Commuting Signatures and Verifiable Encryption
and an Application to Non-Interactively Delegatable Credentials

Georg Fuchsbauer
École normale supérieure, CNRS - INRIA, Paris, France
http://www.di.ens.fr/~fuchsbau

Abstract

Verifiable encryption allows to encrypt a signature and prove that the plaintext is valid. We introduce a new
primitive called commuting signature that extends verifiable encryption in multiple ways: a signer can encrypt
both signature and message and prove validity; more importantly, given a ciphertext, a signer can create a
verifiably encrypted signature on the encrypted message; thus signing and encrypting commute. We instantiate
commuting signatures using the proof system by Groth and Sahai (EUROCRYPT ’08) and the automorphic
signatures by Fuchsbauer (ePrint report 2009/320). As an application, we give an instantiation of delegatable
anonymous credentials, a powerful primitive introduced by Belenkiy et al. (CRYPTO ’09). Our instantiation
is arguably simpler than theirs and it is the first to provide non-interactive issuing and delegation, which is
a standard requirement for non-anonymous credentials. Moreover, the size of our credentials and the cost of
verification are less than half of those of the only previous construction, and efficiency of issuing and delegation
is increased even more significantly. All our constructions are proved secure in the standard model.

1 Introduction

A verifiably-encrypted-signature scheme [BGLS03] enables a signer to make a digital signature on a message,
encrypt the signature under a third party’s encryption key, and produce a proof asserting that the ciphertext contains
a valid signature. Suppose the message is only available as an encryption. The signer cannot make a signature on
it, as this would contradict the security of the encryption scheme (given two messages and the encryption of one
of them, a signature on the plaintext could be used to decide which message was encrypted). However, the following
does not seem a priori impossible: given an encryption, instead of producing a signature on the plaintext, the signer
produces a verifiably encrypted signature on it.

We show that—surprisingly—such a functionality is feasible and moreover give a practical instantiation of it.
We then use this new primitive to build the first non-interactively delegatable anonymous credential scheme: given
an encrypted public key, a delegator can make an encrypted certificate on the key together with a proof of validity.

Delegatable Anonymous Credentials. Access control that respects users’ privacy concerns is a challenging prob-
lem in security. To gain access to resources, a participant must prove to possess the required credential issued by
an authority. To increase manageability of the system, the authority does usually not issue credentials directly to
each user, but relies on intermediate layers in the hierarchy. Belenkiy et al. [BCC+09] give the following example:
a system administrator issues credentials for webmasters to use his server. The latter are entitled to create forums
and delegate rights to moderators, who in turn can give posting privileges to users.

In the real world delegation of rights is realized by certifying the public key of the delegated user. Consecutive
delegation leads to a credential chain, consisting of public keys and certificates linking them, starting with the
original issuer of the credential and ending with a user, say Alice. To delegate her credential to Bob, Alice simply
extends the length of the chain by one by appending a certificate on Bob’s public key under hers.

Anonymous credentials [Cha85, Dam90, LRSW00, Bra99, CL01, CL02, CL04, BCKL08] aim to provide a
similar functionality while at the same time not revealing information about the user’s identity when obtaining
or showing a credential. However, the goal of reconciling delegatability and anonymity remained elusive—until
recently. Chase and Lysyanskaya [CL06] give delegatable anonymous credentials for which the size of a credential is exponential in its length (i.e., the number of delegations), which makes them impractical. In [BCC+09] Belenkiy et al. introduce a new approach using a non-interactive zero-knowledge (NIZK) proof system [BFM88] with randomizable proofs: a credential is a non-interactive proof of knowledge of a certification chain that can be randomized before being re-delegated or shown; this guarantees anonymity and unlinkability.

The functionality of the system can be sketched as follows: each user holds a secret key which she uses to produce multiple pseudonyms $Nym$. A user $A$ can be known to user $O$ as $Nym_A^O$ and to $B$ as $Nym_A^B$. Given a credential issued by $O$ for $Nym_A^O$, $A$ can transform it into a credential for $Nym_A^B$ and show it to $B$. Moreover $A$ can delegate the credential to user $C$, known to $A$ as $Nym_C^A$. $C$ can then show a credential from $O$ for $Nym_C^O$ to user $D$ (without revealing neither $Nym_C^A$ nor $Nym_C^O$), or redelegate it, and so on.

Delegation preserves anonymity, i.e., delegator and delegatee learn nothing more about each other than their respective pseudonyms. In the instantiation of [BCC+09] (BCCKLS), the delegation protocol is fairly complex and highly interactive—as opposed to (non-anonymous) credentials, where it suffices to know a user’s public key in order to issue or delegate a credential to her. We correct this shortcoming by giving an instantiation of the BCCKLS model that enables non-interactive delegation: pseudonyms are commitments of the public key; given a pseudonym $Nym$, the delegator can produce a credential for the holder of $Nym$ without any interaction, since she can make a proof of knowledge of a signature on a public key given to her as an encryption $Nym$. We note that, as for the BCCKLS instantiation, abuse prevention mechanisms such as anonymity revocation [CL01] or limited show [CHK+06] can be added to our construction.

**Commuting Signatures and Verifiable Encryption.** Our main building block to instantiate non-interactively delegatable anonymous credentials will be a new primitive we call commuting signature which we sketch in the following and formally define in Sect. 4. Assume we have a digital signature scheme and an encryption scheme combined with a proof system with the following properties: given a verification key, a message and a signature on it valid under the key, we can encrypt any subset of \{key, message, signature\}, and make a proof that the plaintexts constitute a triple of a key, a message and a valid signature. We also require that the proof does not leak any more information about the encrypted values besides validity.

For consistency with our instantiation using the Groth-Sahai methodology [GS08], we will say commitment instead of encryption. Note that the commitments we use are extractable, and therefore constitute an encryption scheme (see below). We denote committing to signatures by $Com$ and committing to messages by $Com_M$. Besides allowing to prove validity of committed values, a commuting-signature scheme provides the following additional functionalities (sketched in Figure 1). Note that none of them requires the extraction (decryption) key.

**SigCom.** Given a commitment $C_M$ to a message $M$ and a signing key $sk$, SigCom produces a commitment $c_\Sigma$ to a signature $\Sigma$ on $M$ under $sk$, and a proof $\pi$ that the content of $c_\Sigma$ is a valid signature on the content of $C_M$. 

![Figure 1: Commuting signatures](image-url)
AdPrC ("adapt proof when committing"). Given a commitment $C_M$ to $M$, a signature $\Sigma$ on $M$ and a proof $\bar{\pi}$ of validity of $\Sigma$ on the content of $C_M$, we can make a commitment $c_\Sigma$ to $\Sigma$ using randomness $\rho_\Sigma$ and run algorithm AdPrC on $C_M, \Sigma, \rho_\Sigma$ and $\bar{\pi}$. Its output is a proof $\pi$ that the content of $c_\Sigma$ is a valid signature on the content of $C_M$. AdPrDC ("adapt proof when decommitting") does the converse: given a committed message $C_M$, a committed signature $c_\Sigma$, together with the used randomness $\rho_\Sigma$, and a proof $\pi$, AdPrDC outputs a proof $\bar{\pi}$ of validity of the signature $\Sigma$ on the committed message.

AdPrC$_M$. Analogously we define algorithms for proof adaptation when committing and decommitting to the message. Given a message $M$, a commitment $c_\Sigma$ to a signature on $M$ and a proof of validity $\bar{\pi}$, AdPrC$_M$ transforms the proof to the case when the message is committed as well. AdPrDC$_M$ is given commitments $C_M$ and $c_\Sigma$ to a signature and a message $M$, the randomness $\rho_M$ for $C_M$ and a proof $\pi$. It adapts $\pi$ to a proof $\bar{\pi}$ that the content of $c_\Sigma$ is a valid signature on $M$.

AdPrC$_K$. Finally, we can also adapt proofs when committing or decommitting to the verification key. Given commitments $C_M$ and $c_\Sigma$ to a message and a signature, a proof of validity $\pi$, the verification key $vk$ and randomness $\rho_{vk}$, AdPrC$_K$ outputs a proof $\bar{\pi}$ that the content of $c_\Sigma$ is a signature on the content of $C_M$ valid under the key $vk$ given as a commitment $c_{vk}$ with randomness $\rho_{vk}$. AdPrDC$_K$ is given $(vk, \rho_{vk}, C_M, c_\Sigma)$ and adapts a proof $\pi$ for $(c_{vk}, C_M, c_\Sigma)$ to a proof for $(vk, C_M, c_\Sigma)$.

We require that committing, signing and the functionalities above commute with each other, that is, it does not matter in which order we execute them; e.g. signing a message, committing to the message and the signature and proving validity yields the same as committing to the message and then running SigCom. Thus, the diagram in Fig. 1 commutes. Note that due to the argument given in the beginning there cannot exist a functionality XX that given a commitment $C_M$ to a message $M$ and a secret key $sk$ outputs a signature $\Sigma$ on $M$.

**Instantiating Commuting Signatures.** In [Fuc09], Fuchsbauer gave the first efficient implementation of blind signatures [Cha82] with round-optimal issuing [Fis06]: this means that the user who wants to obtain a blind signature sends one message to signer, who replies with one message from which the user can construct the blind signature. The scheme can be sketched as follows: the user randomizes the message by multiplying it with a random term, makes an (extractable) commitment to the message and the randomness and adds a witness-indistinguishable (WI) proof that the commitments contain the correct values. The signer therefore learns nothing about the message. Using the randomized message, the signer can fabricate a “pre-signature”, from which, knowing the randomizer, the user can retrieve an actual signature. To prevent the signer from linking the resulting signature to the signing session, the actual blind signature is a proof of knowledge (PoK) of the signature. The PoK consists of extractable commitments to the signature components and a WI proof that the committed values satisfy the signature-verification equation on the message, in other words, a verifiably encrypted signature.

We observe that the values sent from the user to the signer can be seen as a commitment to (or an encryption of) the message. We show that this commitment can be used by the signer to directly construct a proof of knowledge of a signature on the committed message (that is, extractable commitments to the signature components and a proof that the committed values constitute a valid signature on the committed message). As in [Fuc09], the commitments and WI proofs are instantiated with the Groth-Sahai methodology for committing to elements from a bilinear group and constructing proofs that they satisfy pairing-product equations. These commitments are extractable, thus the extraction key acts as the decryption key and witness indistinguishability implies semantic security (cf. Sect. 3.1). We will use the notions encryption and extractable commitment interchangeably. An extractable commitment to a signature together with a proof of validity is a verifiably encrypted signature (VES) and can also be interpreted as a proof of knowledge of a signature (since by decryption, the signature can be extracted). Our instantiation of commuting signatures is given in Sect. 6.

**Instantiating Delegatable Anonymous Credentials.** Belenkiy et al. [BCC+09] show that Groth-Sahai proofs can be randomized and combine with an authentication scheme for secret keys to construct delegatable credentials. A pseudonym Nym is a commitment to the user’s secret key and a credential is a proof of knowledge of an authentication chain. Such a proof consists of commitments to secret keys, commitments to authenticators between the keys, and proofs of validity. To issue or delegate, the issuer and the user jointly compute a proof of knowledge of an authenticator on the content of the user’s pseudonym. In the case of delegation, the issuer preempts
her own credential, which she randomizes before. The authors note that secret keys cannot be extracted from the commitments, and that an adversary against the authentication scheme must be allowed to ask for authenticators on as well as under the attacked key. They therefore give an \emph{F-unforgeable certification-secure} authentication scheme.

We avoid this notion and interactivity of delegation by replacing the authenticators on secret keys by signatures on public keys, in particular we use the automorphic signatures from \cite{Fuc09}. (Automorphic signatures are Groth-Sahai compatible signatures whose verification keys lie in the message space.) A credential is then a chain of \emph{public keys} and \emph{certificates} (as in the non-anonymous case), which are all given as commitments completed with proofs of validity. Commuting signatures enable non-interactive delegation (and issuing, which is a special case of delegation): given a pseudonym (i.e., a commitment to the public key) of a user, the issuer can produce a commitment $c_\Sigma$ to a signature on the committed user key and a proof $\pi$ of validity using $\text{SigCom}$. If it is a delegation, the issuer then randomizes her own credential $\text{cred}$, yielding a credential $\text{cred}'$ on a new pseudonym $\text{Nym}'$ that is unlinkable to $\text{cred}$. Finally, running $\text{AdPrC}_K$, she adapts the proof $\pi$ to a proof $\hat{\pi}$ of validity of the content of $c_\Sigma$ on the content of the user pseudonym under the content of the issuer’s new pseudonym $\text{Nym}'$. The credential for the user is then $\text{cred}' \parallel \text{Nym}' \parallel (c_\Sigma, \hat{\pi})$.

Replacing the authenticators from \cite{BCC+09}, which consist of 11 group elements and are verified by 8 pairing-product equations (PPE), with automorphic signatures (consisting of 5 group elements and satisfying 3 PPEs) more than doubles efficiency of the scheme. More importantly, our delegation (and issuing) protocol outperforms theirs significantly (see Sect. 5.3 for a more detailed comparison).

Automorphic signatures were combined with Groth-Sahai proofs in \cite{Fuc09} to construct \emph{anonymous proxy signatures} \cite{FP08}. This primitive is related to anonymous credentials in that it considers proving rights in an anonymous way; but it does not achieve mutual anonymity between the delegator and the delegated user. Note that if in our credential scheme we give the extraction key for the commitments to a tracing authority, and if we define a \emph{proxy signing algorithm} which works like delegation but produces a committed signature on a \emph{clear} message rather than a committed user key, we get an instantiation of anonymous proxy signatures with mutually anonymous delegation, a feature not considered so far.

**Overview.** We start with giving an overview of our notation. In Sect. 3, we define extractable commitments and randomizable witness-indistinguishable proofs for them. We also define digital signatures and discuss how they can be combined with extractable commitments and proofs to verifiably encrypted signatures. In Sect. 4 we formally define commuting signatures and give some immediate black-box results, such as blind signatures. In Sect. 5 we recall the model for delegatable credentials from \cite{BCC+09} and describe our instantiation providing non-interactive delegation. We prove security and conclude with a comparison to the BCCKLS instantiation in Sect. 5.3. In Sect. 6, we give the instantiations of the primitives defined in Sect. 3: Groth-Sahai proofs and automorphic signatures. In Sect. 7 we state and prove 5 lemmas about properties of Groth-Sahai proofs which are used in Sect. 8, where we instantiate our commuting-signature scheme. In Sect. 9 we give a variant of the automorphic signatures from \cite{Fuc09} which enables a more efficient instantiation of delegatable credentials. In Sect. 10 we discuss some issues of simulatability of Groth-Sahai proofs that arise when they are used to instantiate delegatable credentials satisfying a simulation-based anonymity definition. The discussion concerns the BCCKLS instantiation as well as ours. Finally, in Appendix C, we give some complementary results. In particular, we describe how to extend commuting signatures when several messages are to be signed at once.

## 2 Notation

Since we are going to combine quite a few concepts we give a guideline on notation. We tried to stick to our framework, but deviated sometimes for the sake of consistency with other work such as Groth and Sahai’s.

- Capital Roman letters denote elements of a bilinear group. \textit{Diffie-Hellman pairs} of group elements are mostly two consecutive letters of the alphabet.

- Lower-case Roman letters denote integers. Mostly they correspond to the logarithm of the corresponding capital letter in a common basis, e.g. $M = G^m$. 

• Greek letters denote the randomness used in the commitments to a group element that is denoted by the corresponding Roman letter, e.g. \( c_M = \text{Com}(ck, M, \mu) \).

• \( \mathcal{M}, \mathcal{V}, \mathcal{R} \) and \( \mathcal{C} \) denote spaces for messages, values, randomness and commitments, respectively; \( \mathcal{E} \) denotes a class of equations, \( \mathcal{H} \) a hash function, \( G \) denotes a group, \( \mathbb{Z} \) denotes integers and \( \mathbb{N} \) non-negative integers.

• Special cases:
  
  - \( p \) denotes a prime, \( e \) denotes a bilinear map (pairing) and \( E \) denotes a generic equation;
  
  - \( \phi \) and \( \theta \) denote the components of Groth-Sahai proofs, whereas \( \pi \) denotes generic proofs;
  
  - \( \vec{u}, \vec{v} \) and their components \( u_{ij} \) and \( v_{ij} \) are the keys for Groth-Sahai commitments;
  
  - \( c, d \) and \( C \) denote commitments;
  
  - \( t_T \) denotes an element from the target group \( G_T \);
  
  - \( \Gamma \) and \( Z \) denote matrices with entries \( \gamma_{ij} \) and \( z_{ij} \) in \( \mathbb{Z}_p \), respectively.

• In a bilinear group, we denote the group operation by “\( \cdot \)”. For vectors of group elements, “\( \circ \)” denotes applying the group operation componentwise.

• By “\( := \)”, we denote either a definition—with the definiens on the right-hand side (RHS) and the definiendum on the left-hand side (LHS)—or an assignment of the value on the RHS to the variable on the LHS.

• “\( \leftarrow \)” denotes a random assignment. If the RHS is a set it denotes choosing a value from it uniformly and assigning it to the LHS. If the RHS is a probabilistic algorithm it denotes choosing its random tape uniformly and assigning the outcome to the LHS.

### 3 Preliminaries

We recall the definitions and security requirements of a number of primitives from the literature, which we will combine to a system of commuting signatures and verifiable encryption in Sect. 4.

#### 3.1 Commitments

A (non-interactive) randomizable extractable commitment scheme \( \text{Com} \) is composed of the algorithms Setup, \( \text{Com} \), RdCom, ExSetup, Extr, and WISetup, which define \( \mathcal{V} \), the space of “committable” values, \( \mathcal{R} \), the randomness space and \( \mathcal{C} \) the space of commitments. Setup outputs a commitment key \( ck \), and \( \text{Com} \), on inputs \( ck \), a message \( M \in \mathcal{V} \) and randomness \( \rho \in \mathcal{R} \) outputs a commitment \( c \in \mathcal{C} \). ExSetup outputs \( (ck, ek) \), where \( ck \) is distributed as the output of Setup, and \( ek \) is the extraction key. We require the following:

- The scheme is perfectly binding, i.e., for any commitment \( c \in \mathcal{C} \) there exists exactly one \( M \in \mathcal{V} \) s.t. \( c = \text{Com}(ck, M, \rho) \) for some \( \rho \). Moreover, \( \text{Extr}(ek, c) \) extracts that value \( M \) from \( c \).

- The scheme is computationally hiding, in particular, WISetup outputs keys \( ck^* \) that are computationally indistinguishable from those output by Setup, and which generate perfectly hiding commitments, i.e., for every \( c \) and \( M \in \mathcal{V} \) there exists a \( \rho \in \mathcal{R} \) s.t. \( c = \text{Com}(ck^*, M, \rho) \).

- The scheme is randomizable, i.e., \( \text{RdCom} \) takes as input a commitment \( c \) and fresh randomness \( \rho' \leftarrow \mathcal{R} \) and outputs a randomized commitment \( c' \). If \( \rho' \) is chosen uniformly from \( \mathcal{R} \) then \( c' \) is distributed as \( \text{Com}(ck, M, \rho) \) where \( \rho \) is picked uniformly from \( \mathcal{R} \).

(In particular we have \( \text{RdCom}(ck, \text{Com}(ck, M, \rho), \rho') = \text{Com}(ck, M, \rho + \rho') \).

A commitment scheme with the above properties is actually a lossy encryption scheme [BHY09]; in particular, it satisfies the IND-CPA definition of semantic security.\(^1\)

\(^1\)Consider the security game for IND-CPA for encryption schemes. The challenger creates a key pair, gives the encryption key to the adversary, who outputs two messages, gets an encryption of one of them and has to guess which one. Replacing the key by a “lossy” encryption key output by WISetup is indistinguishable; and then the encryption is independent of the message.
We write Com also when we commit to a vector in \( \mathcal{V}^n \): if \( M = (M_1, \ldots, M_n) \) and \( \rho = (\rho_1, \ldots, \rho_n) \) then \( \text{Com}(ck, M, \rho) := (\text{Com}(ck, M_1, \rho_1), \ldots, \text{Com}(ck, M_n, \rho_n)) \). Likewise, we define \( \text{Extr}(ck, (c_1, \ldots, c_n)) := (\text{Extr}(ck, c_1), \ldots, \text{Extr}(ck, c_n)) \).

### 3.2 Proofs for Committed Values

A randomizable witness-indistinguishable proof system **Proof** for a commitment scheme Com for a class \( \mathcal{E} \) of equations \( E \) consists of the algorithms \( \text{Prove}, \text{Verify} \) and RdProof. Given values \( M_1, \ldots, M_n \in \mathcal{V} \) satisfying an equation \( E \in \mathcal{E} \), the algorithm \( \text{Prove} \), on input \( ck, (M_1, \ldots, M_n) \) and \( \rho_1, \ldots, \rho_n \in \mathcal{R} \), outputs a proof \( \pi \). On inputs \( ck, E, c_1, \ldots, c_n \) and \( \pi \), \( \text{Verify} \) outputs 0 or 1, indicating acceptance or rejection of a proof. We require that the system satisfies the following:

- **Completeness.** For all \((M_1, \ldots, M_n)\) satisfying \( E \), and \( \rho_1, \ldots, \rho_n \in \mathcal{R} \) we have
  \[
  \text{Verify}(ck, E, \text{Com}(ck, M_1, \rho_1), \ldots, \text{Com}(ck, M_n, \rho_n), \text{Prove}(ck, E, (M_1, \rho_1), \ldots, (M_n, \rho_n))) = 1.
  \]

- **Soundness.** Let \((ck, ek) \leftarrow \text{ExSetup}, E \in \mathcal{E} \), and \( c_1, \ldots, c_n \in \mathcal{C} \). If \( \text{Verify}(ck, E, c_1, \ldots, c_n, \pi) = 1 \) for some \( \pi \), then letting \( M_i := \text{Extr}(ek, c_i) \), we have that \((M_1, \ldots, M_n)\) satisfy \( E \).

- **Witness indistinguishability.** Let \( c^* \leftarrow \text{WISetup} \), and \( M_1, \ldots, M_n, M'_1, \ldots, M'_n \in \mathcal{V} \) be such that both \((M_1, \ldots, M_n)\) and \((M'_1, \ldots, M'_n)\) satisfy an equation \( E \). Let \( \rho_1, \rho_2, \rho'_1, \ldots, \rho'_n \in \mathcal{R} \) be such that for all \( i \), \( \text{Com}(ck^*, M_i, \rho_i) = \text{Com}(ck^*, M'_i, \rho'_i) \). Then the outputs of \( \text{Prove}(ck^*, E, (M_1, \rho_1), \ldots, (M_n, \rho_n)) \) and \( \text{Prove}(ck^*, E, (M'_1, \rho'_1), \ldots, (M'_n, \rho'_n)) \) are equally distributed.

- **Randomizability.** Given commitments \( c_1, \ldots, c_n \), a proof \( \pi \) for \((c_1, \ldots, c_n)\) and \( E \), and \( \rho_1, \ldots, \rho'_n \in \mathcal{R} \), algorithm RdProof outputs a proof \( \pi' \) for the randomized commitments \( c'_i := \text{RdCom}(ck, c_i, \rho'_i) \) for all \( i \), let \( M_i \) be the value committed in \( c_i \) and let \( \rho_i \) be such that \( c_i = \text{Com}(ck, M_i, \rho_i) \). If \( \rho'_1, \ldots, \rho'_n \) are chosen uniformly, then \( \pi' \) and \((c'_i)_{i=1}^n\) are distributed as the output of \( \text{Prove}(ck, E, (M_1, \rho_1), \ldots, (M_n, \rho_n)) \) and \( \text{Com}(ck, M_i, \rho_i) \) with \( \rho_i \) chosen uniformly from \( \mathcal{R} \).

If \( E \) is the conjunction of equations \( E_1, \ldots, E_k \) over variables \( M_1, \ldots, M_n \) then for \( M = (M_1, \ldots, M_n) \) and \( \rho = (\rho_1, \ldots, \rho_n) \) we define \( \text{Prove}(ck, E, (M, \rho)) := (\text{Prove}(ck, E_1, (M_i, \rho_i))_{i=1}^n) \). For \( \pi = (\pi_1, \ldots, \pi_k) \) we define \( \text{Ver}(ck, E, (c_i)_{i=1}^n, \pi) := \bigwedge_{j=1}^k \text{Ver}(ck, E_j, (c_i)_{i=1}^n, \pi_j) \).

### 3.3 Digital Signatures

A digital signature scheme Sig consists of the following algorithms: Setup\(_S\) outputs public parameters pp and defines a message space \( \mathcal{M} \). On input pp, KeyGen\(_S\) outputs a pair \((vk, sk)\) of verification and signing key. Sign\((sk, M)\), for \( M \in \mathcal{M} \) outputs a signature \( \Sigma \), which is verified by \( \text{Ver}(vk, M, \Sigma) \). We require that Sig satisfies the following:

- **Strong unforgeability (under chosen message attack).** No probabilistic polynomial-time (p.p.t.) adversary, given \( vk \) and an oracle for adaptive signing queries on messages of its choice can output a pair \((M, \Sigma)\) s.t. \( \text{Ver}(vk, M, \Sigma) = 1 \) and \((M, \Sigma) \neq (M_i, \Sigma_i)\) for all \( i \); where \( M_i \) are the queried messages and \( \Sigma_i \) the oracle responses.

- **Compatibility with Com and Proof.** The verification keys and signatures are composed of values in \( \mathcal{V} \), the value space of Com, and signature verification consists of checking equations from \( \mathcal{E} \), the class of equations for Proof.

For our application in Sect. 5 we require furthermore that Sig is automorphic, that is, besides being compatible, its verification keys have to lie in \( \mathcal{M} \).
### 3.4 Verifiably Encrypted Signatures

The triple \((\text{Com}, \text{Proof}, \text{Sig})\) constitutes a verifiable encryption scheme satisfying the definitions of Rückert and Schröder [RS09], who revisited those of Boneh et al. [BGLS03]—if a proof that a committed signature satisfies \(\text{Ver}(\text{vk}, M, \cdot)\) can be simulated (see Sect. 10). This is the case for our instantiations\(^2\), given in Sect. 6.

**Definition 1** (Verifiably encrypted signatures (VES)). A verifiably encrypted signature scheme is defined as the tuple \((\text{Kg}, \text{AdjKg}, \text{Sig}, \text{Vf}, \text{Create}, \text{VesVf})\). \(\text{Kg}\) outputs a signature key pair \((\text{vk}, \text{sk})\), \(\text{Sig}\) and \(\text{Vf}\) produce and verify signatures. \(\text{AdjKg}\) outputs a key pair \((\text{apk}, \text{ask})\) for the adjudicator. \(\text{Create}(\text{sk}, \text{apk}, M)\) returns a VES \(\omega\) which is verified by \(\text{VesVf}(\text{apk}, \text{vk}, \omega, M)\) and \(\text{Adj}(\text{ask}, \text{apk}, \text{vk}, \omega, M)\) returns a signature \(\sigma\) on \(M\) under \(\text{vk}\). The security notions from [RS09] are the following:

- **Unforgeability** means that no adversary given the public keys and access to a Create and Adj oracle can output a VES on a message \(M\) that it has never queried to its oracles.
- **Abuse freeness** states that no malicious adjudicator provided with a Create oracle can output a valid VES for a message it never queried.
- **Extractability** means that no malicious signer that can create its own \(\text{vk}\) and has access to a Adj oracle can output a valid VES from which cannot be extracted a valid signature.
- **Opacity** means that no adversary given \(\text{vk}\) and \(\text{ck}\) and access to oracles Create and Adj for messages of its choice, can output a valid pair \((M, \Sigma)\) if it has never queried Adj on \(M\).

**A Straightforward Instantiation.** Based on \((\text{Com}, \text{Proof}, \text{Sig})\) we instantiate a VES scheme as follows: we define the signer’s key generation, the adjudicator’s key generation, signing and verification as \(\text{Kg} := \text{KeyGen}_\Sigma\), \(\text{AdjKg} := \text{ExSetup}\), \(\text{Sig} := \text{Sign}\), \(\text{Vf} := \text{Ver}\). \(\text{Create}(\text{sk}, \text{ck}, M)\) creates a VES on \(M\) by setting \(\Sigma := \text{Sign}(\text{sk}, M)\), choosing \(\rho \leftarrow \mathcal{R}\) and returning \(\omega := (c_\Sigma := \text{Com}(\text{ck}, \Sigma, \rho), \bar{\pi} \leftarrow \text{Prove}(\text{ck}, \text{Vf}_\text{ver}(\text{vk}, M, \cdot), (\Sigma, \rho)))\). Verification \(\text{VesVf}(\text{ck}, \text{vk}, (c_\Sigma, \bar{\pi}), M)\) is defined as \(\text{Verf}(\text{ck}, \text{Vf}_\text{ver}(\text{vk}, M, \cdot), (c_\Sigma, \bar{\pi}))\). Finally, \(\text{Adj}(\text{ek}, \text{ck}, \text{vk}, \omega, M)\) checks if \(\omega = (c_\Sigma, \bar{\pi})\) is valid and if so returns \(\text{Extr}(\text{ek}, c_\Sigma)\).

The scheme \((\text{Kg}, \text{AdjKg}, \text{Sig}, \text{Vf}, \text{Create}, \text{VesVf}, \text{Adj})\) satisfies the security notions from Def. 1 The first three notions are reduced to unforgeability of \(\text{Sig}\) and soundness of \(\text{Proof}\) in a straightforward manner; note in particular that Groth-Sahai proofs are **perfectly** sound, so no adversary even when given the extraction key can make a proof of a false statement.

Opacity can be proved by a reduction with a security loss that is exponential in the number of Adj calls:\(^3\) Let Game 0 be the original game. In Game 1 we abort if the adversary makes an Adj query for a message it never queried Create for; or if it makes an Adj query for \((c_\Sigma, \bar{\pi})\) such that the committed value \(\Sigma\) was never used to answer one of the Create queries. By strong unforgeability of \(\text{Sig}\) and soundness of \(\text{Proof}\), the probability of aborting is negligible. In Game 2, when queried \(\mathcal{O}_{\text{Adj}(\text{ek}, \text{ck}, \text{vk}, \cdot)}((c_\Sigma, \bar{\pi}), M)\), instead of extracting the signature committed to in \(c_\Sigma\), we return the signature produced when answering \(\mathcal{O}_{\text{Create}(\text{sk}, \text{ck}, \cdot)}(M)\); if there have been more such calls then we guess randomly. (If, in the worst case, the adversary queries Create \(q_C\) times and Adj \(q_A\) times on the same message then the probability of correctly simulating Game 2 is \(1/q_C^{q_A}\).) Game 3 is Game 2 but replacing \(\text{ck}\) by \(\text{ck}^*\) output by WISetup. Game 3 can now be simulated by a challenger playing the unforgeability game against \(\text{Sig}\); Given the trapdoor for Groth-Sahai proofs in the WI setting, the challenger simulates the commitments and proofs to answer Create queries, i.e., without using signatures. When queried Adj on a message \(M\), it asks its own Sign oracle for a signature on \(M\) and returns it (or returns a signature it had already returned, depending on the guess). A successful adversary outputs a signature on \(M\) for which it has never queried Adj (and thus never made the challenger query Sign for it) and can therefore be used to break strong unforgeability of \(\text{Sig}\).

**A Fully Secure Instantiation.** The security reduction for opacity can be made tight if outputs of Create are **non-malleable** (i.e., from an \(\omega\) returned by Create on \(M\), one cannot produce a different valid \(\omega'\) for \(M\)):

---

\(^2\)Groth-Sahai proofs for pairing-product equations can be simulated if the equations only contain elements from \(G_1\) and \(G_2\) but not from \(G_T\). This is the case for the equations in (9), which constitute Ver of our instantiation.

\(^3\)If the adversary is only allowed a constant number of Adj queries, this suffices (see also Remark 1).
adversary can then make Adj queries only for ω’s received from a Create query, and in the reduction the challenger need not guess the correct signature for a certain message. Non-malleability can be achieved by replacing ω by \((ω, \text{Sign}(sk, ω))\) in the definition of Create and adding a check of the second component to VesVf. (This relies on the fact that \text{Sig} is strongly unforgeable.)

**Remark 1.** In our application to delegatable credentials, we require somewhat different properties from the triple \((\text{Com}, \text{Proof}, \text{Sig})\). On the one hand, we do not require opacity, since no adversary can query extraction of committed values—the exponential reduction of the straightforward instantiation is thus irrelevant.

On the other hand, we require that the verification keys are in \(M\) (i.e., \text{Sig} is automorphic) and that we can commit to messages and verification keys (for which we will introduce a commitment scheme \(\text{Com}_M\)) and prove validity of an encrypted signature, possibly on an encrypted message or under an encrypted key; we also require that two verifiably encrypted signatures are indistinguishable. The last property is implied by witness-indistinguishability of \text{Proof}, and is not required for VES.

### 4 Commuting Signatures and Verifiable Encryption

#### 4.1 Definition

Recall the primitives introduced in Sect. 3. Let \(\text{Com} = (\text{Setup}, \text{Com}, \text{RdCom}, \text{ExSetup}, \text{Extr}, \text{WISetup})\) be an extractable commitment scheme with value space \(V\); let \(\text{Proof} = (\text{Prove}, \text{Verify}, \text{RdProof})\) be a randomizable WI proof system for \(\text{Com}\); let \(\text{Sig} = (\text{Setup}_S, \text{KeyGen}_S, \text{Sign}, \text{Ver})\) be an strongly unforgeable signature scheme with message space \(M\) that is compatible with \((\text{Com}, \text{Proof})\). We extend \((\text{Com}, \text{Proof}, \text{Sig})\) by the following functionalities presented in the introduction and formally defined in Def. 2:

- \(\text{Com}_M\) is a randomizable extractable commitment scheme whose message space is that of \text{Sig}.
- \(\text{AdPrC}\) and \(\text{AdPrC}_M\) are algorithms that, given a message/signature pair of which one is verifiably encrypted, produce a proof of validity when both are encrypted.
- \(\text{AdPrDC}\) and \(\text{AdPrDC}_M\) are algorithms that, given a verifiably encrypted message/signature pair and the randomness for one of them, return an adapted proof; in particular, \(\text{AdPrDC}\) returns a proof that a signature is valid on a committed message and \(\text{AdPrDC}_M\) returns a proof that a committed signature is valid on a given message.
- \(\text{SigCom}\) takes a \(\text{Com}_M\) commitment and a signing key, and produces a verifiably encrypted signature on the committed value.
- \(\text{AdPrC}_K\) is given a proof of validity for a committed signature and a committed message and adapts it to a proof for when the verification key is also committed. \(\text{AdPrDC}_K\), given the randomness of the committed verification key adapts a proof to when the key is given in the clear.

**Definition 2.** A system of commuting signatures and verifiable encryption consists of an extractable commitment scheme \(\text{Com}\), a (randomizable) WI proof system \(\text{Proof}\) for \(\text{Com}\), a compatible signature scheme \(\text{Sig}\) and the functionalities \(\text{Com}_M, \text{AdPrC}, \text{AdPrDC}, \text{AdPrC}_M, \text{AdPrDC}_M, \text{AdPrC}_K, \text{AdPrDC}_K, \text{SigCom}, \text{SmSigCom}\) defined below.

\(\text{Com}_M\) On input \(pp = (ck, pp_S)\) returned by Setup and \(\text{Setup}_S\), respectively, a message \(M \in M\) and \(μ \in \mathcal{R}_M\), algorithm \(\text{Com}_M\) outputs a commitment \(C \in C_M\), the space of commitments. \(\text{RdCom}_M\) takes inputs \(C, μ' \leftarrow \mathcal{R}_M\) and outputs a randomized commitment \(C'\). On input ek output by \(\text{ExSetup}\), and \(C, \text{Extr}_M\) outputs the committed value \(M\).

We require that \(\text{Com}_M := (\text{Setup}, \text{Com}_M, \text{RdCom}_M, \text{ExSetup}, \text{Extr}_M, \text{WISetup})\) is a randomizable extractable commitment scheme that is perfectly binding and computationally hiding as defined in Sect. 3.1. Moreover, it must be compatible with \(\text{Proof}\), i.e., \(M \subseteq V\), \(\text{Prove}\) and \(\text{RdProof}\) accept inputs from \(\mathcal{R}_M\) and \(\text{Verify}\) accepts \(\text{Com}_M\) commitments as inputs.

In the following we assume that \(ck \leftarrow \text{Setup}, pp_S \leftarrow \text{Setup}_S, (vk, sk) \leftarrow \text{KeyGen}_S(pps), M \in M\) and \(μ \in \mathcal{R}_M\).
AdPrC$(ck, vk, C, (\Sigma, \rho), \hat{\pi})$. If $\text{Verify}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), C, \hat{\pi}) = 1$ then the algorithm outputs $\pi$ which is distributed as

$$\left[ \text{Prove}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), (M, \mu), (\Sigma, \rho)) \right],$$

where $M$ and $\mu$ are such that $C = \text{Com}_M(ck, M, \mu)$.

AdPrDC$(ck, vk, C, (\Sigma, \rho), \pi)$. If $\text{Verify}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), C, \text{Com}(ck, \Sigma, \rho), \pi) = 1$, the algorithm outputs $\bar{\pi}$ which is distributed as

$$\left[ \text{Prove}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), (M, \mu)) \right],$$

where $M$ and $\mu$ are such that $C = \text{Com}_M(ck, M, \mu)$.

AdPrC$_M$(ck, vk, (M, $\mu$), $\mathbf{c}_\Sigma$, $\bar{\pi}$). If $\text{Verify}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), C, \text{Com}_M(ck, M, \mu), \mathbf{c}_\Sigma, \bar{\pi}) = 1$, then it outputs $\pi$ which is distributed as

$$\left[ \text{Prove}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), (M, \mu), (\Sigma, \rho)) \right],$$

where $\Sigma$ and $\rho$ are such that $\mathbf{c}_\Sigma = \text{Com}(ck, \Sigma, \rho)$.

AdPrDC$_M$(ck, vk, (M, $\mu$), $\mathbf{c}_\Sigma$, $\pi$). If $\text{Verify}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), \text{Com}_M(ck, M, \mu), \mathbf{c}_\Sigma, \pi) = 1$, the algorithm outputs $\bar{\pi}$ which is distributed as

$$\left[ \text{Prove}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), (M, \mu), (\Sigma, \rho)) \right],$$

where $\Sigma$ and $\rho$ are such that $\mathbf{c}_\Sigma = \text{Com}(ck, \Sigma, \rho)$.

AdPrC$_K$(ck, (vk, $\xi$), C, $\mathbf{c}_\Sigma$, $\pi$). If $\text{Verify}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), C, \mathbf{c}_\Sigma, \pi) = 1$, the algorithm outputs $\bar{\pi}$ which is distributed as

$$\left[ \text{Prove}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), (vk, \xi), (M, \mu), (\Sigma, \rho)) \right],$$

where $M, \mu, \Sigma$ and $\rho$ are such that $C = \text{Com}_M(ck, M, \mu)$ and $\mathbf{c}_\Sigma = \text{Com}(ck, \Sigma, \rho)$.

AdPrDC$_K$(ck, (vk, $\xi$), C, $\mathbf{c}_\Sigma$, $\bar{\pi}$). If $\text{Verify}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), C, (vk, \xi), C, \mathbf{c}_\Sigma, \bar{\pi}) = 1$, the algorithm outputs $\pi$ which is distributed as

$$\left[ \text{Prove}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), (M, \mu), (\Sigma, \rho)) \right],$$

where $M, \mu, \Sigma$ and $\rho$ are such that $C = \text{Com}_M(ck, M, \mu)$ and $\mathbf{c}_\Sigma = \text{Com}(ck, \Sigma, \rho)$.

SigCom$(ck, sk, C)$. If $C \in \mathcal{C}_M$ then the algorithm outputs a commitment to a signature and a proof of validity ($\mathbf{c}_\Sigma, \pi$) which is distributed as

$$\left[ \Sigma \leftarrow \text{Sign}(sk, M); \ \rho \leftarrow \mathcal{R} : (\text{Com}(ck, \Sigma, \rho), \text{Prove}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), (M, \mu), (\Sigma, \rho))) \right],$$

where $M$ and $\mu$ are such that $C = \text{Com}_M(ck, M, \mu)$.

SmSigCom$(ck, ek, vk, C, \Sigma)$. Assume $(ck, ek) \leftarrow \text{ExSetup}$. If $\text{Ver}(vk, \text{Extr}_M(ek, C), \Sigma) = 1$, then the algorithm outputs $(\mathbf{c}_\Sigma, \pi)$ which is distributed as

$$\left[ \rho \leftarrow \mathcal{R} : (\text{Com}(ck, \Sigma, \rho), \text{Prove}(ck, E_{\text{Ver}}(vk, \cdot, \Sigma), (M, \mu), (\Sigma, \rho))) \right],$$

where $M$ and $\mu$ are such that $C = \text{Com}_M(ck, M, \mu)$.

**Remark 2.** When we verify a signature $\Sigma$ on a message $M$ running $\text{Ver}(vk, M, \Sigma)$, we implicitly assume that $\text{Verify}$ also checks whether $M \in M$. Analogously, we assume that when verifying a proof of validity by running $\text{Verify}$ on $E_{\text{Ver}}$ and $C$; it checks whether $C \in \mathcal{C}_M$, too.

Def. 2 implies that running $\text{Com}_M$ on $M$ and then $\text{SigCom}$ yields the same output as running $\Sigma \leftarrow \text{Sign}(sk, M)$ and then $\text{Com}_M$ on $M$, $\text{Com}$ on $\Sigma$ and $\text{Prove}$ for $E_{\text{Ver}}(vk, \cdot, \Sigma)$; or running $\text{Sign}$, then $\text{Com}_M$ on $M$ and $\text{Prove}$ for $E_{\text{Ver}}(vk, \cdot, \Sigma)$; and then $\text{Com}$ on $\Sigma$ and $\text{AdPrC}$; or running $\text{Sign}$, then $\text{Com}$ on $\Sigma$ and $\text{Prove}$ for $E_{\text{Ver}}(vk, M, \cdot)$, and then $\text{Com}_M$ on $M$ and $\text{AdPrC}_M$. And similar statements hold for sequences of algorithm executions including decommitments and proof adaptation. This means that the diagram in Fig. 1 commutes.

Note that $\text{SmSigCom}$ is not trivial: given $ek$ it might recover the message $M$ but not the randomness $\mu$ used for $C$. The difference to $\text{AdPrC}$ is that $\text{SmSigCom}$ does not get $\bar{\pi}$ as input but $ek$ instead.
4.2 Black-Box Results

Security notions for commuting signatures follow from the security of the used building blocks and the fact that all algorithms perfectly commute.

**Unforgeability.** Extractability of $\text{Com}$ and $\text{Com}_M$, perfect soundness of $\text{Proof}$, strong unforgeability of $\text{Sig}$ and commutativity of $\text{SigCom}$ with signing and verifiably encrypting implies unforgeability, defined as the intractability for a p.p.t. adversary $A$ of winning the following game:

- Run $(\ck, \ek) \leftarrow \text{ExSetup}$ and $(\vk, \sk) \leftarrow \text{KeyGen}_\Sigma(\text{Setup}_\Sigma)$; provide $A$ with $(\ck, \ek, \vk)$ and access to a $\text{SigCom}$ oracle that on input $\ck$, a commitment to a message, outputs $\text{SigCom}(\ck, \sk, \ck)$. Let $\ck_i$ be the value submitted in the $i$-th oracle call and $(c_i, \pi_i)$ be the response; define $M_i := \text{Extr}_{\Sigma}(\ek, c_i)$ and $\Sigma_i := \text{Extr}(\ek, c_i)$. Then $A$ wins if it outputs $(C, c^*, \pi^*)$ such that $\text{Verify}(\ck, \text{E}_{\text{Ver}(\vk, \cdot)}(C), C, c^*, \pi^*) = 1$ and $(\text{Extr}_{\Sigma}(\ek, C^*), \text{Extr}(\ek, c^*)) \notin \{(M_1, \Sigma_1), \ldots, (M_n, \Sigma_n)\}$.

This unforgeability notion is reduced to strong unforgeability of $\text{Sig}$. On receiving $\vk$, run $(\ck, \ek) \leftarrow \text{ExSetup}$ and give $(\ck, \ek, \vk)$ to the adversary. Answer a query for $\ck$ as follows: using $\ek$, extract $M$, query it to the signing oracle to receive $\Sigma$; then run $(c, \pi) \leftarrow \text{SmSigCom}(\ck, \ek, \vk, \ck, \Sigma)$ and return $(c, \pi)$. Since by Def. 2, $\text{SmSigCom}$ and $\text{SigCom}$ both commute with $\text{Com}_\Sigma$, $\text{Com}$ and $\text{Proof}$, this perfectly simulates the adversary’s oracle. If the adversary wins the game, we return $\text{Extr}_{\Sigma}(\ek, C^*)$ and $\text{Extr}(\ek, c^*)$ which yields a valid forgery $(M, \Sigma)$ by perfect soundness of $\text{Proof}$.

**Indistinguishability.** The message in $\ck$ remains hidden to a signer running $\text{SigCom}$, in a computational sense: replacing $\ck$ by $\ck^*$ -- $\text{WISetup}$ is computationally indistinguishable and results in perfectly hiding outputs of $\text{Com}$ and $\text{Com}_\Sigma$.

**Blind Signatures.** Given a system of commuting signatures and verifiable encryption, we can easily build a blind-signature scheme in a black-box way. To get a signature on a message $M$, the user chooses $\mu \leftarrow \mathcal{R}_M$ and sends the commitment $\ck := \text{Com}_\Sigma(\ck, M, \mu)$ to the signer. The latter uses $\text{SigCom}$ to produce and send $(c, \pi)$, a committed signature on $M$ and a proof of validity. The user can produce a proof $\pi \leftarrow \text{AdPrDC}_M(\ck, \vk, (M, \mu), c, \pi)$, which asserts validity of the committed signature on $M$. The blind signature is defined as $(c, \pi)$ and is verified by $\text{Verify}(\ck, \text{E}_{\text{Ver}(\vk, \cdot)}(M), c, \pi)$.

This yields a generically more efficient construction than that from [Fis06], in which, besides $c$ and $\pi$, the blind signature contains $\ck$ and a proof that $\ck$ opens to $M$.

5 Application: Non-Interactively Delegatable Anonymous Credentials.

Our main application of commuting signatures and verifiable encryption is a black-box construction of a delegatable anonymous credential scheme with a non-interactive delegation protocol. Our scheme borrows the idea of combining Groth-Sahai proofs and automorphic signatures from the instantiation of anonymous proxy signatures from [Fuc09]. This primitive is similar in that it enables to prove knowledge of a certification chain, but there is no mutual anonymity of the users in the (non-interactive) delegation protocol. Commuting signatures now allow to define a delegation protocol where both delegator and delegatee remain anonymous w.r.t. each other. Moreover, the protocol is non-interactive, that is, a user can publish a pseudonym, which can be used by the delegator to produce a credential for the user—as it would be in the non-anonymous case with public keys instead of pseudonyms.

We start by presenting the model for delegatable credentials defined in [BCC+09]. In Sect. 5.2 we give our instantiation of it, and compare it to that from [BCC+09] in the subsequent section.

5.1 The BCCKLS Model

The system parameters are set up by a trusted party. Every user holds a secret key $\sk$, of which she can publish pseudonyms $\text{Nym}$. Any user can be an originator of a credential by publishing a pseudonym $\text{Nym}_O$ as the public key. If user $A$ was issued a credential for pseudonym $\text{Nym}_A$, she can transform it into a credential for any
other pseudonym $Nym_I'$. Moreover, credentials can be delegated to other users. A (non-interactively) delegatable anonymous credential system consists of the following algorithms:5

Setup$_C(1^\lambda)$ outputs the system parameters $pp$
KeyGen$_C(pp)$ creates a user secret key $sk$
NymGen$(pp, sk)$ outputs a new pseudonym $Nym$ and auxiliary information $aux$ related to $Nym$
Issue$(pp, Nym_O, sk_I, Nym_I, aux_I, cred, Nym_L, L)$: $sk_I$, $Nym_I$ and $aux_I$ are the issuer’s secret key, pseudonym and auxiliary information. $cred$ is a level $L$ credential for the issuer rooted at $Nym_O$, and $Nym_L$ is the pseudonym of the delegated user. If $L = 0$ then $cred = \emptyset$. The algorithm outputs credproof.
Obtain$(pp, Nym_O, sk_U, Nym_U, aux_U, Nym_I, L, credproof)$: $sk_U$, $Nym_U$ and $aux_U$ are the user’s secret key, pseudonym and auxiliary information. $Nym_O$ and $Nym_I$ are the originator’s and the issuer’s pseudonym, and $credproof$ is the output of Issue. The algorithm outputs a credential $cred$.
CredProve$(pp, Nym_O, cred, sk, Nym, aux, L)$ takes a level $L$ credential from $Nym_O$, and $sk$, $Nym$ and $aux$, and outputs a credproof for $Nym$.
CredVerify$(pp, Nym_O, credproof, Nym, L)$ verifies a level $L$ credproof for a pseudonym $Nym$ rooted at $Nym_O$.

Security is defined by correctness, anonymity and unforgeability, which we sketch below. For a formal security definition we refer to Appendix A of the full version of [BCC+09].

Correctness. A credential is proper if for all user pseudonyms, CredProve outputs a proof that is accepted by CredVerify. Run honestly, Issue and Obtain must produce a proper credential.

Anonymity. There exists a simulator $(\text{SimSetup}_C, \text{SimCredProve}, \text{SimIssue}, \text{SimObtain})$ with the following properties: $\text{SimSetup}_C$ outputs parameters that are indistinguishable from those produced by Setup and a trapdoor $\text{sim}$. Under theses parameters the outputs $Nym$ of NymGen are distributed independently of $sk$.
$\text{SimCredProve}$ gets $\text{sim}$ instead of $cred$, $sk$ and $aux$ and outputs credproof that is indistinguishable from outputs of CredProve. SimIssue has input $\text{sim}$ instead of $sk_I$, $aux_I$ and $cred$ and cannot be distinguished from Issue by an adversary interacting with it. SimObtain gets $\text{sim}$ instead of $sk_U$ and $aux_U$ and cannot be distinguished from Obtain by an adversary interacting with it.

Note that for the case of non-interactive delegation, this means: SimIssue produces credproof that is indistinguishable from outputs of Issue. And SimObtain is obsolete, since the issuer does not interact with it.

Unforgeability. (I) There exists ExSetup$_C$ that outputs parameters $pp$ (distributed as those from Setup$_C$) and an extraction key $ek$. Under $pp$ pseudonyms are perfectly binding for $sk$ and Extract$_C$ using $ek$ outputs the chain of $L$ identities from a level $L$ credproof.

(II) No adversary $A$ can output a valid credential from which can be extracted an unauthorized chain of identities. This means that $A$ is given the parameters and has oracles to add honest users, request pseudonyms from them, request issuings between honest users, request proofs and it can run Issue and Obtain with the simulator. When $A$ requests Issue for $(Nym_O, Nym_I, Nym_U, cred, L)$, the simulator extracts $vk_O, vk_I, vk_U$ from the pseudonyms and adds $(vk_O, L + 1, vk_I, vk_U)$ to a list ValidCredentialChains. The adversary wins if it outputs a valid triple $(Nym_O, credproof, Nym)$, from which can be extracted $(vk_0, \ldots, vk_L)$ s.t. $(vk_0, i, vk_{i-1}, vk_i) \notin$ ValidCredentialChains for some $i$ and $vk_{i-1}$ is an honest user key.

5.2 Our Instantiation

In the instantiation from [BCC+09] the system parameters are a Groth-Sahai (GS) commitment key and parameters for an authentication scheme. Each user holds a secret key $x$ for the authentication scheme. A pseudonym is made up of two GS commitments to $H^x$ and $U^x$ (from which $x$ cannot be extracted), respectively, for parameters $H$ and $U$. To issue and delegate, the issuer and the user jointly compute a proof of knowledge of an authenticator on the user’s secret key, which is valid under the issuer’s secret key. The authors define a complex interactive two-party

---

5Since, as opposed to [BCC+09], we consider non-interactive delegation, Issue and Obtain are non-interactive algorithms; the output of Issue is credproof which is an additional input for Obtain.
A credential is then a chain of pseudonyms and committed authenticators with GS proofs of validity.

We replace the authentication scheme by an automorphic signature scheme. A non-anonymous credential for \( vk_L \) rooted at \( vk_0 \) is a chain of public keys and signatures \((\Sigma_1, vk_1, \Sigma_2, \ldots, vk_{L-1}, \Sigma_L)\), where \( \Sigma_i \) is a signature on \( vk_i \) under \( vk_{i-1} \). To achieve anonymity, the public keys and signatures in the credential are committed to and proofs of validity are added. Using commuting signatures, given a commitment to a public key, the issuer can directly make a commitment to a signature on it and a validity proof. This is what enables non-interactive delegation.

**Commuting Signatures with Partially Public Messages.** To instantiate credentials, merely signing user public keys does not suffice. The issuer of a credential might want to add public information to the credential, such as attributes. For delegatable credentials it is also required to include the originator’s pseudonym and the delegation level in each certificate to prevent combining different credentials and changing the order within a credential.

In Sect. 9.1, we give an automorphic signature scheme \( \text{Sig}'' \), where in addition to the message, the signer can specify some public value. The message space of \( \text{Sig}'' \) is \( \mathbb{Z}_p \times \mathcal{M} \). Our scheme only extends the parameters of \( pp \) by one group element, but is otherwise as efficient as \( \text{Sig} \), in particular, \( \text{Sig} \)- and \( \text{Sig}'' \)-signatures have the same size. We also define \( \text{VK} \) that on input a signing key outputs the corresponding verification key, which allows us to comply with the formal definition of credentials in [BCC+09]. In Sect. 9.2, we define \( \text{SigCom}'' \), which is \( \text{SigCom} \) adapted to \( \text{Sig}'' \) and thus has the public part of the message as additional input. Moreover, we show that all other algorithms defined in Def. 2 and instantiated in Sect. 8 work equally for \( \text{Sig} \) and \( \text{Sig}'' \).

**More Intuition.** We informally describe how our algorithms work. \( \text{Setup}_C \) generates a key for \( \text{Com} \) and parameters for \( \text{Sig}'' \), \( \text{KeyGen}_C \) outputs a secret key for \( \text{Sig}'' \), and given a secret key, \( \text{NymGen} \) outputs a commitment to the corresponding public key and the used randomness as auxiliary information. A level \( L \) credential from \( \text{Nym}_0 \) to \( \text{Nym}_L \) has the form

\[
\text{cred} = (c_1, \pi_1, \text{Nym}_1, c_2, \pi_2, \ldots, \text{Nym}_{L-1}, c_L, \pi_L),
\]

where \( c_i \) is a commitment to a signature \( \Sigma_i \) on the public value \( \mathcal{H}(\text{Nym}_{i-1}, i) \) and the key committed in \( \text{Nym}_i \), valid under the key committed in \( \text{Nym}_{i-1} \); and \( \pi_i \) is a proof of validity of \( \Sigma_i \). We call it a credential if it is valid on a trivial \( \text{Nym}_L \), i.e., when \( \text{Nym}_L = \text{Com}(ck, vk_L, 0) \), and speak of a credential proof otherwise.

\( \text{CredProve} \) takes a credential and turns it into a credential proof by randomizing all its components, using aux s.t. \( \text{Nym}_L = \text{Com}(ck, \text{VK}(sk), aux) \) for the last component. \( \text{CredVerify} \) verifies a credential proof by checking the proofs contained in it. Given a level \( L \) credential, \( \text{Issue} \) extends it by one level to a credential for \( \text{Nym}_{L+1} \): if it is not an original issuing, it first makes a credential proof for the issuer’s pseudonym \( \text{Nym}_I \); using \( \text{SigCom}'' \) it produces \( (c_{L+1}, \pi_{L+1}) \) for \( \text{Nym}_{L+1} \), and turns the proof into a proof for the committed verification key \( \text{Nym}_I \) by running \( \text{AdPrC}_K \) on randomness \( aux_I \). Obtain turns this credential proof into a credential by adapting the randomness to make it valid for a trivial \( \text{Nym}_L \).

**Algorithm Specification.** We now formally define the algorithms of our scheme \( \text{Cred} \).

\( \text{Setup}_C(1^\lambda) \). Run \( ck \leftarrow \text{Setup}; pp_S \leftarrow \text{Setup}' \); return \( pp := (ck, pp_S) \)

\( \text{KeyGen}_C(pp) \). Parse \( pp \leftarrow (ck, pp_S) \); run \( (vk, sk) \leftarrow \text{KeyGen}'(pp_S) \); return \( sk \)

\( \text{NymGen}(pp, sk) \). Choose \( aux \leftarrow \mathcal{R}_M \); return \( \text{Nym} := \text{Com}_M(pp, \text{VK}(sk), aux), aux) \)

\( \text{Issue}(pp, \text{Nym}_O, sk_I, \text{Nym}_I, aux_I, cred, \text{Nym}_U, L) \).

- If \( L = 0 \), and \( \text{cred} \neq \emptyset \) or \( \text{Nym}_O \neq \text{Nym}_I \) then abort;
  - if \( L > 0 \) then set \( \text{credproof} \leftarrow \text{CredProve}(pp, \text{Nym}_O, cred, sk_I, \text{Nym}_I, aux_I, L) \) and abort if it fails
- Parse \( cred \) as in (1); if \( \text{Nym}_L := \text{Nym}_I \neq \text{Com}_M(pp, \text{VK}(sk_I), aux) \) or \( \text{Nym}_U \notin \mathcal{C}_M(pp) \) then abort
- \( (c_{L+1}, \pi_{L+1}) \leftarrow \text{SigCom}''(pp, sk, \mathcal{H}(\text{Nym}_O, L + 1), \text{Nym}_U) \)
  - \( \pi_{L+1} \leftarrow \text{AdPrC}_K(pp, (\text{VK}(sk_I), aux_I), \text{Nym}_U, c_{L+1}, \pi_{L+1}) \)
- Return \( \text{credproof} \parallel (\text{Nym}_I, c_{L+1}, \pi_{L+1}) \)
Obtain\((pp, Nym_O, sk_U, Nym_U, aux_U, Nym_I, L; credproof')\).

- Parse \(pp \rightsquigarrow (ck, pp_S)\); parse \(credproof' \rightsquigarrow credproof_L' \parallel (Nym_I', c_{L'+1}, \pi_{L'+1})\). If \(Nym_I \neq Nym_I'\) or \(Nym_U \neq \text{Com}_M(pp, VK(sk_U), aux_U)\) or CredVerify\((pp, Nym_O, credproof', Nym_U, L + 1) = 0\) then abort
- \(\pi_{L+1} \leftarrow \text{RdProof}(ck, E_{\text{Ver}_{pp_S}(\cdot, \cdot)}(Nym_O, L+1, \cdot), (Nym_I, 0), (Nym_U, \cdot - aux_U), (c_{L+1}, 0), \pi_{L+1})\)
- Return \(credproof_L' \parallel (Nym_I', c_{L'+1}, \pi_{L'+1})\)

CredProve\((pp, Nym_O, cred, sk, Nym, aux, L)\).

- Parse \(pp \rightsquigarrow (ck, pp_S)\) and cred as in (1)
- If \(Nym \neq \text{Com}_M(pp, VK(sk), aux)\) or CredVerify\((pp, Nym_O, cred, \text{Com}_M(pp, VK(sk), 0), L) = 0\) then abort
- For \(i = 1 \ldots L\), pick \(\nu_i \leftarrow \mathcal{R}_M, \gamma_i \leftarrow \mathcal{R}\). Set \(Nym_0 := Nym_O, \nu_0 := 0, Nym_L := Nym_U, \nu_L := aux\)
- For \(i = 1 \ldots L\) do
  \[Nym_i' := \text{RdCom}_M(pp, Nym_i, \nu_i); c_i' := \text{RdCom}(ck, c_i, \gamma_i)\]
  \[\pi_i' \leftarrow \text{RdProof}(ck, E_{\text{Ver}_{pp_S}(\cdot, \cdot)}(Nym_O, i, \cdot), (Nym_{i-1}, \nu_{i-1}), (Nym_i, \nu_i), (c_i, \gamma_i), \pi_i)\]
- Return \((c_1', \pi_1', Nym_1', c_2', \pi_2', \ldots, Nym_L', c_L', \pi_L')\)

CredVerify\((pp, Nym_O, credproof, Nym, L)\).

- Parse \(pp \rightsquigarrow (ck, pp_S)\), \(credproof \rightsquigarrow (c_1, \pi_1, Nym_1, \ldots, c_L, \pi_L)\), let \(Nym_0 := Nym_O, Nym_L := Nym\)
- If \(\forall 1 \leq i \leq L : \text{Verify}(ck, E_{\text{Ver}_{pp_S}(\cdot, \cdot)}(Nym_O, i, \cdot), Nym_{i-1}Nym_i, c_i, \pi_i) = 1\) and \(Nym_i \in \mathcal{C}_M(pp)\), return 1

**Theorem 1.** Let \((\text{Com}, \text{Proof})\) be a randomizable, extractable, composable zero-knowledge non-interactive proof-of-knowledge system, let \(\text{Sig}''\) be a strongly unforgeable automorphic signature scheme, and let \(\mathcal{H}\) be collision resistant. Then \(\text{Cred}\) as defined above is a secure anonymous delegatable credential scheme.

**Proof sketch.** We refer to the full version of [BCC + 09] for the formal definition of the model, which is quite involved. Since the overall construction of our scheme is similar to the BCCKLS construction, in particular the use of (randomizable and simulatable) Groth-Sahai proofs to commit to a delegation chain and prove validity, our scheme is proved to satisfy the security definitions analogously. Our protocol is a lot simpler though, since our certificates are on public keys and one can extract a complete certification chain from our credentials, avoiding thus partial-extractability notions. Moreover, our construction does not make use of interactive secure two-party protocols. We give a sketch of the security proof, highlighting the differences.

**Correctness.** Correctness of our scheme follows from a straightforward argument using the correctness of the underlying building blocks.

**Anonymity.** A witness-indistinguishability based definition of anonymity is an immediate consequence of perfectly hiding commitments and proofs, when \(ck\) is produced by WISetup: pseudonyms \(Nym\) information-theoretically hide the committed value and the proofs in cred do not contain information either. We thus define SimSetup\(_{\text{C}}\) using WISetup. The algorithms CredProve, Issue and Obtain can be simulated without knowledge of any private information; this follows from the zero-knowledge property of Groth-Sahai proofs; in particular, in the witness-indistinguishable (WI) setting, given the simulation trapdoor \(\text{sim}\) of GS proofs, we can make perfectly hiding commitments and proofs for any equation with certain properties—which are satisfied by ours. (See Sect. 10.)

Our simulator SimCredProve is the exact analogue of SimProve, defined in the anonymity proof of the BCCKLS scheme: it constructs a simulated certificate chain from \(Nym_O\) to \(Nym_I\) of length \(L\), by simulating the intermediate pseudonyms, i.e., the \(\text{Com}_M\) commitments (cf. Sect. 10.2), the commitments in \(c_i\), and the proofs \(\pi_i\). SimIssue is defined as SimCredProve for \(L+1\) except that it sets \(Nym_L\) as \(Nym_I\). Since Obtain does not interact with the issuer, SimObtain is the empty algorithm. Our proof is considerably simpler than that of [BCC + 09], due to the fact that Issue readily outputs a credential rather than engaging in a two-party protocol with Obtain. All we need to do is simulate GS commitments and proofs. We refer to Sect. 10 for a discussion on how to actually simulate such proofs.
UNFORGEABILITY. Soundness and extractability of GS proofs, unforgeability of \textbf{Sig}'' and simulatability of \textbf{SigCom}'' imply that our scheme is unforgeable in the sense of \cite{BCC+09}: (I) For ExSetup, we substitute Setup by ExSetup in Setup\textsubscript{C}. This generates an identically distributed key \(ck\) (which leads to perfectly binding commitments) and an extraction key \(ek\) that allows to extract the committed chain of public keys (“identities”) and certificates from a credential.

The notion defined in (II) is reduced to unforgeability of \textbf{Sig}: given a verification key \(vk\) and a signing oracle, we simulate the game as follows: we guess which honest user the adversary will “frame”; we compute the parameters with ExSetup\textsubscript{C}, and use extraction, the signing oracle and Sm\textbf{SigCom} to simulate Issue for that user. Let \((Nym, credproof, Nym_L)\) be a successful forgery, thus when we extract \((vk_0, \Sigma_1, vk_1, \ldots, \Sigma_L, vk_L)\) from it then for some \(i: (vk_0, i, vk_{i-1}, vk_i) \notin \text{ValidCredentialChains and } vk_{i-1}\) is honest. If we guessed correctly (i.e., \(vk_{i-1} = vk\) then we can return the \textbf{Sig}'' forgery \((vk_i, \Sigma_i)\), since (by collision resistance of \(H\)) we have never queried our signing oracle on \((H(Nym, i), vk_i)\) for any \(Nym\) of \(vk_0\).

Optimizing the Black-Box Construction for Concrete Commuting Signatures. When using our implementation of commuting signatures (Sect. 6 and 8), we can make the following optimizations. In the instantiation we have \(M \subseteq G_1 \times G_2\) for an asymmetric bilinear group. A \textbf{Com}\textsubscript{M} commitment to a message \((M, N) \in M\) is defined as \(c_M := \text{Com}(ck, M, \mu), c_N := \text{Com}(ck, N, \nu), \pi_N \leftarrow \text{Prove}(ck, E_{DHM}(M, \mu), (N, \nu))\) and additional components which enable the signer to make a committed signature on \((M, N)\) and a proof of validity by running \textbf{SigCom}. These additional components are however not required for the \(Nym\)’s contained in the credential, where giving \((c_M, c_N, \pi_M)\) is sufficient. Moreover, when issuing a new credential, the “public key” used to verify it can be given in the clear, i.e., as \((X, Y) \in M\), rather than as a commitment.

5.3 A Comparison to the BCCKLS Instantiation

The key building block of a delegatable-credential scheme is a certification scheme signing (or authenticating) user keys and some public information. The certificates (or “authenticator”) on user secret keys in the BCCKLS instantiation \cite{BCC+09} are in \(G_1^8 \times G_2^3\) and are verified by evaluating 16 pairings. Our certificates on a user public key (and a public value) are in \(G_1^6 \times G_2^2\) and verified by evaluating 7 pairings. Proving validity of a committed certificate requires Groth-Sahai proofs (which are in \(G_1^{2 \times 2} \times G_2^{2 \times 2}\) for the SXDH instantiation) for 8 equations for the BCCKLS instantiation and 3 equations for ours. The pseudonyms in \cite{BCC+09} consist of two Groth-Sahai commitments and one proof, and are thus in \(G_1^6 \times G_2^3\), as are the pseudonyms contained in our credentials. The pseudonyms enabling non-interactive delegation are the size of two optimized pseudonyms and 3 \(G_1\) elements.\footnote{Note that if the size of pseudonyms is to be minimized, user could publish \(Nym = (c_M, c_N, \pi_M)\) and send the remaining elements \((c_P, c_Q, \pi_P, k, \pi_U)\) as a first step in Obtain (see Sect. 10.2).}

A tuple \((c_1, \pi_1, Nym_1)\) contained in a warrant is thus in \(G_1^{50} \times G_2^{40}\) for the BCCKLS instantiation, whereas it is in \(G_1^{20} \times G_2^{18}\) for ours.\footnote{This analysis is for the case when Groth-Sahai proofs are applied in a straightforward way and not considering simulatability. To make inhomogeneous equations simulatable, a Groth-Sahai proof is augmented by one commitment in \(G_1^4\), one commitment in \(G_2^2\) and one proof from \(G_1^2 \times G_2^2\) (see Sect. 10.2). Of our 3 equations only 1 is inhomogeneous, the size of a triple \((c_1, \pi_1, Nym_1)\) is thus augmented by 6 \(G_1\) elements and 4 \(G_2\) elements. We note that the 8 equations for BCCKLS authenticators contain 3 inhomogeneous equations.} We conclude that the size of our credentials is less than half the size of BCCKLS credentials.

Most importantly, issuing and delegation in our scheme is substantially more efficient than in the BCCKLS scheme. In the latter the issuer and the user run a secure two-party protocol to jointly compute a proof of knowledge of an authenticator on the user’s secret key. This protocol uses homomorphic encryption and interactive ZK proofs asserting that certain blinding values are in the correct ranges. Since these tools are not made explicit it is not clear how many rounds the protocol requires nor what amount of data needs to be sent in each of them. In contrast, in our instantiation the issuer simply rerandomizes his credential and runs SigCom and AdPr\textsubscript{C}K. She then sends a ready credential to the user.

Concerning the assumptions on which security is based, they are both non-interactive, “\(q\)-type” assumptions and part of the generalized “\(\text{Uber-Assumption}\)” family \cite{Boy08}. What is more, both are comparable variants of the \textbf{strong Diffie-Hellman} assumption \cite{BB04}, as was argued in \cite{Fuc09} (cf. the discussion in Appendix C.1, ibid.).


6 Instantiation of the Building Blocks

In this section we give instantiations of the building blocks **Com**, **Proof** and **Sig** (Sect. 6.2, 6.3 and 6.4, respectively) on which we base our commuting signatures. We start with introducing bilinear groups and the assumptions under which our instantiations are secure.

6.1 Bilinear Groups and Assumptions

A bilinear group (cf. e.g. [GPS08]) is a tuple $\text{grp} = (p, G_1, G_2, G_T, e, G_1, G_2)$ where $G_1, G_2$ and $G_T$ are cyclic groups of prime order $p$, $G_1$ and $G_2$ generate $G_1$ and $G_2$, respectively, and $e : G_1 \times G_2 \to G_T$ is an efficient non-degenerate bilinear map, i.e., $\forall X \in G_1 \forall Y \in G_2 \forall a, b \in \mathbb{Z} : e(X^a, Y^b) = e(X, Y)^{ab}$, and $e(G_1, G_2)$ generates $G_T$. We assume that there exists a probabilistic polynomial-time algorithm $\text{GrpGen}$ that on input $1^\lambda$ outputs a bilinear group $\text{grp}$ for which $p$ is a $\lambda$-bit prime.

Assumption 1 (SXDH). The Symmetric External Diffie-Hellman Assumption states that given $(G_1^r, G_1^s, G_1^t)$ for random $r,s \in \mathbb{Z}_p$, it is hard to decide whether $t = rs$ or $t$ is random; moreover, given $(G_2^r, G_2^s, G_2^t)$ for random $r',s' \in \mathbb{Z}_p$, it is hard to decide whether $t' = r's'$ or $t'$ is random.

The $q$-Asymmetric Double Hidden Strong Diffie-Hellman assumption was introduced in [Fuc09] and is a variant of $q$-HSHD [BW07] in asymmetric bilinear groups. It was shown in [FPV09] that under the $q$-SDH assumption [BB04], given $q - 1$ tuples $((K \cdot G^{c_i})^{1/(x+c_i)}, c_i, v_i)$ for random $c_i, v_i \in \mathbb{Z}_p$, it is hard to produce a new tuple of this form. The assumption below states that if $c_i$ and $v_i$ are given in a hidden form $(F^{c_i}, H^{c_i})$ and $(G^{v_i}, H^{v_i})$, respectively, it is intractable to produce a new tuple $((K \cdot G^{v_i})^{1/(x+c_i)}, F^{c_i}, H^{c_i},G^{v_i}, H^{v_i})$.

Assumption 2 ($q$-ADHSDH). Given $(G, F, K, X = G^x; H, Y = H^x) \in G_1^4 \times G_2^2$, it is hard to output a new tuple $(A, B, V, D, W) \in G_1^4 \times G_2^2$ of this form, i.e., a tuple that satisfies

\[ e(A, Y \cdot D) = e(K \cdot V, H) \quad e(B, H) = e(F, D) \quad e(V, H) = e(G, W) \]  

(2)

The next assumption was also introduced in [Fuc09] and is the weakest variant of the various flexible CDH assumptions, adapted to asymmetric bilinear groups. It states that given $G, G^a$ and $H$, it is hard to output $(G^a, G^{ra}, H^r, H^{ra})$ for an arbitrary $r \neq 0$.

Assumption 3 (AWFCDH). Given random generators $G \in G_1$ and $H \in G_2$, and $A = G^a$ for $a \in \mathbb{Z}_p$, it is hard to output $(G^a, G^{ra}, H^r, H^{ra}) \in (G_1^2) \times (G_2^2)$, i.e., a tuple $(R, M, S, N)$ that satisfies

\[ e(A, S) = e(M, H) \quad e(M, H) = e(G, N) \quad e(R, H) = e(G, S) \]  

(3)

Throughout the paper, we will assume two fixed generators $G$ and $H$ of $G_1$ and $G_2$, respectively. We call a pair $(A, B) \in G_1 \times G_2$ a Diffie-Hellman pair (w.r.t. $(G, H)$), if there exists $a \in \mathbb{Z}_p$ such that $A = G^a$ and $B = H^a$. Using the bilinear map $e$, such pairs are efficiently decidable by checking

\[ E_{DH}(A; B) : e(G^{-1}, B) e(A, H) = 1 . \]

We let $\text{DH} := \{(G^a, H^a) \mid a \in \mathbb{Z}_p\}$ denote the set of DH pairs and implicitly assume them to be w.r.t. $G$ and $H$.

6.2 SXDH Commitments.

We instantiatate **Com**, defined in Sect. 3.1, by the commitment scheme based on SXDH given in [GS08].

Setup, on input $(p, G_1, G_2, G_T, e, G_1, G_2)$ chooses $\alpha_1, \alpha_2, t_1, t_2 \in \mathbb{Z}_p$ and returns $ck = (g^{rp}, u_1, u_2, v_1, v_2)$ with

\[ u_1 := (G_1, G_1^{\alpha_1}) \quad u_2 := (G_1^2, G_1^{\alpha_1 \alpha_2}) \quad v_1 := (G_2, G_2^{\alpha_2}) \quad v_2 := (G_2^2, G_2^{\alpha_2 \alpha_2}) \]

Value and random space are defined as $V := G_1 \cup G_2$ and $\mathcal{R} := \mathbb{Z}_p^2$. 


We define the following two shortcuts for $E$

In order to instantiate $6.3$ SXDH Groth-Sahai Proofs for Pairing-Product Equations given by the values $\text{WISetup}$

Remark 3. Com\(\text{commitments are homomorphic:} \text{Com}(ck, X, r) \odot \text{Com}(ck, X', r') = \text{Com}(ck, X \cdot X', r + r')\); therefore if $c = \text{Com}(ck, X, r)$ then $\text{RdCom}(ck, c, r') = \text{Com}(ck, X, r + r')$.

Security. The scheme is perfectly binding, computationally hiding and randomizable as defined in Sect. 3.1.

6.3 SXDH Groth-Sahai Proofs for Pairing-Product Equations

In order to instantiate Proof, defined in Sect. 3.2, we use the proof system introduced in [GS08]. The class of equations $E$ for our proof system are pairing-product equations (PPE). A PPE over variables $X_1, \ldots, X_m \in \mathbb{G}_1$ and $Y_1, \ldots, Y_n \in \mathbb{G}_2$ is an equation of the form

$$E(X_1, \ldots, X_m; Y_1, \ldots, Y_n) : \prod_{i=1}^n e(A_i, Y_1) \prod_{i=1}^m e(X_i, B_i) \prod_{i=1}^n \prod_{j=1}^m e(X_i, Y_j)^{\gamma_{i,j}} = t_T,$$

defined by $A_j \in \mathbb{G}_1$, $B_i \in \mathbb{G}_2$, $\gamma_{i,j} \in \mathbb{Z}_p$, for $1 \leq i \leq m$ and $1 \leq j \leq n$, and $t_T \in \mathbb{G}_T$.

Proofs. We define $\text{Prove}(ck, E, (X_i, r_i))_{i=1}^n, (Y_j, s_j))_{j=1}^n$ for $X_i \in \mathbb{G}_1, Y_j \in \mathbb{G}_2$ and $r_i, s_j \in \mathbb{Z}_p^n$, and equation $E$ given by the values $((A_j^m)_{j=1}^n, (B_i^m)_{i=1}^n, (\gamma_{i,j})_{i,j}) \in \mathbb{Z}_p^{m \times n}, t_T \in \mathbb{G}_T)$. For notational convenience, let us first define the following two shortcuts for $Z = (z_{ij})_{ij} \in \mathbb{Z}_p^{2 \times 2}$, $\bar{u} \in \mathbb{G}_1^{2 \times 2}, \bar{v} \in \mathbb{G}_2^{2 \times 2}$.

$$Z \otimes \bar{u} := \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \quad Z \otimes \bar{v} := \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix}$$

(5)

The output $(\phi, \theta) \in \mathbb{G}_2^{2 \times 2} \times \mathbb{G}_1^{2 \times 2}$ of $\text{Prove}$ is then defined as:

$$\phi := \begin{bmatrix} v_{11}' & v_{12}' \\ v_{21}' & v_{22}' \end{bmatrix} \cdot \left( \prod_{i=1}^m B_i^{r_i} \right) \left( \prod_{j=1}^n Y_j^{\sum_{i=1}^m r_i \gamma_{i,j}} \right) \cdot \left( \prod_{i=1}^m B_i^{r_i} \right) \left( \prod_{j=1}^n Y_j^{\sum_{i=1}^m r_i \gamma_{i,j}} \right)^{-1} \cdot (Z \otimes \bar{v})$$

$$\theta := \begin{bmatrix} 1 \\ 1 \end{bmatrix} \left( \prod_{j=1}^n A_j^{s_j} \right) \left( \prod_{i=1}^m X_i^{\sum_{j=1}^n s_j \gamma_{i,j}} \right) \cdot (Z \otimes \bar{u})$$

We write $\text{Prove}(ck, E, (X_i, r_i))_{i=1}^n, (Y_j, s_j))_{j=1}^n; Z)$ if we want to make the internal randomness $Z \in \mathbb{Z}_p^{2 \times 2}$ explicit.

Randomization. Randomization of a commitment $c = \text{Com}(ck, X, r)$ via $\text{RdCom}(ck, c, r')$ replaces randomness $r$ by randomness $r + r'$; and similarly for $d = \text{Com}(ck, Y, s)$. Adaptation of a proof by $\text{RdProof}$ must thus
do the same to a proof \( \pi = (\phi, \theta) \). We formally define \( \text{RdProof}(ck, E, (c_i, r_i)_{i=1}^m, (d_j, s_j)_{j=1}^n, \pi) \): choose \( Z = ((z_{11}, z_{12}), (z_{21}, z_{22}))^\top \in \mathbb{Z}_{p}^{2\times 2} \), define \((t_{11}, t_{12}, t_{21}, t_{22})\) as in (6) and output \((\phi', \theta') \in \mathbb{G}_2^{2\times 2} \times \mathbb{G}_2^{2\times 2} \) defined as:

\[
\phi' := \phi \odot \left[ \left( \prod_{j=1}^m d_{ij} \right)^{\sum_{i=1}^m \tilde{r}_i \gamma_{ij}} \right] v_{11} t_{12} \left( \prod_{j=1}^m B_{ij} \right)^{\sum_{i=1}^m \tilde{r}_i \gamma_{ij}} v_{11} t_{12} \left( \prod_{j=1}^m d_{ij} \right)^{\sum_{i=1}^m \tilde{r}_i \gamma_{ij}} v_{11} t_{12} \left( \prod_{j=1}^m B_{ij} \right)^{\sum_{i=1}^m \tilde{r}_i \gamma_{ij}} v_{11} t_{12} \quad (Z \otimes \tilde{v})
\]

\[
\theta' := \theta \odot \left[ \left( \prod_{j=1}^m c_{ij} \right)^{\sum_{i=1}^m \tilde{s}_i \gamma_{ij}} \right] v_{11} t_{12} \left( \prod_{j=1}^m A_{ij} \right)^{\sum_{i=1}^m \tilde{s}_i \gamma_{ij}} v_{11} t_{12} \left( \prod_{j=1}^m c_{ij} \right)^{\sum_{i=1}^m \tilde{s}_i \gamma_{ij}} v_{11} t_{12} \left( \prod_{j=1}^m A_{ij} \right)^{\sum_{i=1}^m \tilde{s}_i \gamma_{ij}} v_{11} t_{12} \quad (Z \otimes \tilde{u})
\]

which has the same distribution as the output of \( \text{Prove}(ck, E, (X_i, r_i + \tilde{r}_i)_{i=1}^m, (Y_j, s_j + \tilde{s}_j)_{j=1}^n) \), where \( r_i' \) and \( s_j' \) are such that \( c_i = \text{Com}(ck, X_i, r_i') \) and \( d_j = \text{Com}(ck, Y_j, s_j') \), as we will show now.

**Remark 4.** In additive notation, the commitments and proofs can be written as follows (cf. the full version of [GS08]), when the cumulated randomness for all variables is \( R = (r_{ik}) \in \mathbb{Z}_{p}^{m\times 2} \) and \( S = (s_{jk}) \in \mathbb{Z}_{p}^{n\times 2} \), and we set \( \Gamma = (\gamma_{ij}) \in \mathbb{Z}_{p}^{m\times n} \) and define \( \iota(\vec{X}) := [\vec{0} | \vec{X}] \).

\[
\bar{c} = \iota(\vec{X}) + R \vec{u} \\
\bar{d} = \iota(\vec{Y}) + S \vec{v}
\]

\[
\phi = R^\top \iota(\vec{B}) + R^\top \Gamma \iota(\vec{Y}) \quad (R^\top G - Z^\top) \vec{v}
\]

\[
\theta = S^\top \iota(\vec{A}) + S^\top \Gamma \iota(\vec{X}) + Z \vec{u}
\]

To randomize the commitments and proofs, choose \( \hat{R} \leftarrow \mathbb{Z}_{p}^{m\times 2} \), \( \hat{S} \leftarrow \mathbb{Z}_{p}^{n\times 2} \), \( \hat{Z} \leftarrow \mathbb{Z}_{p}^{2\times 2} \) and set

\[
\bar{c}' := \bar{c} + \hat{R} \vec{u} \\
\bar{d}' := \bar{d} + \hat{S} \vec{v}
\]

\[
\phi' := \phi + R^\top \iota(\vec{B}) + R^\top \Gamma \iota(\vec{Y}) \quad (R^\top \Gamma S - Z^\top) \vec{v} \quad (Z \otimes \tilde{v}) = (Z + Z \otimes \tilde{S} \Gamma R) \vec{u}
\]

\[
\theta' := \theta + \hat{S}^\top \iota(\vec{A}) + \hat{S}^\top \Gamma \iota(\vec{X}) + Z \vec{u} \quad (S \otimes \tilde{v}) = (S + S \otimes \tilde{Z} \Gamma R) \vec{u}
\]

The output of \( \text{RdProof}(ck, E, (c_i, r_i)_{i=1}^m, (d_j, s_j)_{j=1}^n, \pi) \) using randomness \( ((\hat{z}_{11}, \hat{z}_{12}), (\hat{z}_{21}, \hat{z}_{22}))^\top \) is therefore the same as that of \( \text{Prove}(ck, E, (X_i, r_i + \tilde{r}_i)_{i=1}^m, (Y_j, s_j + \tilde{s}_j)_{j=1}^n) \) when the randomness used is

\[
\begin{bmatrix}
    z_{11} + \hat{z}_{11} + \sum \tilde{s}_{j1} \gamma_{ij} r_{i1} \\
    z_{21} + \hat{z}_{21} + \sum \tilde{s}_{j2} \gamma_{ij} r_{i2}
\end{bmatrix}
\]

which is uniformly distributed over \( \mathbb{Z}_{p}^{2\times 2} \) if \( \hat{Z} \) is.

**Verification.** Let \( ck = (u_1, u_2, v_1, v_2) \in \mathbb{G}_2^{2\times 2} \times \mathbb{G}_2^{2\times 2} \) be a commitment key, let \( \bar{c} \in \mathbb{G}_p^{m\times 2}, \bar{d} \in \mathbb{G}_p^{n\times 2} \) be vectors of commitments, and let \( (\phi, \theta) \in \mathbb{G}_2^{2\times 2} \times \mathbb{G}_2^{2\times 2} \) be a proof for an equation \( E \) given by \( \vec{A} \in \mathbb{G}_1^{m}, \vec{B} \in \mathbb{G}_2^{m}, \Gamma = (\gamma_{i,j})_{i,j} \in \mathbb{Z}_p^{m\times n}, \) and \( \vec{t}_T \in \mathbb{G}_T \). Verify(\( ck, E, c_i, d_j, (\phi, \theta) \)) outputs 1 if and only if the following 4 equations hold.

\[
\begin{align*}
\prod_{i=1}^m e(c_{i1}, \prod_{j=1}^n d_{ij}) &= e(u_{11}, \phi_{11}) e(u_{21}, \phi_{21}) e(\theta_{11}, v_{11}) e(\theta_{21}, v_{21}) \\
\prod_{i=1}^m e(c_{i1}, B_{ij} \prod_{j=1}^n d_{ij}) &= e(u_{11}, \phi_{12}) e(u_{21}, \phi_{22}) e(\theta_{11}, v_{12}) e(\theta_{21}, v_{22}) \\
\prod_{j=1}^n e(A_{ij} \prod_{i=1}^m c_{ij}, d_{ij}) &= e(u_{12}, \phi_{11}) e(u_{22}, \phi_{21}) e(\theta_{12}, v_{11}) e(\theta_{22}, v_{21}) \\
\prod_{j=1}^n e(A_{ij}, d_{ij} \prod_{i=1}^m c_{ij}, B_{ij}) &= e(u_{12}, \phi_{12}) e(u_{22}, \phi_{22}) e(\theta_{12}, v_{12}) e(\theta_{22}, v_{22})
\end{align*}
\]
Remark 5. Blazy et al. [BFI+10] show that by using techniques of batch verification, the number of pairing computations can be reduced from $4m + n + 16$ to $2m + n + 8$.

**Security.** It follows from the results of [GS08] and Remark 4 that $(\text{Prove}, \text{Verify}, \text{RdProof})$ is a randomizable witness-indistinguishable proof system for $\text{Com}$ from Sect. 6.2, as defined in Sect. 3.2.

### 6.4 Automorphic Signatures

We instantiate the signature scheme $\text{Sig} = (\text{Setup}_S, \text{KeyGen}_S, \text{Sign}, \text{Ver})$ with the scheme from [Fuc09]. It is compatible since signature components are in $\mathcal{V} = \mathbb{G}_1 \cup \mathbb{G}_2$, the space for committed values, and the verification equations are pairing-product equations, thus in $\mathcal{E}$. It is moreover automorphic since the verification keys lie in the message space.

**Scheme 1** ($\text{Sig}$). $\text{Setup}_S$ has input $\text{grp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ and outputs $\text{grp}$ and additional generators $F, K, T \leftarrow \mathbb{G}_1$. The message space is $\mathcal{DH} := \{(G^m, H^m) \mid m \in \mathbb{Z}_p\}$. $\text{KeyGen}_S$ chooses $x \leftarrow \mathbb{Z}_p$ and outputs $(vk = (G^x, H^x), sk = x)$.

$\text{Sign}$ has inputs a secret key $x$ and a message $(M, N) \in \mathcal{DH}$. It chooses random $c, r \leftarrow \mathbb{Z}_p$ and outputs

$$((A := (K \cdot T^r \cdot M)^{\frac{1}{p^2}}, B := F^c, D := H^c, R := G^c, S := H^r))$$

$\text{Ver}$ on inputs a public key $(X, Y) \in \mathcal{DH}$, a message $(M, N) \in \mathcal{DH}$ and a signature $(A, B, D, R, S)$ outputs 1 (and 0 otherwise) iff the following hold:

$$e(A, Y \cdot D) = e(K \cdot M, H)e(T, S) \quad e(B, H) = e(F, D) \quad e(R, H) = e(G, S) \quad (9)$$

**Security.** Under $q$-ADHSDH and AWFCDH, $\text{Sig}$ is strongly existentially unforgeable against adversaries making up to $q - 1$ adaptive chosen-message queries [Fuc09].

**Automorphic Signatures on Two Messages.** Fuchsbauer [Fuc09] shows how to transform the above construction into an automorphic signature scheme that signs two messages at once—if we restrict the message space to $\mathcal{DH}^* := \{(G^m, H^m) \mid m \in \mathbb{Z}_p \setminus \{0\}\}$. $\text{Sign}^*(sk, (V, W), (M, N))$ for $(V, W), (M, N) \in \mathcal{DH}^*$ is defined as follows: pick a key pair $(vk^*, sk^*) \leftarrow \text{KeyGen}_S$ and output

$$(vk^*, \text{Sign}(sk, vk^*), \text{Sign}(sk^*, (M, N)), \text{Sign}(sk^*, (V, W) \circ (M, N))), \text{Sign}(sk^*, (V, W)^3 \circ (M, N))) \text{.}$$

$\text{Ver}^*(vk, (V, W), (M, N), \Sigma)$ parses $\Sigma$ as $(vk^*, \Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$ and outputs

$$\text{Ver}(vk, \Sigma_0) \cdot \text{Ver}(vk^*, (M, N), \Sigma_1) \cdot \text{Ver}(vk^*, (V, W) \circ (M, N), \Sigma_2) \cdot \text{Ver}(vk^*, (V, W)^3 \circ (M, N), \Sigma_3) \text{.}$$

$\text{Sig}^* = (\text{Setup}_S, \text{KeyGen}_S, \text{Sign}^*, \text{Ver}^*)$ is strongly unforgeable under ADHSDH and AWFCDH [Fuc09].

In Sect. 9.1, we give a variant of the scheme $\text{Sig}$ with messages in $\mathbb{Z}_p \times \mathcal{DH}$ (required by our application to credentials in Sect. 5) which does not increase the size of a signature.

### 7 Additional Properties of Groth-Sahai Proofs

We identify five properties of Groth-Sahai proofs that will allow us to instantiate commuting signatures. The first is that proofs are constructed independently of the right-hand side of the equation; if the equation is linear, i.e., if $\gamma_{ij} = 0$ for all $i, j$, then they are even independent of the committed values. Given two independent (i.e., with no common variables) equations and commitments and proofs for them then the product of the proofs is a proof of the “product of the equations” and the concatenated vectors of commitments. The fourth property states that if we change a committed value by exponentiation then we can adapt the proof. And lastly, given commitments and a proof for an equation, if we commit to a constant of the equation then we can turn the proof into one for the set of commitments extended by the new commitment and the equation where the constant is now a variable.

---

$^8$Exponentiation of a DH pair is defined componentwise: $(M, N)^k := (M^k, N^k)$. 

18
7.1 Independence of Proofs

In general, proofs are independent of the right-hand side of the equation; moreover, proofs for linear equations are independent of the committed values.

Lemma 1. Consider equation \( E \) from (4). Then the output of \( \text{Prove}(\cdot, E, \cdot, \cdot) \) is independent of \( t_T \).

Proof. The result follows by inspection of the proof definition in (7), or, more generally, the one in Remark 4, which also encompasses other instantiations of Groth-Sahai proofs.

For concreteness, we will give the proofs of the next lemmas for the SXDH instantiation, but we note that they also hold for the other instantiations.

Lemma 2. Linear proofs depend only on the randomness of the commitments, but not on the committed values.

Proof. For an equation \( E \) for which \( \gamma_{ij} = 0 \) for all \( i \) and \( j \) the proof simplifies to

\[
\phi := \left[ \prod_{i=1}^{m} B_{i}^{r_{i1}} \right] \circ (Z \otimes \vec{v}) \quad \theta := \left[ \prod_{j=1}^{n} A_{j}^{s_{j1}} \right] \circ (Z \otimes \vec{u})
\]

which does not contain values \( X_i \) and \( Y_j \).

7.2 Proofs for Composed Equations

Groth-Sahai proofs are homomorphic w.r.t. the equations in the following sense. Given equations

\[
E : \prod_{i=1}^{n} e(A_i, Y_i) \prod_{i=1}^{m} e(X_i, B_i) \prod_{i=1}^{m} e(X_i, Y_j) = t_T
\]

\[
E' : \prod_{i=1}^{n'} e(A'_i, Y'_i) \prod_{i=1}^{m'} e(X'_i, B'_i) \prod_{i=1}^{m'} e(X'_i, Y'_j) = t'_T
\]

and a proof \( \pi \) for commitments \((\vec{c}, \vec{d})\) for equation \( E \) and a proof \( \pi' \) for commitments \((\vec{c}', \vec{d}')\) for equation \( E' \), then \( \pi'' := \pi \circ \pi' \) is a proof for commitments \((\vec{c}, \vec{c}', \vec{d}, \vec{d}')\) and equation \( E'' \) (for arbitrary \( t''_T \in \mathbb{G}_T \)):

\[
E'' : \prod_{i=1}^{n} e(A_i, Y_i) \prod_{i=1}^{n'} e(A'_i, Y'_i) \prod_{i=1}^{m} e(X_i, B_i) \prod_{i=1}^{m'} e(X'_i, B'_i) \prod_{i=1}^{m} e(X_i, Y_j) \prod_{i=1}^{m'} e(X'_i, Y'_j) = t''_T
\]

Lemma 3. If \( \pi = \text{Prove}(ck, E, (X_i, r_i))_{i=1}^{m}, (Y_j, s_j)_{j=1}^{n}; Z) \) and \( \pi' = \text{Prove}(ck, E', (X'_i, r'_i))_{i=1}^{m'}, (Y'_j, s'_j)_{j=1}^{n'}; Z') \) then \( \pi \circ \pi' = \text{Prove}(ck, E'', (X_i, r_i))_{i=1}^{m'}, (Y_j, s_j)_{j=1}^{n'}; Z + Z') \)

Proof. Equation \( E'' \) over \((X_1, \ldots, X_m, X'_1, \ldots, X'_m;' Y_1, \ldots, Y_n, Y'_1, \ldots, Y'_n)\) is determined by the constants \( \vec{A}'' := (\vec{A}, \vec{A}') \), \( \vec{B}'' := (\vec{B}, \vec{B}') \) and \( \Gamma'' := \left[ \begin{array}{c} \Gamma \\ \Gamma' \end{array} \right] \). The proof \( \pi'' = \pi \circ \pi' \) looks as follows (with \( t''_{ij} := t_{ij} + t'_{ij} \))

\[
\phi'' := \begin{bmatrix}
\prod_{i=1}^{m} B_{i}^{r_{i1}} & (\prod_{i=1}^{m'} (B'_i)^{r'_{i1}})(\prod_{j=1}^{n} (Y_j)^{s_{j1}})(\prod_{j=1}^{n'} (Y'_j)^{s'_{j1}})
\prod_{i=1}^{m} B_{i}^{r_{i2}} & (\prod_{i=1}^{m'} (B'_i)^{r'_{i2}})(\prod_{j=1}^{n} (Y_j)^{s_{j1}})(\prod_{j=1}^{n'} (Y'_j)^{s'_{j1}})
\end{bmatrix}
\]

\[
\circ ((Z + Z') \otimes \vec{v})
\]

\[
\theta'' := \begin{bmatrix}
1 & (\prod_{j=1}^{n} A_{j}^{s_{j1}})(\prod_{j=1}^{n'} (A'_j)^{s'_{j1}})(\prod_{i=1}^{m} X_i^{s_{j1}})(\prod_{j=1}^{n'} (X'_i)^{s'_{j1}})
1 & (\prod_{j=1}^{n} A_{j}^{s_{j2}})(\prod_{j=1}^{n'} (A'_j)^{s'_{j2}})(\prod_{i=1}^{m} X_i^{s_{j2}})(\prod_{j=1}^{n'} (X'_i)^{s'_{j2}})
\end{bmatrix}
\]

\[
\circ ((Z + Z') \otimes \vec{u})
\]

which is a proof for \((\vec{c}, \vec{c}', \vec{d}, \vec{d}')\) for \( E'' \) with internal randomness \( Z + Z' \).
7.3 Changing the Committed Value and Adapting Proofs

We give a special case which we require to randomize commitments to non-trivial messages in Appendix C.1. We start with some notation. Let \( \vec{w} \in \mathbb{G}^{m \times n} \), \( Z \in \mathbb{Z}_p^{m \times n} \) and \( k \in \mathbb{Z}_p \). Then by \( \vec{w}^k \) we denote componentwise exponentiation, and by \( k \cdot Z \) we denote standard scalar multiplication, i.e.,

\[
\text{if } \vec{w} = (w_{ij})_{1 \leq i \leq m, 1 \leq j \leq n} \text{ then } \vec{w}^k = (w_{ij}^k)_{1 \leq i \leq m, 1 \leq j \leq n}
\]

\[
\text{if } Z = (z_{ij})_{1 \leq i \leq m, 1 \leq j \leq n} \text{ then } k \cdot Z = (k z_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}
\]

Consider equation \( E^*: e(X, Y) = t_T \); then given a proof \( \pi \) for \( E^* \), \( \text{Com}(ck, X, r) \) and \( \text{Com}(ck, Y, s) \), \( \pi^k \) is a proof for \( e(X, Y) = t_T^k \) and \( \text{Com}(ck, X^k, k \cdot r) \) and \( \text{Com}(ck, Y, s) \).

**Lemma 4.** If \( \pi = \text{Prove}(ck, E^*, (X, r), (Y, s); Z) \) then \( \pi^k = \text{Prove}(ck, E^*, (X^k, k \cdot r), (Y, s); k \cdot Z) \).

**Proof.** By (5), we have \((Z \otimes \vec{u})^k = (k \cdot Z) \otimes \vec{u}\) and \((Z \otimes \vec{v})^k = (k \cdot Z) \otimes \vec{v}\) for \( Z \in \mathbb{Z}_p^{2 \times 2} \) and \( k \in \mathbb{Z}_p \). The proof \( \text{Prove}(ck, E^*, (X, r), (Y, s); Z) \) is defined as

\[
\pi_1 = \left[ v_{11}^{r_1 s_1} v_{21}^{r_1 s_2} \quad v_{12}^{r_2 s_1} v_{22}^{r_2 s_2} \right] \otimes (Z \otimes \vec{v}) \quad \pi_2 = \left[ 1 \quad X^{s_1} \right] \otimes (Z \otimes \vec{u})
\]

so we have

\[
\pi_k^1 = \left[ v_{11}^{kr_1 s_1} v_{21}^{kr_1 s_2} \quad v_{12}^{kr_2 s_1} v_{22}^{kr_2 s_2} \right] \otimes ((k \cdot Z) \otimes \vec{v}) \quad \pi_k^2 = \left[ 1 \quad (X^k)^{s_1} \right] \otimes ((k \cdot Z) \otimes \vec{u})
\]

which is the definition of \( \text{Prove}(ck, E^*, (X^k, k \cdot r), (Y, s); k \cdot Z) \). \( \square \)

7.4 Committing to Constants and Adapting Proofs

Given a proof for an equation, one can commit to one of the constants and adapt the proof. Consider an equation \( E(X_1, \ldots, X_m; Y_1, \ldots, Y_n) \) as in (4) and a proof \((\phi, \theta)\) for commitments \((c_1, \ldots, c_m; d_1, \ldots, d_n)\). Some calculation shows that \((\phi, \theta)\) is also a proof for equation

\[
E'(\vec{X}, A_k; \vec{Y}) : \prod_{i=1}^{n} e(A_i, Y_i) \prod_{i=1}^{m} e(X_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(X_i, Y_j)^{\gamma_{ij} \cdot e(A_k, Y_k)} = t_T
\]

and commitments \((c_1, \ldots, c_m, \text{Com}(ck, A_k, 0); d_1, \ldots, d_n)\). This yields the following result.

**Lemma 5.** Let \( \pi \leftarrow \text{Prove}(ck, E, (X_i, r_i))_{i=1}^{m}, (Y_j, s_j))_{j=1}^{n} \) and for all \( i, j \) let \( c_i = \text{Com}(ck, X_i, r_i) \) and \( d_j = \text{Com}(ck, Y_j, s_j) \). Then \( \text{RdProof}(ck, E', (c_i, 0))_{i=1}^{m}, (\text{Com}(ck, A_k, 0), r), (d_j, 0))_{j=1}^{n}, \pi \) yields a proof that is distributed as the output of \( \text{Prove}(ck, E', (X_i, r_i))_{i=1}^{m}, (A_k, r), (Y_j, s_j))_{j=1}^{n} \). An analogous result holds for committing to a constant \( B_k \in \mathbb{G} \).

8 Instantiation of Commuting Signatures

In [Fuc09], a blind signature scheme is constructed from the scheme \( \text{Sig} \) (Sect. 6.4) as follows. The user, who wishes to obtain a signature on a message \((M, N) \in \mathcal{D}\), chooses a random \( t \leftarrow \mathbb{Z}_p \) and blinded the first message component by the factor \( T^t \). The user then sends the following: \( U := T^t \cdot M \), commitments \( c_M \) and \( c_N \) to \( M \) and \( N \), respectively, and commitments \( c_P \) and \( c_Q \) to \( G^t \) and \( H^t \), respectively; moreover, proofs \( \pi_M, \pi_P \) and \( \pi_U \) of well-formedness of \((M, N), (P, Q) \) and \( U \), respectively. The signer replies with a “pre-signature” on \( U \) (which is constructed as a signature on \( U \), but on a message that lacks the second component):

\[
A := (K \cdot T^r \cdot U)^{\frac{1}{n+1}} \quad B := F^c \quad D := H^c \quad R' := G^r \quad S' := H^r
\]
Knowing \( t \), the user can fabricate an actual signature on \((M, N)\) from this “pre-signature” by setting \( R := R' \cdot G^t \) and \( S := S' \cdot H^t \). Then \((A, B, D, R, S)\) is a signature with randomness \((c, r + t)\) because \( A = (K \cdot T^r \cdot U)^{1/(x+c)} = (K \cdot T^{r+t} \cdot M)^{1/(x+c)} \), \( R = G^{r+t} \), and \( S = H^{r+t} \). To prevent linking a signature to the signing session, the blind signature is defined as a Groth-Sahai proof of knowledge of the signature. Now to turn this into a commuting signature, there are two key observations.

1. The values \((c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U)\) which the user sends to the signer can actually be considered as a commitment to the message \((M, N)\), which is extractable and randomizable, and which perfectly hides the message when the values are produced using a key \( \text{ek}^* \leftarrow \text{WISetup} \).

2. Since \( \text{Com} \) is homomorphic, the values \( c_P \) and \( c_Q \) can be used to produce commitments on the actual signature components \( R \) and \( S \). Moreover, we show how \( \pi_P \) and \( \pi_U \) can be used to produce a proof of validity of the committed values using the results from Lemmas 1, 2, 3 and 5.

For the blind signature scheme in [Fuc09], the values \( c_p, c_Q, \pi_P \) and \( \pi_U \) are mainly used in the proof of unforgeability, when the simulator needs to extract the message, query it to its signing oracle and then use the values \( P \) and \( Q \) to turn the signature into a pre-signature. We show that all these values can be directly used by the signer to produce commitments to the signature components and even a proof of validity.

Our exposition will use Groth-Sahai proofs and the results from Sect. 7 in a black-box manner. We refer to Appendix A for a detailed and self-contained presentation of the instantiations.

### 8.1 Commitments to Messages

We define a commitment on a message \((M, N) \in \mathcal{DH}\) as the values \( C = (c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U)\) the user sends to the signer in the issuing protocol for blind signatures from [Fuc09]. We then show how to randomize a commitment and how to extract the committed value. Since the committed values are the messages of \( \text{Sig} \), the algorithms also get the parameters \( pp_S \) as input.

\[ \text{Com}_M(p) = (c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U) \in \mathcal{C}_M(p). \]

\[ \text{RandCom}, \text{on input } (c, pp_S), C \text{ and randomness } (t', \mu', \nu', \rho', \sigma'), \text{ first defines } \hat{c}_P := c_P \circ \text{Com}(c, G^t', 0), \hat{c}_Q := c_Q \circ \text{Com}(c, H^t', 0) \text{ and } U' := U \cdot T^t'. \]

Then it sets

\[ c'_M := \text{RdCom}(c_M, \mu'), \hat{\pi}'_M := \text{RdProof}(c, E_{DH}(c_M, \mu'), (c_N, \nu'), \pi_M) \]

\[ c'_N := \text{RdCom}(c_N, \nu'), \hat{\pi}'_N := \text{RdProof}(c, E_{DH}(c_P, \rho'), (c_Q, \sigma'), \pi_P) \]

\[ c'_P := \text{RdCom}(c_P, \rho'), \hat{\pi}'_P := \text{RdProof}(c, E_{DH}(c_Q, \sigma'), (c_P, \sigma'), \pi_U) \]

and returns \( C' = (c'_M, c'_N, \pi'_M, c'_P, c'_Q, \pi'_P, U', \pi'_U) \in \mathcal{C}_M(p). \) Randomness \((t, \mu, \nu, \rho, \sigma)\) was thus replaced by \((t + t', \mu + \mu', \nu + \nu', \rho + \rho', \sigma + \sigma')\).
\( C_M(pp), \) the space of valid Com\(_M\) commitments under parameters \( pp = (ck, ppS) \) is defined as

\[
C_M(pp) := \{ (c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U) \in \mathbb{G}_1^{17} \times \mathbb{G}_2^{16} \mid \text{Verify}(ck, E_{DH}, c_M, c_N, \pi_M) \wedge \text{Verify}(ck, E_{DH}, c_P, c_Q, \pi_P) \wedge \text{Verify}(ck, E_U, c_M, c_Q, \pi_U) \}.
\]

See Appendix A.1 for a proof that Com\(_M\) is a randomizable extractable commitment scheme that is perfectly binding and computationally hiding.

### 8.2 Making Committents to a Signature on a Committed Message and a Proof of Validity

We show how the signer can use the values in \( \mathbb{C} \) to produce a proof of knowledge

\[
(c_A, c_B, c_D, c_R, c_S, \pi_A, \pi_B, \pi_R) \in \mathbb{G}_1^{18} \times \mathbb{G}_2^{16}
\]

of a signature \((A, B, D, R, S)\), where \( \pi_A, \pi_B \) and \( \pi_R \) are proofs that the committed values satisfy the equations in (9), respectively, i.e.,

\[
E_A(A; M; S, D) := e(T^{-1}, S) e(A, Y) e(M, H^{-1}) e(A, D) = e(K, H)
\]

\[
E_B(B; D) := e(F^{-1}, D) e(B, H) = 1
\]

\[
E_R(R; S) := e(G^{-1}, S) e(R, H) = 1
\]

In the blind signature from [Fuc09], on receiving \( C \), the signer checks the proofs contained in it, and then produces a pre-signature by choosing \( c, r \leftarrow \mathbb{Z}_p \) and computing

\[
A := (K \cdot T^r \cdot U)^{\frac{1}{1+r}} \quad B := F^c \quad D := H^c \quad R := G^r \quad S' := H^r
\]

Knowing \( t \) s.t. \( U = T^t \cdot M \), these values are turned into a signature by setting \( R := R' G^t \) and \( S := S' H^t \). Since the commitments are homomorphic, the signer can—without knowledge of the values \( P = G^t \) and \( Q = H^t \)—make commitments to \( R \) and \( S \):

\[
c_R := c_P \circ \text{Com}(ck, R', 0) = \text{Com}(ck, R, \rho) \quad c_S := c_Q \circ \text{Com}(ck, S', 0) = \text{Com}(ck, S, \sigma)
\]

The signer also chooses \( \alpha, \beta, \delta \leftarrow \mathbb{Z}_p^2 \), and makes the remaining commitments:

\[
e_A := \text{Com}(ck, A, \alpha) \quad e_B := \text{Com}(ck, B, \beta) \quad e_D := \text{Com}(ck, D, \delta)
\]

The vector \( \bar{\Sigma} := (c_A, c_B, c_D, c_R, c_S) \) is thus a commitment on the actual signature \( \Sigma = (A, B, D, R, S) \). It remains to construct proofs \( \pi_A, \pi_B \) and \( \pi_R \) that the committed values satisfy the 3 equations in (12)—without knowledge of \( \mu, \rho \) and \( \sigma \), the randomness of the commitments \( c_M, c_R \) and \( c_S \), respectively! This can be done using the following observations:

1. Equation \( E_R(R; S) \) is actually \( E_{DH}(R; S) \) from (10). Since \( c_R \) and \( c_P \) have the same randomness \( \rho \), and \( c_S \) and \( c_Q \) have the same randomness \( \sigma \), and since by Lemma 2, the proof for the linear equation \( E_{DH} \) is independent of the committed values, we can set \( \pi_R := \pi_P \).

2. Lemmas 1 and 2 state that linear proofs only depend on the randomness of the commitments. Since \( c_S = \text{Com}(ck, S, \sigma) \) and \( c_Q = \text{Com}(ck, Q, \sigma) \) have the same randomness, \( \pi_U \) is a proof for \( E_U(M; S) \) for \( c_M \) and \( c_S \). Moreover, define

\[
E_{A^1}(A; D) := e(A, Y) e(A, D) = 1
\]

and let \( \pi_{A^1} \leftarrow \text{Prove}(ck, E_{A^1}, (A, \alpha), (D, \delta)) \). Since the product of the left-hand sides of \( E_U(M; S) \) and \( E_{A^1}(A; D) \) is the left-hand side of \( E_A(A, M; S, D) \), by Lemma 3 we have \( \pi_A := \pi_U \circ \pi_{A^1} \).
ΣigCom(ck, sk, C) Parse C as (cM, cN, cπM, cπP, cπU, U, πU) and sk as x. If πM, πP and πU are valid then choose c, r ← Zp and α, β, ρ, σ′ ← Z2p and compute the following values:

\[ A := (K \cdot T' \cdot U)^{\pi_e} \quad \text{and} \quad c_B := \text{Com}(ck, F', \beta) \]
\[ c_A := \text{Com}(ck, A, \alpha) \quad \text{and} \quad c_D := \text{Com}(ck, H', \delta) \]
\[ \pi_A' \leftarrow \text{Prove}(ck, E_{A'}, (A, \alpha), (H', \delta)) \quad \text{and} \quad \pi_A \leftarrow \text{RdProof}(ck, E_{A}, (cA, 0), (cD, 0), (cM, 0), (cS, \sigma'), \pi_A') \]
\[ \pi_R \leftarrow \text{RdProof}(ck, E_{R}, (c_{R'}, \rho'), (cS, \sigma'), \pi_P) \quad \text{and} \quad \pi_B \leftarrow \text{Prove}(ck, E_{DH}, (F', \beta), (H', \delta)) \]

Return \( (c_A, c_B, c_R, c_S, \pi_A, \pi_B, \pi_R) \).

Figure 2: Making commitments to a signature and proving knowledge.

The remaining proof \( \pi_B \) can be constructed regularly, since randomness \((\beta, \delta)\) is known to the signer. Finally, to get a random proof of knowledge, the signer randomizes all commitments and proofs using RdCom and RdProof as defined in Sect. 6.3. Algorithm SigCom is summarized in Fig. 2. In Appendix A.2, we formally prove that the output of SigCom distributed as required by Def. 2.

**Instantiation of SmSigCom.** This algorithm is similar to SigCom but instead of the signing key sk it is directly given a signature \((A, B, D, R, S)\). It proceeds like SigCom but starting from a signature instead of producing a pre-signature: choose \(\alpha, \beta, \delta \leftarrow \mathcal{R}\) and set \(c_A, c_B\) and \(c_D\) as in (13); use \(ek\) to extract \(P\) and \(Q\) from \(C\) and set

\[ c_R := c_P \circ \text{Com}(ck, R\cdot P^{-1}, 0) = \text{Com}(ck, R, \rho) \quad \text{and} \quad c_S := c_Q \circ \text{Com}(ck, S\cdot Q^{-1}, 0) = \text{Com}(ck, S, \sigma) \]

Now \(\pi_A, \pi_B\) and \(\pi_R\) can be computed as in SigCom (see Fig. 2).

### 8.3 Instantiations of Proof Adaptation for Committing and Decommitting

We define equations \(E_{\bar{A}}\) and \(E_{\bar{A}}\) and recall \(E_A\):

\[ E_A(A, M; S, D) := e(T^{-1}, S) e(A, Y) e(M, H^{-1}) e(A, D) = e(K, H) \]
\[ E_{\bar{A}}(A; S, D) := e(T^{-1}, S) e(A, Y) e(A, D) = e(K \cdot M, H) \]
\[ E_{\bar{A}}(M) := e(M, H^{-1}) = e(A, Y \cdot D)^{-1} e(K, H) e(T, S) \]

Recall equations \(E_B\) and \(E_R\) from (12). Then we have

\[ E_{\text{Verify}}((X, Y), \ldots)((M, N), (A, B, D, R, S)) \equiv E_A(A, M; S, D) \land E_B(B; D) \land E_R(R; S) \]
\[ E_{\text{Verify}}((X, Y), (M, N), \ldots)(A, B, D, R, S) \equiv E_{\bar{A}}(A; S, D) \land E_B(B; D) \land E_R(R; S) \]
\[ E_{\text{Verify}}((X, Y), \ldots)((A, B, D, R, S))(M, N) \equiv E_{\bar{A}}(M) \]

Since the product of the left-hand sides of \(E_{\bar{A}}\) and \(E_{\bar{A}}\) is the left-hand side of \(E_A\), by Lemma 3 we have

\[ \pi_A = \pi_{\bar{A}} \circ \pi_{\bar{A}}, \]

which allows us to implement the algorithms AdPrC, AdPrC_M, AdPrDC and AdPrDC_M as follows:

AdPrC(pp, vk, C, ((A, B, D, R, S), (α, β, δ, ρ, σ), \(\bar{\pi}\)). The proof \(\bar{\pi}\) is a proof for equation \(E_{\bar{A}}\). The algorithm sets

\[ \pi_{\bar{A}} \leftarrow \text{Prove}(ck, E_{\bar{A}}, (A, \alpha), (S, \sigma), (D, \delta)) \quad \pi_B \leftarrow \text{Prove}(ck, E_B, (B, \beta), (D, \delta)) \]
\[ \pi_R \leftarrow \text{Prove}(ck, E_R, (R, \rho), (S, \sigma)) \]

for \(E_B\) and \(E_R\) as defined in (12). It then returns \(\pi := (\pi_{\bar{A}} \circ \pi_B, \pi_B, \pi_R)\).
AdPrC_M(pp, vk, ((M, N), (t, μ, ν, σ)), E_S, π). The proof π is of the form (π_A, π_B, π_R). The algorithm sets π_A ← Prove(ck, E_A, (M, μ)) and returns a randomization of π := (π_A ⊕ π_A, π_B, π_R).

AdPrC_D(pp, vk, C, ((A, B, D, R, S), (α, β, δ, ρ, σ)), π). The proof π is of the form (π_A, π_B, π_R). The algorithm sets π_A ← Prove(ck, E_A, A, α), (S, σ), (D, δ) and returns π := π_A ⊕ π_A (where “⊕” denotes component-wise division, that is: replace all the components of the second argument by their inverses and then multiply them with those of the first argument).

AdPrC_M(pp, vk, ((M, N), (t, μ, ν, σ)), E_S, π). The proof π is of the form (π_A, π_B, π_R). The algorithm produces π_A ← Prove(ck, E_A, (M, μ)) and returns a randomization of π := (π_A ⊕ π_A, π_B, π_R).

Instantiation of AdPrC_K and AdPrC_D. In applications (such as the credentials in Sect. 5) where the signer wants to remain anonymous, she makes a commitment
c_vk := (c_X = Com(ck, X, ξ), c_Y = Com(ck, Y, ψ), π_X = Prove(ck, E_DH, (X, ξ), (Y, ψ))}
to her public key vk = (X, Y) ∈ DH and wishes to prove that the values in c_v are a valid signature on the value (M, N) in C under the public key that is committed in c_vk. The first equation of verification is thus

E_A(A, M; S, Y, D) : e(T^{-1}, S) e(M, H^{-1}) e(A, Y) e(A, D) = e(K, H),

whereas E_B and E_R remain unchanged. Given a commitment C to a message, c_v = (c_A, c_B, c_C, c_D, c_v), a commitment to a signature, and a proof π = (π_A, π_B, π_R) of validity, π_A can be adapted to π_A using Lemma 5 from Sect. 7: set π_A ← RdProof(ck, E_A, (c_A, 0), (c_M, 0), (c_S, 0), (Com(ck, Y, ψ), (c_D, 0), π_A). (See Appendix A.3 for the details.) To adapt to a decommitment of c_vk, we have to reset the randomness of c_v to 0. AdPrC_D does thus the converse: it sets π_A ← RdProof(ck, E_A, (c_A, 0), (c_M, 0), (c_S, 0), (Com(ck, Y, ψ), (c_D, 0), π_A).

9 Commuting Signatures with Partially Public Messages

9.1 Automorphic Signatures on an Integer and a Message

The scheme Sig from Sect. 6.4 can be adapted to sign a value from Z_p and an element from DH at the same time, as it is required for our application to delegatable credentials. Note that while this requires one extra element in the parameters it does not increase the size of a signature.

Intuition. ADHS is states that given “weak signatures” ((KV) 1/m, Fc, Hc) on random messages (V, W) ∈ DH, it is hard to forge such a signature on a new message. Now to turn this into a CMA secure scheme, Fuchsbauer [Fuc09] implicitly defines a trapdoor commitment TCom((M, N), r) := M · Tr with opening (G^r, H^r). The actual signature is then a weak signature on TCom((M, N), r) together with the opening (G^r, H^r). AWFCDH implies that it is hard to open a TCom commitment in two different ways, thus TCom is computationally binding.

In order to sign a message pair consisting of an integer value v and a DH-pair (M, N), we replace TCom by TCom" having an additional parameter L: TCom"(v, (M, N), r) := L^v · M · Tr, which is also computationally binding by AWFCDH: consider an adversary producing (v, (M, N), (R, S)) and (v', (M', N'), (R', S')) with TCom"(v, (M, N), r) = TCom"(v', (M', N'), r'); then the case v ≠ v' is reducible to AWFCDH—as for TCom—and r = r' is reducible to CDH, which is implied by AWFCDH.

Scheme 2 (Sig"). Setup" has input grp = (p, G_1, G_2, G_T, e, G, H) and outputs grp and additional generators F, K, L, T ← G_1. The message space is DH := {G^m, H^m} \ m \in Z_p}. KeyGen" chooses x ← Z_p and outputs (vk = VK(x), sk = x), with VK(x) := (G^x, H^x) Sign" has inputs a secret key x and a message (v, (M, N)) ∈ Z_p × DH. It chooses random c, r ← Z_p and outputs

(A := (K · L^v · M · Tr) 1/m, B := F^c, D := H^c, R := G^r, S := H^r).
Ver" on inputs a public key \((X, Y) \in \mathcal{DH}\), a message \((v, (M, N)) \in \mathbb{Z}_p \times \mathcal{DH}\) and a signature \((A, B, D, R, S)\) outputs 1 (and 0 otherwise) iff the following hold:

\[
e(A, Y \cdot D) = e(K \cdot L^v, M, H) e(T, S) \quad e(B, H) = e(F, D) \quad e(R, H) = e(G, S)
\] (15)

Theorem 2. Assuming q-ADHSDH and AWFCDH, \(\text{Sig}_A\) is strongly existentially unforgeable against adversaries making at most \(q - 1\) adaptive chosen-message queries.

A formal proof of Theorem 2 can be found in Appendix B.

9.2 Verifiably Encrypting a Signature on a Public Integer and a Committed Message

A commitment to a signature on an integer \(v\) and a message committed in C is of the form \((\pi_A, \pi_B, \pi_R)\) for equations \(E_B\) and \(E_R\) as in (12) and \(E_{A''}\) defined as

\[
E_{A''}(A, M; S, D) : = e(T^{-1}, S) e(A, Y) e(M, H^{-1}) e(A, D) = e(K \cdot L^v, H)
\] (16)

By Lemma 1, proofs are independent of the right-hand side of the equation, thus \(\pi_{A''}\) is defined like \(\pi_A\). This also holds for proofs about all other equations such as \(E_A\), \(E_A\) and \(E_{A'}\) and their variants for \(\text{Sig}''\), since going from \(\text{Sig}\) to \(\text{Sig}''\) only affects the right-hand sides of the equations.

Proofs for \(\text{E}_{\text{Ver}(\cdot)}\) and \(\text{E}_{\text{Ver}''(\cdot)}\) are thus the same for all combinations of keys, messages and signatures given as commitments or in the clear. This means that the proof-adaptation algorithms \(\text{AdPrC}, \text{AdPrC}_M, \text{AdPrC}_K, \text{AdPrDC}, \text{AdPrDC}_M\) and \(\text{AdPrDC}_K\) defined in Sect. 8.3 all can be used for proofs about committed \(\text{Sig}''\) signatures as well. The only functionality that has to be slightly adapted is \(\text{SigCom}\). We define \(\text{SigCom}''(ck, sk, v, C)\) as \(\text{SigCom}\) in Fig. 2, except that \(A\) is defined as \(A := (K \cdot L^v \cdot T^{-1} \cdot U)\). We do not need to modify SmSigCom, since \(\Sigma\) is given as input to it.

Note that if in the construction of a blind signature in Sect. 4.2 we replace \(\text{Sig}\) and \(\text{SigCom}\) by \(\text{Sig}''\) and \(\text{SigCom}''\), respectively, we obtain partially blind signatures [AF96], where the signer controls part of the message.

10 A Note on Simulatability of Proofs

Groth and Sahai [GS08] show that pairing-product equations with a right-hand side \(t_T\) of the form

\[
t_T = e(P_1, Q_1) \cdots e(P_n, Q_n)
\] (17)

can be simulated: in the witness-indistinguishability setting (i.e., when \(ck^* \leftarrow \text{WISetup}\); cf. Sect. 6.2), given as simulation trapdoor \(\text{sim}\) the values \((\alpha_1, t_1, \alpha_2, t_2)\) used to construct \(ck^*\) one can construct commitments and proofs of validity for an equation of the above form without knowing a witness, i.e., elements that satisfy the equation.

Equations with right-hand side 1 ("homogeneous equations") can be simulated directly, since they have a trivial witness. Equations with a non-trivial right-hand side as in (17) must be transformed to a new set of equations to be simulatable: in the original equation the values \(P_i\) are replaced by variables \(V_i\) (which makes the equation homogeneous) and for each \(i\) we add the multi-scalar multiplication equation\(^9\) \(V_i \cdot P_i = 1\), where the commitment for \(d\) will be a trivial commitment to 1 (Since the randomness for the commitment of \(d\) is 0, we can check that the committed value is 1, which gives us \(V_i = P_i\) from the additional equations, and thus soundness of the construction.) In the simulation, we can now set all variables from \(G_1\) and \(G_2\) to 1 (which is a satisfying witness for our transformed PPE), and can thus give commitments and proofs. The additional equations can be simulated, since given the trapdoor \(\text{sim}\), the commitment to \(d\) can be trapdoor-opened to 0 (see [GS08] for the details).

In Sect. 10.2 we show that modifying our verification equations for commuting signatures does not interfere with its functionality; thus we get simulatability, as required for anonymity of our credentials.

\(^9\)An equation of the form \(E(X_1, \ldots, X_m; y_1, \ldots, y_n) : \prod_{i=1}^m A_i^{x_i} \prod_{j=1}^m X_j^{y_j} = T\), over \(X_i \in G_1\) and \(y_j \in \mathbb{Z}_p\), is called multi-scalar-multiplication equation in \(G_1\). [GS08] show how to construct WI proofs for this type of equation.
10.1 Simulating Proofs of Knowledge with Given Commitments.

Groth and Sahai show that given $\text{sim}$ in the WI setting, for any simulatable PPE, a simulator can produce commitments and a proof of validity. However, they do not consider the case where some of the commitments are given to the simulator, i.e., it cannot produce them itself and in particular, it does not know their randomness.

In [BCC+09], to prove anonymity, simulations of this kind are required (see, e.g., the definition of SimProve in Appendix B of the full version). However, the authors do not explain how to achieve such simulation. We will show that the proofs used in our construction can all be simulated even when some of the commitments are fixed in advance.

Lemma 6. Let $E$ be as in (4) with $t_T = 1$ and $A_j = 1$ for indices $j \in J \subseteq \{1, \ldots, n\}$. Given commitments $d_j$ for $j \in J$, we can simulate $c_1, \ldots, c_m$ and $d_j$ for $j \notin J$ and a proof $\pi$ for $E$ and $(c_1, \ldots, c_m, d_1, \ldots, d_m)$ if we are given the simulation trapdoor sim for $ck^*$. A symmetric result holds for $c_i$ and $d_j$ interchanged, and $A_j$ replaced with $B_i$.

Proof. If the simulator can choose all the commitments $c_i$ and $d_j$, it sets the committed values to 1. Since these values satisfy an homogeneous equation, the simulator can make an honest proof using the randomness for the commitments. But if the commitments $(d_j)_{j \in J}$ are fixed and given to the simulator, it does not know the randomness $s_j$ s.t. $d_j = \text{Com}(ck^*, 1, s_j)$ for all $j \in J$. We show that the proof can nonetheless be construct. Let us look at the definition of $\text{Prove}(ck, E, (X_i, r_i)_{i=1}^n, (Y_j, s_j)_{j=1}^n, Z)$ in (7). For the case when $X_i = 1 = Y_j$ we have

$$
\phi := \left[ \sum_{j=1}^n \sum_{i=1}^m r_{ij} \gamma_{ij} s_j \right] \left[ \sum_{j=1}^n \sum_{i=1}^m r_{ij} \gamma_{ij} s_j \right] \otimes (Z \otimes \bar{v})
$$

$$
\theta := \left[ 1 \left( \prod_{j=1}^n A_j^{s_j} \right) \right] \otimes (Z \otimes \bar{u})
$$

which has to be construct without knowledge of $(s_j)_{j \in J}$, i.e., the values satisfying $d_j = \text{Com}(ck^*, 1, s_j)$. Let the simulation trapdoor $\text{sim} = (\alpha_1, \alpha_2, \beta_1, \beta_2)$ be s.t. $v_1 = (G_2, G_2^{\alpha_2})$ and $v_2 = (G_2^{\beta_2}, G_2^{\alpha_2 \beta_2 - 1})$ (see Sect. 6.2). Let $(k_j, l_j)$ be the (unknown) logarithms of $d_j$, i.e., $d_{j,1} = G_2^{k_j}$ and $d_{j,2} = G_2^{l_j}$. Then we have

$$(G_2^{k_j}, G_2^{l_j}) = d_j = \text{Com}(ck^*, 1, s_j) = (v_{11}^{s_j}, v_{21}^{s_j}, v_{12}^{s_j}, v_{22}^{s_j}) = (G_2^{s_j + \beta_1 s_j}, G_2^{s_j + \beta_2 s_j}, G_2^{\alpha_2 s_j + \alpha_2 \beta_2 s_j - s_j})$$

Solving for $s_j$ and $s_j$ we get $s_j = k_j - \alpha_2 \beta_2 k_j + \beta_2 l_j$ and $s_j = \alpha_2 k_j - l_j$. The simulator can thus compute

$$G_2^{s_j} = d_{j,1}^{(1-\alpha_2 \beta_2)} \cdot d_{j,2}^{\alpha_2 \beta_2} \quad G_2^{s_j} = d_{j,1}^{\alpha_2 \beta_2} \cdot d_{j,2}^{(1-\alpha_2 \beta_2)}$$

and use these values to compute $\phi$, since it knows the logarithms of all $v_{ij}$ as well as all $r_{ij}$ and $\gamma_{ij}$, and $\theta$, e.g.

$$v_{22} = \sum_{j=1}^n s_j \sum_{i=1}^m r_{ij} = (G_2^{s_j + \beta_1 s_j})^{\alpha_2 \beta_2 - 1} = \prod_{j=1}^n (G_2^{s_j})^{\alpha_2 \beta_2 - 1} \sum_{i=1}^m r_{ij} \gamma_{ij} = (\prod_{j \in J} (d_{j,1}^{\alpha_2 \beta_2} \cdot d_{j,2}^{(1-\alpha_2 \beta_2)})^{\alpha_2 \beta_2 - 1} \sum_{i=1}^m r_{ij} \gamma_{ij}) (\prod_{j \notin J} G_2^{s_j})^{\alpha_2 \beta_2 - 1} \sum_{i=1}^m r_{ij} \gamma_{ij})$$

Since $A_j = 1$ for $j \in J$, it is straightforward to compute $\theta$. \\[QED\]

The above lemma lets us simulate a committed message, a committed signature and a proof of validity for a given committed public key (since in $E_{\tilde{A}^\theta}$ given in (19) below, the implicit) constant that is paired with $Y$ is 1, thus the premise of the lemma is satisfied). To prove anonymity of our construction of a delegatable-credential scheme in Sect. 5 we moreover need to simulate a verifiably encrypted signature on a given committed message; this requires simulation of a proof for equation $E_{\tilde{A}^\theta}$, where the constant ($H^{-1}$) that is paired with the value whose commitment is given ($M$) is not trivial; however, its logarithm $-1$ is known to the simulator.

We give a strengthening of Lemma 6, where the $A_j$ are of the form $G_2^{a_j}$ with $a_j$ known to the simulator.
Lemma 7. Let $E$ be as in (4) with $t^\tau = 1$ and $A_j = G_i^{a_j}$ (with $a_j$ known) for indices $j \in J$. Given commitments $d_j$ for $j \in J$, we can simulate $c_1, \ldots, c_n$ and $d_j$ for $j \notin J$ and a proof $\pi$ for $E$ and $(c_1, \ldots, c_m, d_1, \ldots, d_m)$ if we are given the simulation trapdoor sim for $ck^*$. A symmetric result holds for $c_i$ and $d_j$ interchanged, and $A_j = G_i^{a_j}$ replaced with $B_i = G_i^{b_i}$.

Proof. For simplicity, we give a proof in the additive notation of Remark 4. Since $\vec{X} = (0, \ldots, 0)^\top = \vec{Y}$, we have

$$\vec{c} = R\vec{u}, \quad \phi = R^\top \iota(\vec{B}) + (R^\top \Gamma S - Z^\top)\vec{v}$$
$$\vec{d} = S\vec{v}, \quad \theta = S^\top \iota(\vec{A}) + Z\vec{u}$$

Let us denote by $\vec{\tilde{A}}^\prime$ the vector $\vec{A}$ where all $A_j$ with $j \notin J$ are replaced by 0, and by $\vec{\tilde{A}}''$ the vector $\vec{A}$ where all $A_j$ with $j \in J$ are replaced by 0, and let $S'$ and $S''$ be defined analogously. We have $\vec{A} = \vec{\tilde{A}}^\prime + \vec{\tilde{A}}''$ and $S = S' + S''$, and moreover $S^\top \iota(\vec{A}) = (S')^\top \iota(\vec{A}^\prime) + (S'')^\top \iota(\vec{A}'')$. Note that the simulator only knows the logarithms of $\vec{A}'$ and the values in $S''$.

Since in the WI setting the matrix $\vec{u}$ is invertible, there exists $\Omega \in \mathbb{Z}_p^{2\times 2}$ s.t. $\iota(\vec{A}^\prime) = \Omega \vec{u}$. The logarithms of $\iota(\vec{A}^\prime)$ and $\vec{u}$ being known to the simulator, it can efficiently compute $\Omega$. We now show how the simulator computes the proof $(\phi, \theta)$: it chooses $\vec{Z} \leftarrow \mathbb{Z}_p^{2\times 2}$ and (knowing the values in $S''$) computes $\theta := (S'')^\top \iota(\vec{A}'') + \vec{Z}\vec{u} = S^\top \iota(\vec{A}) - (S')^\top \iota(\vec{A}^\prime) + \vec{Z}\vec{u}$, which is a proof $\theta = S^\top \iota(\vec{A}) + Z\vec{u}$ with $Z := \vec{Z} - (S')^\top \Omega$. The first part of the proof is then

$$\phi = R^\top \iota(\vec{B}) + (R^\top \Gamma S - \vec{Z}^\top + \Omega^\top S')\vec{v},$$

which can be constructed using the techniques of the proof of Lemma 6: it suffices to construct the elements in (18) and use the known logarithms of $\vec{v}$ as well as the known values $R$ and $\Omega$. \hfill \Box

10.2 Making the Equations for Ver'' Simulatable

In our application in Sect. 5.2 we have to simulate proofs for the equations of $E_{\text{ver''}^\cdot(v,\cdot)}(vk, (M, N), \Sigma)$, when the commitments for $vk = (X, Y)$ or $(M, N)$ (or both!) are given to the simulator.

While $E_B$ and $E_R$ from (12) have a trivial right-hand side, we replace $E_{A''}$ from (16) by the equations

$$E_{A''} (A ; W, S, N, D) : e(K \cdot L', W) e(T^{-1}, S) e(G^{-1}, N) e(A, Y) e(A, D) = 1$$
$$E_{d''} (d ; W) : W^d \cdot H^{-d} = 1$$

where, besides transforming $E_{A''}$ into a homogeneous equation and a multi-scalar multiplication equation\(^{10}\) as described by [GS08], we replaced $e(M, H^{-1})$ by $e(G^{-1}, N)$ which by $E_M(M; N)$ is equal. Accordingly, we replace $E_U$ by

$$E_{U''} (Q, N) : e(T^{-1}, Q) e(G^{-1}, N) = e(U, H)^{-1}$$

which, together with $E_M$ and $E_P$, still asserts that $U = T^4 \cdot M$. Note that in addition, this equation is linear in the sense of Groth-Sahai and a proof $\pi_{U''}$ thus reduces to an element from $\mathbb{G}_1^3$, whereas $\pi_U \in \mathbb{G}_1^4 \times \mathbb{G}_2^4$.

Next, we show how to adapt SigCom'' (which is the algorithm in Fig. 2 with $A$ defined as $(K \cdot L', \cdot T^r \cdot U)^{\pi''}$ at the beginning). All that needs to be done to define SigCom'' is replacing $E_{A''}$ by

$$E_{A''} (A ; W, D) : e(K \cdot L', W) e(A, Y) e(A, D) = 1.$$

Setting $\pi_{A''} \leftarrow \pi_{U''} \circ \text{Prove}(ck, E_{A''}^1, (A, \alpha), (W, \omega), (H^c, \delta))$ in Fig. 2 yields thus a proof for $E_{A''}$ by Lemma 3, since the product of the left-hand sides of $E_{A''}$ and $E_{U''}$ is the left-hand side of $E_{A''}$. The proof for the additional (multi-scalar multiplication) equation can be produced by the signer herself.

We demonstrated how our instantiation of commuting signatures based on Sig'' can be adapted to make the equations for Ver'' simulatable. Below, we show that they can even be simulated when commitments to the verification key and/or the message are given to the simulator.

\(^{10}\)We chose to turn $H$ into a variable, since proofs for equations in $\mathbb{G}_2$ are in $\mathbb{G}_1^4 \times \mathbb{G}_2^4$ and thus smaller than proofs for equations in $\mathbb{G}_1$. 27
**Simulating Com**. When the simulator needs to simulate a Com commitment it does the following: set c_M and c_N to commitments to 1. This enables simulation of other proofs for equations about M (such as those in Ver”). Since Com also contains the value U = T^1.M, the simulator has to choose t randomly, which defines P and Q. Now the simulator can produce c_P, c_Q, π_M, π_P, π_U honestly. Note that the fact that c_P and c_Q were not produced as commitments to 1 is not a problem, as they are never used outside of a Com commitment.

**Simulating Ver” for Fixed Commitments.** In the proof of anonymity of our credential scheme, we have to construct algorithms SimCredProve and SimIssue that output Groth-Sahai proofs without being given witnesses. The proofs are for validity of certificates contained in credentials, thus about the equations in Ver” from Scheme 2. The only equation that contains parts of a verification key or the message of is the following

\[ E_{A_S^M}(A; W, S, N, Y, D) : e(K^{-1}, W) e(T^{-1}, S) e(G^{-1}, N) e(A, Y) e(A, D) = 1 . \] (19)

**Corollary 1.** Given commitments c_{sk} and C, the simulator can produce (c_Σ, π) that is distributed as

\[
\left[ \Sigma \leftarrow \text{Sign}''(sk, (M, N)); \rho \leftarrow \mathcal{R} : \left( \text{Com}(ck^*, \Sigma, \rho), \text{Prove}(ck^*, \pi, \text{Ver}''(\cdot, \cdot, \cdot), ((X, \xi), (Y, \psi)), ((M, \mu), (N, \nu)), (\Sigma, \rho)) \right) \right],
\]

where \( vk = (X, Y) \) and \((\xi, \psi)\) are such that \( c_{sk} = \text{Com}(ck^*, vk, (\xi, \psi)) \), sk is such that \( vk = VK(sk) \), and M, N, \( \mu \) and \( \nu \) are such that \( C = (\text{Com}(ck^*, M, \mu), \text{Com}(ck^*, N, \nu), \ldots) \).

**Proof.** Simulating Ver” means simulating \( E_{A_S^M}, E_{A_S}, E_B \) and E_R. The simulator makes commitments c_Σ to \((1, \ldots, 1)\). Proofs \( \pi_B \) and \( \pi_R \) are computed honestly and the first equation satisfies the premises of Lemma 7: the constants (the “A_j” in (4)) that are paired with N and Y are \( G^{-1} \) and 1, respectively, and thus have known logarithms. \( \pi_{A_S} \) can thus be simulated. \( \pi_{A_S} \) is simulated by opening \( c_d := \text{Com}(ck^*, 1, 0) \) to 0, as described in [GS08].

**11 Conclusions**

In this paper we defined and instantiated a new primitive we call commuting signatures. They allow users to encrypt different components of a triple consisting of a verification key, a message and a signature, and prove validity of the encrypted values. Most importantly, they enable signers that are given an encrypted message to produce an encryption of a signature on it together with a proof of validity.

We showed that this primitive enables the first instantiation of delegatable anonymous credentials with noninteractive issuing and delegation. Moreover, using our instantiation, the efficiency of the credential scheme improves significantly compared to the (only) previous instantiation. We believe that commuting signatures are an important tool in the construction of privacy-preserving primitives and that they will find further applications.

**Acknowledgments**

This work was supported by EADS, the French ANR 07-TCOM-013-04 PACE Project, and the European Commission through the ICT Program under Contract ICT-2007-216646 ECRYPT II.

**References**


A Details and Proofs of our Commuting-Signature Instantiation

A.1 Commitments on Messages

We define a commitment on a message \((M, N) \in DH\) as the values \(C = (c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U)\) that the user sends to the signer in the issuing protocol for blind signatures from [Fuc09]. We then show how to randomize algorithms also get the parameters \(pp\) as input.

\[\text{Com}_M\] has inputs \(pp = (ck, grp, F, K, T), (M, N) \in DH\) and \((t, \mu, \nu, \rho, \sigma) \in \mathbb{Z}_p^5 =: \mathcal{R}_M\). Recall that \(ck\) is composed of \(u = (u_1, u_2) \in G_1^{2 \times 2}\) and \(v = (v_1, v_2) \in G_2^{2 \times 2}\). We define the following equations:

\[
E_{DH}(M, N) := e(G^{-1}, N)e(M, H) = 1
\]

and

\[
E_U(M, Q) := e(T^{-1}, Q)e(M, H^{-1}) = e(U, H)^{-1}
\]

\[\text{Com}_M(pp, (M, N), (t, \mu, \nu, \rho, \sigma))\] defines \(P = G^t\) and \(Q = H^t\), computes

\[
c_M := \text{Com}(ck, M, \mu) = (u_{11}^{\mu} u_{12}^{\mu}, M u_{11}^{\mu} u_{22}^{\mu})
\]

\[
c_N := \text{Com}(ck, N, \nu) = (v_{11}^{\nu} v_{12}^{\nu}, N v_{11}^{\nu} v_{22}^{\nu})
\]

\[
c_P := \text{Com}(ck, P, \rho) = (u_{11}^{\rho} u_{12}^{\rho}, P u_{11}^{\rho} u_{22}^{\rho})
\]

\[
c_Q := \text{Com}(ck, Q, \sigma) = (v_{11}^{\sigma} v_{12}^{\sigma}, Q v_{11}^{\sigma} v_{22}^{\sigma})
\]

\[
\pi_M \leftarrow \text{Prove}(ck, E_{DH}, (M, \mu), (N, \nu))
\]

\[
\pi_P \leftarrow \text{Prove}(ck, E_{DH}, (P, \rho), (Q, \sigma))
\]

\[
U := T^t \cdot M
\]

\[
\pi_U \leftarrow \text{Prove}(ck, E_U, (M, \mu), (Q, \sigma))
\]

and returns \(C = (c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U) \in C_M\).

\[\text{RdCom}_M\] on input \((ck, pp_S), C\) and \((t', \mu', \nu', \rho', \sigma')\), the algorithm first defines \(\hat{c}_P := c_P \circ (1, G^t), \hat{c}_Q := c_Q \circ (1, H^t)\) and \(U' := U \cdot T^{t'}\). It sets

\[
\hat{c}_M := \text{RdCom}(ck, c_M, \mu')
\]

\[
\hat{\pi}_M \leftarrow \text{RdProof}(ck, E_{DH}, (c_M, \mu'), (c_N, \nu'), \pi_M)
\]

\[
\hat{c}_N := \text{RdCom}(ck, c_N, \nu')
\]

\[
\hat{\pi}_P \leftarrow \text{RdProof}(ck, E_{DH}, (\hat{c}_P, \rho'), (c_Q, \sigma'), \pi_P)
\]

\[
\hat{c}_P := \text{RdCom}(ck, \hat{c}_P, \rho')
\]

\[
\hat{\pi}_U \leftarrow \text{RdProof}(ck, E_U, (c_M, \mu'), (c_Q, \sigma'), \pi_U)
\]

\[
\hat{c}_Q := \text{RdCom}(ck, \hat{c}_Q, \sigma')
\]

and returns \(C' = (c'_M, c'_N, \pi'_M, c'_P, c'_Q, \pi'_P, U', \pi'_U) \in C_M\).

\[\text{Extr}_M\] has inputs \(ek\) and \(C\). It returns \((\text{Extr}(ek, c_M), \text{Extr}(ek, c_N))\).

\(C \in C_M\) is efficiently verifiable by parsing it as \((c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U)\) and checking the proofs \(\pi_M, \pi_P\) and \(\pi_U\).

**Theorem 3.** \(\text{Com}_M\) is a randomizable extractable commitment scheme that is perfectly binding and computationally hiding.

**Proof.** The commitment \(C = (c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U)\) is binding by the corresponding property of SXDH commitments. A correctly constructed commitment contains valid proofs; in particular, we have \(e(U, H) = e(T^t, H) e(M, H) = e(T, Q) e(M, H)\), thus (21) is satisfied.

The scheme is computationally hiding as defined Sect. 3.1: let \(ck^* \leftarrow \text{WlSetup}\). Then for every \((M, N) \in DH\) there exists \(t\) s.t. \(U = T^t \cdot M\). Moreover there exist \(\mu, \nu, \rho\) and \(\sigma\) s.t. \(c_M = \text{Com}(ck, M, \mu), c_N = \text{Com}(ck, N, \nu), c_P := \text{Com}(ck, G^t, \rho),\) and \(c_Q := \text{Com}(ck, H^t, \sigma)\). So for every \(C\) and every \((M, N) \in DH\) there exists \(r := (t, \mu, \nu, \rho, \sigma) \in \mathcal{R}_M\) s.t. \(C = \text{Com}_M(ck^*, (M, N), r)\).
Moreover, RdComM randomizes a commitment. When \((U, \mathbf{c}, c_Q)\) is replaced by \((U', \mathbf{c}, \tilde{c}_Q)\) in the first step, \(t\) is replaced by \(t + t'\) (since the commitments are homomorphic, \(\tilde{c}_P\) is a commitment to \(P \cdot G^t\) and \(c'_Q\) commits to \(Q \cdot H^t\); note that \(\tau_P\) and \(\tau_U\) do not depend on \(t\) but only on the randomness of the commitments—which is not changed in the first step.) In the second step, \((\mu, \nu, \rho, \sigma)\) is replaced by \((\mu + \mu', \nu + \nu', \rho + \rho', \sigma + \sigma')\).

A.2 Making Commitments to a Signature and a Proof of Validity

We show how the signer can use the values in \(C\) to produce a proof of knowledge

\[
\begin{align*}
(c_A, c_B, c_D, c_R, c_S, \pi_A, \pi_B, \pi_R) & \in \mathbb{G}_1^{18} \times \mathbb{G}_2^{16}
\end{align*}
\]

of a signature \((A, B, D, R, S)\) where \(\pi_A, \pi_B\) and \(\pi_R\) are proofs that the committed values satisfy the equations in (9), respectively, i.e.

\[
\begin{align*}
E_A(A; M; S, D) : e(T^{-1}, S) e(A, Y) e(M, H^{-1}) e(A, D) & = e(K, H) \\
E_B(B; D) : e(F^{-1}, D) e(B, H) & = 1 \\
E_R(R; S) : e(G^{-1}, S) e(R, H) & = 1
\end{align*}
\]

(22)

Instantiating the formulas from Sect. 6.3, for equations \(E_{DH}\) and \(E_U\) (as defined in (20) and (21)), the proofs \(\pi_M, \pi_P\) and \(\pi_U\) are computed by choosing random \(Z_M, Z_P, Z_U \leftarrow \mathbb{Z}_p^{2 \times 2}\) and (using the notation defined in (5)) setting

\[
\begin{align*}
\pi_{M,1} & = \begin{bmatrix} 1 & H_{\mu_1} \\ 1 & H_{\mu_2} \end{bmatrix} \circ (Z_M \oplus \bar{v}) & \pi_{P,1} & = \begin{bmatrix} 1 & H_{\nu_1} \\ 1 & H_{\nu_2} \end{bmatrix} \circ (Z_P \oplus \bar{v}) & \pi_{U,1} & = \begin{bmatrix} 1 & H_{-\mu_1} \\ 1 & H_{-\mu_2} \end{bmatrix} \circ (Z_U \oplus \bar{v}) \\
\pi_{M,2} & = \begin{bmatrix} 1 & G_{-\sigma_1} \\ 1 & G_{-\sigma_2} \end{bmatrix} \circ (Z_M \oplus \bar{u}) & \pi_{P,2} & = \begin{bmatrix} 1 & G_{-\sigma_1} \\ 1 & G_{-\sigma_2} \end{bmatrix} \circ (Z_P \oplus \bar{u}) & \pi_{U,2} & = \begin{bmatrix} 1 & T_{-\sigma_1} \\ 1 & T_{-\sigma_2} \end{bmatrix} \circ (Z_U \oplus \bar{u})
\end{align*}
\]

(23)

In the blind signature from [Fuc09], on receiving \(C\), the signer checks the proofs contained in it, and then produces a pre-signature by choosing \(c, r \leftarrow \mathbb{Z}_p\) and computing

\[
\begin{align*}
A : = (K \cdot T^r \cdot U)^{\frac{1}{x+c}} & \quad B : = F^c & \quad D : = H^c & \quad R' : = G^r & \quad S' : = H^r
\end{align*}
\]

Knowing \(t\) s.t. \(U = T^t \cdot M\), these values can be turned into a signature on \((M, N)\) by setting \(R : = R' \cdot G^t\) and \(S : = S' \cdot H^t\). (Because \(A = (K \cdot T^r \cdot U)^{1/(x+c)} = (K \cdot T^{r+t} \cdot M)^{1/(x+c)}\), \(R = G^{t+t}\), and \(S = H^{t+t}\).) Since the commitments are homomorphic, the signer can—without knowledge of the values \(P = G^t\) and \(Q = H^t\)—make commitments on \(R\) and \(S\):

\[
\begin{align*}
c_R & : = c_P \circ \text{Com}(ck, R', 0) = \text{Com}(ck, R, \mu) & \quad c_S : = c_Q \circ \text{Com}(ck, S', 0) = \text{Com}(ck, S, \nu)
\end{align*}
\]

The signer also chooses \(\alpha, \beta, \delta \leftarrow \mathbb{Z}_p^2\), and makes the remaining commitments:

\[
\begin{align*}
c_A & : = \text{Com}(ck, A, \alpha) & \quad c_C & : = \text{Com}(ck, B, \beta) & \quad c_D & : = \text{Com}(ck, D, \delta)
\end{align*}
\]

The vector \(\mathbf{c}_S : = (c_A, c_B, c_D, c_R, c_S)\) are thus commitments on the actual signature \(\Sigma = (A, B, D, R, S)\). It remains to construct proofs \(\pi_A, \pi_B\) and \(\pi_R\) that the committed values satisfy the 3 equations in (22). Instantiating the proofs given in (7) for the concrete equations, we get the following:

\[
\begin{align*}
\pi_{A,1} & = \begin{bmatrix} v_{11}^{\alpha_1 \delta_1} & v_{21}^{\alpha_1 \delta_2} \\ v_{12}^{\alpha_2 \delta_1} & v_{22}^{\alpha_2 \delta_2} \end{bmatrix} (YD)^{\alpha_1} H^{-\mu_1} v_{12}^{\alpha_1 \delta_1} v_{21}^{\alpha_2 \delta_2} \circ (Z_A \oplus \bar{v}) & \pi_{R,1} & = \begin{bmatrix} 1 & H_{\mu_1} \\ 1 & H_{\mu_2} \end{bmatrix} \circ (Z_R \oplus \bar{v}) \\
& = \begin{bmatrix} 1 & T_{-\sigma_1} A_{\delta_1} \\ 1 & T_{-\sigma_2} A_{\delta_2} \end{bmatrix} \circ (Z_A \oplus \bar{u}) & \pi_{R,2} & = \begin{bmatrix} 1 & G_{-\sigma_1} \\ 1 & G_{-\sigma_2} \end{bmatrix} \circ (Z_R \oplus \bar{u})
\end{align*}
\]

(24)
The algorithm SigCom as defined in Fig. 3 (and Fig. 2), is commutative.

\textbf{Proof.} The algorithm SigCom optimizes the steps discussed informally above by constructing and randomizing \( c_R \) and \( c_S \) in one step. Moreover, \( c_A, c_B \) and \( c_D \) need not be randomized since SigCom chooses their randomness; in addition, being produced “freshly”, \( \pi_B \) need not be randomized either.

We formally prove that the output of the algorithm is correctly distributed. Recall that by \( \text{Prove}(\ldots; Z) \) we denote the output of \( \text{Prove} \) when \( Z \in \mathbb{Z}_p^2 \times 2 \) is the randomness used.

Let \( C = (c_M, c_N, \pi_M, c_P, c_Q, \pi_P, U, \pi_U) \) and let \((M, N) \in DH\) and \( \text{rand} := (t, \mu, \nu, \rho, \sigma) \) be such that \( C = \text{Com}_M(ck, (M, N), \text{rand}) \); in particular, since \( \text{Com}_M \) is perfectly binding, we have \( U = T^s \cdot M \) and

\begin{align*}
\pi_M := & \text{Prove}(ck, E_{DH}, (M, \mu), (N, \nu); Z_M) & \pi_U := & \text{Prove}(ck, E_U, (M, \mu), (H^t, \sigma); Z_U) \\
\pi_P := & \text{Prove}(ck, E_{DH}, (G^t, \rho), (H^t, \sigma); Z_P)
\end{align*}

\textbf{Correctness.} Let \( c, r, \alpha, \beta, \delta, \rho', \sigma' \) be the values chosen by SigCom. We have

\begin{align*}
c_A := & \text{Com}(ck, (K \cdot T^+ \cdot U)^{\frac{1}{2}}, \alpha) & c_B := & \text{Com}(ck, F^t, \beta) & c_D := & \text{Com}(ck, H^t, \delta) \\
c_R := & \text{Com}(ck, G^{t+r}, \rho + \rho') & c_S := & \text{Com}(ck, H^{t+s}, \sigma + \sigma')
\end{align*}
where the first equation follows from the definition of $U$ and the last two from the homomorphic property of $\text{Com}$. Note that values $(A, B, D, R, S)$ committed in $(e_A, e_B, e_D, e_R, e_S)$ compose a valid signature, in particular

$$(A, B, D, R, S) = \text{Sign}(x, (M, N); (c, r + t)) .$$

Define $\pi'_A$ as in Fig. 3 and let $\pi'_R := \pi_P$. In the discussion above we showed the following:

$\pi'_A := \text{Prove}(ck, E_A, (A, \alpha), (M, \mu), (S, \sigma), (D, \delta); Z_U)$ 
$\pi'_R := \text{Prove}(ck, E_{DH}, (P, \rho), (Q, \sigma); Z_P)$

Let $Z_A, Z_B$ and $Z_R$ be the randomness used by RdProof (or Prove) in the construction of $\pi_A, \pi_B$ and $\pi_R$, respectively. By the properties of RdProof (cf. Remark 4) we have the following:

$\pi_A := \text{Prove}(ck, E_A, (A, \alpha), (M, \mu), (S, \sigma + \sigma'), (D, \delta); Z_U + Z_A + Z')$
$\pi_R := \text{Prove}(ck, E_R, (R, \rho + \rho'), (S, \sigma + \sigma'); Z_P + Z_R + Z'')$
$\pi_B := \text{Prove}(ck, E_{DH}, (B, \beta), (D, \delta); Z_B)$

where the $Z'$ is defined by $\alpha, \mu$ and $\sigma'$ and $Z''$ is defined by $\rho$ and $\sigma$ (cf. equation (8) in Remark 4).

The resulting output $(e_A, e_B, e_D, e_R, e_S, \pi_A, \pi_B, \pi_R)$ of $\text{SigCom}$ is thus the same as the values constructed the following way:

$$(A, B, D, R, S) := \Sigma = \text{Sign}(x, (M, N), (\hat{c}, \hat{r}))$$
$$(e_A, e_B, e_D, e_R, e_S) := \text{Com}(ck, (A, B, D, R, S), (\hat{\alpha}, \hat{\beta}, \hat{\delta}, \hat{\rho}, \hat{\sigma}))$$
$$\pi_A := \text{Prove}(ck, E_A, (A, \hat{\alpha}), (M, \hat{\mu}), (S, \hat{\sigma}), (D, \hat{\delta}); \hat{Z}_A)$$
$$\pi_B := \text{Prove}(ck, E_B, (B, \hat{\beta}), (D, \hat{\delta}); \hat{Z}_B)$$
$$\pi_R := \text{Prove}(ck, E_{DH}, (R, \hat{\rho}), (S, \hat{\sigma}); \hat{Z}_R)$$

when $(\hat{\alpha}, \hat{\beta}, \hat{\delta}, \hat{\rho}, \hat{\sigma}, \hat{Z}_A, \hat{Z}_B, \hat{Z}_R)$ are defined as

$\hat{c} = c$  \quad $\hat{r} = r + t$  \quad $\hat{\alpha} = \alpha$  \quad $\hat{\beta} = \beta$  \quad $\hat{\delta} = \delta$
$\hat{\rho} = \rho + \rho'$  \quad $\hat{\sigma} = \sigma + \sigma'$  \quad $\hat{Z}_A = Z_U + Z_A + Z'$  \quad $\hat{Z}_B = Z_B$  \quad $\hat{Z}_R = Z_P + Z_R + Z''$

All these values are uniformly random since $c, r, \alpha, \beta, \delta, \rho', \sigma', Z_A, Z_B,$ and $Z_R$ are chosen uniformly and independently at random by $\text{SigCom}$. \hfill \Box

**Instantiation of $\text{SmSigCom}$**. This algorithm similarly does what $\text{SigCom}$ does but instead of being given the signing key $sk$, it is directly given a signature. It proceeds similarly to $\text{SigCom}$ but starting from a signature instead of a pre-signature. It first uses $ek$ to extract $P$ and $Q$ from $C$. It then chooses $\alpha, \beta, \delta \leftarrow R$ and sets $c_A := \text{Com}(ek, A, \alpha)$, $c_B := \text{Com}(ek, B, \beta)$, and $c_D := \text{Com}(ek, D, \delta)$. It moreover sets

$$e_R := e_P \circ \text{Com}(ck, R; P^{-1}, 0) = \text{Com}(ck, R, \rho) \quad e_S := e_Q \circ \text{Com}(ck, S; Q^{-1}, 0) = \text{Com}(ck, S, \sigma)$$

It can now define $\pi_A$ as in (25) and $\pi_R$ as $\pi_P$ from $C$, and produce $\pi_B$ using $\beta$ and $\delta$. Randomize everything and output $(e_A, e_B, e_D, e_R, e_S, \pi_A, \pi_B, \pi_R)$.

### A.3 Instantiations of Proof Adaptation for Committing and Decommitting

**Adaptation of Proofs for Committing and Decommitting to Signatures and Messages.** In Fig. 4 we rewrote the proofs contained in a commitment $C$ and the proofs for the first verification relation of the signatures, depending on which elements are committed. $\pi_A$ is the proof when both signature and message are committed, $\pi_{\tilde{A}}$ when only the signature is committed and $\pi_{\tilde{A}}$ for when only the message is committed. We also give $\pi_{\tilde{A}}$, a proof for when only the elements $A$ and $D$ of a signature are committed.
The last equation allows us to implement the algorithms AdPrC, AdPrC_M, AdPrDC and AdPrDC_M.

AdPrC(pp, vk, C, ((A, B, D, R, S), (α, β, δ, ρ, σ), π)). The proof π is a proof for equation E_A. The algorithm sets

$$\pi_A \leftarrow \text{Prove}(ck, E_A, (A, α), (S, σ), (D, δ))$$

for E_B and E_R as defined in (22). It then returns $$π := (π_A, π_B, π_R)$$.

AdPrC_M(pp, vk, ((M, N), (t, μ, ν, ρ, σ)), cΣ, π). The proof π is of the form $$(π_A, π_B, π_R)$$. The algorithm sets $$π_A \leftarrow \text{Prove}(ck, E_A, (A, α), (S, σ), (D, δ))$$ and returns a randomization of $$π := (π_A, π_A, π_A)$$.

AdPrDC(pp, vk, C, ((A, B, D, R, S), (α, β, δ, ρ, σ), π)). The proof π is of the form $$(π_A, π_B, π_R)$$. The algorithm sets $$π_A \leftarrow \text{Prove}(ck, E_A, (A, α), (S, σ), (D, δ))$$ and returns $$π := π_A ⊕ π_A$$ (where “⊕” denotes component-wise division, that is: replace all the components of the second argument by their inverses and then multiply them with those of the first argument).
AdPrDC_M(pp, vk, ((M, N), (t, μ, ν, ρ, σ)), c_Σ, π). The proof π is of the form (π_A, π_B, π_R). The algorithm produces π_̂_A ← Prove(ck, E_̂_A, (M, μ)) and returns a randomization of π := (π_A ⊙ π_̂_A, π_B, π_R).

Instantiation of Proof Adaptation when Committing to the Verification Key In some applications (such as the one discussed in Sect. 5), in order to remain anonymous, the signer makes a commitment

(c_X = \text{Com}(ck, X, ξ), c_Y = \text{Com}(ck, Y, ψ), π_X = \text{Prove}(ck, E_DH, (X, ξ), (Y, ψ)))

to his public key and wishes to prove that the values in c_Σ are a valid signature on the values (M, N) in C under the public key that is committed in (c_X, c_Y, π_X). The first equation of verification is thus

E_̂_A(A, M; S, Y, D) := e(T^{-1}, S) e(M, H^{-1}) e(A, Y)e(A, D) = e(K, H),

for which, by (7), the proof is

\begin{align*}
\pi_{A,1} & := \begin{bmatrix}
\nu_{11}^{\alpha_1(\psi_1 + \delta_1)} \nu_{21}^{\alpha_2(\psi_1 + \delta_2)} \\
\nu_{11}^{\alpha_2(\psi_1 + \delta_1)} \nu_{21}^{\alpha_2(\psi_1 + \delta_2)}
\end{bmatrix}
(\begin{bmatrix}
YD & \alpha_1 H^{-\mu_1} \nu_{12}^{\alpha_1(\psi_1 + \delta_1)} \nu_{22}^{\alpha_2(\psi_1 + \delta_2)} \\
YD & \alpha_2 H^{-\mu_2} \nu_{12}^{\alpha_1(\psi_1 + \delta_1)} \nu_{22}^{\alpha_2(\psi_1 + \delta_2)}
\end{bmatrix}) \odot (Z_\hat{A} \otimes \bar{v}) \\
\pi_{A,2} & := \begin{bmatrix}
1 & 1
\end{bmatrix}
(\begin{bmatrix}
T^{-\sigma_1} A^{\psi_1 + \delta_1} \\
T^{-\sigma_2} A^{\psi_1 + \delta_2}
\end{bmatrix}) \odot (Z_\hat{A} \otimes \bar{u})
\end{align*}

(26)

whereas E_B and E_R do not change. Given a commitment C to a message, a commitment c_Σ = (c_A, c_B, c_D, c_R, c_S) to a signature and a proof π = (π_A, π_B, π_R) of validity, π_A can be adapted to π_̂_A using the technique discussed in Sect. 7.4. Using the randomness ψ for c_Y, choose Z′ ∈ \mathbb{Z}_p^{2×2} and set

\begin{align*}
\pi_{A,1} & := \pi_{A,1} \circ (Z′ \otimes \bar{v}) \\
\pi_{A,2} & := \pi_{A,2} \circ \begin{bmatrix}
c_{A,1}^{\psi_1} & c_{A,2}^{\psi_1} \\
c_{A,1}^{\psi_2} & c_{A,2}^{\psi_2}
\end{bmatrix} \circ (Z′ \otimes \bar{u})
\end{align*}

We show why this yields a proof for E_̂_A. Let Z ∈ \mathbb{Z}_p^{2×2} denote the randomness used in π_A. Set

\[\tilde{Z} := \begin{bmatrix}
z_{11} + z_{11}′ + α_1 \psi_1 & z_{12} + z_{12}′ + α_2 \psi_1 \\
z_{21} + z_{21}′ + α_1 \psi_2 & z_{22} + z_{22}′ + α_2 \psi_2
\end{bmatrix}.\]

Then, using the definition of π_A from (24), we have

\begin{align*}
\pi_{A,1} & := \begin{bmatrix}
\nu_{11}^{\alpha_1(\psi_1 + \delta_1)} \nu_{21}^{\alpha_2(\psi_1 + \delta_2)} \\
\nu_{11}^{\alpha_2(\psi_1 + \delta_1)} \nu_{21}^{\alpha_2(\psi_1 + \delta_2)}
\end{bmatrix}
(\begin{bmatrix}
YD & \alpha_1 H^{-\mu_1} \nu_{12}^{\alpha_1(\psi_1 + \delta_1)} \nu_{22}^{\alpha_2(\psi_1 + \delta_2)} \\
YD & \alpha_2 H^{-\mu_2} \nu_{12}^{\alpha_1(\psi_1 + \delta_1)} \nu_{22}^{\alpha_2(\psi_1 + \delta_2)}
\end{bmatrix}) \odot (\tilde{Z} \otimes \bar{v}) \\
\pi_{A,2} & := \begin{bmatrix}
1 & 1
\end{bmatrix}
(\begin{bmatrix}
(T^{-\sigma_1} A^{\psi_1} & \psi_1 \\
(T^{-\sigma_2} A^{\psi_2} & \psi_2
\end{bmatrix}) \circ ((Z + Z′) \otimes \bar{u}) = \begin{bmatrix}
1 & 1
\end{bmatrix}
\begin{bmatrix}
T^{-\sigma_1} A^{\psi_1} \\
T^{-\sigma_2} A^{\psi_2}
\end{bmatrix} \odot (\tilde{Z} \otimes \bar{u})
\end{align*}

which is a proof for equation E_̂_A as detailed in (26) when Z_\hat{A} := \tilde{Z}.

B Proof of Theorem 2

Consider an adversary that after receiving parameters (G, F, K, L, T, H) and public key (X, Y) is allowed to ask for q − 1 signatures (A_i, B_i, D_i, R_i, S_i) on messages (u_i, (M_i, N_i)) ∈ \mathbb{Z}_p × D_H of its choice and then outputs (u, (M, N)) ∈ \mathbb{Z}_p × D_H and a valid signature (A, B, D, R, S) on it, such that either (u, (M, N)) was never
queried, or \((u_i(M, N)) = (u_i(M_i, N_i))\) and \((A, B, D, R, S) \neq (A_i, B_i, D_i, R_i, S_i)\). We distinguish three kinds of forgers: An adversary is called of Type I if its output satisfies the following
\[
\forall 1 \leq i \leq q - 1: \left[ e(T, S \cdot S_i^{-1}) \neq e(L^{u_i} \cdot M_i \cdot L^{-u_i} \cdot M^{-1}, H) \lor B \neq B_i \right]
\] (27)
An adversary is called of Type IIa if its output satisfies
\[
\exists 1 \leq i \leq q - 1: \left[ e(T, S \cdot S_i^{-1}) = e(L^{u_i} \cdot M_i \cdot L^{-u_i} \cdot M^{-1}, H) \land B = B_i \land S \neq S_i \right]
\] (28)
otherwise it is called of Type IIb. We will use the first type to break \(q\)-ADHSDH, Type IIa to break AWFCDH and Type IIb to break CDH, which is implied by AWFCDH.

**Type I**  Let \((G, F, K, X, H, Y, (A_i, B_i, V_i, D_i, W_i)_{i=1}^{q-1})\) be a \(q\)-ADHSDH challenge. It satisfies thus
\[
e(A_i, Y-D) = e(K \cdot L^{u_i} \cdot M, H) e(T, S).
\] (30)
The tuple \((A, B, D, V := G^{1-u} \cdot M^R, W := H^{1-u} \cdot N \cdot S^t)\) satisfies (27), since \((B, D)\) and \((V, W)\) are Diffie-Hellman pairs and \(e(K, V, H) = e(K \cdot L^{u_i} \cdot M, H) e(T, S)\). Moreover, it is a solution for the ADHSDH instance, since it is a new tuple: assume that for some \(i\) we have \(A = B_i\) and \(W = W_i\), that is \(H^{1-u} \cdot N \cdot S^t = H^{1-u_i} \cdot N_i \cdot S_i^t\). Since \((M, N), (M_i, N_i) \in \mathcal{D} H\), we have \(e(T, S) e(L^{u_i} \cdot M, H) = e(T, S) e(G, H^{1-u} \cdot N) = e(G, H^{1-u} \cdot N \cdot S^t) = e(G, H^{1-u_i} \cdot N_i \cdot S_i^t) = e(T, S_i) e(G, H^{1-u} \cdot N_i) = e(T, S_i) e(L^{u_i} \cdot M_i, H)\). We have thus \(e(T, S \cdot S_i^{-1}) = e(L^{u_i} \cdot M_i \cdot L^{-u_i} \cdot M^{-1}, H) = B = B_i\) which contradicts (27) and thus the fact that \(A\) is of Type I.

**Type IIa**  Let \((G, H, T = G^t)\) be an AWFCDH instance; let \(A\) be a forger of Type IIa. Pick \(F, K \leftarrow \mathbb{G}_1\) and \(l, x \leftarrow \mathbb{Z}_t\), set \(X := G^x, Y := H^x\) and give the adversary parameters \((G, F, K, L := G^t, T, H)\) and public key \((X, Y)\). Answer a signing query on \((u_i, (M_i, N_i)) \in \mathbb{Z}_t \times \mathcal{D} H\) by returning a signature \((A_i, B_i, D_i, R_i, S_i)\) produced by \(\text{Sign}_A(x, \cdot)\). Suppose \(A\) returns \(((A, B, D, R, S), (u_i, (M, N)))\) satisfying (15) s.t. \(e(T, S \cdot S_i^{-1}) = e(L^{u_i} \cdot M_i \cdot L^{-u_i} \cdot M^{-1}, H)\), \(B = B_i\) and \(S \neq S_i\) for some \(i\). Then \((M^* := L^{u_i} \cdot M_i \cdot L^{-u_i} \cdot M^{-1}, N^* := H^{1-u_i} \cdot N_i \cdot H^{-l} \cdot N^{-1}, R^* := R \cdot R_i^{-1}, S^* := S \cdot S_i^{-1})\) is a AWFCDH solution: \((S^*, M^*), (M^*, N^*)\) and \((R^*, S^*)\) satisfy the respective equations in (3), and since \(S \neq S_i\) it is non-trivial.

**Type IIb**  Let \((G, H, L := G^l)\) be a CDH instance, i.e., we have to produce \(H^l\). Let \(A\) be a forger of Type IIb. Pick \(F, K, T \leftarrow \mathbb{G}_1\) and \(x \leftarrow \mathbb{Z}_t\), set \(X := G^x, Y := H^x\) and give the adversary parameters \((G, F, K, L, T, H)\) and public key \((X, Y)\). Answer a signing query on \((u_i, (M_i, N_i)) \in \mathbb{Z}_t \times \mathcal{D} H\) by returning a signature \((A_i, B_i, D_i, R_i, S_i)\) produced by \(\text{Sign}_A(x, \cdot)\). Suppose \(A\) returns \(((A, B, D, R, S), (u_i, (M, N)))\) satisfying (15) of Type IIa, i.e., \(e(T, S \cdot S_i^{-1}) = e(L^{u_i} \cdot M_i \cdot L^{-u_i} \cdot M^{-1}, H)\), \(B = B_i\) and \(S = S_i\) for some \(i\); which implies \(L^{u_i} \cdot M_i = L^u \cdot M\).

We first show that \(u \neq u_i\): Suppose \(u = u_i\); then by the above we have \(M = M_i\), and moreover \(B = B_i\) and \(S = S_i\). Since these values completely determine \(A, D, R, \text{ and } N\), we have \((A, B, D, R, S, u, M, N) = (A_i, B_i, D_i, R_i, S_i, u_i, M_i, N_i)\), which means that \(A\) did not break strong unforgeability.

From \(L^{u_i} \cdot M_i = L^u \cdot M\) we have \(L^{-u_i} = M_i \cdot M^{-1}\) and since \(u \neq u_i\) we have \(L = (M_i \cdot M^{-1})^{1/u_i}\), which for \(m := \log_G M = \log_H N, m_i := \log_G M_i = \log_H N_i\) can be written as \(G^{m_i - m} = H^{m_i - m_i}\) is a CDH solution.  

\(\Box\)
C Additional Tools and Their Instantiations

In Sect. 6.4, we introduced the scheme from [Fuc09] to sign two public keys at once, which can easily be extended for arbitrary many messages. The scheme Sig* however has message space \( \mathcal{M}^* := \mathcal{D} \mathcal{H}^* = \mathcal{D} \mathcal{H} \setminus \{(1, 1)\} \). In this section, we define randomizable, extractable commitments to elements from \( \mathcal{M}^* \), and show how to make a commitments to a signature and a proof of validity on a clear and a committed message from \( \mathcal{M}^* \). We define the following:

**Committing to \( \mathcal{M}^* \) Elements.** We define \( \text{Com}_{\mathcal{M}^*}^* \) that has the same properties as \( \text{Com}_{\mathcal{M}} \), but with value space \( \mathcal{M}^* \) rather than \( \mathcal{M} \). By \( \text{Com}_{\mathcal{M}^*}^* \) we denote the commitment space and by \( \mathcal{R}_{\mathcal{M}^*}^* \) the space of randomness. Our instantiation is given in Appendix C.1.

**Partially Blind Automorphic Signatures.** Since Sig* (cf. Sect. 6.4) signs two messages, we can define a variant of SigCom that gets one message in the clear and one committed message. Based on Sig* and \( \text{Com}_{\mathcal{M}^*}^* \), we define PSigCom that is given a message \( M \in \mathcal{M}^* \) and a \( \text{Com}_{\mathcal{M}^*}^* \) commitment and outputs a proof of knowledge of a Sig* signature on \( M \) and the committed value:

\[
\text{PSigCom}(ck, sk, M, C).
\]

If \( M \in \mathcal{M}^* \) and \( C \in \mathcal{C}_{\mathcal{M}^*}^* \) then the algorithm outputs a commitment to a signature and a proof of validity \((c_{\Sigma}, \pi)\) which is distributed as:

\[
\left[ \Sigma \leftarrow \text{Sign}^*(sk, (M, V)); \rho \leftarrow \mathcal{R} : \left( \text{Com}(ck, \Sigma, \rho), \text{Prove}(ck, \text{E}_\text{Ver}^*(\text{sk}, (M, \cdot)), (V, \nu), (\Sigma, \rho)) \right) \right],
\]

where \( V \) and \( \nu \) are such that \( C = \text{Com}_{\mathcal{M}^*}^*(ck, V, \nu) \).

Note that if in the construction of a blind signature in Sect. 4.2 we replace Sig, \( \text{Com}_{\mathcal{M}} \) and SigCom by Sig*, \( \text{Com}_{\mathcal{M}^*}^* \) and PSigCom, we obtain partially blind signatures [AF96] (where the signer controls part of the message), which are automorphic themselves.

C.1 Commitments to Non-Trivial Messages

We instantiate \( \text{Com}_{\mathcal{M}^*}^* \), with message space \( \mathcal{M}^* := \mathcal{D} \mathcal{H}^* = \{(G^m, H^m) \mid m \in \mathbb{Z}_p \setminus \{0\}\} \). To guarantee that the committed value is not \((1, 1)\), \( \text{Com}_{\mathcal{M}^*}^* \) contains additional elements. Intuitively, given \( (M, N) \in \mathcal{M}^* \) if we choose \( l \leftarrow \mathbb{Z}_p^* \) and publish \( W = N^l \) then \( W \neq 1 \) if \( N \neq 1 \). We add a commitment \( c_L \) to \( G^l \) and a proof \( \pi_W \) that \( e(G^l, N) = e(G, W) \), which proves well-formedness of \( W \). In the WI setting \( W, c_L \) and \( \pi_W \) perfectly hide \( N \).

Randomization of a \( \text{Com}_{\mathcal{M}^*}^* \) commitment is a bit trickier. Since \( l \) must be from \( \mathbb{Z}_p^* \), randomize it multiplicatively, that is, we choose \( l' \leftarrow \mathbb{Z}_p^* \) and replace \( l \) by \( l \cdot l' \). This also enables randomization of \( W \) as \( W' := W^{l'} \) which without knowledge of \( N \) cannot be done additively. Finally, Lemma 4 (Sect. 7.4) shows how to adapt \( c_L \) and \( \pi_W \) to the new value \( l \cdot l' \).

We define \( \text{Com}_{\mathcal{M}^*}^* \) by extending \( \text{Com}_{\mathcal{M}} \) from Sect. 8.1:

\[
\text{Com}_{\mathcal{M}^*}^*(pp, (M, N), (\kappa = (t, \mu, \nu, \rho, \sigma), \eta, l)).
\]

If \( (M, N) \in \mathcal{D} \mathcal{H}^*, (\kappa, \eta, l) \in \mathcal{R}_{\mathcal{M}} \times \mathcal{R} \times \mathbb{Z}_p^* =: \mathcal{R}_{\mathcal{M}}^* \), then define:

\[
W := N^l \quad c_L := \text{Com}(ck, G^l, \eta) \quad \pi_W \leftarrow \text{Prove}(ck, E_W, (G^l, \eta), (N, \nu))
\]

with \( E_W(L; N) = e(L, \mathbb{N}) = e(G, W) \), and output \((\text{Com}_{\mathcal{M}^*}^*(ck, (M, N), \kappa), W, c_L, \pi_W)\).

The space of valid commitments is \( \mathcal{C}_{\mathcal{M}^*}^* := \{ (C, W, c_L, \pi_W) \mid C \in \mathcal{C}_{\mathcal{M}} \land W \neq 1 \land \text{Verify}(ck, E_W, c_L, c_N, \pi_W) \} \). \( \text{Com}_{\mathcal{M}^*}^* \) is shown to be binding and computationally hiding analogously to \( \text{Com}_{\mathcal{M}} \); in particular if \( ck^* \) is a WI key then for every \( (M, N) \in \mathcal{D} \mathcal{H}^* \), given \((C, W, c_L, \pi_W)\) there exists \((\kappa, \eta, l)\) such that \( C = \text{Com}_{\mathcal{M}^*}^*(ck^*, (M, N), \kappa), W = N^l \), \( c_L = \text{Com}(ck^*, G^l, \eta) \) and \( \pi_W \leftarrow \text{Prove}(ck, E_W, (G^l, \eta), (N, \nu)) \) with \( \kappa = (t, \mu, \nu, \rho, \sigma) \).

Using the results from Sect. 7, we define \( \text{RdCom}_{\mathcal{M}^*}^* \) by extending \( \text{RdCom}_{\mathcal{M}} \):

\[
\text{RdCom}_{\mathcal{M}^*}^*(pp, (C, W, c_L, \pi_W), (\kappa', \eta', l')) \text{ returns a commitment } (C', W', c'_L, \pi'_{W'}) \text{ that is equivalent to the output of } \text{Com}_{\mathcal{M}^*}^*(pp, (M, N), (\kappa + \kappa', l' \cdot \eta + \eta', l \cdot l')) \text{ defined as } C' := \text{RdCom}_{\mathcal{M}^*}^*(pp, C, \kappa') \text{ and}
\]
\[ W' := W^{l''} \quad c_L' := \text{RdCom}(ck, (c_L', \eta')) \quad \pi_W' \leftarrow \text{RdProof}(ck, E_W, (c_L', \eta'), (e_N, \nu'), \pi_W) \]

RdCom\textsubscript{\text$\kappa$}' works as follows: the part C of a commitment is randomized by RdCom\textsubscript{\text$\kappa$} which replaces $\kappa$ by $\kappa + \kappa'$. Now $W'$ is replaced by $W^{l''}$, which implicitly replaces $l$ by $l - l'$. Setting $\hat{c}_L := c_L'$, we get $\hat{c}_L = \text{Com}(ck, L^{l''}, l' \cdot \eta)$ and by Lemma 4, we have that $\hat{\pi}_W := \pi_W'$ is a proof for $E_W$ and $(\hat{c}_L, e_N)$. In $W'$, $\hat{c}_L$ and $\hat{\pi}_W$, randomness $l$ has thus consistently been replaced by $l - l'$. The final step is to set $c_L' := \text{RdCom}(ck, \hat{c}_L, \eta')$ and $\pi_W' \leftarrow \text{RdProof}(ck, E_W, (\hat{c}_L, \eta'), (e_N, \nu'), \hat{\pi}_W)$. Note that $c_L'$ is thus a commitment to $L''$ under randomness $l' \cdot \eta + \eta'$.

If $(\kappa, \eta, l)$ and $(\kappa', \eta', l')$ are both uniformly chosen from $\mathcal{R}_{\text{\text$\kappa$}}$ then the randomness after randomization is also uniform in $\mathcal{R}_{\text{\text$\kappa$}}$.

### C.2 Making Commitments to a Signature on a Public and a Committed Message and a Proof of Validity

We give an instantiation of PSigCom at the beginning of the section. We start by giving a variant of SigCom that has inputs $(ck, sk, (V, W), C)$ with $(V, W) \in \mathcal{M}^*$ and $C \in \mathcal{C}_{\text{\text$\kappa$}}^*$ and outputs a proof of knowledge of a signature on $(V, W) \equiv (M, N)$, where $(M, N)$ is the message committed in $C$. The verification of a signature on such a product $\text{Ver}'((X, Y), (V, W), (M, N), (A, B, D, R, S))$ are the equations $E_{A'}(A, M; S, D) = e(T^{-1}, S) e(A, Y) e(M, H^{-1}) e(A, D) = e(K \cdot V, H)$, and $E_B, E_R$ as defined in (12). Since the left-hand sides of $E_{A'}$ and $E_A$ from (12) are equivalent, by Lemma 1 both equations have the same proofs: $\pi_A = \pi_{A'}$. The only thing that changes w.r.t. SigCom is thus the value $A$ of the pre-signature.

SigCom\textprimed$(ck, x, (V, W), C)$. This variant is defined as SigCom in Fig. 2, except that $A := (K \cdot T' \cdot U \cdot V)^{\frac{1}{1 + \pi}}$.

Using SigCom\textprimed, the definition of PSigCom is straightforward. Note that AdPrC\textsubscript{$\kappa$} can also be applied to outputs of SigCom\textprimed since proofs only depend on the left-hand sides of their equation. Let $E_{\hat{A'}}$ be $E_{A'}$ with $Y$ being a variable. Since $E_{A'}$ and $E_A$ as well as $E_{\hat{A'}}$ and $E_{\hat{A'}}$ have the same left-hand sides, AdPrC\textsubscript{$\kappa$} also transforms a proof for $E_{A'}$ into one for $E_{\hat{A'}}$.

PSigCom$(ck, sk, (V, W), C)$.

- $(vk^*, sk^*) \leftarrow \text{KeyGen}_S; \tau \leftarrow \mathcal{R}_{\text{\text$\kappa$}}; C_{vk^*} := \text{Com}_{\text{\text$\kappa$}}(ck, vk^*, \tau); (c_{\Sigma_0}, \pi_0) \leftarrow \text{SigCom}(ck, sk, C_{vk^*})$
- $(c_{\Sigma_1}, \pi_1') \leftarrow \text{SigCom}(ck, sk^*, C); \pi_1 \leftarrow \text{AdPrC\textsubscript{$\kappa$}}(ck, (vk^*, \tau), C, c_{\Sigma_1}, \pi_1')$
- $(c_{\Sigma_2}, \pi_2') \leftarrow \text{SigCom}(ck, sk^*, (V, W), C); \pi_2 \leftarrow \text{AdPrC\textsubscript{$\kappa$}}(ck, (vk^*, \tau), C, c_{\Sigma_2}, \pi_2')$
- $(c_{\Sigma_3}, \pi_3') \leftarrow \text{SigCom}(ck, sk^*, (V, W)^3, C); \pi_3 \leftarrow \text{AdPrC\textsubscript{$\kappa$}}(ck, (vk^*, \tau), C, c_{\Sigma_3}, \pi_3')$
- Return $(c_{\Sigma} = (C_{vk^*}, c_{\Sigma_0}, c_{\Sigma_1}, c_{\Sigma_2}, c_{\Sigma_3}), \pi = (\pi_0, \pi_1, \pi_2, \pi_3)$

A proof of knowledge of a signature $(c_{\Sigma}, \pi)$ under $vk$ on the message pair $(V, W) \in \mathcal{M}^*$ and $(M, N)$, which is given as a commitment $C \in \mathcal{C}_{\text{\text$\kappa$}}^*$ is then verified by checking the following:

\[ C_{vk^*} \in \mathcal{C}_{\text{\text$\kappa$}}; \text{Verify}(ck, E_{\text{Ver}(vk^*, \cdot, \cdot)}, C_{vk^*}, c_{\Sigma_0}, \pi_0), \text{Verify}(ck, E_{\text{Ver}((\cdot, \cdot), \cdot, \cdot)}, (c_{vk^*}, C, c_{\Sigma_1}, \pi_1), \text{Verify}(ck, E_{\text{Ver}^{*}((\cdot, V, W), \cdot, \cdot)}, C_{vk^*}, C, c_{\Sigma_2}, \pi_2), \text{Verify}(ck, E_{\text{Ver}^{*}((\cdot, V, W), (\cdot, \cdot), \cdot)}, C_{vk^*}, C, c_{\Sigma_3}, \pi_3) \]

Proof adaptation AdPrC\textsubscript{$\kappa$} for a verifiable encrypted signature of the above form $(c_{\Sigma} = (C_{vk^*}, c_{\Sigma_0}, c_{\Sigma_1}, c_{\Sigma_2}, c_{\Sigma_3}), \pi = (\pi_0, \pi_1, \pi_2, \pi_3)$) is done by running $\hat{\pi}_0 \leftarrow \text{AdPrC\textsubscript{$\kappa$}}(ck, (vk, \xi), C_{vk^*}, c_{\Sigma_0}, \pi_0)$ and outputting $(\hat{\pi}_0, \pi_1, \pi_2, \pi_3)$. 

38