}$$

and sets

$$\begin{aligned} \mathsf{sk} &= (x, y), \\ \mathsf{pk} &= (g, \tilde{g}, X, \{Y_i\}, \{\tilde{Z}_i\}, \{\tilde{W}_{i,j}\}_{i\neq j}). \end{aligned} \tag{103}$$

- $\mathsf{RS\_Poly.Sign}(\mathsf{sk}, \{m_i\}_{i\in[q]})$: Checks that $m_i \neq 0$, $\forall i$, randomly selects a group generator $\tilde{\sigma}_1 \overset{R}{\leftarrow} \mathbb{G}_2^*$, computes $\tilde{\sigma}_2 \leftarrow \tilde{\sigma}_1^{x+\sum_i m_i y^i}$ and $\tilde{\sigma}_0 = \tilde{g}^{\sum_i z^i}$, sets $\sigma_1 = 1_{\mathbb{G}_1}$, and outputs the aggregated signature $\sigma = (\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$ for messages $\{m_i\}$.

- $\mathsf{RS\_Poly.Derive}(\mathsf{pk}, \sigma, \{m_i\}_{i\in[q]}, \mathcal{I})$: On input a subset $\mathcal{I} \subseteq \{1, \ldots, q\}$, and an aggregate signature σ on messages $\{m_i\}_{i\in[q]}$ under public key pk, parses σ as $(\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$, and checks whether $\bar{\mathcal{I}} = \emptyset$, where $\bar{\mathcal{I}}$ is the compliment of $\mathcal{I}$ (namely $\bar{\mathcal{I}} = \{1, \ldots, q\}\backslash\mathcal{I}$). If $\bar{\mathcal{I}} \neq \emptyset$, generates the accumulator $\sigma_1 \leftarrow \prod_{j\in\bar{\mathcal{I}}} Y_j^{m_j}$, and the enforcer:

$$\tilde{\sigma}_0 \leftarrow \prod_{i\in\mathcal{I}} \prod_{j\in\bar{\mathcal{I}}} \tilde{W}_{i,j}^{m_j}. \tag{104}$$

  Otherwise, if $\bar{\mathcal{I}} = \emptyset$, sets $(\sigma_1, \tilde{\sigma}_0) = (1_{\mathbb{G}_1}, 1_{\mathbb{G}_2})$. In either case, outputs the derived signature:

$$\sigma_{\mathcal{I}} = (\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2), \tag{105}$$

  for messages $\{m_i\}_{i\in\mathcal{I}}$.

- $\mathsf{RS\_Poly.Verify}(\mathsf{pk}, \{m_i\}_{i\in\mathcal{I}}, \sigma)$: Parses σ as $(\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$, checks that $m_i \neq 0$, $\forall i \in \mathcal{I}$, and executes the checks:

$$\begin{aligned} &1.\quad e(X \cdot \sigma_1 \cdot \prod_{i\in\mathcal{I}} Y_i^{m_i}, \tilde{\sigma}_1) = e(g, \tilde{\sigma}_2) \\ &2.\quad e(\sigma_1, \prod_{i\in\mathcal{I}} \tilde{Z}_i) = e(g, \tilde{\sigma}_0). \end{aligned} \tag{106}$$

  If both checks are satisfied, outputs accept, otherwise reject.

Scheme $\mathsf{RS\_Poly}$ only uses three random field elements (namely x, y, z), compared to the $q+1$ field elements required in $\mathsf{RS\_PS}$. Secret key sk is therefore shorter (constant-sized), irrespective of the number of messages q, while the public key pk has the same size for both $\mathsf{RS\_Poly}$ and $\mathsf{RS\_PS}$.

The intuition for the new scheme $\mathsf{RS\_Poly}$ is as follows. Algorithm $\mathsf{RS\_PS}$ leverages the fact that the discrete log of $e\left(\sigma_1, \prod_{i\in\mathcal{I}} \tilde{g}^{y_i}\right)$ contains $\mathcal{O}(y_i^2)$ terms, for some $i \in \mathcal{I}$, if the accumulator σ_1 hides a forgery. For a polynomial-based scheme, the obvious generalisation would include elements $\tilde{Y}_i \leftarrow \tilde{g}^{y^i}$ in the public key, and check whether the exponent of the target group element

$$e\left(\sigma_1, \prod_{i\in\mathcal{I}} \tilde{Y}_i\right) = e\left(\sigma_1, \prod_{i\in\mathcal{I}} \tilde{g}^{y^i}\right), \tag{107}$$

contained $\mathcal{O}(y^{i+i})$ terms, for some $i \in \mathcal{I}$. However, for a polynomial-based signature scheme, a valid accumulator may give rise to an $\mathcal{O}(y^{i+i})$ term in Eq. (107) for some $i \in [q]$. For example, the set $\{y^i\}_{i\in[q]}$ may contain elements y^i and $y^{i'}$ such that $2i = i'$. Similarly, the product $y^i y^j$, for $i \in \mathcal{I}$ and $j \in \bar{\mathcal{I}}$, may contain elements such that $i + j = j' \in \bar{\mathcal{I}}$ for some $j' \neq j$. Consequently this straightforward generalisation of $\mathsf{RS\_PS}$ is inadequate. Instead, the public elements $\tilde{Y}_i$ in $\mathsf{RS\_PS}$ were replaced by the elements $\tilde{Z}_i \leftarrow \tilde{g}^{z^i}$, for a random field element z. In the resulting scheme $\mathsf{RS\_Poly}$, the discrete log of $e\left(\sigma_1, \prod_{i\in\mathcal{I}} \tilde{Z}i\right)$ only contains $\mathcal{O}(y^i z^i)$ terms if the accumulator σ_1 involves a forgery. Including the elements $\{\tilde{W}_{i,j}\}_{i,\neq j} \leftarrow \{\tilde{g}^{z^i y^j}\}_{i\neq j}$ in the public key, and requiring users to construct the enforcer $\tilde{\sigma}_0 \leftarrow \prod_{i\in\mathcal{I}} \prod_{j\in\bar{\mathcal{I}}} \tilde{W}_{i,j}^{m_j}$, allows the verifier to confirm that σ_1 has the correct form and does not hide a forgery, using the second test in Eq. (106). The new scheme $\mathsf{RS\_Poly}$ therefore achieves the same redactable signature functionality as $\mathsf{RS\_PS}$ but with a constant-sized secret key.

Notice that the enforcer element $\sigma_2 \in \mathbb{G}_1$ in $\mathsf{RS\_PS}$ was replaced with an enforcer $\tilde{\sigma}_0 \in \mathbb{G}_2$ in $\mathsf{RS\_Poly}$, and the elements $\tilde{W}_{i,j}$ now belong to $\mathbb{G}_2$. These changes reflect the fact that the elements $\tilde{Z}_i$ and Y_i in the public key $\mathsf{pk}_{\mathsf{RS\_Poly}}$ have distinct exponents (z^i versus y^i), differing from the relationship between elements $\tilde{Y}_i$ and Y_i in $\mathsf{RS\_PS}$ (which have the same exponent, y_i). It is possible to modify $\mathsf{RS\_Poly}$ to implement a redactable polynomial-based signature scheme with an enforcer element $\sigma_2 \in \mathbb{G}_1$, analogous to $\mathsf{RS\_PS}$, by replacing $\tilde{W}_{i,j} \in \mathbb{G}_2$ with elements $W_{i,j} \in \mathbb{G}_1$. However, when considering the generalisation to unlinkable signatures below, shifting the enforcer σ_2 into the group $\mathbb{G}_2$ (as done in $\mathsf{RS\_Poly}$ above) allows one to retain a public key with the same size for both the PS and the polynomial-based unlinkable redactable signature schemes. Also note that verification does not explicitly use the elements $\{\tilde{W}_{i,j}\}_{i\neq j}$. Thus, the full public key is not required for verification since one may define a verification key $\mathsf{vk} = (g, X, \{Y_i\}, \{\tilde{Z}_i\})$, which contains $2q + 2$ group elements.

7.3 Security of the Poly-Based Redactable Signature Scheme

The following definition generalises the q-PSR assumption underlying the security of scheme $\mathsf{RS\_PS}$.

Definition 7.2. *q-PolyS Redactable (q-PolyR) Assumption.* Let $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ define a type-3 bilinear group, with generators $(g, \tilde{g}) \in \mathbb{G}_1^* \times \mathbb{G}_2^*$. For random field elements $x, y, z \xleftarrow{R} \mathbb{F}_p^*$, let $X = g^x$, $Y_i = g^{y^i}$, $\tilde{Z}_i = \tilde{g}^{z^i}$, $\tilde{W}_{i,j} = \tilde{g}^{z^i y^j}$, for $i, j \in [q]$ and $i \neq j$. Define oracle $\mathcal{O}_{\mathsf{RS\_Poly}}(\{m_i\}_{i\in[q]})$, which takes as input messages $m_i \in \mathbb{F}_p^*$, and outputs an $\mathsf{RS\_Poly}$ signature. Given $(g, \tilde{g}, X, \{Y_i\}_{i\in[q]}, \{\tilde{Z}_i\}_{i\in[q]}, \{\tilde{W}_{i,j}\}_{i,j\in[q];\, i\neq j})$, and unlimited access to oracle $\mathcal{O}_{\mathsf{RS\_Poly}}$, no adversary can efficiently generate a valid $\mathsf{RS\_Poly}$ signature for a message set $\{m_i^*\}_{i\in\mathcal{I}}$, for some $\mathcal{I} \subseteq [q]$, such that at least one of the messages $m_j^* \in \{m_i^*\}_{i\in\mathcal{I}}$ was not previously submitted to the oracle.

The security proof for $\mathsf{RS\_Poly}$ proceeds in three steps. First, an unforgeability security game is defined and the equivalence between the game-based security definition and an assumption-based security definition is proven. Next, it is shown that the q-PolyR assumption holds in the generic bilinear group model. Finally, these ingredients are leveraged to construct the desired proof.

- <u>EUF-CMA RS-Poly Security Game</u>
  - Challenger $\mathcal{C}$ initialises an empty message table $T_M = \emptyset$, and executes algorithms $\mathsf{RS\_Poly.Setup}$ and $\mathsf{RS\_Poly.Keygen}$, generating the keys
  $$\begin{aligned} \mathsf{sk}_* &= (x, y) \\ \mathsf{pk}_* &= (g, \tilde{g}, X, \{Y_i\}_{i\in[q]}, \{\tilde{Z}_i\}_{i\in[q]}, \{\tilde{W}_{i,j}\}_{i,j\in[q];\, i\neq j}). \end{aligned} \tag{108}$$
  The challenger shares the public key pk_* with the PPT adversary $\mathcal{A}_{\mathsf{RS\_Poly}}$, and provides access to a signing oracle.
  - When the adversary requests a signature on a set of messages $\{m_{i,\kappa}\}_{i\in[q]}$, with $\kappa \in [Q_O]$ labeling the oracle queries, the oracle outputs an $\mathsf{RS\_Poly}$ signature $\sigma_{(\kappa)}$, and adds the elements $\{m_{i,\kappa}\}_{i\in[q]}$ to T_M.
  - Eventually the adversary outputs a signature σ_* and messages $\{m_i^*\}_{i\in\mathcal{I}}$. The adversary wins the game if (i) $\mathcal{I} \neq \emptyset$; (ii) the signature σ_* is a valid $\mathsf{RS\_Poly}$ signature for messages $\{m_i^*\}_{i\in\mathcal{I}}$ under public key pk_*; and (iii) there exists $m_j^* \in \{m_i^*\}_{i\in[q]}$, such that $m_j^* \notin T_M$.

Theorem 7.1. *A PPT adversary can win the EUF-CMA RS-Poly security game with non-negligible probability if and only if they can break the q-PolyR assumption with non-negligible probability.*

Proof. Let $\mathcal{A}_{\mathrm{UF}}$ be an adversary in the EUF-CMA RS-Poly security game, and $\mathcal{A}_{q\mathrm{PolyR}}$ an adversary against the q-PolyR assumption challenger. Assume that $\mathcal{A}_{\mathrm{UF}}$ can win the EUF-CMA RS-Poly security game with non-negligible probability. Upon receipt of a q-PolyR challenge public key:

$$\mathsf{pk}_* = (g, \tilde{g}, X, \{Y_i\}_{i\in[q]}, \{\tilde{Z}_i\}_{i\in[q]}, \{\tilde{W}_{i,j}\}_{i,j\in[q];\, i\neq j}), \tag{109}$$

adversary $\mathcal{A}_{q\mathrm{PolyR}}$ passes pk_* to $\mathcal{A}_{\mathrm{UF}}$, and responds to $\mathcal{A}_{\mathrm{UF}}$'s oracle queries using the q-PolyR oracle. With non-negligible probability, $\mathcal{A}_{\mathrm{UF}}$ eventually outputs a verifying $\mathsf{RS\_Poly}$ signature for a set of messages, winning the EUF-CMA RS-Poly security game. Adversary $\mathcal{A}_{q\text{PolyR}}$ passes this output to the q-PolyR challenger, successfully breaking the q-PolyR assumption with the same non-negligible probability.

Conversely, assume that $\mathcal{A}_{q\text{PolyR}}$ can break the q-PolyR assumption with non-negligible probability. Upon receipt of the EUF-CMA RS-Poly security game public key pk_*, adversary $\mathcal{A}_{\text{UF}}$ acts as a q-PolyR challenger to $\mathcal{A}_{q\text{PolyR}}$ and passes the public key to $\mathcal{A}_{q\text{PolyR}}$. Adversary $\mathcal{A}_{\text{UF}}$ answers $\mathcal{A}_{q\text{PolyR}}$'s oracle queries using the EUF-CMA RS-Poly security game oracle. With non-negligible probability, $\mathcal{A}_{q\text{PolyR}}$ eventually outputs a verifying $\mathsf{RS\_Poly}$ signature for a set of messages, breaking the q-PolyR assumption. Adversary $\mathcal{A}_{\text{UF}}$ passes this output to the EUF-CMA RS-Poly security game challenger, and successfully wins the game with the same non-negligible probability.

$\blacksquare$

Theorem 7.2. *The q-PolyR assumption holds in the generic bilinear group model.*

Proof. Let the PPT adversary $\mathcal{A}$ have access to the q-PolyR public parameters pp, public key:

$$(g, \tilde{g}, X, \{Y_i\}_{i\in[q]}, \{\tilde{Z}_i\}_{i\in[q]}, \{\tilde{W}_{i,j}\}_{i,j\in[q];\, i\neq j}), \tag{110}$$

signing oracle $\mathcal{O}_{\mathsf{RS\_Poly}}$, and a generic bilinear group oracle $\mathcal{O}_{\mathbb{G}}$. The signing oracle outputs an $\mathsf{RS\_Poly}$ signature $\sigma_\kappa = (\sigma_{1,\kappa}, \tilde{\sigma}_{0,\kappa}, \tilde{\sigma}_{1,\kappa}, \tilde{\sigma}_{0,\kappa})$ on input messages $\{m_{i,\kappa}\}_{i\in[q]}$, where $\kappa \in [Q_O]$ labels the Q_O oracle queries. To generate new group elements in the generic bilinear group model, the adversary uses generic group operations such as group multiplications and pairing operations. Hence, there exist scalars

$$\begin{aligned}
&(a, b, \{c_i\}_{i\in[q]}),\\
&(\mu, \{\nu_i\}_{i\in[q]}, \{\phi_{i,j}\}_{i,j\in[q];i\neq j}, \{\gamma_\kappa\}_{\kappa\in[Q_O]}, \{\delta_\kappa\}_{\kappa\in[Q_O]}),\\
&(\mu', \{\nu'_i\}_{i\in[q]}, \{\phi'_{i,j}\}_{i,j\in[q];i\neq j}, \{\gamma'_\kappa\}_{\kappa\in[Q_O]}, \{\delta'_\kappa\}_{\kappa\in[Q_O]}),\\
&(\mu'', \{\nu''_i\}_{i\in[q]}, \{\phi''_{i,j}\}_{i,j\in[q];i\neq j}, \{\gamma''_\kappa\}_{\kappa\in[Q_O]}, \{\delta''_\kappa\}_{\kappa\in[Q_O]}),
\end{aligned} \tag{111}$$

such that the elements in the adversary's forgery $(\sigma_1^*, \tilde{\sigma}_0^*, \tilde{\sigma}_1^*, \tilde{\sigma}_2^*)$ may be written as:

$$\begin{aligned}
\sigma_1^* &= g^a \cdot X^b \cdot \prod_{i\in[q]} Y_i^{c_i},\\
\tilde{\sigma}_1^* &= \tilde{g}^{\mu} \cdot \prod_{i\in[q]} \tilde{Z}_i^{\nu_i} \cdot \prod_{i\neq j} \tilde{W}_{i,j}^{\phi_{i,j}} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{1,\kappa}^{\gamma_\kappa} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{2,\kappa}^{\delta_\kappa},\\
\tilde{\sigma}_2^* &= \tilde{g}^{\mu'} \cdot \prod_{i\in[q]} \tilde{Z}_i^{\nu'_i} \cdot \prod_{i\neq j} \tilde{W}_{i,j}^{\phi'_{i,j}} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{1,\kappa}^{\gamma'_\kappa} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{2,\kappa}^{\delta'_\kappa},\\
\tilde{\sigma}_0^* &= \tilde{g}^{\mu''} \cdot \prod_{i\in[q]} \tilde{Z}_i^{\nu''_i} \cdot \prod_{i\neq j} \tilde{W}_{i,j}^{\phi''_{i,j}} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{1,\kappa}^{\gamma''_\kappa} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{2,\kappa}^{\delta''_\kappa}.
\end{aligned} \tag{112}$$

The elements $\sigma_{1,\kappa}$ and $\tilde{\sigma}_{0,\kappa}$, from the κ-th signing oracle query, are not explicitly included in the above expressions as they are built from publicly known parameters (i.e. they are implicitly included). If the forgery is valid, it must satisfy the verification checks. The second check gives (in the exponent):

$$\begin{aligned}
&(a + bx + \sum_{i\in[q]} c_i\, y^i)(\sum_{i\in\mathcal{I}} z^i) \\
= \; & \mu'' + \sum_{i\in[q]} z^i \nu''_i + \sum_{i\neq j} z^i y^j \phi''_{i,j} + \sum_{\kappa\in[Q_O]} v_\kappa \gamma''_\kappa + \sum_{\kappa\in[Q_O]} v_\kappa (x + \sum_{i\in[q]} m_i y^i)\delta''_\kappa, \qquad (113)
\end{aligned}$$

where the elements v_κ are unknown to the adversary, and derive from the signature element $\tilde{\sigma}_{1,\kappa} = \tilde{g}^{v_\kappa}$. Inspecting the $\mathcal{O}(1)$ term, the $\mathcal{O}(x)$ terms, the $\mathcal{O}(v_\kappa)$ terms, and the $\mathcal{O}(v_\kappa x)$ terms, in turn, shows that a valid forgery requires $\mu'' = b = \gamma''_\kappa = \delta''_\kappa = 0$, leaving

$$(a + \sum_{i\in[q]} c_i\, y^i)(\sum_{i\in\mathcal{I}} z^i) = \sum_{i\in[q]} z^i \nu''_i + \sum_{i\neq j} z^i y^j \phi''_{i,j}. \qquad (114)$$

This expression requires that $c_i = 0$ for $i \in \mathcal{I}$ (there are no $i = j$ terms on the right hand side), and $\nu''_i = 0$ for $i \in \bar{\mathcal{I}}$, with $\nu''_i = a$ for all $i \in \mathcal{I}$. Moreover, one requires that $\phi''_{i,j} = 0$ for $i \in \bar{\mathcal{I}}$ or $j \in \mathcal{I}$, and $\phi''_{i,j} = c_j$ for $j \in \bar{\mathcal{I}}$. The second verification check therefore requires:

$$\begin{aligned}
\sigma_1 &= g^a \cdot \prod_{j\in\bar{\mathcal{I}}} Y_j^{c_j}, \\
\tilde{\sigma}_0 &= \prod_{i\in\mathcal{I}} \tilde{Z}_i^a \cdot \prod_{i\in\mathcal{I},\, j\in\bar{\mathcal{I}}} \tilde{W}_{i,j}^{c_j}. \qquad (115)
\end{aligned}$$

Using this expression, the first verification check requires:

$$\begin{aligned}
&(x + a + \sum_{j\in\bar{\mathcal{I}}} y^j c_j + \sum_{i\in\mathcal{I}} y^i m^*_i) \times \\
&\left(\mu + \sum_{i\in[q]} z^i \nu_i + \sum_{i,j\in[q], i\neq j} z^i y^j \phi_{i,j} + \sum_{\kappa\in[Q_O]} \gamma_\kappa v_\kappa + \sum_{\kappa\in[Q_O]} \delta_\kappa v_\kappa (x + \sum_{i\in[q]} y^i m_{i,\kappa})\right) \\
= \; & \left(\mu' + \sum_{i\in[q]} z^i \nu'_i + \sum_{i,j\in[q], i\neq j} z^i y^j \phi'_{i,j} + \sum_{\kappa\in[Q_O]} \gamma'_\kappa v_\kappa + \sum_{\kappa\in[Q_O]} \delta'_\kappa v_\kappa (x + \sum_{i\in[q]} y^i m_{i,\kappa})\right).
\end{aligned}$$

Inspecting the $\mathcal{O}(v_\kappa x^2)$ terms, the $\mathcal{O}(x)$ term, the $\mathcal{O}(xz^i)$ terms, the $\mathcal{O}(xz^iy^j)$ terms, the $\mathcal{O}(z^i)$ terms, the $\mathcal{O}(z^iy^j)$ terms, and the $\mathcal{O}(1)$ term, in turn, shows that one requires

$$\delta_\kappa = \mu = \nu_i = \phi_{i,j} = \nu'_i = \phi'_{i,j} = \mu' = 0, \qquad (116)$$

leaving

$$(x + a + \sum_{j\in\bar{\mathcal{I}}} y^j c_j + \sum_{i\in\mathcal{I}} y^i m^*_i)(\sum_{\kappa\in[Q_O]} \gamma_\kappa v_\kappa) = \sum_{\kappa\in[Q_O]} \gamma'_\kappa v_\kappa + \sum_{\kappa\in[Q_O]} \delta'_\kappa v_\kappa (x + \sum_{i\in[q]} y^i m_{i,\kappa}).$$

Treating this expression as a polynomial in the constants v_κ, one has:

$$(x + a + \sum_{j\in\bar{\mathcal{I}}} y^j c_j + \sum_{i\in\mathcal{I}} y^i m^*_i) \times \gamma_\kappa = \gamma'_\kappa + \delta'_\kappa (x + \sum_{i\in[q]} y^i m_{i,\kappa}). \qquad (117)$$

For the adversary to produce a forgery, given each oracle query labelled by $\kappa \in [Q_O]$, there must be at least one message $m_{i,\kappa}$, for $i \in [q]$, such that $m_i^* \neq m_{i,\kappa}$. Similar to Sanders' observations for scheme $\mathsf{RS\_PS}$ [San19], a PPT adversary cannot solve the above expression in the generic group model for all κ unless $\gamma_\kappa = \gamma'_\kappa = \delta'_\kappa = 0$, for all $\kappa \in [Q_O]$, which does not constitute a forgery. Hence, construction of a forgery is not possible.

Finally, we bound the probability that two polynomials in the adversary's set accidentally evaluate to the same value [BBG05]. The public key contains $3 + q(q+3)/2$ elements, all with monomial exponents of degree $\leq (2q-1)$ in the secret elements x, y, z. A query to the signing oracle outputs four group elements, $(\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$, with exponents of maximal degree $\deg(\sigma_1) \leq q$, $\deg(\tilde{\sigma}_0) \leq (2q-1)$, $\deg(\tilde{\sigma}_1) \leq 1$, and $\deg(\tilde{\sigma}_2) \leq (q+1)$, in the secret elements and the random elements drawn during signature generation. The largest-degree polynomial returned by a call to the bilinear group oracle occurs for a pairing call. The maximal degree of a polynomial in the exponent of a $\mathbb{G}_2$ element is $(2q-1)$, and for a $\mathbb{G}_1$ element it's $(q+1)$. Thus, the maximal degree of an exponent outputted by the group oracle is $\leq (2q^2+q-1)$. An adversary that makes Q_O signing oracle queries and $Q_\mathbb{G}$ group oracle queries generates $(4Q_O + Q_\mathbb{G} + 1 + q(q+3)/2)$ polynomials in total, all with degree $\leq (2q^2+q-1)$. By the Schwartz-Zippel lemma [DL78, Zip79, Sch80], the probability of collision among these polynomials is at most:

$$\frac{(2q^2+q-1)}{2p}(4Q_O + Q_\mathbb{G} + 3 + q(q+3)/2)(4Q_O + Q_\mathbb{G} + 2 + q(q+3)/2), \tag{118}$$

which is negligible for a PPT adversary, with $q \leq \mathsf{poly}(\lambda)$. Thus, the q-PolyR assumption holds in the generic bilinear group model, except with negligible probability. ■

Theorem 7.3. *Redactable signature scheme* $\mathsf{RS\_Poly}$ *is EUF-CMA secure against PPT adversaries in the generic bilinear group model. That is, a PPT adversary has a negligible probability of winning the EUF-CMA RS-Poly security game.*

Proof. Assume that there exists a PPT adversary which can win game EUF-CMA RS-Poly with non-negligible probability in the generic bilinear group model. By Theorem 7.1, this adversary can break the q-PolyR assumption with non-negligible probability, which contradicts Theorem 7.2. Hence, no such adversary can exist in the generic bilinear group model. ■

This completes the security proofs for $\mathsf{RS\_Poly}$. Security was proven with respect to both the q-PolyR assumption and the EUF-CMA RS-Poly security game. One may similarly prove security of $\mathsf{RS\_PS}$ with respect to the q-PSR assumption (introduced above) by leveraging Sanders' game-based security proof [San19].

8 Unlinkable Redactable Signatures

Redactable signatures allow users to selectively disclose signed attributes. However, malicious verifiers (or other adversaries) can potentially link different reveals of a user's signature and extract additional information. Consequently, for some use cases, it is preferable that users can obtain multiple distinct redacted signatures for a given subset of messages, obscuring the relationship between signatures and availing a degree of unlinkability. Signatures with this property are known as *unlinkable redactable signatures* (URS) [CL01]. One may extend the redactable signature scheme $\mathsf{RS\_PS}$ to construct an unlinkable redactable signature scheme [San19]. This extension leverages two features of $\mathsf{RS\_PS}$:

- Additional group elements may be included in σ_1, without spoiling the verification checks, provided they do not have the form Y_i^a for $i \in \mathcal{I}$ and some field element a.
- Scheme $\mathsf{RS\_PS}$ uses PS signatures, which may be sequentially aggregated.

These properties allow one to sequentially aggregate a valid signature with a dummy message to create random noise, generating an unlinkable redactable signature scheme. In what follows, the PS-based URS is described and then generalised to a related polynomial-based URS.

8.1 Unlinkable Redactable PS Signatures

The PS-based URS scheme (here called $\mathsf{URS\_PS}$) is defined by the same algorithms as $\mathsf{RS\_PS}$, namely

$$\mathsf{URS\_PS}.\mathcal{X} \quad = \quad \mathsf{RS\_PS}.\mathcal{X} \qquad \text{for} \qquad \mathcal{X} \in \{\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify}\},$$

but with the following new definition for the Derive algorithm:

- $\mathsf{URS\_PS.Derive}(\mathsf{pk}, \sigma, \{m_i\}_{i\in[q]}, \mathcal{I})$: On input a subset $\mathcal{I} \subseteq \{1, \ldots, q\}$, signature σ, messages $\{m_i\}_{i\in[q]}$, and public key pk, parses σ as $(\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$ and checks that σ verifies on messages $\{m_i\}_{i\in[q]}$ under pk_*. Selects random field elements $(r, t) \xleftarrow{R} (\mathbb{F}_p^*)^2$. If $\bar{\mathcal{I}} \neq \emptyset$, generates the elements:

$$\begin{aligned} \tilde{\sigma}'_1 &\leftarrow \tilde{\sigma}_1^t \\ \tilde{\sigma}'_2 &\leftarrow \tilde{\sigma}_2^t \cdot (\tilde{\sigma}_1^t)^r \\ \sigma'_1 &\leftarrow g^r \cdot \prod_{j\in\bar{\mathcal{I}}} Y_j^{m_j} \\ \sigma'_2 &\leftarrow (\prod_{i\in\mathcal{I}} Y_i^r) \cdot \prod_{i\in\mathcal{I}} \prod_{j\in\bar{\mathcal{I}}} W_{i,j}^{m_j}. \end{aligned} \tag{119}$$

  Otherwise, if $\bar{\mathcal{I}} = \emptyset$, sets $(\sigma'_1, \sigma'_2) \leftarrow (g^t, \prod_{i\in[q]} Y_i^t)$. In either case, outputs the derived signature:

$$\sigma_{\mathcal{I}} = (\sigma'_1, \sigma'_2, \tilde{\sigma}'_1, \tilde{\sigma}'_2), \tag{120}$$

  for messages $\{m_i\}_{i\in\mathcal{I}}$.

The verification checks remain the same as in $\mathsf{RS\_PS.Verify}$, namely:

$$\begin{aligned} &1. \quad e(X \cdot \sigma_1' \cdot \prod_{i \in \mathcal{I}} Y_i^{m_i}, \tilde{\sigma}_1') = e(g, \tilde{\sigma}_2') \\ &2. \quad e(\sigma_1', \prod_{i \in \mathcal{I}} \tilde{Y}_i) = e(\sigma_2', \tilde{g}). \end{aligned} \tag{121}$$

Note that $\mathsf{URS\_PS}$ signatures have the same complexity as $\mathsf{RS\_PS}$ signatures; indeed, taking the limits $t \to 1$, and $r \to 0$ reduces the $\mathsf{URS\_PS}$ signature to an $\mathsf{RS\_PS}$ signature [San19]. Moreover, rewriting $\tilde{\sigma}_2'$ as

$$\tilde{\sigma}_2' \;\leftarrow\; \tilde{\sigma}_2^t \, (\tilde{\sigma}_1^r)^t \;=\; \lim_{\tilde{y} \to 1} \tilde{\sigma}_2^t \, (\tilde{\sigma}_1^{\tilde{y}r})^t, \tag{122}$$

suggests that $\mathsf{URS\_PS}$ employs the sequential aggregate signature methods of $\mathsf{SAS\_SM}$ but with secret key $\tilde{y}$ set to $\tilde{y} = 1$. Thus, in the present terminology, scheme $\mathsf{URS\_PS}$ is constructed by leveraging the sequential aggregate signature scheme $\mathsf{SAS\_SM}$ to inject noise into the redactable signature scheme $\mathsf{RS\_PS}$.

The unforgeability security of $\mathsf{URS\_PS}$ was proven using a tight reduction between a $\mathsf{URS\_PS}$ adversary and an $\mathsf{RS\_PS}$ adversary [San19]. The proof is not reproduced here but the proof strategy is similar to the reduction reported below for the polynomial-based scheme $\mathsf{URS\_Poly}$.

8.2 Polynomial-Based Unlinkable Redactable Signatures

A variant of scheme $\mathsf{URS\_PS}$ may be constructed by generalising the polynomial-based redactable signature scheme $\mathsf{RS\_Poly}$, instead of $\mathsf{RS\_PS}$, and adding noise sequentially using the techniques of $\mathsf{SAS\_SM}$. The resulting new URS scheme ($\mathsf{URS\_Poly}$) is defined by the algorithms:

$$\mathsf{URS\_Poly}.\mathcal{X} \;=\; \mathsf{RS\_Poly}.\mathcal{X} \qquad \text{for} \qquad \mathcal{X} \in \{\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify}\},$$

along with the following Derive algorithm:

- $\mathsf{URS\_Poly.Derive}(\mathsf{pk}, \sigma, \{m_i\}_{i \in [q]}, \mathcal{I})$: On input a subset $\mathcal{I} \subseteq \{1, \ldots, q\}$, signature σ, messages $\{m_i\}_{i \in [q]}$, and public key pk, parses σ as $(\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$ and checks that σ verifies on messages $\{m_i\}_{i \in [q]}$ under pk_*. Generates random field elements $(r, t) \overset{R}{\leftarrow} (\mathbb{F}_p^*)^2$. If $\bar{\mathcal{I}} \neq \emptyset$, generates the elements:

$$\begin{aligned} \tilde{\sigma}_1' \;&\leftarrow\; \tilde{\sigma}_1^t \\ \tilde{\sigma}_2' \;&\leftarrow\; \tilde{\sigma}_2^t \cdot (\tilde{\sigma}_1^t)^r \\ \sigma_1' \;&\leftarrow\; g^r \cdot \prod_{j \in \bar{\mathcal{I}}} Y_j^{m_j} \\ \tilde{\sigma}_0' \;&\leftarrow\; (\prod_{i \in \mathcal{I}} \tilde{Z}_i^r) \cdot \prod_{i \in \mathcal{I}} \prod_{j \in \bar{\mathcal{I}}} \tilde{W}_{i,j}^{m_j}. \end{aligned} \tag{123}$$

Otherwise, if $\bar{\mathcal{I}} = \emptyset$, sets $(\sigma_1', \tilde{\sigma}_0') \leftarrow (g^t, \prod_{i\in[q]} \tilde{Z}_i^t)$. In either case, outputs the derived signature:

$$\sigma_{\mathcal{I}} = (\sigma_1', \tilde{\sigma}_0', \tilde{\sigma}_1', \tilde{\sigma}_2'), \tag{124}$$

for messages $\{m_i\}_{i\in\mathcal{I}}$.

The signature complexity for URS_Poly is equivalent to RS_Poly and the verification checks remain as given in RS_Poly, namely:

$$\begin{aligned} &1.\quad e(X \cdot \sigma_1' \cdot \prod_{i\in\mathcal{I}} Y_i^{m_i}, \tilde{\sigma}_1') = e(g, \tilde{\sigma}_2') \\ &2.\quad e(\sigma_1', \prod_{i\in\mathcal{I}} \tilde{Z}_i) = e(g, \tilde{\sigma}_0'). \end{aligned} \tag{125}$$

Note that the limit $t \to 1$, $r \to 0$ reduces URS_Poly to RS_Poly and one may interpret the addition of noise via the element r as an implementation of the sequential signature scheme SAS_SM with secret key set to unity. Algorithm URS_Poly.Derive is identical to URS_PS.Derive, modulo the replacement $W_{i,j}^{m_j} \to \tilde{W}_{i,j}^{m_j}$ and $\tilde{Y}_i \to \tilde{Z}_i$ in the definition of $\tilde{\sigma}_0'$. Accordingly, the differences between URS_Poly and URS_PS are identical to the differences between RS_Poly and RS_PS. For example, the secret key for URS_Poly is constant-sized at $\mathcal{O}(1)$ elements, independent of the number of messages q (similar to RS_Poly), whereas the secret key for URS_PS contains $\mathcal{O}(q)$ elements (similar to RS_PS).

Note that, for $\bar{\mathcal{I}} \neq \emptyset$, computation of the enforcer element $\tilde{\sigma}_0' \in \mathbb{G}_2$ uses the elements $\{\tilde{Z}_i\}$ and $\{\tilde{W}_{i,j}\}_{i\neq j}$. If one instead employed an enforcer element $\tilde{\sigma}_0' \to \sigma_0' \in \mathbb{G}_1$, the corresponding calculation would require $\{\tilde{Z}_i\}$ and $\{\tilde{W}_{i,j}\}_{i\neq j}$ to also be $\mathbb{G}_1$ elements, namely $\{\tilde{Z}_i\} \to \{Z_i\}$ and $\{\tilde{W}_{i,j}\}_{i\neq j} \to \{W_{i,j}\}_{i\neq j}$. However, the verification checks use the elements $\{\tilde{Z}_i\}$. Thus, to achieve $\tilde{\sigma}_0' \to \sigma_0' \in \mathbb{G}_1$, one must extend the public key to include both the elements $\{\tilde{Z}_i\}$ and $\{Z_i\}$, and replace $\{\tilde{W}_{i,j} = \tilde{g}^{z^i y^j}\}_{i\neq j} \to \{W_{i,j} = g^{z^i y^j}\}_{i\neq j}$. This modification would increase the size of the public key, relative to URS_PS. By employing $\tilde{\sigma}_0' \in \mathbb{G}_2$ in the definition of URS_Poly above, the public key for URS_Poly has the same number of elements as the public key for URS_PS. Moreover, this approach does not increase the cost of verification in URS_Poly, relative to URS_PS.

To provide an example, consider a user who calls URS_Poly.Sign and receives a signature $\sigma = (1_{\mathbb{G}_1}, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$, for messages $\{m_i\}_{i\in[q]}$, where $\tilde{\sigma}_2 = \tilde{\sigma}_1^{x+\sum_i m_i y^i}$. Then, to obtain a derived signature on a subset of messages $\{m_i\}_{i\in\mathcal{I}}$, the user calls URS_Poly.Derive, which outputs the signature $\sigma_{\mathcal{I}} = (\sigma_1', \tilde{\sigma}_0', \tilde{\sigma}_1', \tilde{\sigma}_2')$, with:

$$\begin{aligned} \sigma_1' &= g^{r+\sum_{j\in\bar{\mathcal{I}}} m_j y^j}, \\ \tilde{\sigma}_0' &= \tilde{g}^{(\sum_{i\in\mathcal{I}} r z^i)+\sum_{i\in\mathcal{I}}\sum_{j\in\bar{\mathcal{I}}} m_j y^j z^i}, \\ \tilde{\sigma}_1' &= \tilde{\sigma}_1^t, \\ \tilde{\sigma}_2' &= \tilde{\sigma}_1^{t(x+r+\sum_{i=1}^q m_i y^i)}. \end{aligned} \tag{126}$$

The signature $\tilde{\sigma}_2'$ again has the form of a sequentially aggregated signature with initial message set $\{m_i\}_{i\in[q]}$, and a sequentially aggregated message r (under 'secret key' $\tilde{y} \to 1$).

The user may pass the derived signature $\sigma_{\mathcal{I}}$ to a verifier, who calls $\mathsf{URS\_PS.Verify}$. The first check in Eq. (125) tests (in the exponent):

$$\left(x + r + (\sum_{j \in \bar{\mathcal{I}}} m_j y^j) + (\sum_{i \in \mathcal{I}} m_i y^i)\right) \cdot t \overset{?}{=} t \cdot \left(x + r + \sum_{i \in [q]} m_i y^i\right), \tag{127}$$

which passes for a valid signature. The second check gives:

$$\left(r + \sum_{j \in \bar{\mathcal{I}}} m_j y^j\right) \cdot \left(\sum_{i \in \mathcal{I}} z^i\right) \overset{?}{=} \left((\sum_{i \in \mathcal{I}} r z^i) + \sum_{i \in \mathcal{I}} \sum_{j \in \bar{\mathcal{I}}} m_j y^j z^i\right), \tag{128}$$

which also passes, provided the accumulator does not contain any disallowed $\mathcal{O}(y^i)$ terms for $i \in \mathcal{I}$. If an adversary uses a malicious element σ_1' that contains some dependence on $Y_{i'}$ for $i' \in \mathcal{I}$ (say $\sigma_1' = Y_{i'}^u g^\tau$ for some element τ), the second verification check gives

$$e(\sigma_1', \prod_{i \in \mathcal{I}} \tilde{Z}_i) = e(g, \tilde{g})^{(y_{i'} u + \tau)(\sum_{i \in \mathcal{I}} z_i)}. \tag{129}$$

The exponent therefore contains a term $y_{i'} z_{i'}$ for $i' \in \mathcal{I}$, but no such term appears in the public key. Thus, the adversary is unable to use the public key to construct an enforcer element $\tilde{\sigma}_0'$ that would successfully verify the correct form of the accumulator σ_1' in the second check in Eq. (125) (i.e. verification precludes the malicious accumulator).

The second verification test does not forbid the inclusion of the random element r (which obscures the signature). Consequently users making several reveals of a signature on a particular subset of messages $\{m_i\}_{i \in \mathcal{I}}$, to different verifiers, may call $\mathsf{URS\_PS.Derive}$ for each reveal and introduce distinct random elements into each signature, thereby achieving unlinkability.

8.3 Poly-Based URS Security: Unforgeability and Unlinkability

Unforgeability in unlinkable redactable signature schemes may be defined in multiple ways [CDHK15, San19]. Sanders [San19] noted that the definition introduced by Camenisch *et al.* [CDHK15] may be too strong for some use cases. A distinction was therefore made between *strong unforgeability* (i.e. the definition introduced by Camenisch *et al.* [CDHK15]) and a weaker notion simply called *unforgeability* [San19]. In what follows, it is shown that protocol $\mathsf{URS\_Poly}$ is both unforgeable and unlinkable. Section 8.4 demonstrates that $\mathsf{URS\_Poly}$ is also strongly unforgeable.

Concretely, the EUF-CMA URS-Poly security game is defined as follows:

- EUF-CMA URS-Poly Security Game
  - Challenger $\mathcal{C}$ initialises an empty message table $T_M = \emptyset$, and executes algorithms $\mathsf{URS\_Poly.Setup}$ and $\mathsf{URS\_Poly.Keygen}$, generating public parameters pp, and keys

$$\begin{aligned} \mathsf{sk}_* &= (x, y) \\ \mathsf{pk}_* &= (g, \tilde{g}, X, \{Y_i\}_{i \in [q]}, \{\tilde{Z}_i\}_{i \in [q]}, \{\tilde{W}_{i,j}\}_{i,j \in [q]; i \neq j}). \end{aligned} \tag{130}$$

The challenger shares pp and pk_* with the PPT adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$, and provides access to a signing oracle $\mathcal{O}^{UF}_{\mathsf{URS\_Poly}}$, which executes $\mathsf{URS\_Poly.Sign}(\mathsf{sk}, \{m_i\}_{i\in[q]})$, outputs the signature $\sigma_{(\kappa)}$, and adds $\{m_{i,\kappa}\}_{i\in[q]}$ to table T_M.

- When the adversary requests a signature on messages $\{m_{i,\kappa}\}_{i\in[q]}$, with $\kappa \in [Q_O]$ (for Q_O oracle queries), the signing oracle $\mathcal{O}^{UF}_{\mathsf{URS\_Poly}}$ outputs the signature $\sigma_{(\kappa)}$, and adds $\{m_{i,\kappa}\}_{i\in[q]}$ to table T_M.
- Eventually the adversary outputs a signature σ_* and messages $\{m_i^*\}_{i\in\mathcal{I}}$. The adversary wins the challenge if (i) $\mathcal{I} \neq \emptyset$; (ii) the signature σ_* verifies for messages $\{m_i^*\}_{i\in\mathcal{I}}$ under public key pk_*; and (iii) there exists $m_j^* \in \{m_i^*\}_{i\in[q]}$, such that $m_j^* \notin T_M$.

The unforgeability security of scheme $\mathsf{URS\_Poly}$ is proven using a tight reduction to an adversary against scheme $\mathsf{RS\_Poly}$.

Theorem 8.1. *An adversary that can break the EUF-CMA security of the unlinkable redactable signature scheme* $\mathsf{URS\_Poly}$ *by winning the EUF-CMA URS-Poly security game can be leveraged to break the EUF-CMA security of protocol* $\mathsf{RS\_Poly}$ *by winning the EUF-CMA RS-Poly security game with the same success probability.*

Proof. Assume that adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$ can win the EUF-CMA URS-Poly security game with non-negligible probability. Let $\mathcal{A}_{\mathsf{RS\_Poly}}$ be an adversary against the EUF-CMA RS-Poly security game. Upon receipt of the challenge key pk_* and parameters pp, adversary $\mathcal{A}_{\mathsf{RS\_Poly}}$ acts as EUF-CMA URS-Poly challenger and passes pk_* and pp to $\mathcal{A}_{\mathsf{URS\_Poly}}$. When $\mathcal{A}_{\mathsf{URS\_Poly}}$ requests a signature on a set of messages, $\mathcal{A}_{\mathsf{RS\_Poly}}$ passes the messages to its signing oracle $\mathcal{O}_{\mathsf{RS\_Poly}}$. Upon receiving the oracle output $\sigma = (\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$, $\mathcal{A}_{\mathsf{RS\_Poly}}$ selects $t, r \xleftarrow{R} \mathbb{F}_p^*$, computes

$$\sigma_1' \;=\; g^r \cdot \sigma_1, \qquad \tilde{\sigma}_0' = \prod_{i\in\mathcal{I}} \tilde{Z}_i^r \cdot \tilde{\sigma}_0, \qquad \tilde{\sigma}_1' = \tilde{\sigma}_1^t, \qquad \tilde{\sigma}_2' = (\tilde{\sigma}_1^t) \cdot (\tilde{\sigma}_1')^r, \tag{131}$$

and returns $\sigma = (\sigma_1', \tilde{\sigma}_0', \tilde{\sigma}_1', \tilde{\sigma}_2')$ to $\mathcal{A}_{\mathsf{URS\_Poly}}$. This process perfectly simulates a $\mathsf{URS\_Poly}$ signing oracle. Eventually $\mathcal{A}_{\mathsf{URS\_Poly}}$ outputs a signature σ_* which is a valid forgery for a set of messages $\{m_i^*\}_{i\in\mathcal{I}}$. This signature verifies under $\mathsf{RS\_Poly.Verify}$, as the verification checks for $\mathsf{RS\_Poly}$ and $\mathsf{URS\_Poly}$ are identical. Moreover, as all oracle queries made by $\mathcal{A}_{\mathsf{URS\_Poly}}$ are forwarded to $\mathcal{O}_{\mathsf{RS\_Poly}}$, both sets of queried messages are identical. Hence, adversary $\mathcal{A}_{\mathsf{RS\_Poly}}$ succeeds whenever adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$ succeeds. ■

The unlinkability security game for a redactable polynomial-based signature scheme is defined as follows:

- URS-Poly Unlinkability Security Game
  - Challenger $\mathcal{C}$ executes $\mathsf{URS\_Poly.Setup}$ and $\mathsf{URS\_Poly.Keygen}$, and passes the parameters pp and public key pk_* to adversary $\mathcal{A}$.

- Adversary $\mathcal{A}$ provides $\mathcal{C}$ with two sets $\{\{m_i^{(a)}\}_{i\in[q]}, \sigma_a\}$ and $\{\{m_i^{(b)}\}_{i\in[q]}, \sigma_b\}$, and a set $\mathcal{I} \subset [q]$, where $\sigma_{a,b}$ are valid $\mathsf{URS\_Poly}$ signatures for message sets $\{m_i^{(a,b)}\}_{i\in[q]}$, and $m_i^{(a)} = m_i^{(b)}$ for all $i \in \mathcal{I}$ (a and b are arbitrary labels).
- The challenger checks that the signatures verify, that $\mathcal{I} \neq \emptyset$, and that $m_i^{(a)} = m_i^{(b)}$ for all $i \in \mathcal{I}$. If any check fails, the challenger aborts.
- The challenger selects $\beta \xleftarrow{R} \{a, b\}$, computes $\mathsf{URS\_Poly.Derive}(\mathsf{pk}_*, \sigma_\beta, \{m_i^{(\beta)}\}_{i\in[q]}, \mathcal{I})$, and outputs the derived signature $\sigma_{\mathcal{I}}^{\beta}$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs $\beta_{\mathcal{A}} \in \{a, b\}$ and wins the challenge if $\beta_{\mathcal{A}} = \beta$.

The point here is that the adversary should not be able to distinguish between derived unlinkable signatures as the associated distributions for derived signatures should be indistinguishable.

Theorem 8.2. *Derived signatures generated by redactable signature scheme* $\mathsf{URS\_Poly}$ *are information-theoretically unlinkable.*

Proof. The proof proceeds by showing that a signature $\sigma_{\mathcal{I}}$, derived from the $\mathsf{URS\_Poly}$ signature $\sigma = (\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$ for messages $\{m_i\}_{i\in[q]}$, using $\mathcal{I} \subset [q]$ with $\mathcal{I} \neq \emptyset$, is randomly distributed, independent of the revealed messages $\{m_i\}_{i\in\mathcal{I}}$. Recall that algorithm $\mathsf{URS\_Poly.Derive}$ draws uniform elements $r, t \xleftarrow{R} \mathbb{F}_p^*$, and computes the derived signature $\sigma_{\mathcal{I}} = (\sigma'_1, \tilde{\sigma}'_0, \tilde{\sigma}'_1, \tilde{\sigma}'_2)$, where:

$$\begin{aligned}
\sigma'_1 &= g^{r+\sum_{j\in\bar{\mathcal{I}}} m_j y^j}, \\
\tilde{\sigma}'_0 &= \tilde{g}^{(\sum_{i\in\mathcal{I}} r z^i)+\sum_{i\in\mathcal{I}}\sum_{j\in\bar{\mathcal{I}}} m_j y^j z^i}, \\
\tilde{\sigma}'_1 &= \tilde{\sigma}_1^t, \\
\tilde{\sigma}'_2 &= \tilde{\sigma}_1^{t(x+r+\sum_{i=1}^q m_i y^i)}.
\end{aligned} \tag{132}$$

Consider each element in turn. As $\tilde{\sigma}_1 \in \mathbb{G}_2^*$ is a random generator, element $\tilde{\sigma}'_1 = \tilde{\sigma}_1^t$ is randomly distributed in $\mathbb{G}_2$. For any value of $\sum_{j\in\bar{\mathcal{I}}} m_j y^j$, one may define u such that $r = u - \sum_{j\in\bar{\mathcal{I}}} m_j y^j$. Moreover, r and u are both uniformly distributed as r is chosen at random and there exists a one-to-one mapping between r and u. Thus, $\sigma'_1 = g^u$ is a random element in $\mathbb{G}_1$. Similarly, this substitution gives $\tilde{\sigma}'_0 = \tilde{g}^{u\cdot\sum_{i\in\mathcal{I}} z^i}$, for random u, meaning the distribution of $\tilde{\sigma}'_0$ in $\mathbb{G}_2$ is independent of both the signature σ and the revealed messages $\{m_i\}_{i\in\mathcal{I}}$. Finally, substituting for r gives $\tilde{\sigma}'_2 = \tilde{\sigma}_1^{t(x+u+\sum_{j\in\bar{\mathcal{I}}} m_j y^j)}$. For any given set of redacted messages $\{m_j\}_{j\in\bar{\mathcal{I}}}$, there exists a value of the random field element u, such that $u + \sum_{j\in\bar{\mathcal{I}}} m_j y^j = c$, for any $c \in \mathbb{F}_p$. Thus, the exponent does not reveal any information about the redacted messages and the distribution of $\tilde{\sigma}'_2$ in $\mathbb{G}_2$ is also independent of both the non-redacted messages $\{m_i\}_{i\in\mathcal{I}}$, and the signature σ (recall that $\tilde{\sigma}_1^t$ is uniformly distributed in $\mathbb{G}_2$).

Since the distribution of a derived $\mathsf{URS\_Poly}$ signature is independent of both the original signature and the revealed messages, with the dependence on the redacted messages masked by the random variable u, the derived signature is information-theoretically unlinkable. ■

Theorem 8.3. *No adversary can gain an advantage in the URS-Poly unlinkability security game.*

Proof. By Theorem 8.2, derived signatures are information-theoretically unlinkable. Consequently an adversary in the URS-Poly unlinkability security game has no available strategy bar brute force (i.e. guessing) and therefore cannot gain an advantage. ■

8.4 Poly-Based URS Security: Strong Unforgeability

In addition to satisfying the unforgeability and unlinkability properties described above, scheme $\mathsf{URS\_Poly}$ also satisfies the stronger definition of unforgeability introduced by Camenisch *et al.* [CDHK15]. Concretely, this stronger notion of unforgeability is defined by the following security game:

- <u>SUF-CMA URS-Poly Security Game</u>
  - Challenger $\mathcal{C}$ initialises three empty tables, $T_{1,2,3} = \emptyset$, and executes algorithms $\mathsf{URS\_Poly.Setup}$ and $\mathsf{URS\_Poly.Keygen}$, generating public parameters pp and keys

$$\begin{aligned} \mathsf{sk}_* &= (x, y) \\ \mathsf{pk}_* &= (g, \tilde{g}, X, \{Y_i\}_{i\in[q]}, \{\tilde{Z}_i\}_{i\in[q]}, \{\tilde{W}_{i,j}\}_{i,j\in[q];\, i\neq j}). \end{aligned} \tag{133}$$

    The challenger shares pp and pk_* with the PPT adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$, and provides access to the following oracles:
    * A signing oracle $\mathcal{O}^{\mathrm{S}}_{\mathsf{URS\_Poly}}$, which takes as input a set of messages $\{m_i\}_{i\in[q]}$, and computes $\sigma \leftarrow \mathsf{URS\_Poly.Sign}(\mathsf{sk}_*, \{m_i\}_{i\in[q]})$, but does not output the signature. Adds $(\sigma, \{m_i\}_{i\in[q]})$ to table T_1, namely $T_1[c] = (\sigma, \{m_i\}_{i\in[q]})$.
    * A derived-signature oracle $\mathcal{O}^{\mathrm{DS}}_{\mathsf{URS\_Poly}}$, which takes as input an index c, and subset $\mathcal{I} \subset [q]$, and returns $\mathsf{URS\_Poly.Derive}(\mathsf{pk}_*, \sigma, \{m_i\}_{i\in[q]}, \mathcal{I})$, using the entry $T_1[c]$. Aborts if $T_1[c] = \emptyset$. Adds the set $\{m_i\}_{i\in\mathcal{I}}$ to table T_2.
    * A reveal oracle $\mathcal{O}^{\mathrm{R}}_{\mathsf{URS\_Poly}}$, which takes as input an index c, returns $T_1[c] = (\sigma, \{m_i\}_{i\in[q]})$, and adds $\{m_i\}_{i\in[q]}$ to table T_3. Aborts if $T_1[c] = \emptyset$.
  - Eventually the adversary outputs a derived signature σ_* and messages $\{m_i^*\}_{i\in\mathcal{I}}$. The adversary wins the challenge if (i) $\mathcal{I} \neq \emptyset$; (ii) σ_* is a valid $\mathsf{URS\_Poly}$ signature for the messages $\{m_i^*\}_{i\in\mathcal{I}}$ under public key pk_*; (iii) the messages $\{m_i^*\}_{i\in\mathcal{I}}$ are not in table T_2; and (iv) there exists $m_j^* \in \{m_i^*\}_{i\in\mathcal{I}}$, such that $m_j^* \notin \{\hat{m}_i\}_{i\in[q]}$, for all messages $\{\hat{m}_i\}_{i\in[q]} \in T_3$.

The adversary succeeds by constructing a valid derived signature on messages $\{m_i^*\}_{i\in\mathcal{I}}$, even if the full set of messages were previously signed by the signer, namely $\{m_i^*\}_{i\in\mathcal{I}} \subset \{m_{i,\kappa}\}_{i\in[q]}$ for some κ (where κ labels an oracle call), provided the set $\{m_i^*\}_{i\in\mathcal{I}}$ does not appear in T_2 (the table of messages for which a derived signature was previously requested). This notion is stronger than unforgeability, which does not accept a forgery if the full set of messages were previously signed, even if the forgery is a derived signature.

Theorem 8.4. *The unlinkable redactable signature scheme* $\mathsf{URS\_Poly}$ *is strongly unforgeable in the generic bilinear group model. That is, a PPT adversary has a negligible chance of winning the SUF-CMA URS-Poly security game.*

Proof. Let the PPT adversary $\mathcal{A}$ receive the SUF-CMA URS-Poly security game public parameters pp, and public key:

$$(g, \tilde{g}, X, \{Y_i\}_{i\in[q]}, \{\tilde{Z}_i\}_{i\in[q]}, \{\tilde{W}_{i,j}\}_{i,j\in[q];\, i\neq j}), \tag{134}$$

along with the oracles $\mathcal{O}^{\mathrm{S}}_{\mathsf{URS\_Poly}}$, $\mathcal{O}^{\mathrm{DS}}_{\mathsf{URS\_Poly}}$, and $\mathcal{O}^{\mathrm{R}}_{\mathsf{URS\_Poly}}$, and a generic bilinear group oracle $\mathcal{O}_{\mathbb{G}}$. Signatures returned by $\mathcal{O}^{\mathrm{R}}_{\mathsf{URS\_Poly}}$ or $\mathcal{O}^{\mathrm{DS}}_{\mathsf{URS\_Poly}}$ are collectively (and sequentially) labelled as $\sigma_\kappa = (\sigma_{1,\kappa}, \tilde{\sigma}_{0,\kappa}, \tilde{\sigma}_{1,\kappa}, \tilde{\sigma}_{0,\kappa})$, for messages $\{m_{i,\kappa}\}_{i\in[q]}$ or $\{m_{i,\kappa}\}_{i\in\mathcal{I}_\kappa}$, respectively. Here $\kappa \in [Q_O]$ labels the Q_O queries to $\mathcal{O}^{\mathrm{R}}_{\mathsf{URS\_Poly}}$ and $\mathcal{O}^{\mathrm{DS}}_{\mathsf{URS\_Poly}}$. The adversary eventually outputs a set of messages $\{m^*_i\}_{i\in\mathcal{I}}$, for set $\mathcal{I} \subseteq [q]$, and a forged signature $(\sigma^*_1, \tilde{\sigma}^*_0, \tilde{\sigma}^*_1, \tilde{\sigma}^*_2)$. The forgery may be for a complete set of q messages ($\mathcal{I} = [q]$) or a derived signature for a subset of messages ($\mathcal{I} \subset [q]$). For $\mathcal{I} = [q]$, a valid forgery occurs if, for all message sets $\{m_{i,\kappa}\}_{i\in[q]}$ in table T_3, there exists at least one index $i_\kappa \in \mathcal{I}$, such that $m_{i_\kappa,\kappa} \neq m^*_{i_\kappa}$ (where $m^*_{i_\kappa} \in \{m^*_i\}_{i\in[q]}$, the final messages outputted by the adversary). If $\mathcal{I} \subset [q]$, the derived signature is a valid forgery if, in addition to the previous requirement, the elements $\{m^*_i\}_{i\in\mathcal{I}}$ do not appear in table T_2 (i.e. no previous derived signature request was made for the set $\{m^*_i\}_{i\in\mathcal{I}}$).

To generate new group elements in the generic bilinear group model, the adversary uses generic group operations such as group multiplications and pairing operations. Thus, there exist scalars

$$\begin{aligned}
&(a, b, \{c_i\}_{i\in[q]}, \{d_\kappa\}_{\kappa\in[Q_O]}),\\
&(\mu, \{\nu_i\}_{i\in[q]}, \{\phi_{i,j}\}_{i,j\in[q];i\neq j}, \{\gamma_\kappa\}_{\kappa\in[Q_O]}, \{\delta_\kappa\}_{\kappa\in[Q_O]}, \{\epsilon_\kappa\}_{\kappa\in[Q_O]}),\\
&(\mu', \{\nu'_i\}_{i\in[q]}, \{\phi'_{i,j}\}_{i,j\in[q];i\neq j}, \{\gamma'_\kappa\}_{\kappa\in[Q_O]}, \{\delta'_\kappa\}_{\kappa\in[Q_O]}, \{\epsilon'_\kappa\}_{\kappa\in[Q_O]}),\\
&(\mu'', \{\nu''_i\}_{i\in[q]}, \{\phi''_{i,j}\}_{i,j\in[q];i\neq j}, \{\gamma''_\kappa\}_{\kappa\in[Q_O]}, \{\delta''_\kappa\}_{\kappa\in[Q_O]}, \{\epsilon''_\kappa\}_{\kappa\in[Q_O]}),
\end{aligned} \tag{135}$$

such that the elements in an adversary's forgery $(\sigma^*_1, \tilde{\sigma}^*_0, \tilde{\sigma}^*_1, \tilde{\sigma}^*_2)$ may be written as:

$$\begin{aligned}
\sigma^*_1 &= g^a \cdot X^b \cdot \prod_{i\in[q]} Y_i^{c_i} \cdot \prod_{\kappa\in[Q_O]} g^{d_\kappa\cdot r_\kappa},\\
\tilde{\sigma}^*_1 &= \tilde{g}^{\mu} \cdot \prod_{i\in[q]} \tilde{Z}_i^{\nu_i} \cdot \prod_{i\neq j} \tilde{W}_{i,j}^{\phi_{i,j}} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{1,\kappa}^{\gamma_\kappa} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{2,\kappa}^{\delta_\kappa} \cdot \prod_{\kappa\in[Q_O]} \prod_{i\in\mathcal{I}_\kappa} \tilde{Z}_i^{\epsilon_\kappa\cdot r_\kappa},\\
\tilde{\sigma}^*_2 &= \tilde{g}^{\mu'} \cdot \prod_{i\in[q]} \tilde{Z}_i^{\nu'_i} \cdot \prod_{i\neq j} \tilde{W}_{i,j}^{\phi'_{i,j}} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{1,\kappa}^{\gamma'_\kappa} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{2,\kappa}^{\delta'_\kappa} \cdot \prod_{\kappa\in[Q_O]} \prod_{i\in\mathcal{I}_\kappa} \tilde{Z}_i^{\epsilon'_\kappa\cdot r_\kappa},\\
\tilde{\sigma}^*_0 &= \tilde{g}^{\mu''} \cdot \prod_{i\in[q]} \tilde{Z}_i^{\nu''_i} \cdot \prod_{i\neq j} \tilde{W}_{i,j}^{\phi''_{i,j}} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{1,\kappa}^{\gamma''_\kappa} \cdot \prod_{\kappa\in[Q_O]} \tilde{\sigma}_{2,\kappa}^{\delta''_\kappa} \cdot \prod_{\kappa\in[Q_O]} \prod_{i\in\mathcal{I}_\kappa} \tilde{Z}_i^{\epsilon''_\kappa\cdot r_\kappa}.
\end{aligned} \tag{136}$$

Here the element g^{r_κ} comes from $\sigma_{1,\kappa}$, and the elements $\prod_{i\in\mathcal{I}_\kappa} \tilde{Z}_i^{r_\kappa}$ come from the elements $\tilde{\sigma}_{0,\kappa}$, produced by the κ-th oracle query. The other components of the signature elements $\sigma_{1,\kappa}$ and $\tilde{\sigma}_{0,\kappa}$ are not explicitly included in the above as they are built using publicly known parameters (i.e. the missing elements are implicitly included).

If the forgery is valid, it must satisfy the verification checks. The second check gives (in the exponent):

$$\begin{aligned}
&(a + bx + \sum_{i\in[q]} c_i\, y^i + \sum_{\kappa\in[Q_O]} d_\kappa \cdot r_\kappa)(\sum_{i\in\mathcal{I}} z^i) \\
= \quad &(\mu'' + \sum_{i\in[q]} z^i \nu_i'' + \sum_{i\neq j} z^i y^j \phi_{i,j}'' + \sum_{\kappa\in[Q_O]} v_\kappa \gamma_\kappa'' + \sum_{\kappa\in[Q_O]} v_\kappa (x + \sum_{i\in[q]} m_i y^i)\delta_\kappa'' \\
&+ \sum_{\kappa\in[Q_O]} \sum_{i\in\mathcal{I}_\kappa} z^i r_\kappa \epsilon_\kappa''). \qquad (137)
\end{aligned}$$

where the elements v_κ are unknown to the adversary (they're defined through the elements $\tilde{\sigma}_{1,\kappa} = \tilde{g}^{v_\kappa}$). Inspecting the $\mathcal{O}(1)$ term, the $\mathcal{O}(x)$ terms, the $\mathcal{O}(v_\kappa)$ terms, and the $\mathcal{O}(v_\kappa x)$ terms, in turn, shows that one requires

$$\mu'' = b = \gamma_\kappa'' = \delta_\kappa'' = 0, \qquad (138)$$

giving

$$(a + \sum_{i\in[q]} c_i\, y^i + \sum_{\kappa\in[Q_O]} d_\kappa \cdot r_\kappa)(\sum_{i\in\mathcal{I}} z^i) = \sum_{i\in[q]} z^i \nu_i'' + \sum_{i\neq j} z^i y^j \phi_{i,j}'' + \sum_{\kappa\in[Q_O]} \sum_{i\in\mathcal{I}_\kappa} z^i r_\kappa \epsilon_\kappa''.$$

This expression requires $c_i = 0$ for $i \in \mathcal{I}$ (there are no $i = j$ terms on the right hand side), and $\nu_i'' = 0$ for $i \in \bar{\mathcal{I}}$, with $\nu_i'' \equiv \nu'' = a$ for all $i \in \mathcal{I}$. Moreover, one requires $\phi_{i,j}'' = 0$ for $i \in \bar{\mathcal{I}}$ or $j \in \mathcal{I}$, and $\phi_{i,j}'' = c_j$ for $j \in \bar{\mathcal{I}}$. Finally, one requires $d_\kappa = \epsilon_\kappa'' = 0$ for all κ such that $\mathcal{I} \neq \mathcal{I}_\kappa$, and $d_\kappa = \epsilon_\kappa'' = 0$ for all κ with $\mathcal{I} = \mathcal{I}_\kappa$. The second verification check therefore requires:

$$\begin{aligned}
\sigma_1 &= g^a \cdot \prod_{j\in\bar{\mathcal{I}}} Y_j^{c_j} \cdot \prod_{\kappa\in[Q_O]:\mathcal{I}=\mathcal{I}_\kappa} g^{d_\kappa \cdot r_\kappa}, \\
\tilde{\sigma}_0 &= \prod_{i\in\mathcal{I}} \tilde{Z}_i^a \cdot \prod_{i\in\mathcal{I},\, j\in\bar{\mathcal{I}}} \tilde{W}_{i,j}^{c_j} \cdot \prod_{\kappa\in[Q_O]:\mathcal{I}=\mathcal{I}_\kappa} \prod_{i\in\mathcal{I}_\kappa} \tilde{Z}_i^{d_\kappa \cdot r_\kappa}. \qquad (139)
\end{aligned}$$

Using this expression, the first verification check requires:

$$\begin{aligned}
&(x + a + \sum_{j\in\bar{\mathcal{I}}} y^j c_j + \sum_{\kappa\in[Q_O]:\mathcal{I}=\mathcal{I}_\kappa} d_\kappa r_\kappa + \sum_{i\in\mathcal{I}} y^i m_i^*) \times \\
&\Bigg(\mu + \sum_{i\in[q]} z^i \nu_i + \sum_{i,j\in[q],i\neq j} z^i y^j \phi_{i,j} + \sum_{\kappa\in[Q_O]} \gamma_\kappa v_\kappa + \sum_{\kappa\in[Q_O]} \delta_\kappa v_\kappa (x + \sum_{i\in[q]} y^i m_{i,\kappa}) \\
&\qquad + \sum_{\kappa\in[Q_O]} \sum_{i\in\mathcal{I}_\kappa} z^i r_\kappa \epsilon_\kappa \Bigg) \\
= \quad &\Bigg(\mu' + \sum_{i\in[q]} z^i \nu_i' + \sum_{i,j\in[q],i\neq j} z^i y^j \phi_{i,j}' + \sum_{\kappa\in[Q_O]} \gamma_\kappa' v_\kappa + \sum_{\kappa\in[Q_O]} \delta_\kappa' v_\kappa (x + \delta_{RD} r_\kappa + \sum_{i\in[q]} y^i m_{i,\kappa}) \\
&\qquad + \sum_{\kappa\in[Q_O]} \sum_{i\in\mathcal{I}_\kappa} z^i r_\kappa \epsilon_\kappa' \Bigg).
\end{aligned}$$

Here $\delta_{RD} = 1$ if the κ-th signature was returned by the derive oracle, and $\delta_{RD} = 0$ if returned by the reveal oracle. Inspecting the $\mathcal{O}(v_\kappa x^2)$ terms, the $\mathcal{O}(x)$ term, the $\mathcal{O}(xz^i)$ terms, the $\mathcal{O}(xz^i y^j)$ terms, the $\mathcal{O}(z^i)$ terms, the $\mathcal{O}(z^i y^j)$ terms, and the $\mathcal{O}(1)$ term, in turn, shows that one requires

$$\delta_\kappa = \mu = \nu_i = \phi_{i,j} = \nu'_i = \phi'_{i,j} = \mu' = 0,$$

giving

$$\begin{aligned}(x + a + \sum_{j\in\bar{\mathcal{I}}} y^j c_j + \sum_{\kappa\in[Q_O]:\mathcal{I}=\mathcal{I}_\kappa} d_\kappa r_\kappa + \sum_{i\in\mathcal{I}} y^i m_i^*)(\sum_{\kappa\in[Q_O]} \gamma_\kappa v_\kappa + \sum_{\kappa\in[Q_O]}\sum_{i\in\mathcal{I}_\kappa} z^i r_\kappa \epsilon_\kappa) \\ = \sum_{\kappa\in[Q_O]} \gamma'_\kappa v_\kappa + \sum_{\kappa\in[Q_O]} \delta'_\kappa v_\kappa (x + \delta_{RD} r_\kappa + \sum_{i\in[q]} y^i m_{i,\kappa}) + \sum_{\kappa\in[Q_O]}\sum_{i\in\mathcal{I}_\kappa} z^i r_\kappa \epsilon'_\kappa.\end{aligned}$$

Inspecting the $\mathcal{O}(z^i r_\kappa x)$ terms, one requires $\epsilon_\kappa = 0$, which then requires $\epsilon'_\kappa = 0$ by the $\mathcal{O}(z^i r_\kappa)$ terms, finally giving

$$\begin{aligned}(x + a + \sum_{j\in\bar{\mathcal{I}}} y^j c_j + \sum_{\kappa\in[Q_O]:\mathcal{I}=\mathcal{I}_\kappa} d_\kappa r_\kappa + \sum_{i\in\mathcal{I}} y^i m_i^*)(\sum_{\kappa\in[Q_O]} \gamma_\kappa v_\kappa) \\ = \sum_{\kappa\in[Q_O]} \gamma'_\kappa v_\kappa + \sum_{\kappa\in[Q_O]} \delta'_\kappa v_\kappa (x + \delta_{RD} r_\kappa + \sum_{i\in[q]} y^i m_{i,\kappa}).\end{aligned}$$

Treating this expression as a polynomial in the constants v_κ, one has:

$$(x + a + \sum_{j\in\bar{\mathcal{I}}} y^j c_j + \sum_{\kappa\in[Q_O]:\mathcal{I}=\mathcal{I}_\kappa} d_\kappa r_\kappa + \sum_{i\in\mathcal{I}} y^i m_i^*) \times \gamma_\kappa \;=\; \gamma'_\kappa + \delta'_\kappa (x + \delta_{RD} r_\kappa + \sum_{i\in[q]} y^i m_{i,\kappa}).$$

Two cases must be considered. If the κ-th signature was produced by a call to the reveal oracle for messages $\{m_{i,\kappa}\}_{i\in[q]}$, then $\delta_{RD} = 0$. For the adversary to produce a valid forgery, there must exist one index $i_\kappa \in \mathcal{I}$, such that $m_{i_\kappa,\kappa} \neq m^*_{i_\kappa}$. In this case, a solution implies $\gamma_\kappa = \gamma'_\kappa = \delta'_\kappa = 0$, for all $\kappa \in [Q_O]$, as the adversary is unable to construct non-trivial polynomials in the unknown elements x, y^i, r_κ. If instead the signature was returned by the derive oracle, then $\delta_{RD} = 1$, and $\mathcal{I} \neq \mathcal{I}_\kappa$ (i.e. the outputted subset must not have been previously submitted to the derive oracle). This solution requires $\delta'_\kappa = 0$, which in turn requires $\gamma_\kappa = \gamma'_\kappa = 0$, for all $\kappa \in [Q_O]$. Hence, construction of a forgery is not possible in the generic bilinear group model.

We must also consider the probability that two polynomials in the adversary's set accidentally collide [BBG05]. The public key contains $3+q(q+3)/2$ elements, all with monomial exponents of degree $\leq (2q-1)$ in the secret field elements x, y, z. A query to the signing oracle outputs four group elements, $(\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$, with exponents of maximal degree $\deg(\sigma_1) \leq q$, $\deg(\tilde{\sigma}_0) \leq (2q-1)$, $\deg(\tilde{\sigma}_1) \leq 1$, and $\deg(\tilde{\sigma}_2) \leq (q+1)$, in the secret elements and the random elements drawn during signature generation. The largest-degree polynomial in the exponent of a group element returned by a call to the bilinear group oracle occurs for a pairing call, with maximal degree $\leq (2q^2 + q - 1)$. An adversary that makes Q_O signing oracle queries and $Q_{\mathbb{G}}$ group oracle queries generates $(4Q_O + Q_{\mathbb{G}} + 1 + q(q+3)/2)$ polynomials in total, all with degree $\leq (2q^2 + q - 1)$. By the Schwartz-Zippel lemma [DL78, Zip79, Sch80], the probability of collision among these polynomials is at most:

$$\frac{(2q^2 + q - 1)}{2p}(4Q_O + Q_{\mathbb{G}} + 3 + q(q+3)/2)(4Q_O + Q_{\mathbb{G}} + 2 + q(q+3)/2), \tag{140}$$

which is negligible in $\lambda = \log_2 p$ for a PPT adversary, with $q \leq \mathsf{poly}(\lambda)$. Hence, in the generic bilinear group model, PPT adversaries in the SUF-CMA URS-Poly security game have a negligible chance of winning. ■

In summary, the preceding sections proved that (i) signature scheme RS_Poly is unforgeable in the generic bilinear group model; (ii) signature scheme URS_Poly is unforgeable if RS_Poly is unforgeable; (iii) URS_Poly is information-theoretically unlinkable; and (iv) URS_Poly is strongly unforgeable in the generic bilinear group model.

9 Anonymous Credentials with Selective Disclosure

Previous authors [CDHK15, FHS19] noted that redactable unlinkable signature schemes are similar to anonymous credential schemes [Cha85, CL01], such that realisations of the former permit construction of the latter. Consequently one may leverage the unlinkable redactable signature scheme URS_PS to construct an anonymous credential scheme (here called AC_PS) [San19]. Below, the mapping between PS signature schemes and polynomial signature schemes is leveraged to construct a new polynomial-based anonymous credential scheme (AC_Poly) that leverages the unlinkable redactable signature scheme URS_Poly. The presentation and notation below closely follows Sanders [San19], to allow easy comparison with that work.

9.1 Polynomial-Based Redactable Anonymous Credentials

The objective here is to extend URS_Poly to allow the issuance of credentials tied to a user's secret key usk. In essence, users will obtain a signature on a message set $\{m_i\}_{i\in[q]} \cup \{\mathsf{usk}\}$, though the signature must be generated without explicitly revealing usk to the credential issuer; i.e. the key must remain undisclosed and therefore cannot simply be treated as a redacted message. Instead the protocol is made interactive to accommodate proofs of knowledge for secret attributes (as opposed to redacted attributes). Explicitly, users reveal a derived credential on a subset of attributes $\{m_i\}_{i\in\mathcal{I}}$ by deriving a signature on $\{m_i\}_{i\in\mathcal{I}} \cup \{\mathsf{usk}\}$ and proving knowledge of usk. However, whereas the distribution of derived URS_Poly signatures was independent of the revealed messages $\{m_i\}_{i\in\mathcal{I}}$, the distribution of derived signatures must be tied to (and therefore dependent on) the users secret key. Consequently anonymity is no longer information-theoretic and is instead assumption-dependent. Concretely, anonymity will hold under the DDH assumption in $\mathbb{G}_2$. Note that the interactive presentation protocol described below can be made non-interactive via standard use of the Fiat-Shamir heuristic [FS87].

The scheme $\mathsf{AC\_Poly}$ involves three actors - a credential issuer, a credential user, and a credential verifier. At high level, an issuer generates a public key pk corresponding to the issuer secret key sk. Users execute $\mathsf{AC\_Poly.IssuerKeygen}$ to obtain a user key pair $(\mathsf{usk}, \mathsf{upk})$. Credential issuance involves an interactive protocol between algorithms $\mathsf{AC\_Poly.Obtain}$ and $\mathsf{AC\_Poly.Issue}$, wherein the user proves knowledge of their secret key usk. Credential verification is also interactive. Algorithm $\mathsf{AC\_Poly.Show}$ is analogous to $\mathsf{URS\_Poly.Derive}$ - on input a valid credential for a set of attributes, it derives a credential for a subset of attributes. This algorithm interacts with $\mathsf{AC\_Poly.Verify}$, seeking to convince the latter that the derived credential is valid. Explicitly, anonymous credential scheme $\mathsf{AC\_Poly}$ is defined by the following algorithms and protocols:

- $\mathsf{AC\_Poly.IssuerKeygen}(1^\lambda, q)$: On input a security parameter λ, and an integer q (which specifies the number of attributes to be certified by the issuer), outputs the public parameters $\mathsf{pp} = (p, \mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a type-3 pairing. Generates three random field elements:

$$(x, y, z) \xleftarrow{R} (\mathbb{F}_p^*)^3, \tag{141}$$

  and computes the following group elements:

  - $X \leftarrow g^x$
  - $Y_i \leftarrow g^{y^i},\ \forall i \in [q+1]$
  - $\tilde{Z}_i \leftarrow \tilde{g}^{z^i},\ \forall i \in [q+1]$
  - $\tilde{W}_{i,j} \leftarrow \tilde{g}^{z^i y^j},\ \forall i \in [q+1],\ \forall j \in [q]$, such that $i \neq j$.

  Outputs the credential issuer's secret key $\mathsf{sk} \leftarrow (x, y)$ and corresponding public key:[5]

$$\mathsf{pk} = \Big(g, \tilde{g}, X, \{Y_i\}_{i\in[q+1]}, \{\tilde{Z}_i\}_{i\in[q+1]}, \{\tilde{W}_{i,j}\}_{i\in[q+1],j\in[q]}^{i\neq j}\Big)\,. \tag{142}$$

- $\mathsf{AC\_Poly.UserKeygen}(\mathsf{pk})$: On input an issuer public key pk, generates a random field element and assigns it to the user's secret key, $\mathsf{usk} \xleftarrow{R} \mathbb{F}_p^*$. Computes the user's corresponding private key, $\mathsf{upk} = \tilde{g}^{\mathsf{usk}}$.

- $\big(\mathsf{AC\_Poly.Obtain}(\mathsf{usk}, \mathsf{pk}, \{m_i\}_{i\in[q]}),\ \mathsf{AC\_Poly.Issue}(\mathsf{upk}, \rho_{\mathsf{usk}}, \mathsf{sk}, \{m_i\}_{i\in[q]})\big)$: The user executes $\mathsf{AC\_Poly.Obtain}$ to obtain an anonymous credential on messages $\{m_i\}_{i\in[q]}$, and interacts with the issuer, who runs $\mathsf{AC\_Poly.Issue}$. The user inputs their public key upk and engages in a proof of knowledge π_{usk} for the corresponding secret key usk (using e.g. Schnorr's protocol). Upon verifying the proof of knowledge, $\mathsf{AC\_Poly.Issue}$ selects $s \xleftarrow{R} \mathbb{F}_p^*$, and outputs the credential:

$$\sigma \;=\; (\tilde{\sigma}_1,\, \tilde{\sigma}_2) \;\leftarrow\; (\tilde{g}^s, \tilde{g}^{s(x+\sum_{i\in[q]} m_i y^i)} \cdot \mathsf{upk}^{s\cdot y^{q+1}}). \tag{143}$$

  The protocol concludes when $\mathsf{AC\_Poly.Obtain}$ returns the credential σ or aborts.

[5]The element $j = q+1$ of $\tilde{W}_{i,j}$ is omitted as it is not explicitly used in the protocol.

- $\big(\mathsf{AC\_Poly.Show}(\mathsf{usk}, \mathsf{pk}, \{m_i\}_{i\in[q]}, \mathcal{I}, \sigma),\ \mathsf{AC\_Poly.Verify}(\mathsf{pk}, \{m_i\}_{i\in\mathcal{I}})\big)$: The user executes algorithm $\mathsf{AC\_Poly.Show}(\mathsf{usk}, \mathsf{pk}, \{m_i\}_{i\in[q]}, \mathcal{I}, \sigma)$ to derive a credential for a subset of attributes $\{m_i\}_{i\in\mathcal{I}}$, from the credential σ, which was (purportedly) certified by the issuer with public key pk, as part of the certification of the (possibly) larger set of attributes $\{m_i\}_{i\in[q]}$. Here $\mathcal{I} \subseteq [q]$ labels the non-redacted attributes. Algorithm $\mathsf{AC\_Poly.Verify}$ takes pk and the revealed attributes $\{m_i\}_{i\in\mathcal{I}}$ as input and engages in a protocol to establish the validity of the derived credential. The verification algorithm outputs either accept or reject.

  Specifically, algorithm $\mathsf{AC\_Poly.Show}$ parses σ as $(\tilde{\sigma}_1, \tilde{\sigma}_2)$, generates three random elements:

$$(k, r, t) \xleftarrow{R} (\mathbb{F}_p^*)^3, \tag{144}$$

  computes the derived credential:

  - $\sigma_1' \leftarrow g^r \prod_{j\in\bar{\mathcal{I}}} Y_j^{m_j}$
  - $\tilde{\sigma}_0' \leftarrow (\prod_{i\in\mathcal{I}\cup\{q+1\}} \tilde{Z}_i^r) \cdot (\prod_{i\in\mathcal{I}\cup\{q+1\}} \prod_{j\in\bar{\mathcal{I}}} \tilde{W}_{i,j}^{m_j})$
  - $\tilde{\sigma}_1' \leftarrow \tilde{\sigma}_1^t$
  - $\tilde{\sigma}_2' \leftarrow \tilde{\sigma}_2^t\,(\tilde{\sigma}_1^t)^r$,

  where, $\bar{\mathcal{I}} = [q]\backslash\mathcal{I}$ is the compliment of $\mathcal{I} \subseteq [q]$, and computes the element:

$$\mathcal{C} \quad\leftarrow\quad e(Y_{q+1}^k, \tilde{\sigma}_1') \in \mathbb{G}_T. \tag{145}$$

  The protocol then proceeds as follows:

  - The user sends the derived credential $(\sigma_1', \tilde{\sigma}_0', \tilde{\sigma}_1', \tilde{\sigma}_2')$ and the element $\mathcal{C}$ to the verifier.
  - The verifier selects $c \xleftarrow{R} \mathbb{F}_p^*$, and sends it to the user.
  - The user computes $u = k + c \cdot \mathsf{usk}$, and sends it to the verifier.
  - The verifier evaluates the checks:

$$\begin{aligned} &1.\quad e(Y_{q+1}^u, \tilde{\sigma}_1') \cdot e(X \cdot \sigma_1' \cdot \prod_{i\in\mathcal{I}} Y_i^{m_i}, (\tilde{\sigma}_1')^c) = \mathcal{C} \cdot e(g^c, \tilde{\sigma}_2'), \\ &2.\quad e(\sigma_1', \tilde{Z}_{q+1} \cdot \prod_{i\in\mathcal{I}} \tilde{Z}_i) = e(g, \tilde{\sigma}_0'), \end{aligned} \tag{146}$$

    and outputs accept if both checks pass, else reject.

Verification enforces the following relationship:

$$e(X \cdot Y_{q+1}^{\mathsf{usk}} \cdot \prod_{i\in[q]} Y_i^{m_i}, \tilde{\sigma}_1') = e(g, \tilde{\sigma}_2'), \tag{147}$$

which should be satisfied by a valid credential. Note, however, that Eq. (147) cannot be checked directly by the verifier as that would require access to the user's secret key. Instead Eq. (147) is checked via the first verification check, as observed by recasting the check as:

$$\begin{aligned} e(g, (\tilde{\sigma}_2')^c) &= e(Y_{q+1}^u, \tilde{\sigma}_1') \cdot \mathcal{C}^{-1} \cdot e(X \cdot \sigma_1' \cdot \prod_{i\in\mathcal{I}} Y_i^{m_i}, \tilde{\sigma}_1')^c \\ &= e(Y_{q+1}^{c\cdot\mathsf{usk}}, \tilde{\sigma}_1') \cdot e(X \cdot \sigma_1' \cdot \prod_{i\in\mathcal{I}} Y_i^{m_i}, \tilde{\sigma}_1')^c \\ &= e(X \cdot Y_{q+1}^{\cdot\mathsf{usk}} \cdot \sigma_1' \cdot \prod_{i\in\mathcal{I}} Y_i^{m_i}, (\tilde{\sigma}_1')^c), \end{aligned} \tag{148}$$

which is equivalent to Eq. (147) if the accumulator σ_1' has the correct form. The second check uses the enforcer $\tilde{\sigma}_0'$ to ensure that the accumulator σ_1' has the correct form, preventing the user from hiding malicious elements.

The protocol for showing a derived credential consists (roughly) of generating a signature on the message set $\mathsf{usk}\cup\{m_i\}_{i\in\mathcal{I}}$, and proving knowledge of usk. Note that Schnorr's protocol could also be invoked to prove knowledge of the non-disclosed attributes $\{m_j\}_{j\in\bar{\mathcal{I}}}$, in addition to usk, as both appear as exponents in the verification equation. Hence, the protocol is compatible with a proof of knowledge for non-disclosed attributes, if desired. Relative to protocol AC_PS, AC_Poly has the following differences:

- AC_Poly uses a shorter secret key, comprising $\mathcal{O}(1)$ field elements, compared to $q+1$ field elements in AC_PS.
- The public key elements $\tilde{Y}_i = \tilde{g}^{y_i}$ in AC_PS are replaced by $\tilde{Z}_i = \tilde{g}^{z^i}$.
- To achieve a public key with the same size as AC_PS, the elements $\tilde{W}_{i,j}$ and the enforcer $\tilde{\sigma}_0$ here appear in $\mathbb{G}_2$ (for the same reasons discussed in Section 8.2).

Intuitively, the issuer in the credential scheme AC_Poly treats the user's secret key as another message, and encodes this "message" in the exponent using a new secret key (the element y^{q+1}). Thus, relative to signature scheme URS_Poly, the AC_Poly "message" set is enlarged:

$$\{m_i\}_{i\in[q]} \longrightarrow \{\mathsf{usk}\} \cup \{m_i\}_{i\in[q]}. \tag{149}$$

Hence one could rewrite the pairing argument in Eq. (147) as:

$$Y_{q+1}^{\mathsf{usk}} \cdot \prod_{i\in[q]} Y_i^{m_i} \longrightarrow \prod_{i\in[q+1]} Y_i^{m_i}, \tag{150}$$

with $m_{q+1} \equiv \mathsf{usk}$. However, the issuer should not be able to learn the user's secret key usk in the process of achieving this encoding. Consequently, to issue a credential, algorithm AC_Poly.Issue takes the user's public key upk as input and transforms it so that, in effect, the resulting credential generates a signature over an expanded message set which includes the user's secret key.

9.2 Security: Unforgeability

An adversary is able to forge a signature if it can prove possession of a valid credential for a set of attributes $\{m_i\}_{i\in\mathcal{I}}$. Unforgeability therefore involves two components. Firstly, the adversary may prove ownership of a credential issued for an honest user (i.e. falsely prove ownership of an honest users' secret key). Below it is shown that such a forgery is only possible if the DL assumption does not hold in $\mathbb{G}_2$. Secondly, in principle an adversary could generate a valid forged signature on the attributes $\{m_i\}_{i\in\mathcal{I}}$ (i.e. the adversary generates a valid signature for a malicious user). Note that the signature on the attributes is a $\mathsf{URS\_Poly}$ signature, the unforgeability of which was previously proven - to generate a forgery of the second type, the adversary must therefore break the unforgeability of $\mathsf{URS\_Poly}$ (as shown below).

Defining Honest as a set of identities for honest users, $\mathsf{Corrupt}$ as a set of identities for corrupt users, and Attr as a set of pairs $\{a, \{m_i\}_{i\in[q]}\}$, where a labels a user's identity, the following oracles are employed in the unforgeability and anonymity security games defined below [FHS19, San19]:

- $\mathcal{O}_{\mathsf{Honest}}(a)$: On input an identity a, rejects if $a \in \mathsf{Honest} \cup \mathsf{Corrupt}$, else executes $(\mathsf{usk}_a, \mathsf{upk}_a) \leftarrow \mathsf{AC\_Poly.IssuerKeygen}(\mathsf{pk})$, returns the public key upk_a, and adds identity a to Honest, thereby creating a new honest user.

- $\mathcal{O}_{\mathsf{Corrupt}}(a, \mathsf{upk})$: On input an identity a and an optional public key upk, aborts if $a \in \mathsf{Corrupt}$, otherwise adds a to $\mathsf{Corrupt}$. If $a \notin \mathsf{Honest}$, creates a new corrupt user with key $\mathsf{upk}_a = \mathsf{upk}$. If $a \in \mathsf{Honest}$, removes a from Honest, and returns usk_a and all associated credentials, thereby transferring an honest user into a corrupted user.

- $\mathcal{O}_{\mathsf{Obtain}}(a, \{m_i\}_{i\in[q]})$: Used by an adversary to impersonate a dishonest issuer and issue a credential to an honest user. Takes as input an identity $a \in \mathsf{Honest}$ and attributes $\{m_i\}_{i\in[q]}$ (aborts if $a \notin \mathsf{Honest}$). Executes algorithm

$$\mathsf{AC\_Poly.Obtain}(\mathsf{usk}_a, \mathsf{pk}, \{m_i\}_{i\in[q]}), \tag{151}$$

  which interacts with the adversary (executing the Issue role) and stores the outputted credential (i.e. adds $\{a, \{m_i\}_{i\in[q]}\}$ to Attr).

- $\mathcal{O}_{\mathsf{Issue}}(a, \{m_i\}_{i\in[q]})$: Adversaries use this oracle to impersonate a malicious user and engage with an honest issuer to receive a credential. Takes as input an identity $a \in \mathsf{Corrupt}$ and attributes $\{m_i\}_{i\in[q]}$ (aborts if $a \notin \mathsf{Corrupt}$). Executes algorithm

$$\mathsf{AC\_Poly.Issue}(\mathsf{upk}_a, \mathsf{sk}, \{m_i\}_{i\in[q]}), \tag{152}$$

  which interacts with the adversary (executing the Obtain role). Adds $\{a, \{m_i\}_{i\in[q]}\}$ to Attr.

- $\mathcal{O}_{\mathsf{ObtIss}}(a, \{m_i\}_{i\in[q]})$: Takes as input an identity $a \in \mathsf{Honest}$ and attributes $\{m_i\}_{i\in[q]}$ (aborts if $a \notin \mathsf{Honest}$). Executes the protocol

$$(\mathsf{AC\_Poly.Obtain}(\mathsf{usk}_a, \mathsf{pk}, \{m_i\}_{i\in[q]}), \mathsf{AC\_Poly.Issue}(\mathsf{upk}_a, \mathsf{sk}, \{m_i\}_{i\in[q]})), \tag{153}$$

stores the output and adds $\{a, \{m_i\}_{i\in[q]}\}$ to Attr.

- $\mathcal{O}_{\mathsf{Show}}(\kappa, \mathcal{I})$: On input a query label κ, and set $\mathcal{I} \subseteq [q]$, aborts if the user whose credential was formed by the κ-th query is not honest (i.e. aborts if $a_\kappa \notin \mathsf{Honest}$). If $a_\kappa \in \mathsf{Honest}$, executes:

$$\mathsf{AC\_Poly.Show}(\mathsf{pk}, \mathsf{usk}_{a_\kappa}, \{m_{i,\kappa}\}_{i\in[q]}, \mathcal{I}, \sigma^{(\kappa)}), \tag{154}$$

  with the adversary acting as malicious verifier. Here $\sigma^{(\kappa)}$ denotes the credential generated on attributes $\{m_{i,\kappa}\}_{\in[q]}$ by the κ-th query to either $\mathcal{O}_{\mathsf{Obtain}}$ or $\mathcal{O}_{\mathsf{ObtIss}}$ (made by user a_κ). This oracle allows an adversary to act as a malicious verifier for a showing by an honest user.

Using these oracles, the AC-Poly unforgeability security game is defined as follows:

- <u>EUF-CMA AC-Poly Security Game</u>:
  - Challenger $\mathcal{C}$ executes $\mathsf{AC\_Poly.IssuerKeygen}(1^\lambda, q)$ and provides the public parameters and issuer public key pk to the PPT adversary $\mathcal{A}$.
  - Adversary $\mathcal{A}$ is given access to oracles $\mathcal{O}_{\mathsf{Honest}}$, $\mathcal{O}_{\mathsf{Corrupt}}$, $\mathcal{O}_{\mathsf{Issue}}$, $\mathcal{O}_{\mathsf{ObtIss}}$ and $\mathcal{O}_{\mathsf{Show}}$.
  - Eventually, $\mathcal{A}$ outputs a set of attributes $\{m_i\}_{i\in\mathcal{I}}$, for an index $\mathcal{I} \subseteq [q]$, and a signature σ, purportedly for the attributes $\{m_i\}_{i\in\mathcal{I}}$. The adversary engages in the show-verify protocol, using the supplied oracles to execute $\mathsf{AC\_Poly.Show}$ and interact with the challenger via $\mathsf{AC\_Poly.Verify}(\mathsf{pk}, \{m_i\}_{i\in\mathcal{I}})$, aiming to convince the challenger of the validity of signature σ.
  - If the show-verify protocol outputs accept, and $\{a, \{m_i\}_{i\in\mathcal{I}}\} \not\subset \mathsf{Attr}$ with $a \in \mathsf{Corrupt}$, the adversary wins the game.
  - Conversely, the adversary loses if the show-verify protocol outputs reject or if $\{a, \{m_i\}_{i\in\mathcal{I}}\} \not\subset \mathsf{Attr}$ with $a \in \mathsf{Corrupt}$.

The idea here is that the adversary may play various roles in the protocols comprising the credential scheme but must ultimately output a set of attributes and a valid signature. The adversary wins if the signature verifies on the provided attributes, and both the identity *and* the attributes used by the adversary were not previously classified as $\mathsf{Corrupt}$.

Theorem 9.1. *An adversary that can win the AC-Poly unforgeability security game with probability ϵ, by successfully convincing the verifier of ownership of a credential under secret key usk_a, where usk_a belongs to an honest user $a \in \mathsf{Honest}$, can be leveraged to break the discrete log assumption in $\mathbb{G}_2$ with probability ϵ/Q_U. Here Q_U bounds the number of honest users.*

Proof. Assume that adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$ can win the AC-Poly unforgeability security game with probability ϵ, by creating a valid forgery under secret key usk_a for user $a \in \mathsf{Honest}$. Let adversary $\mathcal{A}_{\mathsf{DL}}$ receive the discrete log challenge $(\tilde{g}, \tilde{g}^w)$. Upon receipt of the challenge,

$\mathcal{A}_{\mathsf{DL}}$ constructs an issuer key pair $(\mathsf{sk}_*, \mathsf{pk}_*)$, with $\tilde{g}$ as the $\mathbb{G}_2$ generator. Adversary $\mathcal{A}_{\mathsf{DL}}$ provides the public parameters and key pk_* to $\mathcal{A}_{\mathsf{AC\_Poly}}$, and acts as an AC-Poly unforgeability challenger.

Adversary $\mathcal{A}_{\mathsf{DL}}$ knows that the forgery outputted by $\mathcal{A}_{\mathsf{AC\_Poly}}$ will be valid under secret key usk_a for some identity a that appears in the list of honest users constructed during the challenge (i.e. $a \in \mathsf{Honest}$). However, $\mathcal{A}_{\mathsf{DL}}$ does not *a priori* know which identity will be used by $\mathcal{A}_{\mathsf{AC\_Poly}}$. Adversary $\mathcal{A}_{\mathsf{DL}}$ therefore assumes that the honest identity labelled by $b \xleftarrow{R} [Q_U]$ will be used by $\mathcal{A}_{\mathsf{AC\_Poly}}$. Adversary $\mathcal{A}_{\mathsf{DL}}$ then simulates the AC-Poly challenge oracles for $\mathcal{A}_{\mathsf{AC\_Poly}}$ as follows:

- $\mathcal{O}_{\mathsf{Honest}}(a')$: When $\mathcal{A}_{\mathsf{AC\_Poly}}$ calls this oracle for identity a', adversary $\mathcal{A}_{\mathsf{DL}}$ returns the public key $\mathsf{upk}_{a'}$ if $a' \neq b$, otherwise it outputs the discrete log challenge as the secret key, $\mathsf{upk}_b = \tilde{g}^w$.
- $\mathcal{O}_{\mathsf{Corrupt}}(a')$: When $\mathcal{A}_{\mathsf{AC\_Poly}}$ calls this oracle for an honest user $a' \neq b$, adversary $\mathcal{A}_{\mathsf{DL}}$ returns $\mathsf{usk}_{a'}$. Else aborts.
- $\mathcal{O}_{\mathsf{Issue}}(a', \{m_i\}_{i\in[q]})$: Adversary $\mathcal{A}_{\mathsf{DL}}$ knows the issuer's secret key sk_* and can therefore respond to any query from $\mathcal{A}_{\mathsf{AC\_Poly}}$.
- $\mathcal{O}_{\mathsf{ObtIss}}(a', \{m_i\}_{i\in[q]})$: Adversary $\mathcal{A}_{\mathsf{DL}}$ knows the issuer's secret key sk_* and can therefore perfectly simulate the issuance side of the protocol. Regarding the user side of the protocol, $\mathcal{A}_{\mathsf{DL}}$ can perfectly execute the user if $a' \neq b$, otherwise $\mathcal{A}_{\mathsf{DL}}$ provides a simulated proof of knowledge for usk_b.
- $\mathcal{O}_{\mathsf{Show}}(\kappa, \mathcal{I})$: When adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$ queries this oracle for κ such that $a_\kappa \neq b$, adversary $\mathcal{A}_{\mathsf{DL}}$ can perfectly execute the user role in the interactive show protocol. If $a_\kappa = b$, the adversary can execute the show protocol user role but must simulate the proof of knowledge for usk_b.

With probability $1/Q_U$, adversary $\mathcal{A}_{\mathsf{DL}}$ correctly guesses the honest identity used by adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$ (i.e. $b = a$) and perfectly simulates the unforgeability game. In such cases, with probability ϵ, adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$ proves knowledge of the secret key usk_b during execution of the show-verify protocol for the forged credential (by assumption). Adversary $\mathcal{A}_{\mathsf{DL}}$ can execute the corresponding extractor to extract w from the proof of knowledge, and thereby output the solution to the DL challenge. Combined, $\mathcal{A}_{\mathsf{DL}}$ therefore has a success probability of ϵ/Q_U.

■

Theorem 9.2. *An adversary that can win the AC-Poly unforgeability security game, with non-negligible probability, by successfully convincing the verifier of ownership of a credential under secret key* usk*, where* $\mathsf{usk} \neq \mathsf{usk}_a$ *for all* $a \in \mathsf{Honest}$*, can be leveraged to win the URS-Poly unforgeability security game with the same probability.*

Proof. Assume that adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$ can win the AC-Poly unforgeability security game by creating a valid forgery under secret key usk, where $\mathsf{usk} \neq \mathsf{usk}_a$ for all $a \in \mathsf{Honest}$. Let $\mathcal{A}_{\mathsf{URS\_Poly}}$ denote an adversary in the EUF-CMA URS-Poly security game. $\mathcal{A}_{\mathsf{URS\_Poly}}$ executes the game for $q+1$ messages, receiving the public key:

$$\mathsf{pk}_* = \left(g, \tilde{g}, X, \{Y_i\}_{i\in[q+1]}, \{\tilde{Z}_i\}_{i\in[q+1]}, \{\tilde{W}_{i,j}\}^{i\neq j}_{i\in[q+1],j\in[q]}\right). \tag{155}$$

Adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$ passes pk_* to $\mathcal{A}_{\mathsf{AC\_Poly}}$ as the challenge public key. To answer oracle queries, $\mathcal{A}_{\mathsf{URS\_Poly}}$ acts as follows:

- $\mathcal{O}_{\mathsf{Honest}}(a)$: When $\mathcal{A}_{\mathsf{AC\_Poly}}$ calls this oracle for identity a, adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$ executes the oracle faithfully and stores the secret key, upk_a.
- $\mathcal{O}_{\mathsf{Corrupt}}(a)$: $\mathcal{A}_{\mathsf{URS\_Poly}}$ executes this oracle faithfully.
- $\mathcal{O}_{\mathsf{Issue}}(a, \{m_i\}_{i\in[q]})$: Adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$ produces a proof of knowledge for the secret key usk_a (where $a \in \mathsf{Corrupt}$). Adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$ executes the corresponding extractor to extract usk_a, and submits $(m_1, \ldots, m_q, \mathsf{usk}_a)$ to the $\mathsf{URS\_Poly}$ signing oracle $\mathcal{O}^{UF}_{\mathsf{URS\_Poly}}$, generating a signature $(\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$. $\mathcal{A}_{\mathsf{URS\_Poly}}$ discards $(\sigma_1, \tilde{\sigma}_0)$ but stores the credential $(\tilde{\sigma}_1, \tilde{\sigma}_2)$.
- $\mathcal{O}_{\mathsf{ObtIss}}(a, \{m_i\}_{i\in[q]})$: For $a \in \mathsf{Honest}$, adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$ recovers the secret key usk_a previously constructed for identity a, and submits $(m_1, \ldots, m_q, \mathsf{usk}_a)$ to the $\mathsf{URS\_Poly}$ signing oracle $\mathcal{O}^{UF}_{\mathsf{URS\_Poly}}$, generating a signature $(\sigma_1, \tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$. $\mathcal{A}_{\mathsf{URS\_Poly}}$ discards $(\sigma_1, \tilde{\sigma}_0)$ but stores the credential $(\tilde{\sigma}_1, \tilde{\sigma}_2)$.
- $\mathcal{O}_{\mathsf{Show}}(\kappa, \mathcal{I})$: Adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$ can query this oracle for a show on query call κ, made to the oracle $\mathcal{O}_{\mathsf{ObtIss}}$. The κ-th query generated a signature σ on $(m_1, \ldots, m_q, \mathsf{usk}_{a_\kappa})$ via oracle $\mathcal{O}^{UF}_{\mathsf{URS\_Poly}}$. This signature is therefore stored in the $\mathsf{URS\_Poly}$ challenge table T_M. Adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$ may thus execute $\mathsf{URS\_Poly.Derive}$ for the signature σ and message set $\{m_i\}_{i\in\mathcal{I}} \cup \{\mathsf{usk}_{a_\kappa}\}$, receiving the derived signature $(\sigma_1', \tilde{\sigma}_0', \tilde{\sigma}_1', \tilde{\sigma}_2')$, with:
  - $\tilde{\sigma}_2' \leftarrow (\tilde{\sigma}_1')^{r+x+\sum_{i\in[q]} m_i y^i + \mathsf{usk}_{a_\kappa} y^{q+1}}$
  - $\sigma_1' \leftarrow g^r \prod_{j\in\bar{\mathcal{I}}} Y_j^{m_j}$
  - $\tilde{\sigma}_0' \leftarrow (\prod_{i\in\mathcal{I}\cup\{q+1\}} \tilde{Z}_i^r) \cdot \prod_{i\in\mathcal{I}\cup\{q+1\}} \prod_{j\in\bar{\mathcal{I}}} \tilde{W}_{i,j}^{m_j}$.

  This signature has the same distribution as signatures generated by $\mathsf{AC\_Poly.Show}$. Finally $\mathcal{A}_{\mathsf{URS\_Poly}}$ selects $k \xleftarrow{R} \mathbb{F}_p^*$, computes $C = e(Y_{q+1}^k, \tilde{\sigma}_1')$, and uses k and usk_{a_κ} to return a valid u in the interactive protocol.

By assumption, $\mathcal{A}_{\mathsf{AC\_Poly}}$ eventually generates a valid forged $\mathsf{AC\_Poly}$ credential with non-negligible probability. Since $\mathcal{A}_{\mathsf{URS\_Poly}}$ does not abort and can respond to all oracle queries by $\mathcal{A}_{\mathsf{AC\_Poly}}$, $\mathcal{A}_{\mathsf{AC\_Poly}}$ is able to prove possession of the credential for attributes $\{m_i\}_{i\in\mathcal{I}}$. Adversary $\mathcal{A}_{\mathsf{URS\_Poly}}$ executes the extractor on the proof of knowledge for usk, the secret key used in the outputted valid derived forgery $(\sigma_1', \tilde{\sigma}_0', \tilde{\sigma}_1', \tilde{\sigma}_2')$, for messages $\{m_i\}_{i\in\mathcal{I}} \cup \{\mathsf{usk}\}$. By assumption, $\mathsf{usk} \neq \mathsf{usk}_a$ for all $a \in \mathsf{Honest}$, and no element in Attr contains attributes $\{m_i\}_{i\in\mathcal{I}}$ for any corrupt user $a \in \mathsf{Corrupt}$. Hence, for all $a \in \mathsf{Corrupt}$, any credential generated for a either used a secret key $\mathsf{usk}_a \neq \mathsf{usk}$ or included (at least) a message $m_j \notin \{m_i\}_{i\in\mathcal{I}}$. In both cases, $(\sigma'_1, \tilde{\sigma}'_0, \tilde{\sigma}'_1, \tilde{\sigma}'_2)$ is a valid $\mathsf{URS\_Poly}$ signature for messages $\{m_i\}_{i\in\mathcal{I}} \cup \{\mathsf{usk}\}$, which may be used as a forgery by $\mathcal{A}_{\mathsf{URS\_Poly}}$ to win the EUF-CMA URS-Poly security game.

■

In combination, Theorems 9.1 and 9.2 show that $\mathsf{AC\_Poly}$ is EUF-CMA secure unless either the DL assumption in $\mathbb{G}_2$ is invalid or the EUF-CMA security of protocol $\mathsf{URS\_Poly}$ can be broken. This completes the proof of EUF-CMA security for $\mathsf{AC\_Poly}$.

9.3 Security: Anonymity

Anonymity of a credential scheme requires that, during a showing of a derived credential, malicious verifiers and/or issuers are unable to acquire any information on a user, beyond the knowledge that the user possesses a valid credential for the revealed attributes [FHS19]. Ideally, anonymity should hold for colluding issuers and verifiers. In the present context, this notion is formalised by the following security game:

- AC-Poly Anonymity Security Game:
  - Challenger $\mathcal{C}$ executes $\mathsf{AC\_Poly.IssuerKeygen}(1^\lambda, q)$ and provides the public parameters and issuer keys $(\mathsf{sk}_*, \mathsf{pk}_*)$ to the PPT adversary $\mathcal{A}$, along with access to oracles $\mathcal{O}_{\mathsf{Honest}}$, $\mathcal{O}_{\mathsf{Corrupt}}$, $\mathcal{O}_{\mathsf{Obtain}}$, and $\mathcal{O}_{\mathsf{Show}}$. The adversary acts as both issuer and verifier, seeking to break anonymity of the protocol.
  - Leveraging its knowledge of sk_*, $\mathcal{A}$ eventually outputs a set of attributes $\{m_i\}_{i\in\mathcal{I}}$, and two identity indexes a and a', namely:
  $$(a, a', \{m_i\}_{i\in\mathcal{I}}) \leftarrow \mathcal{A}. \tag{156}$$
  - The challenger checks that there exists:
  $$\{a, \{m_i^{(a)}\}_{i\in[q]}\} \subset \mathsf{Attr} \quad \text{and} \quad \{a', \{m_i^{(a')}\}_{i\in[q]}\} \subset \mathsf{Attr}, \tag{157}$$
  such that:
  $$\{m_i\}_{i\in\mathcal{I}} \subseteq \{m_i^{(a)}\}_{i\in[q]} \cap \{m_i^{(a')}\}_{i\in[q]}. \tag{158}$$
  If not, $\mathcal{C}$ aborts.
  - The challenger selects an identity $b \xleftarrow{R} \{a, a'\}$, and engages in the show-verify protocol with $\mathcal{A}$ acting as verifier:
  $$\Big(\mathsf{AC\_Poly.Show}(\mathsf{usk}_b, \mathsf{pk}, \{m_i^{(b)}\}_{i\in[q]}, \mathcal{I}, \sigma), \mathcal{A}(\mathsf{pk}, \{m_i\}_{i\in\mathcal{I}})\Big) \tag{159}$$

– Eventually $\mathcal{A}$ outputs an identity $b_* \in \{a, a'\}$.

– If either a or a' were corrupted during $\mathcal{A}$'s execution, the game aborts (i.e. abort if $a \in \mathsf{Corrupt}$ or $a' \in \mathsf{Corrupt}$).

– Otherwise, accept if $b_* = b$ and reject if $b_* \neq b$.

In this game the adversary acts as both issuer and verifier, and must output two user identities and a set of attributes $\{m_i\}_{i\in\mathcal{I}}$, such that both identities were previously added to Attr. The challenger randomly selects one of the outputted identities and shows a derived credential for the common attributes $\{m_i\}_{i\in\mathcal{I}}$. Adversary $\mathcal{A}$ is tasked with determining which identity was selected by the challenger. The adversary wins the game if (i) neither of the outputted identities was corrupted during execution, and (ii) the adversary successfully determines the identity chosen by the challenger. Hence, anonymity requires that, even if the two identities have different redacted attributes $\{m_j^{(a)}\}_{j\in\bar{\mathcal{I}}} \neq \{m_j^{(a')}\}_{j\in\bar{\mathcal{I}}}$, the adversary cannot distinguish between derived credentials for identical sets of revealed attributes.

Theorem 9.3. *An adversary that can win the AC-Poly anonymity security game, with non-negligible advantage, may be leveraged to break the decisional Diffie-Hellman assumption in* $\mathbb{G}_2$ *with non-negligible advantage.*

Proof. Assume that the PPT adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$ can win the AC-Poly anonymity security game with non-negligible advantage ϵ. Let adversary $\mathcal{A}_{\mathsf{DDH}}$ receive a DDH challenge in $\mathbb{G}_2$, from challenger $\mathcal{C}_{\mathsf{DDH}}$, namely $(\tilde{g}, \tilde{g}^\alpha, \tilde{g}^\beta, \tilde{g}^\gamma) \leftarrow \mathcal{C}_{\mathsf{DDH}}$. Adversary $\mathcal{A}_{\mathsf{DDH}}$ is tasked with determining whether $\gamma = \alpha \cdot \beta$. To accomplish this, $\mathcal{A}_{\mathsf{DDH}}$ act as an AC-Poly anonymity challenger to $\mathcal{A}_{\mathsf{AC\_Poly}}$. $\mathcal{A}_{\mathsf{DDH}}$ executes $\mathsf{AC\_Poly.IssuerKeygen}$ and passes the keys $(\mathsf{sk}_*, \mathsf{pk}_*)$ and public parameters to $\mathcal{A}_{\mathsf{AC\_Poly}}$, as AC-Poly anonymity challenge parameters.

To execute the challenge, $\mathcal{A}_{\mathsf{DDH}}$ randomly selects an identity and assigns it to the user that will ultimately be used for the show-verify interactive component of the challenge. Denote this randomly selected identity as $b \xleftarrow{R} \mathbb{F}_p^*$. To answer oracle queries made by $\mathcal{A}_{\mathsf{AC\_Poly}}$, $\mathcal{A}_{\mathsf{DDH}}$ acts as follows:

- $\mathcal{O}_{\mathsf{Honest}}(a)$: When $\mathcal{A}_{\mathsf{AC\_Poly}}$ calls this oracle for identity a, adversary $\mathcal{A}_{\mathsf{DDH}}$ executes the oracle faithfully if $a \neq b$, and sets the DDH challenge item $\tilde{g}^\alpha$ as the public key if $a = b$, namely $\mathsf{upk}_b = \tilde{g}^\alpha$.

- $\mathcal{O}_{\mathsf{Corrupt}}(a)$: $\mathcal{A}_{\mathsf{URS\_Poly}}$ executes this oracle faithfully if $a \neq b$ and aborts if $a = b$.

- $\mathcal{O}_{\mathsf{Obtain}}(a, \{m_i\}_{i\in[q]})$: Since $\mathcal{A}_{\mathsf{DDH}}$ possesses the secret key usk_a, if $a \neq b$, $\mathcal{A}_{\mathsf{DDH}}$ may execute $\mathsf{AC\_Poly.Obtain}$ and respond to oracle queries. If $a = b$, $\mathcal{A}_{\mathsf{DDH}}$ responds by sending the public key upk_b and a simulated proof of knowledge for α.

- $\mathcal{O}_{\mathsf{Show}}(\kappa, \mathcal{I})$: $\mathcal{A}_{\mathsf{DDH}}$ responds to queries faithfully if the κ-th query was made by $a \neq b$, and simulates a proof of knowledge of α if the κ-th query was made by user b.

Eventually $\mathcal{A}_{\mathsf{AC\_Poly}}$ outputs two identities a, a', and some attributes $\{m_i\}_{i\in\mathcal{I}}$. If $b \notin \{a, a'\}$, $\mathcal{A}_{\mathsf{DDH}}$ aborts. Otherwise $\mathcal{A}_{\mathsf{DDH}}$ selects random field elements k, s, and uses the DDH challenge to set $\tilde{\sigma}'_1 = \tilde{g}^{\beta}$. Then $\mathcal{A}_{\mathsf{DDH}}$ computes:

$$\begin{aligned} \tilde{\sigma}'_2 &\leftarrow (\tilde{\sigma}'_1)^{s+x+\sum_{i\in\mathcal{I}} m_i y^i} \cdot (\tilde{g}^{\gamma})^{y^{q+1}}, \\ \sigma'_1 &\leftarrow g^s, \\ \tilde{\sigma}'_0 &\leftarrow (\tilde{g}^s)^{\sum_{i\in\mathcal{I}\cup\{q+1\}} z^i}, \end{aligned} \tag{160}$$

and simulates knowledge of α. The element k is used in the interactive show-verify protocol.

If $\gamma = \alpha \cdot \beta$, the credential elements $(\sigma'_1, \tilde{\sigma}'_0, \tilde{\sigma}'_1, \tilde{\sigma}'_2)$ have the same distribution as a valid $\mathsf{AC\_Poly}$ credential. This equivalence is explicitly observed by setting:

$$s' = s - \sum_{j\in\bar{\mathcal{I}}} m_j y^j, \tag{161}$$

where s' is uniformly distributed if s is uniformly random. If $\gamma \neq \alpha\cdot\beta$, then γ is a random field element, and therefore $\tilde{\sigma}'_2$ is a random $\mathbb{G}_2$ element. In this case, $\mathcal{A}_{\mathsf{AC\_Poly}}$ cannot succeed with non-negligible advantage since the remaining signature elements $\sigma'_1, \tilde{\sigma}'_0, \tilde{\sigma}'_1$ are all independent of both α and $\{m_j\}_{j\in\bar{\mathcal{I}}}$. Moreover, for $\gamma \neq \alpha \cdot \beta$, the credential elements $\tilde{\sigma}'_1$ and $\tilde{\sigma}'_2$ are not correctly related as γ is not dependent on $\mathsf{usk}_b = \alpha$. Hence, by observing any changes in adversary $\mathcal{A}_{\mathsf{AC\_Poly}}$'s behaviour, adversary $\mathcal{A}_{\mathsf{DDH}}$ can win the DDH challenge with non-negligible probability $2\epsilon/Q_U$, where Q_U bounds the number of honest users.

■

10 Conclusion

Pointcheval and Sanders leveraged the properties of type-3 pairings to construct an efficient digital signature scheme [PS16]. This scheme permits multiple generalisations and has found use in threshold credential issuance protocols [SAB+19] and anonymous credential schemes [San19]. The present work studied a mapping between PS signatures and a related class of polynomial-based signatures, referred to as PolySig. This mapping was leveraged to construct many new signature and credential schemes. In particular, new protocols for multi-message signatures, sequential aggregate signatures, signatures for message commitments, redactable signatures, and unlinkable redactable signatures were presented. A variant redactable anonymous credential scheme was also constructed. Security properties of the new protocols were analysed. All new schemes employ constant-sized secret keys, rather than linear-sized (in the number of messages/attributes). The new signatures therefore achieve similar functionality to their PS-based counterparts but simplify key generation/distribution. Security proofs derived in this work are relevant for recently proposed switched threshold signature schemes [McD20]. Future work could leverage the new signature schemes and explore their utility within e.g. credential issuance protocols.

Acknowledgments

The author thanks A. Rondelet, D. Tebbs, and M. Zajac for helpful discussions and comments on the manuscript, and O. Sanders for a clarifying communication.

References

[AGHO11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 649–666. Springer, 2011.

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-taa. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks*, pages 111–125, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, 2008.

[BBD$^+$10] Christina Brzuska, Heike Busch, Oezguer Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In Jianying Zhou and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 87–104, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 440–456, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 41–55, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BCN$^+$10] Patrick Bichsel, Jan Camenisch, Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Get shorty via group signatures without encryption. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 381–398. Springer, 2010.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 213–229, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[BFG$^+$13] David Bernhard, Georg Fuchsbauer, Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. *Int. J. Inf. Sec.*, 12(3):219–249, 2013.

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography – PKC 2011*, pages 403–422, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BFPV13] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Short blind signatures. *Journal of Computer Security*, 21(5):627–661, 2013.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[CDHK15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 262–288, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[Cha83] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985.

[CL01] Jan Camenisch and Anna Lysyanskaya. An identity escrow scheme with appointed verifiers. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 388–407, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 56–72, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[CPS10] Liqun Chen, Dan Page, and Nigel P. Smart. On the design and implementation of an efficient daa scheme. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application*, pages 223–237, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CPST15] Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical. In Jonathan Katz, editor, *Public-Key Cryptography – PKC 2015*, pages 77–100, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DL78] Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193 – 195, 1978.

[DLST14] Nicolas Desmoulins, Roch Lescuyer, Olivier Sanders, and Jacques Traoré. Direct anonymous attestations with dependent basename opening. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis Askoxylakis, editors, *Cryptology and Network Security*, pages 206–221, Cham, 2014. Springer International Publishing.

[FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, Apr 2019.

[Fis00] Marc Fischlin. A note on security proofs in the generic model. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, pages 458–469, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[Gha16] Essam Ghadafi. Short structure-preserving signatures. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016*, pages 305–321, Cham, 2016. Springer International Publishing.

[GMR87] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen message attack. In David S. Johnson, Takao Nishizeki, Akihiro Nozaki, and Herbert S. Wilf, editors, *Discrete Algorithms and Complexity*, pages 287 – 310. Academic Press, 1987.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113 – 3121, 2008. Applications of Algebra to Cryptography.

[GR04] Steven D. Galbraith and Victor Rotger. Easy decision diffie-hellman groups. *LMS Journal of Computation and Mathematics*, 7:201–218, 2004.

[Gro15] Jens Groth. Efficient fully structure-preserving signatures for large messages. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 239–259, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[HHH$^+$08] Stuart Haber, Yasuo Hatano, Yoshinori Honda, William Horne, Kunihiko Miyazaki, Tomas Sander, Satoru Tezoku, and Danfeng Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification.

In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '08, page 353–362, New York, NY, USA, 2008. Association for Computing Machinery.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie-hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, ANTS-IV, page 385–394, Berlin, Heidelberg, 2000. Springer-Verlag.

[KM07] Neal Koblitz and Alfred Menezes. Another look at generic groups. *Advances in Mathematics of Communications*, 1(1):13–28, 2007.

[LLY13] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Aggregating cl-signatures revisited: Extended functionality and better efficiency. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security*, volume 7859 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2013.

[LMPY16] Benoundefinedt Libert, Fabrice Mouhartem, Thomas Peters, and Moti Yung. Practical "signatures with efficient protocols" from simple assumptions. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, page 511–522, New York, NY, USA, 2016. Association for Computing Machinery.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 74–90, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[LRSW00] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard Heys and Carlisle Adams, editors, *Selected Areas in Cryptography*, pages 184–199, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[McD20] Kristian L. McDonald. Switched threshold signatures from K-private PolyShamir secret sharing. *In preparation*, 2020.

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[NTKK09] Ryo Nojima, Jin Tamura, Youki Kadobayashi, and Hiroaki Kikuchi. A storage efficient redactable signature in the standard model. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio A. Ardagna, editors, *Information Security*, pages 326–337, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016*, pages 111–126, Cham, 2016. Springer International Publishing.

[PS18] David Pointcheval and Olivier Sanders. Reassessing security of randomizable signatures. In Nigel P. Smart, editor, *Topics in Cryptology – CT-RSA 2018*, pages 319–338, Cham, 2018. Springer International Publishing.

[SAB$^+$19] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[San19] Olivier Sanders. Efficient redactable signature and application to anonymous credentials. Cryptology ePrint Archive, Report 2019/1201, 2019.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[Sch90] C. P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[Sco02] Mike Scott. Authenticated id-based key exchange and remote log-in with simple token and pin number. *IACR Cryptology ePrint Archive*, 2002:164, 12 2002.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 256–266, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, pages 216–226, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.