# Malleable Signatures: Complex Unary Transformations and Delegatable Anonymous Credentials

Melissa Chase
Microsoft Research Redmond
melissac@microsoft.com

Markulf Kohlweiss
Microsoft Research Cambridge
markulf@microsoft.com

Anna Lysyanskaya
Brown University
anna@cs.brown.edu

Sarah Meiklejohn
UC San Diego
smeiklej@cs.ucsd.edu

March 29, 2013

**Abstract**

A signature scheme is malleable if, on input a message $m$ and a signature $\sigma$, it is possible to efficiently compute a signature $\sigma'$ on a related message $m' = T(m)$, for a transformation $T$ that is *allowable* with respect to this signature scheme. Previous work considered various useful flavors of allowable transformations, such as quoting and sanitizing messages. In this paper, we explore a connection between malleable signatures and anonymous credentials, and give the following contributions:

- We define and construct malleable signatures for a broad category of allowable transformation classes, with security properties that are stronger than those that have been achieved previously. Our construction of malleable signatures is generically based on malleable zero-knowledge proofs, and we show how to instantiate it under the Decision Linear assumption.
- We construct delegatable anonymous credentials from signatures that are malleable with respect to an appropriate class of transformations; we also show that our construction of malleable signatures works for this class of transformations. The resulting concrete instantiation is the first to achieve security under a standard assumption (Decision Linear) while also scaling linearly with the number of delegations.

## 1 Introduction

A signature scheme is malleable — alternatively, homomorphic — if, given a signature $\sigma$ on a message $m$, it is possible to efficiently derive a signature $\sigma'$ on a message $m' = T(m)$ for an "allowable" transformation $T$; as an example, we might allow $m'$ to be any excerpt from $m$ (in this sense, the malleability of the signature scheme is "controlled"). Formal definitions of such signatures were first given by Ahn et al. [3], and their definitions were recently refined by Attrapadung et al. [4]. Without any additional restrictions on the size of the signature or the privacy of the original message $m$, a construction of a malleable signature is trivial: the signature $\sigma$ on $m$ is also automatically a signature on $T(m)$ for any allowable $T$. What makes this problem non-trivial is the following additional twist, called *context hiding*: the derived signature $\sigma'$ should not reveal anything about the original message $m$ that is not revealed by the message $m'$, and in fact it should be impossible to tell that $\sigma'$ was computed from $\sigma$, rather than issued by the signer directly. This definition can also apply to $n$-ary transformations $T$, and implicitly requires that the size of the signature depends only on the message (so, in particular, it cannot grow as

it is transformed unless the transformation outputs a message that requires a longer signature). Using appropriate classes of allowable transformations, malleable signatures are a generalization of a variety of existing signatures, such as quotable or redactable signatures.

A natural application for malleable signatures that has not been previously considered is an anonymous credential system [21, 14, 15]. In such a system, a user Alice is known in different contexts by different unlinkable pseudonyms (it's best to think of a pseudonym as an unconditionally binding, computationally hiding commitment to Alice's secret key), yet she can demonstrate possession of a credential issued to pseudonym $\mathsf{nym}$ to a verifier who knows her by another pseudonym, $\mathsf{nym}'$. Let $\mathcal{T}$ be the set of transformations that, on input some pseudonym $\mathsf{nym}$, output another pseudonym $\mathsf{nym}'$ of the same user, so that there exists some $T \in \mathcal{T}$ such that $\mathsf{nym}' = T(\mathsf{nym})$. Then a signature scheme that is malleable with respect to the set of transformations $\mathcal{T}$ gives us an anonymous credential system: a credential is simply an authority's signature $\sigma$ on $\mathsf{nym}$; malleability makes it possible for Alice to transform it into $\sigma'$ that is a signature on $\mathsf{nym}'$; context hiding should ensure that $\sigma'$ cannot be linked to the original pseudonym $\mathsf{nym}$ to which it was issued; and finally, unforgeability should ensure that Alice cannot compute $\sigma'$ unless she received a signature from the authority on one of her pseudonyms.

Somewhat surprisingly, not only do malleable signatures with respect to appropriate sets of transformations $\mathcal{T}$ yield anonymous credentials, but, as we show in this paper, they also yield *delegatable* anonymous credentials (DACs). A DAC system [20, 6] allows users to delegate their anonymous credentials; for example, a company employee can use his employee credential to issue a guest pass to a company visitor, who can in turn issue a credential to a taxi service that comes to pick her up; when presenting their credentials (and also when obtaining and delegating them), the various participants (the employee, his guest, and her driver) need not reveal any persistent identifiers — or in fact anything — about themselves. DACs are much more privacy-friendly than the traditional anonymous credential model, which assumes that the verifying party knows the public key of the credential issuer(s), as who issued Alice's credentials reveals a lot of information about Alice. For example, the identity of the local DMV that issued her driver's license might reveal her zip code. If, in addition to this, her date of birth and gender are leaked (as could happen in the context of a medical application), this is often enough to uniquely identify her [36, 27]. Verifiers that require more than one credential might learn Alice's identity even more easily. Since DACs protect the identity of every link on Alice's certification chain, they make it impossible to infer any information about Alice based on who issued her credentials.

In order to construct a DAC from malleable signatures, we essentially follow the same outline as for the (non-delegatable) anonymous credential, but consider a different class of transformations. In the non-delegatable scenario, the transformation took as input Alice's $\mathsf{nym}$ and output Alice's $\mathsf{nym}'$. In the DAC scenario, when Alice is delegating her credential to Bob, she uses a transformation $T$ that takes as input Alice's pseudonym $\mathsf{nym}_A$ and the length $\ell$ of her certification chain, and outputs Bob's pseudonym $\mathsf{nym}_B$ and the length of his new certification chain $\ell + 1$; to be allowable, $T$'s description must include Alice's secret key (recall that her pseudonym $\mathsf{nym}_A$ is a commitment to her secret key), so that only Alice can perform this transformation. Intuitively, it is easy to see that this construction yields a DAC; yet its security crucially relies on a transformation being allowable not by virtue of its input-output behavior, but by virtue of what its description contains. As a result, previous definitions of security for malleable and homomorphic signatures are not strong enough to be useful for this application: DAC require not only that an allowable transformation exist, but that the party applying it "know" its description; i.e. that there exist an extractor that, with a special extraction trapdoor, can compute the description of the transformation. Additionally, to ensure Alice's privacy even in the face of adversarial root authorities, context hiding must hold even for adversarially generated public keys for the signature scheme. This flavor of context hiding has not previously been achieved.

**Our contributions.** In this paper, we overcome the definitional obstacle explained above by first proposing, in Section 3, new definitions for malleable signatures. Our definition of context hiding, extending that of Attrapadung et al., allows for adversarially-generated keys and signatures. Our unforgeability definition requires that the transformation $T$ and the original message $m$ that was signed by the signing oracle be extractable from $(m', \sigma')$. To ensure that these definitions are not overly strong, we relate them to the relevant definitions of Ahn et al. and Attrapadung et al. and observe that, for many classes of transformations, the definitions of unforgeability are equivalent (whereas working with adversarially-generated keys makes our definition of context hiding strictly stronger than their computational definitions). With these new definitions in hand, we then follow the intuition developed above and construct, in Section 4, a delegatable anonymous credentials scheme generically from a malleable signature and — because the pseudonyms used in credentials seem to require it — a commitment scheme. Our new definitions for malleable signatures also conveniently allow us to provide a new definition, presented in Section 2, for credential unforgeability; our definition is both more powerful and considerably simpler than existing definitions. In addition to satisfying this new definition, our construction provides several desirable functional features (non-interactive delegation and the ability to delegate polynomially many times), all while relying on a standard assumption (in our concrete instantiation of choice, Decision Linear [11]).

Finally, to construct the signature that we generically rely on in our DAC construction, we provide in Section 5 a general construction of context-hiding malleable signatures for a large range of unary transformation classes. Our construction relies generically on non-interactive zero-knowledge (NIZK) proofs that provide controlled malleability; such proofs were recently defined and realized by Chase et al. [17] and allow, on input a proof $\pi$ for an instance $x \in L$, for the efficient computation of a proof $\pi'$ of a related instance $T_{\text{inst}}(x)$ for an allowable transformation $T = (T_{\text{inst}}, T_{\text{wit}})$, such that if $w$ is a witness for $x \in L$, then $T_{\text{wit}}(w)$ is a witness for $T_{\text{inst}}(x) \in L$. Aside from its usefulness in our construction of delegatable anonymous credentials, our signature construction enjoys other nice properties. Although it is not the first construction of signatures from zero-knowledge proofs — the Fiat-Shamir heuristic [24] is an example of this approach, as are the signatures of knowledge due to Chase and Lysyanskaya [20] and the construction using PRFs due to Bellare and Goldwasser [8] — ours is the first such construction to achieve malleability. In terms of constructions that focus on malleability, previous work gives *ad-hoc* constructions of malleable signatures for various allowable transformations (such as redactable signatures [31, 34, 16], quotable signatures [35], and transitive signatures [33, 9]), but ours is the first *general* efficient construction of malleable signatures. The only previous work that gave a general approach to homomorphic signatures was by Ahn et al. [3], who gave (among other contributions, such as an efficient construction of a quotable signature) an *inefficient* general construction for which a malleable signature on $m$ is essentially a set of non-malleable signatures on $\{m' \mid m' = T(m) \wedge T \in \mathcal{T}\}$.

**Related work on malleable signatures.** Here we distinguish between work on unary and $n$-ary transformations. As mentioned above, some specific types of unary homomorphic signatures have been studied over the last decade or so, such as redactable and quotable signatures in which, given a signature on a document $m$, one can derive signatures on a redacted version of $m$ in which some parts are blacked out, or signatures on quotations from $m$. These can be viewed as special motivating cases of context-hiding malleable signatures (although some of the constructions in these early papers do not meet more recent definitions). A somewhat related but different type of signature is an incremental signature scheme [7], in which a signature on a document can be efficiently updated when the document is updated. Recent work on computing on authenticated data [3, 4] gives a general definitional framework for the problem (which we draw on in our definitions) and some general (but inefficient, as discussed above) constructions for unary transformations, as well as some efficient and elegant provably secure constructions for specific unary transformation classes, such as quoting and subsets.

As far as $n$-ary homomorphisms are concerned, this research direction was initiated by work on transitive signatures [33, 9] (in which, given signatures on the edges $(u, v)$ and $(v, w)$ of a graph, one can derive a signature on the edge $(u, w)$), and by Rivest in a series of talks; the first paper to address this subject more generally and consider several binary transformations was by Johnson et al. [32]. A more recent line of work explored homomorphic signatures under linear and polynomial functions [13, 12, 25, 5]; the emphasis in the papers cited is in transforming $n$ message-signature pairs into a message-signature pair in which the message is some linear or polynomial function of the input messages. This is incomparable to our work because we are interested in more general transformations.

**Related work on delegatable anonymous credentials.** The first construction of delegatable anonymous credentials, by Chase and Lysyanskaya [20], allowed a constant number of delegations. Belenkiy et al. [6] gave the first DAC system that allowed for a polynomial number of delegations using Groth-Sahai proofs [30]; their construction, however, was somewhat ad-hoc and relied on ad-hoc assumptions. Finally, Fuchsbauer [26] gave a construction that built on the construction of Belenkiy et al. and allows for non-interactive issuing and delegation of credentials, also based on ad-hoc assumptions.

Our construction essentially combines many of the nicest features of each of these previous constructions. First, we support non-interactive issuing and delegation of credentials, as do Chase and Lysyanskaya, and Fuchsbauer, but not Belenkiy et al. Second, we support credentials that scale linearly with the number of times they are delegated, as do Belenkiy et al. and Fuschbauer, but not Chase and Lysyanskaya. Third, we can instantiate cm-NIZKs, and thus our malleable signature and entire credentials construction, under the standard and well-established Decision Linear assumption, whereas Belenkiy et al. and Fuschbauer both use less desirable assumptions. (The construction of Chase and Lysyanskaya is based on general assumptions.) Finally, we realize a simulation-extractable notion of delegatable anonymous credentials that is simpler and more efficiently realizable than any of the previous definitions.

## 2 Preliminaries and Notation

As our construction of a malleable signature in Section 5 depends on malleable proofs, we first discuss the definitions for such proofs here. We next recall existing definitions for delegatable anonymous credentials, and propose our new definition for credential unforgeability.

**Malleable proofs** In Section 5, we generically construct malleable signatures using malleable proofs. Briefly, a *malleable proof* [17] is a tuple of algorithms $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$, in which the first three algorithms constitute a standard non-interactive zero-knowledge proof of knowledge (defined formally in Appendix A). The fourth algorithm, $\mathsf{ZKEval}$, given a transformation $T = (T_{\text{inst}}, T_{\text{wit}})$, an instance $x$, and a proof $\pi$ such that $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$, outputs a proof $\pi'$ such that $\mathcal{V}(\mathsf{crs}, T_{\text{inst}}(x), \pi') = 1$; i.e., outputs a valid proof for the transformed instance. The proof system is then malleable with respect to some set of transformations $\mathcal{T}$ if for every $T \in \mathcal{T}$, $\mathsf{ZKEval}$ can be efficiently computed. (This is defined formally in a manner similar to our Definition 3.1.)

In addition to this basic definition of malleability, Chase et al. give amplified security notions for such proofs that are analogous to our amplified notions for signatures; we recall the formal definitions of their notions in Appendix A. The first, controlled-malleable simulation-sound extractability (CM-SSE) is a strong notion of extractability in which, from a valid proof $\pi$ for an instance $x$, an extractor can extract either a witness $w$ such that $(x, w) \in R$, or a previously proved instance $x'$ and transformation $T$ such that $x = T(x')$ and $T \in \mathcal{T}$. The second, derivation privacy, requires that a proof does not reveal whether it was constructed fresh (i.e., using a witness) or by transformation; a related definition, *strong* derivation privacy, requires instead that a proof does not reveal whether it was constructed by

the simulator or by transformation. Putting these together, if a proof is zero knowledge, CM-SSE, and strongly derivation private, then it is called a *cm-NIZK*.

**Delegatable anonymous credentials** At a high level, delegatable anonymous credentials (DAC) allow credentials to be both delegated and issued within the context of a system in which users are *pseudonymous*; i.e., may be using a different pseudonym for each of the different people with whom they interact. As such, algorithms are required for generating each of these pseudonyms, as well as issuing and delegating credentials, and proving (in an anonymous way) the possession of a credential. More formally, a delegatable anonymous credentials scheme consists of eight algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{NymGen}, \mathsf{NymVerify}, \mathsf{Issue}, \mathsf{CredProve}, \mathsf{CredVerify}, \mathsf{Delegate}$) that behave as follows:

- $\mathsf{Setup}(1^k)$: Generate public parameters *params* for the system.
- $\mathsf{KeyGen}(params)$: Generate public and secret keypair $(pk, sk)$; the secret key $sk$ represents a user's "true identity."
- $\mathsf{NymGen}(params, sk)$: Compute a pseudonym for the user corresponding to $sk$.
- $\mathsf{NymVerify}(params, \mathsf{nym}, sk, \mathsf{open})$: Check that a given pseudonym $\mathsf{nym}$ belongs to the user corresponding to $sk$. (In practice, this algorithm will never be run; instead, a user might form a proof of knowledge of a $sk$ corresponding to $\mathsf{nym}$ such that this holds.)
- $\mathsf{Issue}(params, sk_0, pk_0, \mathsf{nym}_r)$: Issue a credential, rooted at the authority who owns $pk_0$, to the pseudonym $\mathsf{nym}_r$.
- $\mathsf{CredProve}(params, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}', \mathsf{open}', \mathsf{cred})$: Prove possession of credential $\mathsf{cred}$ that has been delegated to $\mathsf{nym}$, where the owner of $\mathsf{nym}$ also owns a pseudonym $\mathsf{nym}'$.
- $\mathsf{CredVerify}(params, pk_0, \mathsf{nym}, \ell, \pi)$: Verify that the pseudonym $\mathsf{nym}$ is in possession of a level-$\ell$ credential, rooted at $pk_0$.
- $\mathsf{Delegate}(params, sk_{old}, \mathsf{nym}_{old}, \mathsf{open}_{old}, \mathsf{nym}_{new}, \mathsf{cred})$: Delegate the credential $\mathsf{cred}$, currently delegated to the pseudonym $\mathsf{nym}_{old}$, to the pseudonym $\mathsf{nym}_{new}$.

The main security requirements are anonymity and unforgeability. For anonymity, we require that pseudonyms hide their owners' secret keys, and that a proof of possession of a credential does not reveal the pseudonym to which the credential was initially issued by the root authority, or the pseudonyms to which the credential was delegated. For unforgeability, we require that one cannot prove possession or delegate a credential without knowing a secret key corresponding to some pseudonym to which a credential has been issued.

In order to conceptually identify users with pseudonyms, we require that a $\mathsf{nym}$ output by $\mathsf{NymGen}$ is a computationally hiding, unconditionally binding commitment to the underlying $sk$, such that $\mathsf{NymVerify}$ verifies the opening $(sk, \mathsf{open})$.

Our definition of anonymity is fairly similar to the definition given by Belenkiy et al.; the main modifications are our non-interactive $\mathsf{Issue}$ protocol and that simulated parameters are distributed identically to those output by $\mathsf{Setup}$. Essentially, we require that there exist a simulator $(\mathsf{SimSetup}, \mathsf{SimCred}, \mathsf{SimProve})$ such that (1) $\mathsf{SimSetup}$ produces parameters and the simulation trapdoor, (2) $\mathsf{SimCred}$ takes as input the parameters, the simulation trapdoor, the root authority public key, a pseudonym and a level, and produces credentials indistinguishable from those produced by $\mathsf{Issue}$ and $\mathsf{Delegate}$, and (3) $\mathsf{SimProve}$, given the same set of inputs as $\mathsf{SimCred}$, produces proofs indistinguishable from those produced by $\mathsf{CredProve}$.

Our definition of unforgeability, on the other hand, is a departure from that of Belenkiy et al. It is conceptually similar to the definition of simulation-sound extractability for non-interactive zero knowledge, in that we require that unforgeability should hold in the presence of a simulator that grants and proves possession of credentials at any level for any pseudonym of the adversary's choice without

access to the root authority's secret key. (In contrast, Belenkiy et al.'s unforgeability game required a parameter setting for which the simulator was undefined.)

Formally, we define an augmented setup algorithm $\mathsf{SimExtSetup}$ that produces $(params, \tau_s, \tau_e)$ such that $(params, \tau_s)$ is distributed identically to the output of $\mathsf{SimSetup}$. We then have the following definition.

**Definition 2.1** (Unforgeability)**.** *A delegatable anonymous credentials scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{NymGen}, \mathsf{NymVerify}, \mathsf{Issue}, \mathsf{CredProve}, \mathsf{CredVerify}, \mathsf{Delegate})$ *with simulator* $(\mathsf{SimSetup}, \mathsf{SimCred}, \mathsf{SimProve})$ *is* unforgeable *if there exists a pair* $(\mathsf{SimExtSetup}, \mathsf{Extract})$ *(where* $\mathsf{SimExtSetup}$ *augments* $\mathsf{SimSetup}$*) such that (1)* $\mathsf{nym}$ *is a computationally hiding, unconditionally binding commitment when params are chosen by* $\mathsf{SimExtSetup}$*, even when* $\tau_s$ *and* $\tau_e$ *are given; and (2) it is hard to form a proof of a credential for a pseudonym* $\mathsf{nym}$ *at level* $\ell$ *without knowing the secret key corresponding to* $\mathsf{nym}$ *as well as the secret key corresponding to some* $\mathsf{nym}_1$ *to which a credential at level* $\ell' \leq \ell$ *has been issued; formally, for any adversary* $\mathcal{A}$*, consider the following game, wherein* $\mathsf{SimCred}$*,* $\mathsf{SimProve}$*, and* $\mathsf{Extract}$ *share state:*

- *Step 1.* $(params, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimExtSetup}(1^k)$*;* $(vk_0, sk_0) \xleftarrow{\$} \mathsf{KeyGen}(params)$*.*
- *Step 2.* $((\mathsf{nym}, \ell), \pi) \xleftarrow{\$} \mathcal{A}^{\mathsf{SimCred}(params, \tau_s, vk_0, \cdot, \cdot), \mathsf{SimProve}(params, \tau_s, vk_0, \cdot, \cdot)}(params, vk_0)$*.*
- *Step 3.* $(\{(\mathsf{nym}_i, sk_i, \mathsf{open}_i)\}_{i=1}^k, \ell') \leftarrow \mathsf{Extract}(params, \tau_e, (vk_0, \mathsf{nym}, \ell), \pi)$*, where* $\mathsf{nym}_k = \mathsf{nym}$ *and* $\ell' + k - 2 = \ell$*.*

*Then for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that the probability (over the choices of* $\mathsf{SimExtSetup}$*,* $\mathsf{SimCred}$*,* $\mathsf{SimProve}$*, and* $\mathcal{A}$*) that* $\mathsf{CredVerify}(params, vk_0, \mathsf{nym}, \ell, \pi) = 1$ *and* $(\mathsf{nym}, \ell)$ *was not queried to* $\mathsf{SimProve}$ *but either*

1. $\mathcal{A}$ *created a new credential; i.e.,* $(\mathsf{nym}_1, \ell')$ *was not queried to* $\mathsf{SimCred}$ *or* $\ell < \ell'$*,*
2. $\mathcal{A}$ *delegated through pseudonyms it did not own; i.e.,* $\mathsf{NymVerify}(params, \mathsf{nym}_j, sk_j, \mathsf{open}_j) = 0$ *for some* $j$*,* $1 \leq j \leq k$*, or*
3. $\mathcal{A}$ *proved possession for a credential it did not own; i.e.,* $sk_{k-1} \neq sk_k$*,*

*is at most* $\nu(k)$*.*

Note that in the definition above, it is important that a pseudonym hides the underlying secret key even given the simulation and extraction trapdoors because otherwise the extractor's job would be too easy: it could just pick any pseudonym $\mathsf{nym}_1$ (with a low enough level) that the adversary has queried, use the extraction trapdoor to learn the corresponding $sk_1$, and pretend that that's the one from which the credential was delegated. This way, however, the extractor can only do this if the proof of possession $\pi$ is a proof of knowledge of, among other things, $sk_1$.

In addition to our new definition of credential unforgeability, we also give formal notions of correctness and of anonymity, which we briefly sketched above. Before we can establish the definition of correctness for a credentials scheme, we first need to define what it means for a credential to be valid.

**Definition 2.2** (Valid credential)**.** *For parameters params, we say that a value* $\mathsf{cred}$ *is a* valid *level-*$\ell$ *credential rooted at* $pk_0$ *and belonging to* $\mathsf{nym}$ *if proofs of the credential with respect to compatible pseudonyms always verify. More formally, for all* $sk, \mathsf{open}, \mathsf{open}'$ *such that* $\mathsf{NymVerify}(params, \mathsf{nym}, sk, \mathsf{open}) = 1$ *and* $\mathsf{NymVerify}(params, \mathsf{nym}', sk, \mathsf{open}') = 1$*,* $Pr[\pi \xleftarrow{\$} \mathsf{CredProve}(params, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}', \mathsf{open}', \mathsf{cred}) : \mathsf{CredVerify}(params, pk_0, \mathsf{nym}', \ell, \pi) = 1] = 1$*.*

In what follows, we use the notation $y \in D(x)$ to denote that $Pr[D(x) = y] > 0$; e.g., $params \in \mathsf{Setup}(1^k)$ indicates that *params* might have been produced by $\mathsf{Setup}$ (using some appropriate randomness).

**Definition 2.3** (Correctness)**.** *A delegatable anonymous credentials scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{NymGen}, \mathsf{NymVerify}, \mathsf{Issue}, \mathsf{CredProve}, \mathsf{CredVerify}, \mathsf{Delegate})$ *is* correct *if for all* $params \in \mathsf{Setup}(1^k)$ *and* $(pk_0, sk_0) \in \mathsf{KeyGen}(params)$*, the following properties are satisfied:*

- *Valid pseudonyms verify; i.e., for all* $(\mathsf{nym}, \mathsf{open}) \in \mathsf{NymGen}(params, sk_0)$*,* $\mathsf{NymVerify}(params, \mathsf{nym}, sk_0, \mathsf{open}) = 1$*.*
- *Only valid credentials verify; i.e., if there exist* $sk$*,* $\mathsf{open}$*, and* $\mathsf{open}'$ *with* $\mathsf{NymVerify}(params, \mathsf{nym}, sk, \mathsf{open}) = 1$ *and* $\mathsf{NymVerify}(params, \mathsf{nym}', sk, \mathsf{open}') = 1$*, such that* $\mathsf{CredVerify}(params, pk_0, \mathsf{nym}', \ell, \mathsf{CredProve}(params, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}', \mathsf{open}', \mathsf{cred})) = 1$*, then* $\mathsf{cred}$ *is a valid credential.*
- *The credential output by* $\mathsf{Issue}$ *verifies; i.e., for all* $(pk, sk) \in \mathsf{KeyGen}(params)$*,* $(\mathsf{nym}, \mathsf{open}) \in \mathsf{NymGen}(params, sk)$*, and* $\mathsf{cred} \in \mathsf{Issue}(params, sk_0, pk_0, \mathsf{nym})$*,* $\mathsf{cred}$ *is a valid level-*$1$ *credential, rooted at* $pk_0$ *and belonging to* $\mathsf{nym}$*.*
- *An honestly delegated credential verifies; i.e., for all* $(pk, sk), (pk_{new}, sk_{new}) \in \mathsf{KeyGen}(params)$*,* $(\mathsf{nym}, \mathsf{open}) \in \mathsf{NymGen}(params, sk)$*,* $(\mathsf{nym}_{new}, \mathsf{open}_{new}) \in \mathsf{NymGen}(params, sk_{new})$*, valid level-*$\ell$ *credentials* $\mathsf{cred}$ *rooted at* $pk_0$*, and* $\mathsf{cred}' \in \mathsf{Delegate}(params, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}_{new}, \mathsf{cred})$*,* $\mathsf{cred}'$ *is a valid level-*$\ell + 1$ *credential, rooted at* $pk_0$ *and belonging to* $\mathsf{nym}$*.*

**Definition 2.4** (Anonymity)**.** *A delegatable anonymous credentials scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{NymGen}, \mathsf{NymVerify}, \mathsf{Issue}, \mathsf{CredProve}, \mathsf{CredVerify}, \mathsf{Delegate})$ *is* anonymous *if there exist PPT algorithms* $\mathsf{SimSetup}$*,* $\mathsf{SimCred}$*,* $\mathsf{SimProve}$*, and* $\mathsf{VerifyPK}$ *such that the following properties are satisfied:*

- *For all* $params \in \mathsf{Setup}(1^k)$*,* $\mathsf{VerifyPK}(params, pk_0, sk_0) = 1$ *if and only if* $(pk_0, sk_0) \in \mathsf{KeyGen}(params)$*.*
- *The pseudonyms are hiding even given the trapdoors, i.e. for a bit* $b$ *and a PPT adversary* $\mathcal{A}$*, define* $p_b^{\mathcal{A}}(k)$ *to be the probability of the event that* $b' = 0$ *in the following game:*
  - *Step 1.* $(params, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimExtSetup}(1^k)$*.*
  - *Step 2.* $(vk_0, sk_0, vk_1, sk_1, \mathsf{state}) \xleftarrow{\$} \mathcal{A}(params, \tau_s, \tau_e)$*.*
  - *Step 3. If* $\mathsf{VerifyPK}(params, vk_0, sk_0) = 0$ *or* $\mathsf{VerifyPK}(params, vk_1, sk_0) = 0$*, output* $\perp$*. Otherwise, compute* $(\mathsf{nym}, \mathsf{open}) \leftarrow \mathsf{NymGen}(params, sk_b)$*.*
  - *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \mathsf{nym})$*.*

  *Then for all PPT algorithms* $\mathcal{A}$*, there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$*.*
- *The simulator* $\mathsf{SimCred}$ *can simulate issuing credentials; i.e. for all PPT adversaries* $\mathcal{A}$*,*

$$\begin{aligned} & Pr[params \xleftarrow{\$} \mathsf{Setup}(1^k) : \mathcal{A}^{\mathsf{Issue}'(params, \cdot, \cdot, \cdot)}(params) = 1] \\ \approx\ & Pr[(params, \tau_s) \xleftarrow{\$} \mathsf{SimSetup}(1^k) : \mathcal{A}^{\mathsf{SimCred}'(params, \tau_s, \cdot, \cdot, \cdot)}(params) = 1], \end{aligned}$$

  *where these oracles behave as follows: given* $(sk_0, pk_0, \mathsf{nym})$*, both* $\mathsf{Issue}'$ *and* $\mathsf{SimCred}'$ *output* $\perp$ *if* $\mathsf{VerifyPK}(params, pk_0, sk_0) = 0$*; otherwise,* $\mathsf{Issue}'$ *outputs* $\mathsf{Issue}(params, sk_0, pk_0, \mathsf{nym})$ *and* $\mathsf{SimCred}'$ *outputs* $\mathsf{SimCred}(params, \tau_s, pk_0, \mathsf{nym}, 1)$*.*
- *The simulator* $\mathsf{SimCred}$ *can simulate delegation of credentials at any level* $\ell$*; i.e., for a bit* $b$ *and a PPT adversary* $\mathcal{A}$*, define* $p_b^{\mathcal{A}}(k)$ *to be the probability of the event that* $b' = 0$ *in the following game:*
  - *Step 1.* $(params, \tau_s) \xleftarrow{\$} \mathsf{SimSetup}(1^k)$*.*
  - *Step 2.* $(\mathsf{state}, sk_{old}, \mathsf{nym}_{old}, \mathsf{open}_{old}, \mathsf{nym}_{new}, pk_0, \ell, \mathsf{cred}) \xleftarrow{\$} \mathcal{A}(params, \tau_s)$*.*

– *Step 3. If* $\mathsf{NymVerify}(params, \mathsf{nym}_{old}, sk_{old}, \mathsf{open}_{old}) = 0$ *or* $\mathsf{cred}$ *is not a valid level-*$\ell - 1$ *credential rooted at* $pk_0$ *belonging to* $\mathsf{nym}_{old}$*, output* $\perp$*.*[1] *Otherwise,*

$$\mathsf{cred}' \xleftarrow{\$} \begin{cases} \mathsf{Delegate}(params, sk_{old}, \mathsf{nym}_{old}, \mathsf{open}_{old}, \mathsf{nym}_{new}, \mathsf{cred}) & \text{if } b = 0 \\ \mathsf{SimCred}(params, \tau_s, pk_0, \mathsf{nym}_{new}, \ell) & \text{if } b = 1 \end{cases}$$

– *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \mathsf{cred}')$.

*Then for all PPT algorithms* $\mathcal{A}$*, there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

- *The simulator* $\mathsf{SimProve}$ *can simulate proving credentials for any level* $\ell$*; i.e., for a bit* $b$ *and a PPT adversary* $\mathcal{A}$*, define* $p_b^{\mathcal{A}}(k)$ *to be the probability of the event that* $b' = 0$ *in the following game:*
  – *Step 1.* $(params, \tau_s) \xleftarrow{\$} \mathsf{SimSetup}(1^k)$.
  – *Step 2.* $(\mathsf{state}, pk_0, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}', \mathsf{open}', \ell, \mathsf{cred}) \xleftarrow{\$} \mathcal{A}(params, \tau_s)$.
  – *Step 3. If* $\mathsf{NymVerify}(params, \mathsf{nym}, sk, \mathsf{open}) = 0$ *or* $\mathsf{NymVerify}(params, \mathsf{nym}', sk, \mathsf{open}') = 0$ *or* $\mathsf{cred}$ *is not a valid level-*$\ell - 1$ *credential rooted at* $pk_0$ *and belonging to* $\mathsf{nym}$*, output* $\perp$*. Otherwise,*

$$\pi \xleftarrow{\$} \begin{cases} \mathsf{CredProve}(params, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}', \mathsf{open}', \mathsf{cred}) & \text{if } b = 0 \\ \mathsf{SimProve}(params, \tau_s, pk_0, \mathsf{nym}', \ell) & \text{if } b = 1 \end{cases}$$

  – *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \mathsf{cred}')$.

  *Then for all PPT algorithms* $\mathcal{A}$*, there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

# 3 Defining Malleable Signatures

Formally, a malleable signature scheme consists of four algorithms: $\mathsf{KeyGen}$, $\mathsf{Sign}$, $\mathsf{Verify}$, and $\mathsf{SigEval}$. The first three comprise a standard signature; the additional algorithm, $\mathsf{SigEval}$, on input the verification key $vk$, messages $\vec{m} = (m_1, \ldots, m_n)$, signatures $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$, and a transformation $T$ on messages, outputs a signature $\sigma'$ on the message $T(\vec{m})$. (Here, and in all of our definitions, we consider the most general case wherein the transformation may combine many messages. Our construction in Section 5, however, supports only unary transformations; i.e., those operating on a single message.)

**Definition 3.1** (Malleability)**.** *A signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is* malleable *with respect to a set of transformations* $\mathcal{T}$ *if there exists an efficient algorithm* $\mathsf{SigEval}$ *that on input* $(vk, T, \vec{m}, \vec{\sigma})$*, where* $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$*,* $\mathsf{Verify}(vk, \sigma_i, m_i) = 1$ *for all* $i$*, and* $T \in \mathcal{T}$*, outputs a valid signature* $\sigma$ *for the message* $m := T(\vec{m})$*; i.e., a signature* $\sigma$ *such that* $\mathsf{Verify}(vk, \sigma, m) = 1$.

Here we immediately note one key notational difference with the previous definitions of Ahn et al. [3] and Attrapadung et al. [4]; whereas their definitions were given with respect to a predicate on input and output messages, we found it more natural to instead consider the set of allowed *transformations*. (This was especially natural given that our construction of a malleable signature uses a malleable proof, which is itself defined in the context of transformations.) By using transformations, we inherently capture the dual requirements that the result of an operation be efficiently computable, and that an adversary *know* the transformation that was applied; these requirements could also potentially be captured using predicates (e.g., by using an optional witness input to the predicate), but in a more roundabout way.

[1] By correctness, to test this latter property efficiently it suffices to form a proof of possession of $\mathsf{cred}$ and then see if it verifies, as only valid credentials verify.

### 3.1 Simulation-based definitions for malleable signatures

In order to achieve the stronger notions of unforgeability and context hiding needed to support credentials, we begin with the idea of a *simulatable* signature, introduced by Abe, Haralambiev, and Ohkubo [2, 1], in which there are two indistinguishable ways of producing a signature: using the signing key and the standard signing algorithm, or using a global trapdoor and a simulated signing algorithm. Our reasons for using simulatability are two-fold: first, it allows us to easily consider the notion of context hiding with respect to adversarially-chosen keys, as well as a simpler notion for signature unforgeability (much as Ahn et al. used context hiding to achieve a simpler version of their unforgeability definition). Second, simulatability lines up nicely with the anonymity requirements for credentials, in which there should exist a simulator that can simulate credentials.

Before we present our definition of simulatability — which is somewhat modified from the original; in particular they required only that simulated signatures verify, whereas we require them to be indistinguishable from standard signatures — we must expand the standard notion of a signature scheme to consider signature schemes in which the key generation process is split up into two parts: a trusted algorithm $\mathsf{Gen}$ for generating "universal" parameters $\mathsf{crs}$ (we can think of these as the setting; e.g., the description of a group), and an algorithm $\mathsf{KeyGen}$ that, given these parameters, generates a keypair $(vk, sk)$ specific to a given signer.

**Definition 3.2** (Simulatability)**.** *A signature scheme* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is* simulatable *if there exists an additional PPT algorithm* $\mathsf{KeyCheck}$ *that, on input* $\mathsf{crs}$*,* $vk$*, and* $sk$*, outputs whether or not* $(vk, sk)$ *is in the range of* $\mathsf{KeyGen}(\mathsf{crs})$*, and a PPT simulator* $(\mathsf{SimGen}, \mathsf{SimSign})$ *such that the CRS in* $(\mathsf{crs}, \tau_s) \xleftarrow{\$} \mathsf{SimGen}(1^k)$ *is indistinguishable from* $\mathsf{crs} \xleftarrow{\$} \mathsf{Gen}(1^k)$ *and signatures produced by* $\mathsf{SimSign}$ *are indistinguishable from honest signatures; i.e., for all PPT* $\mathcal{A}$*,*

$$Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{Gen}(1^k) : \mathcal{A}^{S(\mathsf{crs},\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1] \approx Pr[(\mathsf{crs}, \tau_s) \xleftarrow{\$} \mathsf{SimGen}(1^k) : \mathcal{A}^{S'(\mathsf{crs},\tau_s,\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1],$$

*where, on input* $(vk, sk, m)$*,* $S$ *outputs* $\perp$ *if* $\mathsf{KeyCheck}(\mathsf{crs}, vk, sk) = 0$ *and* $\mathsf{Sign}(\mathsf{crs}, sk, m)$ *otherwise, and* $S'$ *outputs* $\perp$ *if* $\mathsf{KeyCheck}(\mathsf{crs}, vk, sk) = 0$ *and* $\mathsf{SimSign}(\mathsf{crs}, \tau_s, vk, m)$ *otherwise.*

Although simulatability might seem to be a fairly strong property, we show in Appendix B that any non-simulatable signature scheme in the standard model can be easily transformed into a simulatable signature scheme in the CRS model (i.e., using split $\mathsf{Gen}$ and $\mathsf{KeyGen}$ algorithms) by adding a proof of knowledge to the public key. Our signature construction in Section 5 also achieves simulatability directly by using cm-NIZKs.

### 3.2 Simulation context hiding

With simulatability in hand, we next present a definition of context hiding that requires transformed signatures to be indistinguishable from freshly simulated signatures on the transformed messages; note that if regular signatures were used instead of simulated signatures, this would be quite similar to the standard notion of context hiding. As mentioned above, however, incorporating simulatability allows us to easily build in the notion of adversarially-generated keys, which will be especially useful for our credentials application.

**Definition 3.3** (Simulation context hiding)**.** *For a simulatable signature* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SigEval})$ *with an associated simulator* $(\mathsf{SimGen}, \mathsf{SimSign})$*, malleable with respect to a class of transformations* $\mathcal{T}$*, and an adversary* $\mathcal{A}$ *and a bit* $b$*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $(\mathsf{crs}, \tau_s) \xleftarrow{\$} \mathsf{SimGen}(1^k)$.

- *Step 2.* $(\mathsf{state}, vk, \vec{m}, \vec{\sigma}, T) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}, \tau_s)$.
- *Step 3. If* $\mathsf{Verify}(\mathsf{crs}, vk, \sigma_i, m_i) = 0$ *for some* $i$ *or* $T \notin \mathcal{T}$*, abort and output* $\perp$*. Otherwise, form* $\sigma \xleftarrow{\$} \mathsf{SimSign}(\mathsf{crs}, \tau_s, vk, T(\vec{m}))$ *if* $b = 0$*, and* $\sigma \xleftarrow{\$} \mathsf{SigEval}(\mathsf{crs}, vk, T, \vec{m}, \vec{\sigma})$ *if* $b = 1$.
- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \sigma)$.

*Then the signature scheme satisfies* simulation context hiding *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

Unsurprisingly, as we show in Appendix B, by incorporating the notion of adversarially-generated keys we provide a strictly stronger definition than any of the existing definitions for computational context hiding. While Ahn et al. and Attrapadung et al. both provide statistical variants on context hiding, we cannot hope to simultaneously achieve statistical hiding and a meaningful notion of extractability in our unforgeability definition.

### 3.3 Simulation unforgeability

The main point at which our unforgeability definition diverges significantly from previous definitions is in considering how to check whether a message and signature output by the adversary are in fact a forgery, or whether they were instead obtained using a valid transformation from the signatures issued by the signer. For many simple classes of transformations this may be easy to do given the set of signed messages, but for classes of transformations that are exponentially (or even infinitely) large, it is not clear that this can be done in an efficient manner. Whereas previous definitions for unforgeability were limited to these simple classes of transformations (as was explicitly acknowledged by Ahn et al., who point out that for many predicates it may be impossible to verify whether or not the adversary has won their unforgeability game), for our credentials application it is crucial that the winning conditions be efficiently testable, even in the face of a complex and infinitely large class of unary transformations; beyond efficient testability, credentials inherently require that signatures can't be transformed without knowledge of some appropriate secret information.

Translating this requirement back to malleable signatures, we therefore require that an adversary *knows* which valid transformation it is applying in order to generate a transformed signature; again, this is quite different from previous definitions, that require only that there *exists* some valid transformation corresponding to any signature produced by the adversary. We build in this requirement by using an extractor that — given the produced message and signature, as well as the set of signed messages — produces the transformation (if one exists) that was used to get from the signed message to the produced message; checking if the adversary won is then as simple as checking if this extracted transformation is in the allowed class. Note that giving the extractor the set of signed messages is a deviation from the standard simulation-sound extractability approach for proofs; this is because in a proof system, it is impossible to describe all the proved statements, as the adversary can form proofs himself. Signatures, however, should be created only with the signing key and thus this notion is meaningful; furthermore, it allows us to support larger classes of transformations, such as $n$-ary transformations, that would seem to be impossible without this extra information.

By using simulatability, we are able to replace all honestly generated and transformed signatures with signatures generated by a simulator; this means that we can simply give the adversary access to an oracle that generates simulated signatures, which has the advantage that we end up with a much cleaner definition than the main one of Ahn et al. (they also provide a similar simplification using the notion of statistical context hiding). The result is a definition similar to that of simulation-sound extractability [22, 28], in which we simulate and extract at the same time.

To formalize this game, in which we need to both simulate and extract, we require an amplified setup, $\mathsf{SimExtGen}$, that outputs a tuple $(\mathsf{crs}, \tau_s, \tau_e)$; we then require the $(\mathsf{crs}, \tau_s)$ part of this to be distributed

identically to the output of $\mathsf{SimGen}$, and we now give $\tau_e$ as input to $\mathsf{SigExt}$. To cover the case of simple — but potentially $n$-ary — transformations, where not all the information about a transformation can be encoded in the signature, the extractor is also given access to the query history $Q$.

**Definition 3.4** (Simulation unforgeability)**.** *For a simulatable signature* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SigEval})$ *with an associated PPT simulator/extractor* $(\mathsf{SimExtGen}, \mathsf{SimSign}, \mathsf{SigExt})$ *that is malleable with respect to a class of transformations* $\mathcal{T}$*, an adversary* $\mathcal{A}$*, and a table* $Q = Q_m \times Q_\sigma$ *that contains messages queried to* $\mathsf{SimSign}$ *and their responses, consider the following game:*

- *Step 1.* $(\mathsf{crs}, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimExtGen}(1^k)$*;* $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{crs})$*.*
- *Step 2.* $(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{SimSign}(\mathsf{crs}, \tau_s, vk, \cdot)}(\mathsf{crs}, vk, \tau_e)$*.*
- *Step 3.* $(\vec{m}', T) \leftarrow \mathsf{SigExt}(\mathsf{crs}, vk, \tau_e, m^*, \sigma^*, Q)$*.*

*Then the signature scheme satisfies* simulation unforgeability *if for all such PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that the probability (over the choices of* $\mathsf{KeyGen}$*,* $\mathsf{SimSign}$*, and* $\mathcal{A}$*) that* $\mathsf{Verify}(vk, \sigma^*, m^*) = 1$ *and* $(m^*, \sigma^*) \notin Q$ *but either (1)* $\vec{m}' \not\subseteq Q_m$*, (2)* $m^* \neq T(\vec{m}')$*, or (3)* $T \notin \mathcal{T}$ *is at most* $\nu(k)$*.*

Having now argued that our definition is significantly stronger than previous definitions, one might be concerned that our definition might be overly restrictive, in the sense that it might rule out previous constructions or many interesting classes of transformations. As we show in Appendix B, however, when considering many transformation classes, including essentially all those for which constructions are known, our definition of unforgeability is equivalent to that of Ahn et al. (with respect to simulatable signatures which, as mentioned above, can be easily and generically obtained from non-simulatable signatures). Thus, although our definition does automatically rule out certain classes of transformations (i.e., those where the transformation cannot be efficiently derived given the query list and some limited amount of extra information), to date this does not seem to be a significant limitation on the schemes we can construct.

# 4 Delegatable Anonymous Credentials from Malleable Signatures

Recall the desired functionality of a credential system: there are various users, each in possession of some secret key, who can use different pseudonyms to represent themselves to other participants; a *credential authority* (CA) publishes some public key $pk$. To form a pseudonym, a user Alice can compute a commitment to her secret key $sk_A$; again, each user might know her under a different pseudonym. The CA may issue a credential to Bob, known to it by the pseudonym $B_1$, by forming a signature on $B_1$ using $sk$; this signature attests to the fact that the CA issued a level-1 credential to the user who owns the pseudonym $B_1$. Bob might now wish to do one of two things with this credential: he can either prove possession of this credential to another user, who might know him under a different pseudonym $B_2$, or he can *delegate* this credential to another user, Carol, whom he knows under the pseudonym $C_1$. To delegate, Bob can give to Carol a level-2 credential belonging to her pseudonym $C_1$, that is "rooted" at the authority $pk$; note that Bob shold only be able to perform this operation if he is the rightful owner of the pseudonym $B_1$ to which the credential was issued. In order to fit this framework, credentials must therefore reveal three pieces of information: the root authority, denoted $pk_0$, the recipient of the credential, denoted $\mathsf{nym}_r$, and the level of the credential, denoted $\ell$.

## 4.1 Allowable transformations

First, let us describe all of the components in the above landscape in terms of commitments, messages, signatures, and transformations. We already saw that pseudonyms are represented as commitments;

this leaves delegation and issuing of credentials, which will be represented using transformations, and credentials and proofs of possession, which will be represented using signatures.

Intuitively, to create an initial credential, the root authority signs the message $m = (\mathsf{nym}, 1)$, using a signing key pair $(vk, sk)$, to obtain a signature $\sigma$; the credential is the tuple $(vk, \mathsf{nym}, 1, \sigma)$, and anyone can verify that this is a valid credential by running the signature verification algorithm $\mathsf{Verify}(vk, \sigma, (\mathsf{nym}, 1))$. To delegate, the user corresponding to $\mathsf{nym}$ can maul the signature $\sigma$, using $\mathsf{SigEval}$, to specify a new recipient $\mathsf{nym}'$ and increase the level.

This intuitive construction, however, does not contain enough information to allow users to prove possession of credentials. To do this, we add a bit of information to the message, meaning messages are now of the form $m = (\mathsf{nym}, \ell, \mathsf{flag})$, where either $\mathsf{flag} = \mathsf{credential}$ (to indicate that Alice is delegating a credential) or $\mathsf{flag} = \mathsf{proof}$ (to indicate that Alice is proving possession of a credential). Delegation can still proceed in the same way as before, but now Alice can additionally prove possession of the credential by switching the flag from $\mathsf{credential}$ to $\mathsf{proof}$, and changing the recipient to a fresh pseudonym $\mathsf{nym}'$ (that still belongs to Alice); this means that a proof object is really still a credential.

Formally, messages to be signed are of the form $m = (\mathsf{nym}, \ell, \mathsf{flag})$, where $\mathsf{nym}$ is a pseudonym, $\ell$ is a level, and $\mathsf{flag} = \mathsf{credential}/\mathsf{proof}$. In order to carry out a valid transformation and delegate a credential to another pseudonym, $\mathsf{nym}'$, one must know $(sk, \mathsf{open})$ that corresponds to $\mathsf{nym}$. Thus, the description of a valid transformation that takes as input the message $m = (\mathsf{nym}, \ell, \mathsf{credential})$ and outputs $T(m) = (\mathsf{nym}', \ell{+}1, \mathsf{credential})$ must include $(sk, \mathsf{open})$. In order to carry out the transformation that allows the owner of $\mathsf{nym}$ to prove possession of a credential in a context where she is known by another pseudonym, $\mathsf{nym}'$, one needs to know $(sk, \mathsf{open}, \mathsf{open}')$ such that $(sk, \mathsf{open})$ correspond to $\mathsf{nym}$, and $(sk, \mathsf{open}')$ correspond to $\mathsf{nym}'$. Thus, the description of a valid transformation that takes as input the message $m = (\mathsf{nym}, \ell, \mathsf{credential})$ and outputs $T(m) = (\mathsf{nym}', \ell, \mathsf{proof})$ must include $(sk, \mathsf{open}, \mathsf{open}')$. More generally, transformations may take a level-$\ell$ credential and output a level-$\ell + k$ credential, which means that its description $\langle T \rangle$ is of the form $\langle T \rangle = (\{\mathsf{nym}_j, sk_j, \mathsf{open}_j\}_{j=1}^{k}, \mathsf{nym}_{new}, \mathsf{flag}')$, where $\mathsf{flag}' = \mathsf{credential}$ means the transformation outputs a delegated credential and $\mathsf{flag}' = \mathsf{proof}$ means it outputs a proof, and

$$T(\mathsf{nym}, \ell, \mathsf{flag}) := \begin{cases} (\mathsf{nym}_{new}, \ell + k, \mathsf{credential}) & \text{if } \mathsf{nym}_1 = \mathsf{nym} \text{ and } \mathsf{flag} = \mathsf{flag}' = \mathsf{credential} \\ (\mathsf{nym}_{new}, \ell + (k-2), \mathsf{proof}) & \text{if } \mathsf{nym}_1 = \mathsf{nym},\ \mathsf{flag} = \mathsf{credential}, \text{ and } \mathsf{flag}' = \mathsf{proof} \\ \perp & \text{otherwise.} \end{cases} \tag{1}$$

Thus, the set of allowable transformations $\mathcal{T}_{\text{dac}}$ consists of transformations whose description is $\langle T \rangle = (\{\mathsf{nym}_j, sk_j, \mathsf{open}_j\}_{j=1}^{k}, \mathsf{nym}_{new}, \mathsf{flag}')$, whose input/output behavior is as in Equation 1, and such that

1. A user needs a pseudonym to which a credential was issued before the user can delegate: $k > 0$.
2. A user can only delegate a credential he owns; i.e., he must know the opening of the pseudonym to which it was issued: for commitment parameters $params'$, $\mathsf{nym}_j = \mathsf{Com}(params, sk_j; \mathsf{open}_j)$ for all $1 \leq j \leq k$.
3. If this is a proof of possession, meaning $\mathsf{flag}' = \mathsf{proof}$, then $k \geq 2$ and $\langle T \rangle$ must include the opening of $\mathsf{nym}_{new}$, so $\mathsf{nym}_k = \mathsf{nym}_{new}$. Additionally, the owner of $\mathsf{nym}_{new}$ must be the same as the owner of the pseudonym to which the credential was delegated, so $sk_k = sk_{k-1}$.

In terms of credential size, we can see that messages scale logarithmically with the number of levels (as they need to represent the integer $\ell$), while the size of the description of a transformation scales linearly. Since credentials should (as part of their functionality) explicitly reveal how many times they have been delegated, this dependence seems inevitable.

### 4.2 Our construction

Our construction follows the intuition developed above: to form a pseudonym, a user forms a commitment to a signing secret key; to issue a credential, the root authority signs the recipient's pseudonym and the intended level of the credential; and to delegate and prove possession of a credential, a user mauls the credential (i.e., signature) using one of the allowable transformations defined above. Formally, we use a simulatable signature $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SigEval})$, malleable with respect to $\mathcal{T}_{\text{dac}}$, and a commitment scheme $(\mathsf{ComSetup}, \mathsf{Com})$ as follows:

- $\mathsf{Setup}(1^k)$: Compute $\mathsf{crs} \xleftarrow{\$} \mathsf{Gen}(1^k)$ and $params' \xleftarrow{\$} \mathsf{ComSetup}(1^k)$. Output $params := (\mathsf{crs}, params')$.
- $\mathsf{KeyGen}(params)$: Output $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$.
- $\mathsf{NymGen}(params, sk)$: Pick a random opening $\mathsf{open}$, compute $\mathsf{nym} := \mathsf{Com}(params', sk; \mathsf{open})$, and output $(\mathsf{nym}, \mathsf{open})$.
- $\mathsf{NymVerify}(params, \mathsf{nym}, sk, \mathsf{open})$: Check that $\mathsf{nym} = \mathsf{Com}(params', sk; \mathsf{open})$; output 1 if this holds and 0 otherwise.
- $\mathsf{Issue}(params, sk_0, vk_0, \mathsf{nym}_r)$: Compute $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{crs}, sk_0, (\mathsf{nym}_r, 1, \mathsf{credential}))$ and output $\mathsf{cred} := (vk_0, 1, \mathsf{nym}_r, \sigma)$.
- $\mathsf{CredProve}(params, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}', \mathsf{open}', \mathsf{cred})$: Parse $\mathsf{cred} = (vk_0, \ell, \mathsf{nym}, \sigma)$ and abort if $\mathsf{Verify}(\mathsf{crs}, vk_0, \sigma, (nym, \ell, \mathsf{credential}))$. Otherwise, set $\langle T \rangle := (((\mathsf{nym}, sk, \mathsf{open}), (\mathsf{nym}', sk, \mathsf{open}')), \mathsf{nym}', \mathsf{proof})$, compute $\sigma' \xleftarrow{\$} \mathsf{SigEval}(\mathsf{crs}, vk_0, T, (\mathsf{nym}, \ell, \mathsf{credential}), \sigma)$, and output $\pi := (\mathsf{nym}', \sigma')$.
- $\mathsf{CredVerify}(params, vk_0, \mathsf{nym}, \ell, \pi)$: Parse $\pi = (\mathsf{nym}', \sigma')$; output 0 if $\mathsf{nym}' \neq \mathsf{nym}$. Otherwise, output $\mathsf{Verify}(\mathsf{crs}, vk_0, \sigma', (\mathsf{nym}, \ell, \mathsf{proof}))$.
- $\mathsf{Delegate}(params, sk_{old}, \mathsf{nym}_{old}, \mathsf{open}_{old}, \mathsf{nym}_{new}, \mathsf{cred})$: Parse $\mathsf{cred} = (vk_0, \ell, \mathsf{nym}_{old}, \sigma)$ and abort if $\mathsf{Verify}(\mathsf{crs}, vk_0, \sigma, (nym, \ell, \mathsf{credential}))$. Otherwise, set $\langle T \rangle := ((\mathsf{nym}_{old}, sk_{old}, \mathsf{open}_{old}), \mathsf{nym}_{new}, \mathsf{credential})$, compute $\sigma' \xleftarrow{\$} \mathsf{SigEval}(\mathsf{crs}, vk_0, T, (\mathsf{nym}_{old}, \ell, \mathsf{credential}), \sigma)$, and output $\mathsf{cred}' := (vk_0, \ell + 1, \mathsf{nym}_{new}, \sigma')$.

**Theorem 4.1.** *If the commitment scheme is computationally hiding and perfectly binding, and the signature is simulatable, simulation unforgeable, and simulation context hiding with respect to $\mathcal{T}_{dac}$, then the above construction describes a secure delegatable anonymous credentials scheme, as defined in Section 2.*

In the next section, we see how to instantiate the malleable signature, using cm-NIZKs, to achieve the required malleability and security properties. As we can instantiate both the cm-NIZK — and thus the malleable signature — and the commitment scheme using the Decision Linear assumption [11], the security of our entire credentials construction is based on this relatively mild assumption.

To prove Theorem 4.1, we break it up into three lemmas, one for each desired property: correctness, anonymity, and unforgeability.

**Lemma 4.2.** *If the commitment scheme is correct, and the signature scheme is correct and malleable, as defined in Definition 3.1, the above construction describes a correct DAC scheme.*

*Proof.* By the correctness of the commitment scheme, valid pseudonyms (i.e., pseudonyms produced by $\mathsf{NymGen}$) always verify. To see that $\mathsf{CredProve}$ produces a verifying proof only when given a valid credentials as input, we observe that verifying credentials must contain a valid signatures, as otherwise a proof under any pseudonym would fail. Thus by malleability the $\pi$ output by $\mathsf{CredProve}$ verifies for all pseudonyms and thus the credential is valid. To see that honestly issued credentials also verify, we observe that by signature correctness the credential output by $\mathsf{Issue}$ is a valid credential, and by malleability the $\pi$ output by $\mathsf{CredProve}$ verifies as well. Similarly, by malleability the credential output

by Delegate is a valid credential, and again by malleability the $\pi$ output by CredProve will verify as well. □

**Lemma 4.3.** *If* (Gen, KeyGen, Sign, Verify, SigEval) *is simulatable and simulation context hiding (as defined in Definitions 3.2 and 3.3 respectively), and the commitment scheme is hiding, then the above construction describes an anonymous DAC scheme, as defined in Definition 2.4.*

*Proof.* To prove this, we must prove that the four properties described in Definition 2.4 are satisfied. We take each of them in turn.

We first define the simulators SimSetup and SimCred and VerifyPK. The simulator SimSetup runs $(\mathsf{crs}, \tau_s) \xleftarrow{\$} \mathsf{SimGen}(1^k)$ and $params' \xleftarrow{\$} \mathsf{ComSetup}(1^k)$, and outputs $(params := (\mathsf{crs}, params'), \tau_s)$. The simulator SimCred on input $(params, \tau_s, vk_0, \mathsf{nym}_{new}, \ell)$ outputs $(vk_0, \ell + 1, \mathsf{nym}_{new}, \mathsf{SimSign}(\mathsf{crs}, \tau_s, vk_0, (\mathsf{nym}_{new}, \ell + 1, \mathsf{credential}))$. Finally, VerifyPK is defined in terms of KeyCheck.

**Pseudonym hiding.** This follows directly from the hiding property of the commitment.

**Issue simulatability.** We argue that signature simulatability implies that an adversary cannot distinguish between honest parameters and honestly issued credentials and parameters formed by SimSetup and credentials formed by SimCred.

To do this, we assume there exists an adversary $\mathcal{A}$ that can distinguish between these two cases with some non-negligible advantage $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that breaks simulatability with the same advantage. To start, $\mathcal{B}$ receives as input a CRS crs. It now forms $params' \xleftarrow{\$} \mathsf{ComSetup}(1^k)$ and gives $params := (\mathsf{crs}, params')$ to $\mathcal{A}$. When $\mathcal{A}$ queries for $(sk_0, pk_0, \mathsf{nym}_r)$, $\mathcal{B}$ queries $(pk_0, sk_0, m = (\mathsf{nym}_r, 1, \mathsf{credential}))$ to the signing oracle of the simulatability game to get back a signature $\sigma$; it then returns $(pk_0, 1, \mathsf{nym}_r, \sigma)$ to $\mathcal{A}$. At the end of the game, if $\mathcal{A}$ guesses $b' = 0$ then $\mathcal{B}$ guesses it is using an honest CRS and interacting with the signer, and if $\mathcal{A}$ guesses $b' = 1$ then $\mathcal{B}$ guesses it is using a simulated CRS and interacting with the simulator. As $\mathcal{B}$ gives to $\mathcal{A}$ $\mathsf{crs} \xleftarrow{\$} \mathsf{Gen}(1^k)$ and credentials of the form $(pk_0, 1, \mathsf{nym}_r, \sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{crs}, sk_0, (\mathsf{nym}_r, 1, \mathsf{credential}))) = \mathsf{Issue}(params, sk_0, pk_0, \mathsf{nym}_r)$ when it is interacting with the real signer using an honest CRS, and $\mathsf{crs} \xleftarrow{\$} \mathsf{SimGen}(1^k)$ and credentials of the form $(pk_0, 1, \mathsf{nym}_r, \sigma \xleftarrow{\$} \mathsf{SimSign}(\mathsf{crs}, \tau_s, pk_0, (\mathsf{nym}_r, 1, \mathsf{credential}))) = \mathsf{SimCred}(params, \tau_s, pk_0, \mathsf{nym}_r, 1)$ when it is interacting with SimSign using a simulated CRS, it therefore perfectly simulates the behavior that $\mathcal{A}$ expects. As $\mathcal{B}$ furthermore succeeds whenever $\mathcal{A}$ does, $\mathcal{B}$ succeeds with the same non-negligible advantage.

**Delegation simulatability.** We argue that simulatable context hiding implies that an adversary cannot distinguish between honestly delegated credentials and ones formed by SimCred.

To prove this we assume there exists an adversary $\mathcal{A}$ that can distinguish between credentials from Delegate and from SimCred with some non-negligible advantage $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that breaks simulation context hiding with the same advantage. To start, $\mathcal{B}$ receives as input a pair $(\mathsf{crs}, \tau_s)$; it then forms $params' \xleftarrow{\$} \mathsf{ComSetup}(1^k)$ and gives $params := (\mathsf{crs}, params')$ to $\mathcal{A}$. When $\mathcal{A}$ outputs $(sk_{old}, \mathsf{nym}_{old}, \mathsf{open}_{old}, \mathsf{nym}_{new}, pk_0, \ell, \mathsf{cred})$, $\mathcal{B}$ first checks that $\mathsf{NymVerify}(params, \mathsf{nym}_{old}, sk_{old}, \mathsf{open}_{old}) = 1$ and that cred is a valid level-$\ell$ credential rooted at $pk_0$; it aborts and outputs $\perp$ if any of these checks fail. Otherwise, $\mathcal{B}$ parses $\mathsf{cred} = (pk_0, \ell, \mathsf{nym}_{old}, \sigma)$ and sets $T := ((\mathsf{nym}_{old}, sk_{old}, \mathsf{open}_{old}), \mathsf{nym}_{new}, \mathsf{credential})$ and $m := (\mathsf{nym}_{old}, \ell, \mathsf{credential})$; it then outputs its own tuple $(pk_0, (m, \sigma), T)$ to get back a signature $\sigma'$, and returns $(pk_0, \ell + 1, \mathsf{nym}_{new}, \sigma')$ to $\mathcal{A}$. At the end, $\mathcal{B}$ outputs the opposite guess bit as $\mathcal{A}$. As $\mathcal{B}$ returns $(pk_0, \ell+1, \mathsf{nym}_{new}, \sigma' \xleftarrow{\$} \mathsf{SimSign}(\mathsf{crs}, \tau_s, pk_0, T(m))) = \mathsf{SimCred}(params, \tau_s, pk_0, \mathsf{nym}_{new}, \ell)$ in the case that $b = 0$, and $(pk_0, \ell + 1, \mathsf{nym}_{new}, \sigma' \xleftarrow{\$} \mathsf{SigEval}(\mathsf{crs}, pk_0, T, (m, \sigma))) =$

$\mathsf{Delegate}(params, sk_{old}, \mathsf{nym}_{old}, \mathsf{open}_{old}, \mathsf{nym}_{new}, \mathsf{cred})$ in the case that $b = 1$, it therefore perfectly simulates the behavior that $\mathcal{A}$ expects. As $\mathcal{B}$ furthermore guesses correctly whenever $\mathcal{A}$ does (as its $b = 0$ is $\mathcal{A}$'s $b = 1$ case, and vice versa), $\mathcal{B}$ succeeds with the same non-negligible advantage.

**Proof simulatability.** To prove this, we first define the simulator $\mathsf{SimProve}$: on input $(params, \tau_s, vk_0, \mathsf{nym}', \ell)$, it outputs $\mathsf{SimSign}(\mathsf{crs}, \tau_s, vk_0, (\mathsf{nym}', \ell, \mathsf{proof}))$. Now, we assume there exists an adversary $\mathcal{A}$ that can distinguish between honest proofs formed by $\mathsf{CredProve}$ and those formed by $\mathsf{SimProve}$ with some non-negligible advantage $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that breaks simulation context hiding with the same advantage. To start, $\mathcal{B}$ receives as input a pair $(\mathsf{crs}, \tau_s)$. It then generates $params' \xleftarrow{\$} \mathsf{ComSetup}(1^k)$ and gives $params := (\mathsf{crs}, params')$ to $\mathcal{A}$. When $\mathcal{A}$ outputs $(pk_0, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}', \mathsf{open}', \ell, \mathsf{cred})$, $\mathcal{B}$ first checks that $\mathsf{NymVerify}(params, \mathsf{nym}, sk, \mathsf{open}) = 1$, that $\mathsf{NymVerify}(params, \mathsf{nym}', sk, \mathsf{open}')$, and that $\mathsf{cred}$ is a valid level-$\ell$ credential rooted at $pk_0$; if any of these checks fails, it aborts and outputs $\perp$. Otherwise, it parses $\mathsf{cred} = (vk_0, \ell, \mathsf{nym}, \sigma)$ and sets $T := (((\mathsf{nym}, sk, \mathsf{open}), (\mathsf{nym}', sk, \mathsf{open}')), \mathsf{nym}', \mathsf{proof})$ and $m := (\mathsf{nym}, \ell, \mathsf{credential})$, and outputs $(vk_0, (m, \sigma), T)$ as its own tuple to receive back a signature $\sigma'$. It then gives $(\mathsf{nym}', \sigma')$ back to $\mathcal{A}$, and at the end outputs the opposite guess bit as $\mathcal{A}$.

To argue that interactions with $\mathcal{B}$ are indistinguishable from those that $\mathcal{A}$ expects, we observe that if $b = 0$, $\mathcal{B}$ gives $(\mathsf{nym}', \sigma' \xleftarrow{\$} \mathsf{SigEval}(\mathsf{crs}, pk_0, T, (m, \sigma))) = \mathsf{CredProve}(params, sk, \mathsf{nym}, \mathsf{open}, \mathsf{nym}', \mathsf{open}', \mathsf{cred})$, which is exactly what $\mathcal{A}$ expects in the case that its own $b' = 1$. Similarly, if $b = 1$, then $\mathcal{B}$ gives $(\mathsf{nym}', \sigma' \xleftarrow{\$} \mathsf{SimSign}(\mathsf{crs}, \tau_s, pk_0, T(m) = (\mathsf{nym}', \ell, \mathsf{proof})))$, which is exactly what $\mathcal{A}$ expects in the case that its own $b' = 0$. $\mathcal{B}$ therefore perfectly simulates the bheavior that $\mathcal{A}$ expects; as $\mathcal{B}$ furthermore guesses correctly whenever $\mathcal{A}$ does, $\mathcal{B}$ succeeds with the same non-negligible advantage. □

**Lemma 4.4.** *If* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SigEval})$ *is simulation unforgeable, as defined in Definition 3.4, then the above construction describes an unforgeable DAC scheme, as defined in Definition 2.1.*

*Proof.* To prove this, we first describe the algorithms $\mathsf{SimExtSetup}$ and $\mathsf{Extract}$: $\mathsf{SimExtSetup}$, on input $1^k$, runs $(\mathsf{crs}, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimExtGen}(1^k)$ and computes $params' \xleftarrow{\$} \mathsf{ComSetup}(1^k)$; it then outputs $params := (\mathsf{crs}, params')$, $\tau_s$, and $\tau_e$. The algorithm $\mathsf{Extract}$, on input $(params, \tau_e, (vk_0, \mathsf{nym}, \ell), \pi = (\mathsf{nym}', \sigma'))$, computes $(m' := (\mathsf{nym}'', \ell', \mathsf{flag}'), T := (S, \mathsf{nym}_{new}, \mathsf{flag})) := \mathsf{SigExt}(\mathsf{crs}, \tau_e, vk_0, (\mathsf{nym}', \ell, \mathsf{proof}), \sigma', Q)$[2] and outputs $(S, \ell')$.

To argue that commitments are still hiding and binding even when given the trapdoors $\tau_s$ and $\tau_e$, we observe that the commitment parameters are generated completely independently from the parameters generated by $\mathsf{SimExtGen}$, meaning that $\tau_e$ and $\tau_s$ provide no meaningful information to help break either hiding or binding.

Now, we first assume there exists an adversary $\mathcal{A}$ that can distinguish between honest and simulated parameters with some non-negligible advantage $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that distinguishes between honest and simulated signature parameters with the same advantage. The reduction is simple: $\mathcal{B}$, on input $(\mathsf{crs}, \tau_s)$, generates $params' \xleftarrow{\$} \mathsf{ComSetup}(1^k)$ and gives $params := (\mathsf{crs}, params')$ to $\mathcal{A}$; $\mathcal{B}$ then outputs the same guess bit as $\mathcal{A}$. As $\mathcal{B}$ simulates exactly the behavior that $\mathcal{A}$ expects, and furthermore $\mathcal{B}$ guesses correctly whenever $\mathcal{A}$ does, $\mathcal{B}$ succeeds with the same advantage.

Next, we assume there exists an adversary $\mathcal{A}$ that can break credential unforgeability with some non-negligible advantage $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that breaks simulation unforgeability for the signature with the same advantage. To start, $\mathcal{B}$ receives as input $(\mathsf{crs}, vk_0)$. It then generates $params' \xleftarrow{\$} \mathsf{ComSetup}(1^k)$ and gives $params := (\mathsf{crs}, params')$ and $vk_0$ to $\mathcal{A}$. Now, when $\mathcal{A}$ queries its $\mathsf{SimCred}$ oracle on input $\mathsf{nym}$ and $\ell$, $\mathcal{B}$ queries its $\mathsf{SimSign}$ oracle on input $m = (\mathsf{nym}, \ell, \mathsf{credential})$ to

[2]Recall that, for both signatures and credentials, the extractor is given access to the queries $Q$ made to the simulator.

receive back a signature $\sigma$; it then returns $(vk_0, \ell, \mathsf{nym}, \sigma)$ to $\mathcal{A}$. Similarly, when $\mathcal{A}$ queries its $\mathsf{SimProve}$ oracle on input $\mathsf{nym}'$ and $\ell$, $\mathcal{B}$ queries its $\mathsf{SimSign}$ oracle on input $m = (\mathsf{nym}', \ell, \mathsf{proof})$. At the end, when $\mathcal{A}$ outputs $((\mathsf{nym}, \ell), \pi = (\mathsf{nym}', \sigma'))$, $\mathcal{B}$ outputs $(m^* = (\mathsf{nym}', \ell, \mathsf{proof}), \sigma')$.

To argue that interactions with $\mathcal{B}$ are indistinguishable from those that $\mathcal{A}$ expects, we observe that $\mathcal{B}$ mimics exactly the behavior of $\mathsf{SimExtSetup}$, $\mathsf{SimCred}$, and $\mathsf{SimProve}$. To further see that $\mathcal{B}$ wins at its game whenever $\mathcal{A}$ does, we consider $(m' = (\mathsf{nym}'', \ell'', \mathsf{flag}''), \langle T \rangle = (\{(\mathsf{nym}'_j, sk'_j, \mathsf{open}'_j)\}_{i=1}^{k}, \mathsf{nym}_{new}, \mathsf{flag}') := \mathsf{SigExt}(\mathsf{crs}, sk, vk, m^*, \sigma')$ and $(\{(\mathsf{nym}_j, sk_j, \mathsf{open}_j)\}_{j=1}^{k}, \ell') := \mathsf{Extract}(params, \tau_e, (vk_0, \mathsf{nym}, \ell), \pi)$. By the definition of $\mathsf{Extract}$, we have $\mathsf{nym}'_j = \mathsf{nym}_j$, $sk'_j = sk_j$, and $\mathsf{open}'_j = \mathsf{open}_j$ for all $j$, $1 \leq j \leq k$; we similarly have that $\ell'' = \ell'$. We now examine the three winning conditions for $\mathcal{A}$:

- If $(\mathsf{nym}_1, \ell_1)$ for some $\ell_1 \leq \ell$ was not queried to $\mathsf{SimCred}$, then there are two possibilities: either $m'$ was not queried to $\mathsf{SimSign}$, or $T(m') \neq m^*$. To see this, we observe that, in the case that $\mathsf{nym}_1 = \mathsf{nym}''$ and $\mathsf{flag}'' = \mathsf{credential}$, if $(\mathsf{nym}_1, \ell_1)$ was not queried to the $\mathsf{SimCred}$ oracle then, by how $\mathcal{B}$'s behavior is defined, $(\mathsf{nym}_1, \ell', \mathsf{credential})$ was not queried to $\mathsf{SimSign}$; as in this case $m' = (\mathsf{nym}_1, \ell', \mathsf{credential})$, we therefore have that $m'$ was not queried to $\mathsf{SimSign}$.
  In the case that either $\mathsf{nym}_1 \neq \mathsf{nym}''$ or $\mathsf{flag}'' = \mathsf{proof}$, by the definition of $T$ we would have that $T(m') = \perp$. As $m^* \neq \perp$, this implies that $T(m') \neq m^*$.
  As both of these cases result in $\mathcal{B}$ winning the simulation unforgeability game, we therefore have that $\mathcal{B}$ wins at its game whenever $\mathcal{A}$ wins by creating a new credential (i.e., whenever $\mathcal{A}$ falls into this first winning condition).
- If $\mathsf{NymVerify}(params, \mathsf{nym}_j, sk_j, \mathsf{open}_j) = 0$ for some $j$, then, by definition, $T \notin \mathcal{T}_{\text{dac}}$, which means $\mathcal{B}$ wins as well.
- If $sk_{k-1} \neq sk_k$ then, again by definition, $T \notin \mathcal{T}_{\text{dac}}$ and $\mathcal{B}$ wins as well.

As any winning condition for $\mathcal{A}$ therefore results in a winning condition for $\mathcal{B}$, we can conclude that $\mathcal{B}$ will succeed at the simulation unforgeability game with the same probability that $\mathcal{A}$ succeeds at the credential unforgeability game. □

# 5 Malleable Signatures from cm-NIZKs

Now that we have seen how to use malleable signatures to construct delegatable anonymous credentials, we must also consider how to construct malleable signatures. In this section, we show how to use cm-NIZKs (as defined in Section 2) to construct a malleable signature meeting the requirements for the DAC construction: namely, simulatability, simulation unforgeability, simulation context hiding, and malleability with respect to $\mathcal{T}_{\text{dac}}$. After providing a generic construction of malleable signatures from cm-NIZKs, we then discuss how to instantiate the cm-NIZK (and thus the malleable signature) concretely.

## 5.1 Our construction

Intuitively, our construction is extremely simple: to sign a message, just prove knowledge of the signing key! While this might seem to produce signatures that are independent of the message being signed, we show that by including the message in the instance, we can bind the message and signature together (as was also done, e.g., by Chase and Lysyanskaya [20]); furthermore, defining transformations on signatures is quite straightforward as well, since signatures in our construction are just malleable proofs.

Formally, we use a hard relation $\mathcal{R}_{pk}$ with generator $\mathcal{G}$ and a cm-NIZK $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$, malleable with respect to some class of transformations $\mathcal{T}_{\text{nizk}}$, for the relation $R$ such that $((pk, m), sk) \in R$ if and only if $(pk, sk) \in \mathcal{R}_{pk}$. We then construct a simulatable signature, malleable with respect to a class of transformations $\mathcal{T}_{\text{sig}}$, as follows:

- $\mathsf{Gen}(1^k)$: Output $\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$.
- $\mathsf{KeyGen}(\mathsf{crs})$: Compute $(pk', sk') \xleftarrow{\$} \mathcal{G}(1^k)$; output $vk := pk'$ and $sk := (vk, sk')$.
- $\mathsf{Sign}(\mathsf{crs}, sk, m)$: Parse $sk = (pk', sk')$ and output $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, (pk', m), sk')$.
- $\mathsf{Verify}(\mathsf{crs}, vk, \sigma, m)$: Output $\mathcal{V}(\mathsf{crs}, (vk, m), \sigma)$.
- $\mathsf{SigEval}(\mathsf{crs}, vk, T, m, \sigma)$: Set $T_{\text{inst}}$ to be such that $T_{\text{inst}}(vk, m) = (vk, T(m))$ and $T_{\text{wit}} = \text{id}$; i.e., such that $T_{\text{wit}}(sk') = sk'$. Return $\sigma' \xleftarrow{\$} \mathsf{ZKEval}(\mathsf{crs}, (T_{\text{inst}}, T_{\text{wit}}), (vk, m), \sigma)$.

Looking at the definition of SigEval, we can see that the malleability of our signature construction is very closely linked to the malleability of the proof. If we therefore wished to construct a signature malleable with respect to a specific class of transformations $\mathcal{T}_{\text{sig}}$, we would require a proof malleable with respect to the class $\mathcal{T}_{\text{nizk}}$ that contained transformations of the form $(T_{\text{inst}} = (\text{id}, T), T_{\text{wit}} = \text{id})$ for all $T \in \mathcal{T}_{\text{sig}}$.

**Theorem 5.1.** *If* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *is zero knowledge, then* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SigEval})$ *is simulatable, as defined in Definition 3.2.*

*Proof.* To prove this, we first define KeyCheck, as well as the simulator (SimGen, SimSign). For KeyCheck, we can define $\mathsf{KeyCheck}(\mathsf{crs}, vk, sk)$ to be 1 if $(vk, sk) \in R_{pk}$ and 0 otherwise. For the simulator, we use the zero-knowledge simulator $(S_1, S_2)$: SimGen outputs $(\mathsf{crs}, \tau_s) \xleftarrow{\$} S_1(1^k)$, and SimSign, on input $(\mathsf{crs}, vk, \tau_s, m)$, outputs $S_2(\mathsf{crs}, \tau_s, (vk, m))$. Now, we assume there exists an adversary $\mathcal{A}$ that can distinguish between interactions with SimSign and the honest signer with some non-negligible advantage $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that can distinguish between interactions with the prover and simulator with the same advantage.

To start, $\mathcal{B}$ receives as input a CRS crs, which it forwards to $\mathcal{A}$. When $\mathcal{A}$ queries its oracle on inputs $pk'$, $sk'$, and $m$, $\mathcal{B}$ first checks that $(pk', sk') \in R_{pk}$ and outputs $\perp$ if this does not hold. Otherwise, it sends the query $(x := (pk', m), w = sk')$ to its own oracle to get back a proof $\pi$, which it then forwards (as $\sigma$) back to $\mathcal{A}$. If $\mathcal{A}$ guesses it is interacting with the signer on an honest CRS then $\mathcal{B}$ guesses it is interacting with the prover on an honest CRS, and if $\mathcal{B}$ guesses it is interacting with the simulated signer on a simulated CRS then $\mathcal{B}$ guesses it is interacting with the simulator on a simulated CRS.

To see that interactions with $\mathcal{B}$ are distributed identically to those that $\mathcal{A}$ expects, we observe that, in the case that the CRS is honest and $\mathcal{B}$ is interacting with the prover, $\mathcal{B}$ provides $\mathcal{A}$ with an honest CRS and signatures of the form $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, (pk', m), sk') = \mathsf{Sign}(\mathsf{crs}, sk, m)$, which is exactly what $\mathcal{A}$ expects in this case. Similarly, if the CRS is simulated and $\mathcal{B}$ is interacting with the simulator, then $\mathcal{B}$ provides $\mathcal{A}$ with a simulated CRS and signatures of the form $\pi \xleftarrow{\$} S_2(\mathsf{crs}, \tau_s, (pk', m)) = \mathsf{SimSign}(\mathsf{crs}, \tau_s, pk', m)$, which is again exactly what $\mathcal{A}$ expects. $\mathcal{A}$ will therefore succeed when interacting with $\mathcal{B}$ with the same advantage $\epsilon$; as $\mathcal{B}$ furthermore guesses correctly whenever $\mathcal{A}$ does, $\mathcal{B}$ will succeed with the same advantage. □

**Theorem 5.2.** *If* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *is CM-SSE with respect to the class of transformations* $\mathcal{T}_{nizk}$ *and* $\mathcal{R}_{pk}$ *is a hard relation, then* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SigEval})$ *is simulation unforgeable with respect to transformation class* $\mathcal{T}_{sig}$*, as defined in Definition 3.4.*

*Proof.* To prove this, we first define the algorithms SimExtGen and SigExt, using the algorithms $SE_1$ and $E_2$ that, by CM-SSE, we know exist for the proof. SimExtGen simply outputs $(\mathsf{crs}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)$; because the $(\mathsf{crs}, \tau_s)$ output by $SE_1$ are, by definition, distributed identically to those output by $S_1$, and furthermore SimGen runs $S_1$, SimExtGen satisfies the constraint that its own $(\mathsf{crs}, \tau_s)$ must be distributed identically to that of SimGen. Now, SigExt, given $(vk, \tau_e, m, \sigma)$, runs $E_2(\mathsf{crs}, \tau_e, (vk, m), \sigma)$ to get back a

tuple $(w, x', (T_{\text{inst}}, T_{\text{wit}}))$; it then parses $x' = (pk', m')$ and $T_{\text{inst}} = T$ and outputs $(m', T)$. We also use the same simulator $\mathsf{SimSign}$ as in the proof of Theorem 5.1.

We now observe that, if there exists an adversary $\mathcal{A}$ that can break unforgeability with some non-negligible probability $\epsilon$, it must be the case that one of two events occurs with some non-negligible probability: in the first, $\mathsf{Event}_1$, for $(w, x', T) \leftarrow E_2(\mathsf{crs}, (pk', m^*), \sigma^*)$, $w \neq \perp$ and $(x, w) \in R$. In the second, $\mathsf{Event}_2$, we consider all other winning cases in the CM-SSE game; i.e., $(x, w) \notin R$, $x'$ was not queried to the $S_2$ oracle, $x \neq T_{\text{inst}}(x')$, or $T \notin \mathcal{T}$. We define $e_1$ to be the probability that $\mathsf{Event}_1$ occurs, and $e_2$ to be the probability that $\mathsf{Event}_2$ occurs. To see that, in the event that $\mathcal{A}$ wins the game, either $e_1$ or $e_2$ must be non-negligible (i.e., either $\mathsf{Event}_1$ or $\mathsf{Event}_2$ must have taken place), we observe that for $\mathcal{A}$ to have won, it must be the case that, for $(m', T) := \mathsf{SigExt}(vk, sk, m, \sigma)$, either $m'$ wasn't queried to $\mathsf{SimSign}$, $T \in \mathcal{T}$ and $m^* \neq T(m')$, or $T \notin \mathcal{T}$. Based on the definition of $\mathsf{SigExt}$, this first case happens in the event that $x' = (pk', m')$ wasn't queried to $S_2$, which is part of $\mathsf{Event}_2$. The second case happens in the event that $(x', T) \neq (\perp, \perp)$ and $x \neq T(x')$, which is also part of $\mathsf{Event}_2$. The third case, on the other hand, might happen in the event that $(x', T) \neq (\perp, \perp)$ and $T \notin \mathcal{T}$, or in the event that $(w, x', T) = (\perp, \perp, \perp)$; these are once again part of $\mathsf{Event}_2$. It also might happen in the event that $w \neq \perp$; in this case, if $(x, w) \in R$ then we are in $\mathsf{Event}_1$, and if $(x, w) \notin R$ then we are in $\mathsf{Event}_2$. As each winning case for $\mathcal{A}$ therefore implies that either $\mathsf{Event}_1$ or $\mathsf{Event}_2$ has taken place, it must be the case that either $e_1$ or $e_2$ is non-negligible.

To first use $\mathsf{Event}_1$ to break the hardness of the relation, $\mathcal{B}$ receives as input a public key $pk'$ for $R_{pk}$. It then generates $(\mathsf{crs}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)$ and gives $\mathsf{crs}$ and $pk'$ to $\mathcal{A}$. When $\mathcal{A}$ queries its $\mathsf{SimSign}$ oracle, $\mathcal{B}$ uses its knowledge of $\tau_s$ to execute the code honestly; similarly, when $\mathcal{A}$ queries its $\mathsf{SigExt}$ oracle, $\mathcal{B}$ uses its knowledge of $\tau_e$ to execute the code honestly. Finally, when $\mathcal{A}$ outputs its pair $(m^*, \sigma^*)$, $\mathcal{B}$ computes $(w, x', T) := E_2(\mathsf{crs}, \tau_e, (pk', m^*), \sigma^*)$. If $\mathsf{Event}_1$ has occurred, then by definition $w \neq \perp$ and $((pk', m^*), w) \in R$; by definition of the relation $R$, this means that $(pk', w) \in R_{pk}$, and thus $\mathcal{B}$ can output $w$ to win its game. As $\mathcal{B}$ behaves honestly and interactions with $\mathcal{B}$ are thus distributed identically to those that $\mathcal{A}$ expects, and further $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does and $\mathsf{Event}_1$ occurs, $\mathcal{B}$ succeeds with overall probability $e_1\epsilon$.

To use $\mathsf{Event}_2$ to break CM-SSE, $\mathcal{C}$ receives as input the pair $(\mathsf{crs}, \tau_e)$; it then generates $(pk', sk') \xleftarrow{\$} \mathcal{G}(1^k)$ and gives $\mathsf{crs}$ and $pk'$ to $\mathcal{A}$. When $\mathcal{A}$ queries its $\mathsf{SimSign}$ oracle on input $m$, $\mathcal{C}$ queries its own $S_2$ oracle on input $(pk', m)$ and returns the resulting proof back to $\mathcal{A}$. When $\mathcal{A}$ queries its $\mathsf{SigExt}$ oracle, $\mathcal{C}$ uses its knowledge of $\tau_e$ to execute the code honestly. Now, when $\mathcal{A}$ outputs its pair $(m^*, \sigma^*)$, $\mathcal{C}$ computes $(w, x', T) := E_2(\mathsf{crs}, \tau_e, (pk', m^*), \sigma^*)$. If $\mathsf{Event}_2$ has occurred, then by definition one of the winning cases for the CM-SSE game holds, and thus $\mathcal{C}$ can output $(x := (pk', m^*), \sigma^*)$ to win its game. As $\mathcal{C}$ generates $vk$ honestly and returns values of the form $S_2(\mathsf{crs}, \tau_s, (pk', m)) = \mathsf{SimSign}(vk = (\mathsf{crs}, pk'), \tau_s, m)$, interactions with it are distributed identically to those that $\mathcal{A}$ expects; furthermore, $\mathcal{C}$ succeeds whenever $\mathcal{A}$ does and $\mathsf{Event}_2$ occurs, so $\mathcal{C}$ succeeds with overall probability $e_2\epsilon$. As $\epsilon$ is assumed to be non-negligible, and by our discussion above so is either $e_1$ or $e_2$, the success probability of either $\mathcal{B}$ or $\mathcal{C}$ is therefore non-negligible as well. □

**Theorem 5.3.** *If* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *is strongly derivation private with respect to* $\mathcal{T}_{nizk}$*, as defined in Definition A.3, then* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SigEval})$ *satisfies simulation context hiding with respected to* $\mathcal{T}_{sig}$*, as defined in Definition 3.3.*

*Proof.* To show this, we take an adversary $\mathcal{A}$ that can break simulation context hiding with some non-negligible advantage $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that breaks strong derivation privacy with the same advantage.

To start, $\mathcal{B}$ will get as input a pair $(\mathsf{crs}, \tau_s)$, which it then immediately forwards to $\mathcal{A}$. When $\mathcal{A}$ outputs its challenge $(pk', sk', m, \sigma, T)$, $\mathcal{B}$ first checks that $\mathsf{Verify}((\mathsf{crs}, pk'), \sigma, m) = 1$, $\mathsf{KeyCheck}(\mathsf{crs}, pk', sk') = 1$ and $T \in \mathcal{T}$; it aborts and outputs $\perp$ if any of these checks fails. Otherwise, it sets $T_{\text{inst}} := T$ and

$T_{\text{wit}} := \text{id}$ and outputs $(x := (pk', m), \sigma, (T_{\text{inst}}, T_{\text{wit}}))$ as its own challenge to get back a proof $\pi'$, which it again forwards directly (as $\sigma'$) to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs its guess bit $b'$, $\mathcal{B}$ outputs the same bit.

To see that interactions with $\mathcal{B}$ are distributed identically to those that $\mathcal{A}$ expects, we observe that the pair $(\mathsf{crs}, \tau_s)$ given to $\mathcal{A}$ is honestly computed. In addition, if $\mathsf{Verify}((\mathsf{crs}, pk'), \sigma, m) = 1$ then, by definition, $\mathcal{V}(\mathsf{crs}, (pk', m), \sigma) = 1$; furthermore, if $T \in \mathcal{T}$ then, again by definition, $(T_{\text{inst}}, T_{\text{wit}}) \in \mathcal{T}_{\text{nizk}}$, meaning that if $\mathcal{B}$ interprets $\mathcal{A}$'s challenge tuple as valid, its own challenge tuple will be interpreted as valid as well. As for the response, if $b = 0$ then $\mathcal{B}$ gets back $S_2(\mathsf{crs}, \tau_s, (pk', m)) = \mathsf{SimSign}(vk = (\mathsf{crs}, pk'), \tau_s, m)$, which is exactly what $\mathcal{A}$ expects. Furthermore, if $b = 1$, then $\mathcal{B}$ gets back $\mathsf{ZKEval}(\mathsf{crs}, (T_{\text{inst}}, T_{\text{wit}}), (pk', m), \sigma) = \mathsf{SigEval}((\mathsf{crs}, pk'), T, m, \sigma)$, which is again exactly what $\mathcal{A}$ was expecting. As $\mathcal{A}$ therefore has the same advantage interacting with $\mathcal{B}$ as it does normally, and furthermore $\mathcal{B}$ succeeds in guessing $b$ whenever $\mathcal{A}$ does, $\mathcal{B}$ succeeds with non-negligible advantage $\epsilon$. □

### 5.2 Instantiating our construction

In order to instantiate this malleable signature, we must consider concrete choices for the cm-NIZK upon which it is based. Although Chase et al. already considered this question for certain classes of transformations, we focus in particular on the specific class of credential transformations $\mathcal{T}_{\text{dac}}$ defined in Section 4.1.

Two constructions of cm-NIZKs exist within the literature, both due to Chase et al.: their original construction [17], based on Groth-Sahai proofs [30], and a more recent construction [18], based on succinct non-interactive arguments of knowledge (SNARGs) and fully homomorphic encryption. While the latter construction is less efficient, showing that it supports the class of transformations $\mathcal{T}_{\text{dac}}$ is relatively straightforward: all we need to show is that the language and class of transformations are what is called *t-tiered*; this means that (1) every instance $x$ in the language can be labeled with an integer $i = \mathsf{tier}(x)$, and (2) every transformation $T \in \mathcal{T}_{\text{dac}}$ is such that $\mathsf{tier}(T(x)) > \mathsf{tier}(x)$ for all $x \in L$ such that $\mathsf{tier}(x) < t$, and $T(x) = \perp$ if $\mathsf{tier}(x) = t$.

To see that our language is $t$-tiered, we recall that the relation $R$ in Section 5.1 was defined as $(x = (pk, m), w = sk) \in R \Leftrightarrow (pk, sk) \in \mathcal{R}_{pk}$ for a hard relation $\mathcal{R}_{pk}$. For the credentials, we have $m = (\mathsf{nym}_r, \ell, b)$; we can therefore define $\mathsf{tier}(x) := \ell + b$. To see that $\mathcal{T}_{\text{dac}}$ is also $t$-tiered, we observe that $\langle T \rangle = (\{\mathsf{nym}_i, sk_i, \mathsf{open}_i\}_{i=1}^{k}, \mathsf{nym}_{new}, b')$, where it is required that $k > 0$. Looking at the two cases for transformations in Equation 1, for the first we see that $T(\mathsf{nym}_r, \ell, b) = (\mathsf{nym}_{new}, \ell + k, 0)$ if $b = b' = 0$; then $\mathsf{tier}(x) = \ell$, $\mathsf{tier}(T(x)) = \ell + k$ for $k > 0$, and thus $\mathsf{tier}(T(x)) > \mathsf{tier}(x)$ as desired. In the second case, $T(\mathsf{nym}_r, \ell, b) = (\mathsf{nym}_{new}, \ell + k - 2, 1)$ if $b = 0$ and $b' = 1$; here it is additionally required that $k \geq 2$, so $\mathsf{tier}(T(x)) = \ell + k - 2 + 1 > \ell = \mathsf{tier}(x)$ and the condition is still satisfied. To finally satisfy the requirement that $T(x) = \perp$ if $\mathsf{tier}(x) = t$, we could require that credentials can be delegated at most $t - 1$ times (the last tier $t$ would then be reserved for proving possession).

While the result of Chase et al. [18] therefore assures us that we can construct a cm-NIZK supporting the desired class of transformations, we might also wish to instantiate our cm-NIZK using their first, more efficient, construction. In order to do this, we first observe that the identity transformation must always be allowable when using Groth-Sahai proofs, as they are inherently re-randomizable. After therefore extending $\mathcal{T}_{\text{dac}}$ to include the identity, we show in Appendix C that our relation and class of transformations are what is called *CM-friendly*; this essentially means that all of the objects (instances, witnesses, and transformations) can be represented as elements of a bilinear group and hence the system is compatible with Groth-Sahai proofs. Although we end up using a slightly modified notion of CM-friendliness (that allows for transformations to grow as credentials are delegated), we can show that our notion of CM-friendliness still implies cm-NIZKs. We can therefore instantiate our malleable signature with respect to $\mathcal{T}_{\text{dac}}$ using this more efficient construction.

## Acknowledgments

Anna Lysyanskaya was supported by NSF grants 1012060, 0964379, 0831293, and by a Sloan Foundation fellowship, and Sarah Meiklejohn was supported in part by a MURI grant administered by the Air Force Office of Scientific Research and in part by a graduate fellowship from the Charles Lee Powell Foundation.

## References

[1] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *Proceedings of Crypto 2010*, volume 6223 of *LNCS*, pages 209–236, 2010.

[2] M. Abe, K. Haralambiev, and M. Ohkubo. Signing on elements in bilinear groups for modular protocol design. Cryptology ePrint Archive, Report 2010/133, 2010. `http://eprint.iacr.org/2010/133`.

[3] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, abhi shelat, and B. Waters. Computing on authenticated data. In *Proceedings of TCC 2012*, volume 7194 of *LNCS*, pages 1–20. Springer-Verlag, 2012.

[4] N. Attrapadung, B. Libert, and T. Peters. Computing on Authenticated Data: New Privacy Definitions and Constructions. In *Asiacrypt 2012*, volume 7658 of *Lecture Notes on Computer Science*. Springer, 12 2012.

[5] N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *Proceedings of PKC 2013*, pages 386–404, 2013.

[6] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Delegatable anonymous credentials. In *Proceedings of Crypto 2009*, volume 5677 of *LNCS*, pages 108–125. Springer-Verlag, 2009.

[7] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: the case of hashing and signing. In *Proceedings of Crypto 1994*, pages 216–233, 1994.

[8] M. Bellare and S. Goldwasser. New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs. In *Proceedings of Crypto 1989*, pages 194–211, 1989.

[9] M. Bellare and G. Neven. Transitive signatures: new schemes and proofs. *IEEE Transactions on Information Theory*, 51(6):2133–2151, 2005.

[10] M. Blum, A. de Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[11] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Proceedings of Crypto 2004*, volume 3152 of *LNCS*, pages 41–55. Springer-Verlag, 2004.

[12] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *Proceedings of Eurocrypt 2011*, volume 6632 of *LNCS*, pages 149–168. Springer-Verlag, 2011.

[13] D. Boneh, D. M. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In S. Jarecki and G. Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2009.

[14] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proceedings of Eurocrypt 2001*, volume 2045 of *LNCS*, pages 93–118. Springer-Verlag, 2001.

[15] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Proceedings of Crypto 2004*, volume 3152 of *LNCS*, pages 56–72. Springer-Verlag, 2004.

[16] E.-C. Chang, C. L. Lim, and J. Xu. Short redactable signatures using random trees. In *Proceedings of CT-RSA 2009*, pages 133–147, 2009.

[17] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. In *Proceedings of Eurocrypt 2012*, pages 281–300, 2012.

[18] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Succinct malleable NIZKs and an application to compact shuffles. In *Proceedings of TCC 2013*, pages 100–119, 2013.

[19] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Verifiable elections that scale for free. In *Proceedings of PKC 2013*, pages 479–496, 2013.

[20] M. Chase and A. Lysyanskaya. On signatures of knowledge. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 2006.

[21] D. Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[22] A. de Santis, G. di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Proceedings of Crypto 2001*, volume 2139 of *LNCS*, pages 566–598. Springer-Verlag, 2001.

[23] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal of Computing*, 29(1):1–28, 1999.

[24] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings of Crypto 1986*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, 1986.

[25] D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 697–714. Springer, 2012.

[26] G. Fuchsbauer. Commuting signatures and verifiable encryption and an application to non-interactively delegatable credentials. *IACR Cryptology ePrint Archive*, 2010:233, 2010.

[27] P. Golle. Revisiting the uniqueness of simple demographics in the us population. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, WPES '06, pages 77–80, New York, NY, USA, 2006. ACM.

[28] J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *Proceedings of Asiacrypt 2006*, volume 4284 of *LNCS*, pages 444–459. Springer-Verlag, 2006.

[29] J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero-knowledge for NP. In *Proceedings of Eurocrypt 2006*, volume 4004 of *LNCS*, pages 339–358. Springer-Verlag, 2006.

[30] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Proceedings of Eurocrypt 2008*, volume 4965 of *LNCS*, pages 415–432. Springer-Verlag, 2008.

[31] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *Proceedings of CT-RSA 2002*, pages 244–262, 2002.

[32] R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In B. Preneel, editor, *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer, 2002.

[33] S. Micali and R. Rivest. Transitive signature schemes. In *Proceedings of CT-RSA 2002*, pages 236–243, 2002.

[34] K. Miyazaki, G. Hanaoka, and H. Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *Proceedings of ASIACCS 2006*, pages 343–354, 2006.

[35] R. Steinfeld, L. Bull, and Y. Zheng. Context extraction signatures. In *Proceedings of ICISC 2001*, pages 285–304, 2001.

[36] L. Sweeney. $k$-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

# A Non-Interactive Proof Systems

## A.1 Standard definitions for zero-knowledge proofs of knowledge

**Definition A.1. [17]** *A set of algorithms* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *constitute a non-interactive (NI) proof system for an efficient relation* $R$ *with associated language* $L_R$ *if completeness and soundness below are satisfied. A NI proof system is extractable if, in addition, the extractability property below is satisfied. A NI proof system is witness-indistinguishable (NIWI) if the witness-indistinguishability property below is satisfied. An NI proof system is zero-knowledge (NIZK) if the zero-knowledge property is satisfied. A NIZK proof system that is also extractable constitutes a non-interactive zero-knowledge proof of knowledge (NIZKPoK) system. A NIWI proof system that is also extractable constitutes a non-interactive witness-indistinguishable proof of knowledge (NIWIPoK) system.*

1. *Completeness [10]. For all* $\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ *and* $(x, w) \in R$, $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ *for all proofs* $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, x, w)$.

2. *Soundness [10]. For all PPT $\mathcal{A}$, and for* $\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$*, the probability that* $\mathcal{A}(\mathsf{crs})$ *outputs* $(x, \pi)$ *such that* $x \notin L$ *but* $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ *is negligible.* Perfect *soundness is achieved when this probability is 0.*
3. *Extractability [29]. There exists a PPT extractor* $E = (E_1, E_2)$ *such that* $E_1(1^k)$ *outputs* $(\mathsf{crs}_e, \tau_e)$*, and* $E_2(\mathsf{crs}_e, \tau_e, x, \pi)$ *outputs a value* $w$ *such that (1) any PPT* $\mathcal{A}$ *given* $\sigma$ *cannot distinguish between the honest CRS and one output by* $E_1$*; i.e.,*

$$Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k) : \mathcal{A}(\mathsf{crs}) = 1] \approx Pr[(\mathsf{crs}_e, \tau_e) \xleftarrow{\$} E_1(1^k) : \mathcal{A}(\mathsf{crs}_e) = 1], \text{ and}$$

*and (2) for all PPT* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *outputs* $(x, \pi)$ *such that* $\mathcal{V}(\mathsf{crs}_e, x, \pi) = 1$ *but* $R(x, E_2(\mathsf{crs}_e, \tau_e, x, \pi)) = 0$ *is negligible; i.e., there exists a negligible function* $\nu(\cdot)$ *such that*

$$Pr[(\mathsf{crs}_e, \tau_e) \xleftarrow{\$} E_1(1^k); (x, \pi) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}_e) : \mathcal{V}(\mathsf{crs}_e, x, \pi) = 1 \wedge (x, E_2(\mathsf{crs}_e, \tau_e, x, \pi)) \notin R] < \nu(k).$$

Perfect *extractability is achieved if this probability is 0, and* $\mathsf{crs}_e$ *is distributed identically to* $\mathsf{crs}$*.*
4. *Witness indistinguishability [23]. For all* $(x, w_1, w_2)$ *such that* $(x, w_1), (x, w_2) \in R$*, any PPT* $\mathcal{A}$ *cannot distinguish between proofs for* $w_1$ *and proofs for* $w_2$*; i.e.,*

$$\begin{aligned} &Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k); (x, w_1, w_2) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}); \pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, x, w_0) : \mathcal{A}(\pi) = 1 \wedge (x, w_0), (x, w_1) \in R] \\ \approx &Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k); (x, w_1, w_2) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}); \pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, x, w_1) : \mathcal{A}(\pi) = 1 \wedge (x, w_0), (x, w_1) \in R]. \end{aligned}$$

Perfect *witness indistinguishability is achieved when these two distributions are identical.*
5. *Zero knowledge [23]. There exists a polynomial-time simulator algorithm* $S = (S_1, S_2)$ *such that* $S_1(1^k)$ *outputs* $(\mathsf{crs}_s, \tau_s)$*, and* $S_2(\mathsf{crs}_s, \tau_s, x)$ *outputs a value* $\pi_s$ *such that for all* $(x, w) \in R$*, a PPT adversary* $\mathcal{A}$ *cannot distinguish between proofs produced by the prover and simulator; i.e., for all PPT adversaries* $\mathcal{A}$*,*

$$Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k) : \mathcal{A}^{P(\mathsf{crs}, \cdot, \cdot)}(\mathsf{crs}) = 1] \approx Pr[(\mathsf{crs}_s, \tau_s) \xleftarrow{\$} S_1(1^k) : \mathcal{A}^{S(\mathsf{crs}_s, \tau_s, \cdot, \cdot)}(\mathsf{crs}_s) = 1],$$

*where, on input* $(x, w)$*,* $P$ *outputs* $\perp$ *if* $(x, w) \notin R$ *and* $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, x, w)$ *otherwise, and* $S$ *also outputs* $\perp$ *if* $(x, w) \notin R$*, and returns* $\pi \xleftarrow{\$} S_2(\mathsf{crs}_s, \tau_s, x)$ *otherwise.* Perfect *zero knowledge is achieved if for all* $(x, w) \in R$*, these distributions are identical.*

## A.2 Definitions for malleable proofs

Let $R(\cdot, \cdot)$ be a relation such that the corresponding language $L_R := \{x \mid \exists w \text{ such that } (x, w) \in R\}$ is in NP. As defined for malleable proofs [17], the relation is *closed* with respect to a transformation $T = (T_{\text{inst}}, T_{\text{wit}})$ if for every $(x, w) \in R$, $(T_{\text{inst}}(x), T_{\text{wit}}(w)) \in R$. The formal definition of a malleable proof extends the definition of a non-interactive proof $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ by adding an additional algorithm $\mathsf{ZKEval}$, designed to transform proofs. More formally, $\mathsf{ZKEval}$, on input the CRS $\mathsf{crs}$, a transformation $T$, an instance $x$ and a proof $\pi$ such that $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$, outputs a proof $\pi'$ for $x' := T_{\text{inst}}(x)$ such that $\mathcal{V}(\mathsf{crs}, x', \pi') = 1$. The proof system is then *malleable* with respect to a set of transformations $\mathcal{T}$ if for every $T \in \mathcal{T}$, $\mathsf{ZKEval}$ can be computed efficiently.

In addition to defining this basic notion of malleability, Chase et al. also defined how to meaningfully *control* the malleability of a proof system by extending the strong notion of simulation-sound extractability [28, 22] to deal with malleability; this means requiring that, for a set of transformations $\mathcal{T}$, if an adversary can produce a proof $\pi$ for an instance $x$ then the extractor should be able to extract from $\pi$ either a witness $w$ or a transformation $T \in \mathcal{T}$ and previous instance $x'$ such that $x = T_{\text{inst}}(x')$ (the definition of simulation-sound extractability required only this first condition). More formally, this is defined as follows:

**Definition A.2.** **[17]** *Let* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *be a NIZKPoK system for an efficient relation* $R$*, with a simulator* $(S_1, S_2)$ *and an extractor* $(E_1, E_2)$*. Let* $\mathcal{T}$ *be a set of unary transformations for the relation* $R$ *such that membership in* $\mathcal{T}$ *is efficiently testable. Let* $SE_1$ *be an algorithm that, on input* $1^k$*, outputs* $(\mathsf{crs}, \tau_s, \tau_e)$ *such that* $(\mathsf{crs}, \tau_s)$ *is distributed identically to the output of* $S_1$*. Let* $\mathcal{A}$ *be given, let* $Q := Q_{inst} \times Q_{proof}$ *be a table used to store the instances queried to* $S_2$ *and the proofs given in response, and consider the following game:*

- *Step 1.* $(\mathsf{crs}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)$.
- *Step 2.* $(x, \pi) \xleftarrow{\$} \mathcal{A}^{S_2(\mathsf{crs}, \tau_s, \cdot)}(\mathsf{crs}, \tau_e)$.
- *Step 3.* $(w, x', T) \leftarrow E_2(\mathsf{crs}, \tau_e, x, \pi)$.
- *Step 4.* $b \leftarrow (w \neq \perp \wedge (x, w) \notin R) \vee$
  $((x', T) \neq (\perp, \perp) \wedge (x' \notin Q_{inst} \vee x \neq T_{inst}(x') \vee T \notin \mathcal{T})) \vee$
  $(w, x', T) = (\perp, \perp, \perp))$.

*The NIZKPoK satisfies* controlled-malleable simulation-sound extractability *(CM-SSE, for short) with respect to* $\mathcal{T}$ *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that the probability (over the choices of* $SE_1$*,* $\mathcal{A}$*, and* $S_2$*) that* $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ *and* $(x, \pi) \notin Q$ *but* $b = 1$ *is at most* $\nu(k)$*.*

**Definition A.3.** **[17]** *For a non-interactive zero-knowledge proof system* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *with an associated simulation* $(S_1, S_2)$*, an efficient relation* $R$ *malleable with respect to* $\mathcal{T}$*, an adversary* $\mathcal{A}$*, and a bit* $b$*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $(\mathsf{crs}_s, \tau_s) \xleftarrow{\$} S_1(1^k)$.
- *Step 2.* $(\mathsf{state}, x_1, \pi_1, \ldots, x_q, \pi_q, T) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}_s, \tau_s)$.
- *Step 3. If* $\mathcal{V}(\mathsf{crs}_s, x_i, \pi_i) = 0$ *for some* $i$*,* $1 \leq i \leq q$*, or* $T \notin \mathcal{T}$*, abort and output* $\perp$*. Otherwise, form*
$$\pi \xleftarrow{\$} \begin{cases} S_2(\mathsf{crs}_s, T_{inst}(x_1, \ldots, x_q)) & \text{if } b = 0 \\ \mathsf{ZKEval}(\mathsf{crs}_s, T, \{x_i, \pi_i\}_{i=1}^q) & \text{if } b = 1. \end{cases}$$
- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \pi)$.

*Then the proof system is* strongly derivation private *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$*.*

# B Relations between Malleable Signature Definitions

In this section we relate our simulation-based notions of unforgeability and context hiding (Definitions 3.4 and 3.3, respectively) to definitions for homomorphic signatures as defined by Ahn et al. [3] and Attrapadung et al. [4].

## B.1 Previous definitions

To begin our comparison, we must first recall the specific definitions of Ahn et al. and Attrapadung et al. to which we are comparing our own. We first recall the two main unforgeability definitions of Ahn et al.; we refer to the first as existential unforgeability, and the second as NHU unforgeability.

**Definition B.1** (Existential unforgeability)**. [3]** *For a signature* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SigDerive})$ *malleable with respect to a predicate* $P$*, a table* $Q = Q_m \times Q_\sigma$*, and an adversary* $\mathcal{A}$*, consider the following game:*

- *Step 1.* $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; $S, Q \leftarrow \emptyset$.
- *Step 2.* $(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign},\mathsf{SigDerive},\mathsf{Reveal}}(vk)$, *where these oracles behave as follows:*

| $\underline{\mathsf{Sign}_{sk}(m)}$ | $\underline{\mathsf{SigDerive}_{pk}(\{h_i\}_i, m'))}$ | $\underline{\mathsf{Reveal}(h)}$ |
|---|---|---|
| $h \xleftarrow{\$} \mathcal{H}$ | $(h_i, m_i, \sigma_i) \leftarrow S \ \forall i$ | $(h, m, \sigma) \leftarrow S$ |
| *add* $(h, m, \mathsf{Sign}(sk, m))$ *to* $S$ | *if* $P(\vec{m}, m') = 1$, *pick* $h' \xleftarrow{\$} \mathcal{H}$ | *add* $(m, \sigma)$ *to* $Q$ |
| *return* $h$ | *add* $(h', m', \mathsf{SigDerive}(vk, \vec{m}, \vec{\sigma}, m'))$ *to* $S$ | *return* $\sigma$ |
| | *return* $h'$ | |

*Then the signature scheme satisfies* existential unforgeability *if for all such PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that the probability (over the choices of* KeyGen, Sign, SigDerive, $\mathcal{A}$, *and the handles) that* $\mathsf{Verify}(vk, \sigma^*, m^*) = 1$, *and* $m^* \notin P^*(Q_m)$ *is at most* $\nu(k)$.[3]

The definition of NHU unforgeability is similar to that of existential unforgeability, with the exception that the Sign oracle is the only one provided.

**Definition B.2** (NHU unforgeability)**. [3]** *For a signature* (KeyGen, Sign, Verify, SigDerive) *malleable with respect to a predicate* $P$, *a table* $Q = Q_m \times Q_\sigma$, *and an adversary* $\mathcal{A}$, *consider the following game:*

- *Step 1.* $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; $Q \leftarrow \emptyset$.
- *Step 2.* $(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}(\cdot)}(vk)$, *where* Sign *behaves as follows:*

$\underline{\mathsf{Sign}_{sk}(m)}$
$\sigma \leftarrow \mathsf{Sign}(sk, m)$
*add* $(m, \sigma)$ *to* $Q$
*return* $\sigma$

*Then the signature scheme satisfies* NHU unforgeability *if for all such PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that the probability (over the choices of* KeyGen, Sign, *and* $\mathcal{A}$*) that* $\mathsf{Verify}(vk, \sigma^*, m^*) = 1$ *and* $m^* \notin P^*(Q_m)$ *is at most* $\nu(k)$.

For context hiding, we compare simulation context hiding (as defined in Definition 3.3) to the notion of *adaptive* context hiding given by Attrapadung et al., which is in turn inspired by the computational notion of context hiding given by Ahn et al.

**Definition B.3** (Adaptive context hiding)**. [4]** *For a signature scheme* (KeyGen, Sign, Verify, SigDerive) *malleable with respect to a predicate* $P$, *an adversary* $\mathcal{A}$, *and a bit* $b$, *let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$.
- *Step 2.* $(\mathsf{state}, \vec{m}, \vec{\sigma}, m') \xleftarrow{\$} \mathcal{A}(vk, sk)$.
- *Step 3. If* $\mathsf{Verify}(vk, \sigma_i, m_i) = 0$ *for some* $i$ *or* $P(\vec{m}, m') = 0$, *abort and output* $\perp$. *Otherwise, form* $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m')$ *if* $b = 0$, *and* $\sigma \xleftarrow{\$} \mathsf{SigDerive}(vk, \vec{m}, \vec{\sigma}, m')$ *if* $b = 1$.
- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \sigma)$.

*Then the signature scheme satisfies* adaptive context hiding *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

[3]Here $P^*(M)$ denotes the set of messages derivable from $M$ by repeated derivation, where a message $m'$ is *derivable* from the set $M$ if $P(M, m') = 1$.

Before we move on to consider our new definitions, we relate the two notions of unforgeability using the notion of adaptive context hiding. As we show, when adaptive context hiding holds, NHU unforgeability implies existential unforgeability; as existential unforgeability trivially implies NHU unforgeability, this means the two notions are equivalent. For the rest of this section, we therefore focus mainly on the notion of NHU unforgeability.

**Theorem B.4.** *If a $P$-homomorphic signature scheme* (KeyGen, Sign, Verify, SigDerive) *is NHU unforgeable and adaptive context hiding, and if $m \in P^*(M)$ is efficiently decidable, then it is also existentially unforgeable.*

*Proof.* We first alter the existential unforgeability game by replacing the SigDerive oracle with an oracle that, on input $(\{h_i\}_i, m')$, looks up $\{(m_i, \sigma_i)\}_i$ and outputs $\mathsf{Sign}(sk, m')$ only if $P(\{m_i\}_i, m') = 1$. To see that these outputs are indistinguishable, we take an adversary $\mathcal{A}$ whose success probabilities in the two games differ by some non-negligible $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that can break adaptive context hiding with advantage $\epsilon$.[4] The reduction here is straightforward: on input $(vk, sk)$, $\mathcal{B}$ gives $vk$ to $\mathcal{A}$. $\mathcal{B}$ then uses its knowledge of $sk$ to answer calls to the Sign and Reveal oracles honestly; when $\mathcal{A}$ queries its SigDerive oracle on input $(\{h_i\}_i, m')$, $\mathcal{B}$ looks up the messages $\vec{m} = \{m_i\}_i$ and signatures $\vec{\sigma} = \{\sigma_i\}_i$ stored in $S$ with $\{h_i\}_i$, and outputs $(\vec{m}, \vec{\sigma}, m')$ as its own query to get back a signature $\sigma$; it then stores $(h', m', \sigma)$ in $S$ (for $h' \xleftarrow{\$} \mathcal{H}$), and returns $h'$ to $\mathcal{A}$. At the end, if $\mathcal{A}$ succeeds in producing a forgery, then $\mathcal{B}$ guesses that its signatures came from SigDerive, and otherwise it guesses they came from Sign. (Note that in order for $\mathcal{B}$ to test whether $\mathcal{A}$'s output is a successful forgery, we need the efficient decideability property.) As $\mathcal{B}$ simulates the behavior that $\mathcal{A}$ expects, $\mathcal{A}$'s success probability in the two games differs by $\epsilon$ depending on $\mathcal{B}$'s input. Thus, $\mathcal{B}$ breaks the adaptive context hiding game with advantage $\epsilon$.

Next, we argue that if an adversary $\mathcal{A}$ can win this altered existential unforgeability game with some non-negligible probability $\epsilon$, then we can construct an adversary $\mathcal{B}$ that wins the NHU unforgeability game with the same probability. Again, the reduction is straightforward: on input $vk$, $\mathcal{B}$ gives $vk$ to $\mathcal{A}$. On both Sign and SigDerive queries, $\mathcal{B}$ forwards the queried message $m$ to its own Sign oracle to get back a signature $\sigma$; it then deals with the handles and storage honestly, and returns the appropriate handle to $\mathcal{A}$, and answers Reveal queries completely honestly. When $\mathcal{A}$ outputs $(m^*, \sigma^*)$ at the end, $\mathcal{B}$ outputs the same. As $\mathcal{B}$ correctly simulates the oracles, and wins whenever $\mathcal{A}$ does, $\mathcal{B}$ succeeds with the same advantage as $\mathcal{A}$. □

## B.2 Relating our definitions to prior work

In Section 3, we briefly outlined our reasons for choosing to work in the language of transformations rather than that of predicates. To meaningful relate our security definitions to the predicate-based definitions just presented, however, we first need a way to formally relate transformations and predicates. We do this as follows:

**Definition B.5.** *We say that a transformation class $\mathcal{T}$* implements *a predicate $P$ if for all $M \subset \mathcal{M}$ and $m^* \in \mathcal{M}$, $P(M, m^*) = 1$ if and only if there exist $m_1, \ldots, m_n \in \mathcal{M}$ and $T \in \mathcal{T}$ such that $T(m_1, \ldots, m_n) = m^*$.*

Note that we can always construct a transformation class $\mathcal{T}$ that implements a given predicate as follows: for each pair $(\vec{m}, m^*)$ such that $P(\vec{m}, m^*) = 1$, we consider $\mathcal{T}_P$ that contains an extreme

[4] Technically, the advantage is the same only if we consider a multi-query version of the adaptive context hiding game; otherwise, using a hybrid, it is $\epsilon/q$, where $q$ is the number of the queries to the SigDerive oracle, which is still non-negligible for polynomial $q$.

partial function; i.e., $\mathcal{T}_P = \{\vec{m} \mapsto m^* \mid P(\vec{m}, m^*) = 1\}$. Sometimes, however, there is a more natural transformation class $\mathcal{T}$ for implementing a given predicate.[5]

The other major syntactic difference is that our simulation-based definitions consider signatures schemes in the CRS model, whereas previous definitions considered signatures in the standard model (i.e., without the trusted setup component $\mathsf{Gen}$ defined in Section 3.1). This difference is also easily addressed: for any CRS model signature $\Sigma = (\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, a corresponding signature $\Sigma'$ in the standard model can be defined as $(\mathsf{KeyGen}', \mathsf{Sign}, \mathsf{Verify})$, where $\mathsf{KeyGen}'$ runs $\mathsf{crs} \xleftarrow{\$} \mathsf{Gen}(1^k)$ and $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{crs})$, and outputs $(vk' := (\mathsf{crs}, vk), sk)$.

With these notational differences out of the way, we now proceed as follows: first, we show that for simulatable signatures, our notions of simulation unforgeability and simulation context hiding imply the notions of NHU unforgeability and adaptive context hiding respectively. Next, we show that existing constructions can be easily made simulatable, without changing their security guarantees, by adding a proof of knowledge and shifting to the CRS model. Finally, we show that for certain classes of transformations (that, to the best of our knowledge, includes all classes of transformations for which constructions exist), our notion of simulation unforgeability is implied by, and thus equivalent to, NHU unforgeability (again, with respect to simulatable signatures, as otherwise our definitions are not well defined).

### B.2.1 Simulatability-based definitions imply previous definitions

**Theorem B.6.** *Let $\mathcal{T}$ be a transformation class that implements the predicate $P$. Let $\Sigma$ be a signature in the CRS model that is malleable with respect to $\mathcal{T}$, and let $\Sigma'$ be the corresponding standard model signature. If (1) $\Sigma$ is simulatable, and (2) $m \in P^*(M)$ is efficiently decidable, then $\Sigma'$ is NHU-unforgeable with respect to $P$ if $\Sigma$ is simulation unforgeable with respect to $\mathcal{T}$.*

*Proof.* Given a successful adversary $\mathcal{A}$ that breaks NHU-unforgeability, we want to break either simulatability or simulation unforgeability. We proceed through the following series of games:

- Game 0. The original NHU unforgeability game.
- Game 1. In this game, replace $\mathsf{Gen}$ with $\mathsf{SimGen}$ and $\mathsf{Sign}$ with $\mathsf{SimSign}$. To see that this is indistinguishable from Game 0, we assume there exists an adversary $\mathcal{A}$ whose success probabilities in Games 0 and 1 differ by some non-negligible $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that breaks simulatability with advantage $\epsilon$. To start, $\mathcal{B}$ is given either a real $\mathsf{crs}$ and a $\mathsf{Sign}$ oracle, or a simulated $\mathsf{crs}$ and a $\mathsf{SimSign}$ oracle. $\mathcal{B}$ then gives its own $\mathsf{crs}$ to $\mathcal{A}$ and responds to $\mathcal{A}$'s $\mathsf{Sign}$ queries using its own oracle. At the end of the game, if $\mathcal{A}$ succeeds in producing a forgery then $\mathcal{B}$ guesses it is in the real world, and other $\mathcal{B}$ guesses it is in the simulated world. (Again, we need the predicate to be efficiently decidable in order for $\mathcal{B}$ to decide.) As $\mathcal{B}$ successfully simulates the behavior that $\mathcal{A}$ expects, $\mathcal{A}$'s success probability in the simulated work differs by $\epsilon$ from its success probability in the real world, meaning $\mathcal{B}$ breaks simulatability with advantage $\epsilon$.
- Game 2. In this game, replace $\mathsf{SimGen}$ with $\mathsf{SimExtGen}$. To see that an adversary cannot distinguish this game from Game 2, we observe that the distribution over the CRS is in fact identical, and thus so are the two games. Furthermore, if there exists an adversary $\mathcal{A}$ that wins at this game with some non-negligible probability $\epsilon$ then we can use it to construct an adversary $\mathcal{B}$ that breaks simulation unforgeability with the same probability. To start, $\mathcal{B}$ gives to $\mathcal{A}$ the $\mathsf{crs}$ that it receives as input; it also answers $\mathcal{A}$'s oracle queries using its own oracle and, when $\mathcal{A}$ outputs $(m^*, \sigma^*)$ at the end of the game, it also outputs $(m^*, \sigma^*)$. To see that $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does, observe

[5] Ahn et al. suggest that in some cases one could pass an additional input $w$ into the predicate. This could also be done here if $w$ is efficiently computable from $T$.

that whenever $\sigma^*$ verifies but $m^* \notin P^*(Q_m)$ (and thus $\mathcal{A}$ succeeds), then $(m^*, \sigma^*)$ cannot be in $Q$ and there simply do not exist any $\vec{m}' \in Q_m$ and $T \in \mathcal{T}_P$ such that $m^* = T(\vec{m}')$. Consequently, extraction will fail, and $\mathcal{B}$ succeeds as well.

As each game is indistinguishable from the previous one, and the success probability in the last game is bounded by the success probability in the simulation unforgeability game, simulatability and simulation unforgeability therefore imply NHU unforgeability. □

**Theorem B.7.** *Let $\mathcal{T}$ be a transformation class that implements a predicate $P$. Let $\Sigma$ be a signature in the CRS model that is malleable with respect to $\mathcal{T}$, and let $\Sigma'$ be the corresponding standard model signature. If $\Sigma$ is simulatable and simulation context hiding with respect to $\mathcal{T}$, then $\Sigma'$ is adaptive context hiding with respect to $P$.*

*Proof.* We proceed through the following series of games:

- Game 0. The original adaptive context hiding game.
- Game 1. Instead of checking that $P(\vec{m}, m') = 0$, check that there exists a transformation $T \in \mathcal{T}$ such that $T(m_1, \ldots, m_n) = 1$ and abort otherwise. This is identical to Game 0 because $\mathcal{T}$ implements $P$.
- Game 2. Replace $\mathsf{KeyGen}'$ with $\mathsf{Gen}$ and $\mathsf{KeyGen}$ and replace $\mathsf{SigDerive}(vk', \vec{m}, \vec{\sigma}, m')$ with $\mathsf{SigEval}(\mathsf{crs}, vk, T, \vec{m}, \vec{\sigma})$. Again, this is identical to Game 2, as $\Sigma$ is simply the CRS model version of $\Sigma'$.
- Game 3. Replace $\mathsf{Gen}$ with $\mathsf{SimGen}$ and use $\mathsf{SimSign}$ to form the challenge signature. This is indistinguishable from Game 2 by simulatability; the argument is analogous to the one in the proof of Theorem B.6. Furthermore, this is now the simulation context hiding game, except that the keys are generated by $\mathsf{KeyGen}$. Given an adversary $\mathcal{A}$ that wins at Game 3 with some non-negligible advantage $\epsilon$, it is therefore trivial to construct an adversary $\mathcal{B}$ that breaks simulation context hiding with the same advantage: to generate its keys, $\mathcal{B}$ runs $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$ and gives these to $\mathcal{A}$. Whenever $\mathcal{A}$ outputs its tuple $(\vec{m}, \vec{\sigma}, m')$, $\mathcal{B}$ outputs $(vk, \vec{m}, \vec{\sigma}, T)$, where $T$ maps $\vec{m} \mapsto m'$ (i.e., is the extreme partial function mentioned above); it then forwards the resulting signature $\sigma$ back to $\mathcal{A}$, and outputs the same guess bit as $\mathcal{A}$ at the end. As $\mathcal{B}$ simulates the behavior that $\mathcal{A}$ expects, and wins whenever $\mathcal{A}$ does, $\mathcal{B}$ succeeds with the same advantage.

As each game is indistinguishable from the previous one, and the success probability in the last game is bounded by the success probability in the simulation context hiding game, simulatability and simulation context hiding imply adaptive context hiding. □

By combining these two theorems with Theorem B.4, which relates NHU unforgeability back to existential unforgeability, we obtain the following corollary:

**Corollary B.8.** *Let $\mathcal{T}$ be a transformation class that implements the predicate $P$. Let $\Sigma$ be a signature in the CRS model that is malleable with respect to $\mathcal{T}$, and let $\Sigma'$ be the corresponding standard model signature. Then if (1) $\Sigma$ is simulatable, (2) $\Sigma'$ is adaptive context hiding (alternatively, $\Sigma$ is simulation context hiding), (3) $P$ is efficiently decidable, and (4) $\Sigma$ is simulation unforgeable, then $\Sigma'$ is existentially unforgeable.*

### B.2.2 Adding simulatability to existing constructions

After demonstrating that our definitions imply previous definitions for simulatable signatures, we now demonstrate that building simulatability into existing constructions is not too difficult. In fact, for any standard model signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ we consider a CRS model signature scheme $\Sigma^* = (\mathsf{Gen}^*, \mathsf{KeyGen}^*, \mathsf{Sign}, \mathsf{Verify})$, in which $\mathsf{Gen}^*$ outputs the $\mathsf{crs}$ for a non-interactive zero-knowledge

proof of knowledge (NIZKPoK) for the relation $R := \{(vk, (sk, r)) \mid \mathsf{KeyGen}(1^k; r) = (vk, sk)\}$, and $\mathsf{KeyGen}^*(\mathsf{crs})$ runs $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k; r)$ and $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, vk, (sk, r))$, and outputs $(vk^* := (vk, \pi), sk)$. We denote this CRS model signature as the *extended* CRS model signature.

**Theorem B.9.** *Let $\Sigma$ be a signature scheme in the standard model, let $\Sigma^*$ be the extended CRS model signature using $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$, and let $\Sigma'$ be the standard model scheme corresponding to $\Sigma^*$. Then, if $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ is a zero-knowledge proof of knowledge,*

1. *$\Sigma^*$ is simulatable;*
2. *if $\Sigma$ is NHU unforgeable, then $\Sigma'$ is NHU unforgeable;*
3. *if $\Sigma$ is adaptive context hiding, then $\Sigma'$ is adaptive context hiding.*

*Proof.* To show simulatability, $\mathsf{SimGen}$ outputs $(\mathsf{crs}, \tau_e) \xleftarrow{\$} E_1(1^k)$; i.e., the extraction trapdoor for the proof of knowledge. Given this trapdoor, $\mathsf{SimSign}$ extracts the secret key from the proof of knowledge, and then sign using the honest signing algorithm. By extractability, the value that $\mathsf{SimSign}$ extracts will be a valid signing key $sk$ with overwhelming probability; as simulation is therefore perfect with all but negligible probability, the signature is simulatable.

To show that unforgeability is preserved, consider an adapted version of the NHU unforgeability game in which $\pi$ is replaced by a simulated proof; this adapted game is indistinguishable from the regular game by zero knowledge. Furthermore, if we are given an adversary $\mathcal{A}$ that can win at this adapted game with non-negligible probability $\epsilon$ (against $\Sigma'$), we can construct an adversary $\mathcal{B}$ that wins at the NHU unforgeability game with the same probability (against $\Sigma$). To start, $\mathcal{B}$ gets as input a verification key $vk$; it then forms the simulated proof $\pi$ (so $(\mathsf{crs}, \tau_s) \xleftarrow{\$} S_1(1^k)$ and $\pi \xleftarrow{\$} S_2(\mathsf{crs}, \tau_s, vk)$) and gives $vk' := (vk, \pi)$ to $\mathcal{A}$. When $\mathcal{A}$ then queries its $\mathsf{Sign}$ oracle, $\mathcal{B}$ responds by querying its own oracle, and when $\mathcal{A}$ outputs $(m^*, \sigma^*)$ at the end of the game $\mathcal{B}$ does the same. As $\mathcal{B}$ perfectly simulates the parameters and the oracle behavior that $\mathcal{A}$ expects, and furthermore $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does, $\mathcal{B}$ succeeds with the same probability as $\mathcal{A}$.

Finally, to show that context hiding is preserved, we first consider the context hiding game for $\Sigma'$ when the verification key is generated using a simulated CRS and simulated proof; this is indistinguishable from the original game by zero knowledge. Then take an adversary $\mathcal{A}$ that wins this adaptive context hiding game for this modified $\Sigma'$ with some non-negligible advantage $\epsilon$ and use it to construct an adversary $\mathcal{B}$ that wins the adaptive context hiding game for $\Sigma$ with the same advantage. Because $\mathcal{B}$ gets the signing keypair, this reduction is quite straightforward: to start, $\mathcal{B}$ receives $(vk, sk)$ and computes $(\mathsf{crs}, \tau_s) \xleftarrow{\$} S_1(1^k)$ and $\pi \xleftarrow{\$} S_2(\mathsf{crs}, \tau_s, vk)$; it then gives $(vk' := (vk, \pi), sk)$ to $\mathcal{A}$. $\mathcal{B}$ then outputs the same tuple as $\mathcal{A}$, forwards the resulting signature back to $\mathcal{A}$, and at the end outputs the same guess bit as $\mathcal{A}$. As $\mathcal{B}$ once again perfectly simulates the parameters and signature, and wins whenever $\mathcal{A}$ does, $\mathcal{B}$ wins with the same advantage $\epsilon$. □

### B.2.3 When existing constructions meet our definitions

For context hiding, our notion of simulation context hiding seems clearly stronger than existing definitions, as it guarantees privacy even in case of adversarially-generated verification keys. What is unclear, however, is whether existing malleable signature schemes (when adapted to be simulatable) can meet our stronger notion of simulation unforgeability. Perhaps unsurprisingly, this depends largely on the class of transformations they require.

As it turns out, for most existing schemes the relevant class of transformations is such that it is easy to determine, given a set of "base" messages and a transformed message, the transformation that was applied. More formally, we have the following definition:

**Definition B.10** (Trivial extractability)**.** *We say a transformation class $\mathcal{T}$ is* trivially extractable *if there exists a PPT algorithm* $\mathsf{TrivExt}$ *such that for all polynomial-sized $M \subset \mathcal{M}$ and $m^* \in \{m \mid \exists T \in \mathcal{T}, \vec{m} \in M : m = T(\vec{m})\}$, the algorithm $\mathsf{TrivExt}(M, m^*)$ produces $\vec{m} = (m_1, \ldots, m_n) \in M$ and transformation $T$ such that $m^* = T(\vec{m})$.*

As an example, we consider the linear homomorphic signature construction given by Attrapadung et al. Phrased in the language of transformations, their construction allows for linear combinations of (similarly tagged) messages; i.e., for any vector $\vec{\beta} = (\beta_1, \ldots, \beta_n) \in \mathbb{Z}^n$, $T_{\vec{\beta}}((\tau_1, \vec{y}_1), \ldots, (\tau_n, \vec{y}_n)) = (\tau_1, \sum_i \beta_i \vec{y}_i)$ if $\tau_1 = \ldots = \tau_n$ and $\perp$ otherwise. To see how this transformation class is trivially extractable, we construct the $\mathsf{TrivExt}$ algorithm as follows: on input $M$ and $m^* = (\tau, \vec{y})$, it first lets $M_\tau$ define the set of messages in $M$ with tag $\tau$. Second, it identifies a subset of $M_\tau$ that forms a basis for the subspace spanned by the vectors in $M_\tau$. Third, it finds coefficients to represent $\vec{y}$ as a linear combination of those basis vectors. Fourth and last, it outputs the basis vectors as $(m_1, \ldots, m_n)$, and the linear combination defined by the coefficients as the transformation $T$. This algorithm is furthermore efficient, as each step involves basis linear algebra.

In a similar manner, one can show that subset signatures, quotable signatures, and transitive signatures are all trivially extractable as well; briefly, for each of these constructions $\mathsf{TrivExt}$ respectively corresponds to finding a superset, searching for a substring, and finding a path in a graph, all of which have simple efficient algorithms.

We argue that for such transformation classes, NHU unforgeability and simulation unforgeability are equivalent.

**Theorem B.11.** *Let $\mathcal{T}$ be a transformation class that implements the predicate $P$. Let $\Sigma$ be a signature in the CRS model that is malleable with respect to $\mathcal{T}$, and let $\Sigma'$ be the corresponding standard model signature. If $\Sigma$ is simulatable and $\mathcal{T}$ is trivially extractable, then $\Sigma'$ is NHU unforgeable with respect to $P$ if and only if $\Sigma$ is simulation unforgeable with respect to $\mathcal{T}$.*

*Proof.* Let $(\mathsf{SimSetup}, \mathsf{SimSign})$ be the simulator for $\Sigma$, and let $(\mathsf{SimExtSetup}, \mathsf{SigExt})$ be the extractor, where $\mathsf{SimExtSetup}$ runs $(\mathsf{crs}, \tau_s) \xleftarrow{\$} \mathsf{SimSetup}(1^k)$ and outputs $(\mathsf{crs}, \tau_s, \perp)$, and $\mathsf{SigExt}$, given $m^*$ and the messages $\vec{m} \in Q_m$ (i.e., the messages queried to to the $\mathsf{SimSign}$ oracle), outputs $(\vec{m}', T) \xleftarrow{\$} \mathsf{TrivExt}(\vec{m}, m^*)$. We argue that the success probability of any adversary interacting with this simulator and extractor in the simulation unforgeability game is at most negligibly different from its success probability in the NHU unforgeability game.

To do this, we proceed through the following series of games:

- Game 0. The regular simulation unforgeability game, using the extractor defined above.
- Game 1. Replace $\mathsf{SimExtGen}$ with $\mathsf{Gen}$ and the $\mathsf{SimSign}$ oracle with a $\mathsf{Sign}$ oracle. This is indistinguishable from Game 0 by simulatability (the argument is analogous to the one in the proof of Theorem B.6).
- Game 2. Replace the winning conditions of the game; rather than check for the $(\vec{m}', T)$ produced by $\mathsf{SigExt}$ that $\vec{m}' \subseteq Q_m$ and $m^* = T(\vec{m}')$ and $T \in \mathcal{T}$, check instead that there exists a transformation $T \in \mathcal{T}$ and a set of messages $(m_1, \ldots, m_n) \in Q_m$ such that $T(m_1, \ldots, m_n) = m^*$, and have the adversary win if no such transformation exists. Because $\mathsf{SigExt}$ runs $\mathsf{TrivExt}$, this is identical to Game 1 by the definition of trivial extractability.
- Game 3. Replace the winning conditions again; this time, have the adversary win if there does not exist a set of messages $(m_1, \ldots, m_n) \subseteq Q_m$ such that $P(m_1, \ldots, m_n, m^*) = 1$. This is identical to Game 2 because $\mathcal{T}$ implements $P$.

As Game 3 is now the NHU unforgeability game, and furthermore each game is (at least) indistinguishable from the previous one, this implies that an adversary can behave at most negligibly differently in the simulation unforgeability and NHU unforgeability games. □

Once again, we can combine Theorems B.9 and B.11 to get a corollary that shows that, for any of the standard types of transformation considered in previous work (i.e., any trivially extractable transformations), we can construct simulation-unforgeable signature schemes based on ones that are NHU unforgeable.

**Corollary B.12.** *If $\Sigma$ is NHU unforgeable for a predicate $P$ implemented by a trivially extractable transformation class $T$, then the extended CRS model signature $\Sigma^*$ is simulation unforgeable.*

# C CM-Friendliness for Delegatable Anonymous Credentials

## C.1 Variable-length CM-friendliness

To show that Groth-Sahai proofs [30] could be used to construct cm-NIZKs, Chase et al. [17] relied on a property, called CM-friendliness, over languages and transformations. This property, however, assumes that there is a fixed-length representation for all transformations; the size of the resulting proofs is then dependent on this length. As we saw in Section 4, however, in our application of delegatable anonymous credentials, the size of the transformation depends on how many times a credential has been delegated. To address this gap, one could of course place a maximum on the number of times a credential can be delegated and then pad the representation of each transformation up to the corresponding maximum length; as this solution seems far from ideal, we instead adopt a solution in which the length depends only on the number of times the current credential has been delegated so far.

To do this, we slightly modify the notion of CM-friendliness; we work from the already revised notion due to Chase et al. [19] that captures the original definition as *perfect* CM-friendliness. The additional change we must make is because, in many cases, the instance $x$ determines an upper bound on the length of the transformation that produced $x$. (For example, in our credential application the level of the credential determines an upper bound on the number of times it has been delegated so far, and thus on the length of the transformation applied.) Thus, we now want to allow the representation of a transformation to depend on the instances that it can produce. Similarly, we can generalize the notion of CM-friendliness to allow the representation of witnesses to depend on the statements; while we do not need this for our application, we nevertheless include it here for completeness.

**Definition C.1** (Variable-length CM-friendliness)**.** *For sets $S$ and $S'$ of pairing product equations and a PPT setup algorithm params $\xleftarrow{\$}$ $\mathsf{CRSSetup}(1^k)$ that specifies some group $G$, we say that $(S, S', \mathsf{CRSSetup})$ is a* CM-friendly instantiation *for a relation $R$ and transformation class $\mathcal{T}$ if the following six properties hold:*

1. *Representable statements. Any instance can be represented as a set of group elements of a fixed length, and any witness can be represented as a set of group elements whose length depends only on the associated instance. More formally, there is a positive integer $d_x$ and an efficiently computable invertible function $F_x(params, \cdot)$ that maps $L_R \mapsto G^{d_x}$, an efficiently computable function $D_w : L_R \to \mathbb{Z}^+$, and an efficiently computable invertible function $F_w(params, \cdot, \cdot)$ that, for any $x \in L_R$, maps $W_x \mapsto G^{D_w(x)}$, where $W_x := \{w \mid (x, w) \in R\}$. For $(x, w) \in R$ we call $F_x(params, x)$ the representation of $x$ and $F_w(params, x, w)$ the representation of $w$.*
2. *Representable transformations. Any transformation can be represented as a set of group elements whose length depends only on the possible resulting instances. More formally, there is an efficiently computable function $D_t : L_R \to \mathbb{Z}^+$, and an efficiently computable invertible function $F_t(params, \cdot, \cdot)$ such that for any $x \in L_R$, $F_t$ maps $T_x \mapsto G^{D_t(x)}$, where $T_x := \{T | \exists x' \in L_R \text{ s.t. } T(x') = x\}$. For $x \in L_R$ and $T \in T_x$ we call $F_t(params, x, T)$ the representation of $T$ for $x$.*

3. *Provable statements. Proving satisfaction of the set $S$ constitutes a computationally sound proof for the statement "$(x, w) \in R$;" i.e., for params $\xleftarrow{\$}$ $\mathsf{CRSSetup}(1^k)$ it holds that (1) if $(x, w) \in R$ then $F_x(params, x)$ and $F_w(params, x, w)$ satisfy $S$, and (2) for a PPT adversary $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that the probability (over the randomness used in $\mathsf{CRSSetup}$ and $\mathcal{A}$) that $\mathcal{A}$ can produce $(X, W)$ such that $X$ and $W$ satisfy $S$ but $(F_x^{-1}(params, X), F_w^{-1}(params, W)) \notin R$ is at most $\nu(k)$.*
4. *Provable transformations. Proving satisfaction of the set $S'$ constitutes a computationally sound proof for the statement "$T_{inst}(x') = x$ for $T \in \mathcal{T}$" using the above representations for $x$ and $T$; i.e., for params $\xleftarrow{\$}$ $\mathsf{CRSSetup}(1^k)$ it holds that (1) if $T \in \mathcal{T}$ and $T_{inst}(x') = x$ then $F_t(params, x, T)$, $F_x(params, x)$, and $F_x(params, x')$ satisfy $S'$, and (2) for a PPT adversary $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that the probability (over the randomness used in $\mathsf{CRSSetup}$ and $\mathcal{A}$) that $\mathcal{A}$ can produce $(T', X, X')$ such that $T'$, $X$, and $X'$ satisfy $S'$ but $F_t^{-1}(params, T') \notin \mathcal{T}$ or $F_t^{-1}(params, T')(F_x^{-1}(params, X') \neq F_x^{-1}(params, X)$ is at most $\nu(k)$.*
5. *Transformable statements. For any $T \in \mathcal{T}, (x, w) \in R$, there is a valid transformation that changes the statement "$(x, w) \in R$" (phrased using $S$ as above) into the statement "$(\widehat{x}, \widehat{w}) \in R$" for $\widehat{x} := T_{inst}(x)$ and $\widehat{w} := T_{wit}(w)$.*
6. *Transformable transformations: for any $T, \widehat{T} \in \mathcal{T}, x, x' \in L_R$ with $x = T(x')$, there is a valid transformation that changes the statement "$x = T(x')$ for $T \in \mathcal{T}$" (phrased using $S'$ as above) into the statement "$\widehat{x} = \widehat{T_{inst}} \circ T_{inst}(x')$ for $\widehat{T} \circ T \in \mathcal{T}$" where $\widehat{x} := \widehat{T_{inst}}(x)$.*

*We say that $(S, S', \mathsf{CRSSetup})$ is a* perfect *CM-friendly instantiation if the probabilities in the third and fourth properties are zero. A relation and transformation class $(R, \mathcal{T})$ are (perfectly) CM-friendly if they have a (perfect) CM-friendly instantiation.*

Using this definition, Chase et al. show that, given any CM-friendly relation $R$ and transformation $\mathcal{T}$, one can use Groth-Sahai proofs to construct the NIWI proof system needed for the cm-NIZK construction (which, briefly, is a proof of knowledge of $(w, \sigma, x', T)$ such that $(x, w) \in R \vee (\mathsf{Verify}(vk, x', \sigma) = 1 \wedge T_{\text{inst}}(x') = x \wedge T \in \mathcal{T})$). Their proof of this relies on two main arguments: first, that Groth-Sahai proofs can be used to construct NIWI proofs for any statement that is a disjunction of two pairing product statements, and second, that the transformations can be applied independently to both sides of the disjunction.

To ensure that we can still construct cm-NIZKS using our modified definition, we observe that the same argument still goes through. To see this, both the prover and verifier for the proof system can first use the instance $x$ to determine which pairing product equations to include on each side of the disjunction; afterwards, the argument proceeds exactly as before, as transformations can still be applied independently to each side of the disjunction to produce the appropriate transformed proof.

## C.2 Proof of credential CM-friendliness

As discussed in Section 5.2, in order to use Groth-Sahai proofs, we require that the identity always be a valid transformation (as GS proofs are inherently re-randomizable). While this may not be a restriction for many classes of transformations, for the set of allowable credential transformations $\mathcal{T}_{\text{dac}}$ defined in Section 4.1, the identity transformation is not allowed; as such, we must modify it slightly to include the identity. We also replace the flags with bits in order to represent them as group elements; this means we use $b = 0$ in place of $\mathsf{credential}$ and $b = 1$ in place of $\mathsf{proof}$ (for messages; for transformations we denote the bit $b_{new}$). Formally, $\mathcal{T}_{\text{dac}}$ must consist of all transformations $T_{\text{dac}} = (\{\mathsf{nym}_j, sk_j, \mathsf{open}_j\}_{j=1}^k, \mathsf{nym}_{new}, b_{new})$, $k \geq 0$, such that

1. $\mathsf{NymVerify}(params, \mathsf{nym}_j, sk_j, \mathsf{open}_j) = 1$ for all $j$, $1 \leq j \leq k$,

2. if $b_{new} = 1$ then $\mathsf{nym}_k = \mathsf{nym}_{new}$ and $sk_{k-1} = sk_k$, and

3. $T_{\text{dac}}(\mathsf{nym}_r, \ell, b) := \begin{cases} (\mathsf{nym}_r, \ell, b) & \text{if } \mathsf{nym}_{new} = \mathsf{nym}_r,\ b = b_{new} = 0 \text{ and } k = 0 \\ (\mathsf{nym}_{new}, \ell + k, 0) & \text{if } \mathsf{nym}_1 = \mathsf{nym}_r,\ b = b_{new} = 0 \text{ and } k > 0 \\ (\mathsf{nym}_{new}, \ell + k - 2, 1) & \text{if } \mathsf{nym}_1 = \mathsf{nym}_r,\ b = 0,\ b_{new} = 1, \text{ and } k > 1 \\ \bot & \text{otherwise.} \end{cases}$

In order to construct a malleable signature for $\mathcal{T}_{\text{dac}}$ using the construction in Section 5, we recall that we are given a hard relation $R_{pk}$, and we need to construct a cm-NIZK for a relation $R$ such that $((pk, (\mathsf{nym}_r, \ell, b)), sk) \in R$ if and only if $(pk, sk) \in R_{pk}$. This cm-NIZK must be malleable with respect to the related transformation class $\mathcal{T}_{\text{nizk}}$, which contains transformation $T = (T_{\text{inst}}, T_{\text{wit}})$ such that $T_{\text{inst}} = (\text{id}, T_{\text{dac}} \in \mathcal{T}_{\text{dac}})$ and $T_{\text{wit}} = \text{id}$. By constructing a CM-friendly instantiation for this relation and transformation class, we thus conclude that we can instantiate the cm-NIZK using the Chase et al. construction [17].

We first describe our concrete choices for the hard relation $\mathcal{R}_{pk}$ and the commitment scheme used to produce pseudonyms; because we consider only bilinear groups, these constructions are both with respect to some setup algorithm $\mathsf{CRSSetup}$ that outputs a bilinear group $(p, G, G_T, e, g)$ such that $|G| = |G_T| = p$, $G = \langle g \rangle$, and $e : G \times G \to G_T$.[6] For the hard relation, the generator $\mathcal{G}$ samples pairs $(pk = (g^a, g^b), sk = g^{ab})$ for $a, b \xleftarrow{\$} \mathbb{F}_p$; note that by CDH it should be hard to find the secret key $g^{ab}$ given only the public key $(g^a, g^b)$, but we can use a pairing to verify that a given pair $(pk, sk)$ is correct; i.e., the $\mathsf{KeyCheck}$ algorithm required for the signature can check that $e(pk_1, pk_2) = e(g, sk)$. For the commitment scheme, we use Boneh-Boyen-Shacham encryption [11]; this means $\mathsf{ComSetup}$ generates $f, h \xleftarrow{\$} G$ and sets $params' := (f, h)$, and that a commitment $c = \mathsf{Com}(params', sk) = (c_1 := f^r, c_2 := h^s, c_3 := g^{r+s} sk)$ for $r, s \xleftarrow{\$} \mathbb{F}_p$. The opening for this commitment is $\mathsf{open} := (R, S) = (g^r, g^s)$, which can be verified (in combination with $sk$) by checking that $e(c_1, g) = e(R, f)$, $e(c_2, g) = e(S, h)$, and $e(c_3, g) = e(R \cdot S \cdot sk, g)$.

**Representable statements:** To see that instances and witnesses are group elements, we observe that instances are of the form $(pk, (\mathsf{nym}_r, \ell, b))$, and witnesses are of the form $sk$, where (using our concrete choice of $\mathcal{R}_{pk}$ above) $sk = g^{ab}$ is a group element. As for instances, using our concrete choices above we see that $pk$ is a pair $(pk_1, pk_2) \in G$, and $\mathsf{nym}$ is a triple $(n_1, n_2, n_3) \in G$. We furthermore represent $\ell$ by the group element $g^\ell$, and $b$ by the group element $g^b$.

**Representable transformations:** As described above, each transformation $T$ is described by a set of values $(\{\mathsf{nym}_j, sk_j, \mathsf{open}_j\}_{j=1}^k, \mathsf{nym}_{new}, b_{new})$. As also described above, in our instantiations, $\mathsf{nym}_j$, $sk_j$, and $\mathsf{open}_j$ are all elements of $G$ for all $j$. The value $k$ for transforming an instance $(pk, (\mathsf{nym}_{old}, \ell_{old}, b_{old}))$ into $x = (pk, (\mathsf{nym}_{new}, \ell_{new}, b_{new}))$ is computed as $k = \ell_{new} - \ell_{old} + 2(b_{new} - b_{old})$.

Rather than use this representation, however, which depends on how many times a credential has been delegated, we would like to switch to a representation whose length depends only on the resulting instance $x$. To achieve this, we prepend the list $\{\mathsf{nym}_j, sk_j, \mathsf{open}_j\}_{j=1}^k$ with a list of $\ell_{new} + 2b_{new} - k - 1$ dummy values $\{\mathsf{nym}_j = (1,1,1), sk_j = 1, \mathsf{open}_j = (1,1)\}_{j=1}^{\ell_{new}+2b_{new}-k-1}$.

The values $\ell_{new}$ and $b_{new}$, together with $k$, uniquely determine $\ell_{old}$ and $b_{old}$, and we include all four of these values in the representation $F_t(x, T)$ of the transformation $T$ for $x$. We represent $b_{new}$ and $b_{old}$ as $g^{b_{new}}$ and $g^{b_{old}}$. The value of $\ell_{new}$ is then uniquely determined by the number of $\mathsf{nym}_j, sk_j, \mathsf{open}_j$ tuples included and the value of $g^{b_{new}}$. We represent $\ell_{old}$ by the vector of values

[6] We stick with a symmetric prime-order setting for simplicity, but we note that using an asymmetric setting, with the SXDH instantiation of GS proofs, should also be feasible.

$(L_1, \ldots, L_{\ell_{new}+2b_{new}})$ such that $L_i = 1$ for $i \neq \ell_{old} + 2b_{old}$ and $L_{\ell_{old}+2b_{old}} = g$. (Note that since there is only one $g$ in this list, its position together with $g^{b_{old}}$ uniquely determines $\ell_{old}$.)

The final representation $F_t(x, T)$ of a transformation $T$ for $x$ is then

$$(g^{b_{old}}, \{L_j\}_{j=1}^{\ell_{new}+2b_{new}}, \{(n_{j1}, n_{j2}, n_{j3}, sk_j, R_j, S_j)\}_{j=1}^{\ell_{new}+2b_{new}-1}, (n_{new1}, n_{new2}, n_{new3}), g^{b_{new}}).$$

**Provable statements:** To prove the statement $(x, w) \in R$ for $x = ((pk_1, pk_2), m)$, $m = (\mathsf{nym}_r, \ell, b)$, and $w = sk$, we can prove that

$$e(pk_1, pk_2) = e(sk, g).$$

**Provable transformations:** Given representations for $x = (pk, (\mathsf{nym}_r, \ell, b))$, $x' = (pk', (\mathsf{nym}'_r, \ell', b'))$, and $T$ as follows

$$\begin{aligned}
F_x(x) &= ((pk_1, pk_2), ((n_{r1}, n_{r2}, n_{r3}), g^{\ell}, g^{b})), \\
F_x(x') &= ((pk'_1, pk'_2), ((n'_{r1}, n'_{r2}, n'_{r3}), g^{\ell'}, g^{b'})), \text{ and} \\
F_t(x', T) &= (g^{b'}, \{L_j\}_{j=1}^{\ell+2b}, \{(n_{j1}, n_{j2}, n_{j3}, sk_j, R_j, S_j)\}_{j=1}^{\ell+2b-1}, (n_{new1}, n_{new2}, n_{new3}), g^{b}),
\end{aligned}$$

we prove the statement "$x = T_{\text{inst}}(x')$ and $T \in \mathcal{T}$."

We capture the requirements that $\{n_{newi} = n_{ri}\}_{i=1}^{3}$ by using the values $n_{ri}$ in place of $n_{newi}$ in the equations below. When $b = 1$ we can capture the requirement that $g^{sk_{\ell'+k-2}} = g^{sk_{\ell'+k-1}}$, and $\{n_{(\ell'+k-1)i} = n_{newi}\}_{i=1}^{3}$ by using the same variable to represent each pair of values.

We first prove that $pk_1$ and $pk_2$ are unchanged:

$$e(g^{pk_1}/g^{pk'_1}, g) = 1 \qquad (pk_1)$$

$$e(g^{pk_2}/g^{pk'_2}, g) = 1. \qquad (pk_2)$$

Next, we prove that $\mathsf{NymVerify}(params, \mathsf{nym}_j, sk_j, \mathsf{open}_j) = 1$ using the following set of equations for $j = 1 \ldots \ell + 2b - 1$:

$$e(n_{j1}, g)^{-1} e(R_j, f) = 1 \qquad (\mathsf{nym},1,\mathrm{j})$$

$$e(n_{j2}, g)^{-1} e(S_j, h) = 1 \qquad (\mathsf{nym},2,\mathrm{j})$$

$$e(n_{j3}, g)^{-1} e(R_j, g) \cdot e(S_j, g) \cdot e(sk_j, g) = 1. \qquad (\mathsf{nym},3,\mathrm{j})$$

Finally, we want to prove that $\mathsf{nym}'_r = \mathsf{nym}_{\ell'+2b'}$. We do this as follows: first we prove that $L_j \neq 1$ when $j = \ell' + 2b'$ and that $\mathsf{nym}'_r$ and $\mathsf{nym}_j$ are the same in the position $j$ where $L_j \neq 1$, using the following set of equations for all $j$, $1 \leq j \leq \ell + 2b - 1$:

$$e(L_j, g^{\ell'} (g^{b'})^2 / g^{j}) = 1 \qquad (L,\mathrm{j})$$

$$e(n_{r1'}, L_j)^{-1} e(n_{j1}, L_j) = 1 \qquad (\mathsf{nym}',1,\mathrm{j})$$

$$e(n_{r2'}, L_j)^{-1} e(n_{j2}, L_j) = 1 \qquad (\mathsf{nym}',2,\mathrm{j})$$

$$e(n_{r3'}, L_j)^{-1} e(n_{j3}, L_j) = 1. \qquad (\mathsf{nym}',3,\mathrm{j})$$

To allow for the identity transformation we also have to prove

$$e(L_{\ell+2b}, g^{\ell'} (g^{b'})^2 / g^{\ell+2b}) = 1 \qquad (L,\ell+2b)$$

$$e(n_{r1'}, L_{\ell+2b})^{-1} e(n_{r1}, L_{\ell+2b}) = 1 \qquad (\mathsf{nym}',1,\ell+2b)$$

$$e(n_{r2'}, L_{\ell+2b})^{-1} e(n_{r2}, L_{\ell+2b}) = 1 \qquad (\mathsf{nym}',2,\ell+2b)$$

$$e(n_{r3'}, L_{\ell+2b})^{-1} e(n_{r3}, L_{\ell+2b}) = 1. \qquad (\mathsf{nym}',3,\ell+2b)$$

To prove $\ell' + 2b' \leq \ell + 2b$, we prove that

$$\prod_{j=1}^{\ell+2b} e(L_j, g) = e(g, g). \qquad (\mathsf{sum}, \ell + 2b)$$

The last thing we need to prove is that either $b' = 0$ (which allows arbitrary transformations including the identity transformation) or $b = b' = 1$ and this is the identity transformation (i.e. $\ell' = \ell$, $k = 0$). We do this by proving that if $b' \neq 0$ then $b \neq 0$, and proving that if $b' \neq 0$, then this must be an identity transformation, i.e. $\ell' + 2b' = \ell + 2b$:

$$e(g^{b'}, g^b/g) = 1 \qquad (b)$$
$$e(g^{b'}, L_{\ell+2b}/g) = 1 \qquad (b')$$

**Transformable statements:** To change the statement "$(x = (pk, m), w) \in R$" into "$(\widehat{x}, \widehat{w}) \in R$" for $\widehat{x} = T_{\text{inst}}(x)$ and $\widehat{w} = T_{\text{wit}}(w)$, we observe that $T_{\text{wit}} = \text{id}$ and that $T_{\text{inst}}$ does not change $pk$, meaning we in fact prove the same statement $(pk, sk) \in R_{pk}$ regardless of the transformation.

Finally, to define how transformations can be transformed, we need to ensure that we use *valid* transformations, meaning transformations that can be applied to Groth-Sahai proofs. We use the following four transformations on sets of pairing product equations:

- $\mathsf{AddTriv}(S)$ takes in a set of variables, and adds the equation $e(s, 1) = 1$ for all $s \in S$.
- $\mathsf{ConstToVar}(a, z, S)$ takes in a constant $a$ and changes it to a variable $z$; it then adds the equation $e(a, w)^{-1}e(z, w) = 1$ for all $s \in S$.
- $\mathsf{MergeEq}(i.j)$ adds a new equation that is the product of equations $i$ and $j$.
- $\mathsf{RemoveEq}(i)$ removes the $i$-th equation.

As Chase et al. already showed that these transformations were valid in the above sense, we can use them to transform transformations as follows:

**Transformable transformations:** We show how to transform a proof of the statement "$T_{\text{inst}}(x') = x$" into a proof of the statement "$\widehat{T_{\text{inst}}} \circ T_{\text{inst}}(x') = \hat{x}$" where $\hat{x} = \widehat{T_{\text{inst}}}(x)$.

For $x' = (pk, (\mathsf{nym}'_r, \ell', b'))$, $x = (pk, (\mathsf{nym}_r, \ell, b))$, $\widehat{x} = (pk, (\widehat{\mathsf{nym}}_r, \widehat{\ell}, \widehat{b}))$ we consider only transformations

$$T = (\{\mathsf{nym}_j, sk_j, \mathsf{open}_j\}_{j=1}^{k}, \mathsf{nym}_{new}, b) \text{ where } k = \ell - \ell' + 2(b - b')$$
$$\widehat{T} = (\{\widehat{\mathsf{nym}}_j, \widehat{sk}_j, \widehat{\mathsf{open}}_j\}_{j=1}^{\widehat{k}}, \widehat{\mathsf{nym}}_{new}, \widehat{b}) \text{ where } \widehat{\mathsf{nym}}_1 = \mathsf{nym}_{new} \text{ and } \widehat{k} = \widehat{\ell} - \ell + 2(\widehat{b} - b) \text{ and,}$$
$$\widehat{T} \circ T = (\{\mathsf{nym}_j, sk_j, \mathsf{open}_j\}_{j=1}^{k} \cup \{\widehat{\mathsf{nym}}_j, \widehat{sk}_j, \widehat{\mathsf{open}}_j\}_{j=1}^{\widehat{k}}, \widehat{\mathsf{nym}}_{new}, \widehat{b}) \ .$$

If this is not the case, then the transformations simply produce $\perp$.[7]

To change the statement $T_{\text{inst}}(x') = x$ into $\widehat{T_{\text{inst}}} \circ T_{\text{inst}}(x') = \hat{x}$, we use the same representations for $x$, $x'$, and $T$ as above. Let $\widehat{T}$ be the transformation described above with $(\widehat{n}_{j1}, \widehat{n}_{j2}, \widehat{n}_{j3})$ and $(\widehat{R}_j, \widehat{S}_j)$ for each $j$, and $(\widehat{n}_{new1}, \widehat{n}_{new2}, \widehat{n}_{new3})$ the representations of $\widehat{\mathsf{nym}}_j$, $\widehat{\mathsf{open}}_j$, and $\widehat{\mathsf{nym}}_{new}$ respectively. Let $\hat{x}$ be the result of the transformation, i.e. $\hat{x} = (\widehat{pk}, (\widehat{\mathsf{nym}}_r, \widehat{\ell}, \widehat{b}))$, and let $F_x(\widehat{x}) = ((\widehat{pk}_1, \widehat{pk}_2), ((\widehat{n}_{r1}, \widehat{n}_{r2}, \widehat{n}_{r3}), g^{\widehat{\ell}}, g^{\widehat{b}}))$ be its representation.

[7] $\perp$ is unprovable, or provable only insofar as any proof that doesn't verify constitutes a valid proof for $\perp$.

First add constant equations for $j = 1 \dots \widehat{k}$:

$$e(\widehat{n}_{j1}, g)^{-1} e(\widehat{R}_j, f) = 1 \qquad (\mathsf{nym}, 1, \ell + 2b + j - 1)$$
$$e(\widehat{n}_{j2}, g)^{-1} e(\widehat{S}_j, h) = 1 \qquad (\mathsf{nym}, 2, \ell + 2b + j - 1)$$
$$e(\widehat{n}_{j3}, g)^{-1} e(\widehat{R}_j, g) \cdot e(\widehat{S}_j, g) \cdot e(\widehat{sk}_j, g) = 1 \qquad (\mathsf{nym}, 3, \ell + 2b + j - 1)$$

and for $j = 2 \dots \widehat{k}$:

$$e(n'_{r1}, 1)^{-1} e(\widehat{n}_{j1}, 1) = 1 \qquad (\mathsf{nym}', 1, \ell + 2b + j - 1)$$
$$e(n'_{r2}, 1)^{-1} e(\widehat{n}_{j2}, 1) = 1 \qquad (\mathsf{nym}', 2, \ell + 2b + j - 1)$$
$$e(n'_{r3}, 1)^{-1} e(\widehat{n}_{j3}, 1) = 1 \qquad (\mathsf{nym}', 3, \ell + 2b + j - 1)$$
$$e(1, g) = 1. \qquad (\mathsf{temp}, \ell + 2b + j - 1)$$

where $\widehat{n}_{1i}$ uses the same constant as $n_{ri}$ from $x$, for $i = 1 \dots 3$.

To allow for the identity transformation we add

$$e(n_{r1'}, 1)^{-1} e(\widehat{n}_{r1}, 1) = 1 \qquad (\mathsf{nym}', 1, \ell + 2b + \widehat{k})$$
$$e(n_{r2'}, 1)^{-1} e(\widehat{n}_{r2}, 1) = 1 \qquad (\mathsf{nym}', 2, \ell + 2b + \widehat{k})$$
$$e(n_{r3'}, 1)^{-1} e(\widehat{n}_{r3}, 1) = 1 \qquad (\mathsf{nym}', 3, \ell + 2b + \widehat{k})$$
$$e(1, g) = 1. \qquad (\mathsf{temp}, \ell + 2b + \widehat{k})$$

For $j = \ell + 2b + 1, \dots, \widehat{\ell} + 2\widehat{b}$, use AddTriv to add equations:

$$e(1, g^{\ell'} (g^{b'})^2 / g^j) = 1 \qquad (L, j)$$

(Note that AddTriv is constructed so that this can be done even without knowing the values $g^{\ell'}$, $g^{b'}$ used in the previous proof.)

Next, use ConstToVar to replace 1 in all the new $(\mathsf{nym}', i, j)$, $(\mathsf{temp}, j)$, and $(L, j)$ equations with $L_j$ to obtain the following for $j = \ell + 2b - 1 \dots \widehat{\ell} + 2\widehat{b} - 1$:

$$e(L_j, g^{\ell'} (g^{b'})^2 / g^j) = 1 \qquad (L, j)$$
$$e(n'_{r1}, L_j)^{-1} e(\widehat{n}_{j1}, L_j) = 1 \qquad (\mathsf{nym}', 1, j)$$
$$e(n'_{r2}, L_j)^{-1} e(\widehat{n}_{j2}, L_j) = 1 \qquad (\mathsf{nym}', 2, j)$$
$$e(n'_{r3}, L_j)^{-1} e(\widehat{n}_{j3}, L_j) = 1 \qquad (\mathsf{nym}', 3, j)$$
$$e(L_j, g) = 1 \qquad (\mathsf{temp}, j)$$

and replace 1 with $L_{\widehat{\ell}+2\widehat{b}}$ in the $(\mathsf{nym}', i, \widehat{\ell} + 2\widehat{b})$, $(\mathsf{temp}, \widehat{\ell} + 2\widehat{b})$, and $(L, \widehat{\ell} + 2\widehat{b})$ equations to obtain:

$$e(L_{\widehat{\ell}+2\widehat{b}}, g^{\ell'} (g^{b'})^2 / g^{\widehat{\ell}+2\widehat{b}}) = 1 \qquad (L, \widehat{\ell} + 2\widehat{b})$$
$$e(n_{r1'}, L_{\widehat{\ell}+2\widehat{b}})^{-1} e(\widehat{n}_{r1}, L_{\widehat{\ell}+2\widehat{b}}) = 1 \qquad (\mathsf{nym}', 1, \widehat{\ell} + 2\widehat{b})$$
$$e(n_{r2'}, L_{\widehat{\ell}+2\widehat{b}})^{-1} e(\widehat{n}_{r2}, L_{\widehat{\ell}+2\widehat{b}}) = 1 \qquad (\mathsf{nym}', 2, \widehat{\ell} + 2\widehat{b})$$
$$e(n_{r3'}, L_{\widehat{\ell}+2\widehat{b}})^{-1} e(\widehat{n}_{r3}, L_{\widehat{\ell}+2\widehat{b}}) = 1 \qquad (\mathsf{nym}', 3, \widehat{\ell} + 2\widehat{b})$$
$$e(L_{\widehat{\ell}+2\widehat{b}}, g) = 1. \qquad (\mathsf{temp}, \widehat{\ell} + 2\widehat{b})$$

Then use ConstToVar to replace $\widehat{n}_{ji}$ with a variable in all equations for $j = \ell + 2b \ldots \widehat{\ell} + 2\widehat{b} - 1$. Use ConstToVar to replace $\widehat{sk}_j$ with a variable in all equations for $j = \ell + 2b \ldots \widehat{\ell} + 2\widehat{b} - 1$. If $\widehat{b} = 1$, use the same variable for $\widehat{sk}_{\widehat{\ell}}$ and $sk_{\widehat{\ell}+1}$.

Use MergeEq to combine (sum, $\ell + 2b$) with (temp,$\ell + 2b + 1$), ..., (temp,$\widehat{\ell} + 2\widehat{b}$) to obtain:

$$\prod_{j=1}^{\widehat{\ell}+2\widehat{b}} e(L_j, g) = e(g, g) \qquad (\mathsf{sum},\widehat{\ell} + 2\widehat{b})$$

and RemoveEq to remove (sum,$\ell + 2b$) and (temp,$\ell + 2b = 1$), ..., (temp,$\widehat{\ell} + 2\widehat{b}$).

Finally, if $b = 1$ then $\widehat{T}$ must be the identity transform, so $\widehat{b} = b$ and $\widehat{\ell}+2\widehat{b} = \ell+2b$. Thus the equations $(b)$ and $(b')$ remain exactly the same.

$$e(g^{b'}, g^{\widehat{b}}/g) = 1 \qquad (b)$$

$$e(g^{b'}, L_{\widehat{\ell}+2\widehat{b}}/g) = 1. \qquad (b')$$

Similarly if $b = 0$ and $k = 0$ the equations $(b)$ and $(b')$ remain exactly the same.

If $b = 0$, then equation $(b)$ gives us $e(g^{b'}, 1/g) = 1$. We can make a copy of this equation, and include it when we use the ConstToVar for variable $L_{\widehat{\ell}+2\widehat{b}}$ above so that we get

$$e(g^{b'}, L_{\widehat{\ell}+2\widehat{b}}/g) = 1. \qquad (b')$$

For the equation $(b)$, if $\widehat{b} = b = 0$, then we can use exactly the same equation. If $b = 0$ and $\widehat{b} = 1$, then we can use AddTriv to add the equation $e(g^{b'}, 1) = 1$, which since $\widehat{b} = 1$ is equivalent to

$$e(g^{b'}, g^{\widehat{b}}/g) = 1. \qquad (b)$$

Note that all the aliasing equations will still be trivially verifiable because $\widehat{n}_{newi}, \widehat{n}_{ri}, g^{\widehat{b}_{new}}$ are in the clear, and when $\widehat{b}_{new} = 1$ $sk_{\widehat{\ell}}$ and $sk_{\widehat{\ell}+1}$ use the same variable and $\widehat{n}_{(\widehat{\ell}+1)i}$ and $\widehat{n}_{ri}$ use the same constant.