On-line Non-transferable Signatures Revisited

Jacob C. N. Schuldt[1] and Kanta Matsuura[2]

[1] Research Center for Information Security,
National Institute of Advanced Industrial Science and Technology, Japan
`jacob.schuldt@aist.go.jp`
[2] Institute of Industrial Science, The University of Tokyo, Japan
`kanta@iis.u-tokyo.ac.jp`

Abstract. Undeniable signatures, introduced by Chaum and van Antwerpen, and designated confirmer signatures, introduced by Chaum, allow a signer to control the verifiability of his signatures by requiring a verifier to interact with the signer to verify a signature. An important security requirement for these types of signature schemes is *non-transferability* which informally guarantees that even though a verifier has confirmed the validity of a signature by interacting with the signer, he cannot prove this knowledge to a third party. Recently Liskov and Micali pointed out that the commonly used notion of non-transferability only guarantees security against an off-line attacker which cannot influence the verifier while he interacts with the signer, and that almost all previous schemes relying on interactive protocols are vulnerable to on-line attacks. To address this, Liskov and Micali formalized on-line non-transferable signatures which are resistant to on-line attacks, and proposed a generic construction based on a standard signature scheme and an encryption scheme.

In this paper, we revisit on-line non-transferable signatures. Firstly, we extend the security model of Liskov and Micali to cover not only the sign protocol, but also the confirm and disavow protocols executed by the confirmer. Our security model furthermore considers the use of multiple (potentially corrupted or malicious) confirmers, and guarantees security against attacks related to the use of signer specific confirmer keys. We then present a new approach to the construction of on-line non-transferable signatures, and propose a new concrete construction which is provably secure in the standard model. Unlike the construction by Liskov and Micali, our construction does not require the signer to issue "fake" signatures to maintain security, and allows the confirmer to both confirm and disavow signatures. Lastly, our construction provides noticeably shorter signatures than the construction by Liskov and Micali.

Keywords: non-transferable signatures, standard model, provable security.

1 Introduction

An ordinary signature scheme provides public verifiability i.e. anyone is able to verify the validity of a given signature using the public key of the signer. While this property is useful in many scenarios, it might not always be desirable. For example, a signer who signs a sensitive message might prefer to be able to control who can verify the validity of his signature. Chaum *et al.* [10] addressed this problem with their proposal of undeniable signatures in which a verifier is required to interact with the signer to verify a signature. Furthermore, to preserve non-repudiation, the signer is also able to prove invalidity of a signature through a disavow protocol. Hence, in a dispute, the signer will either be able to confirm or disavow a purported signature. However, in some scenarios, a signer might become unavailable or might refuse to cooperate with a verifier, in which case the validity of a signature cannot be determined. To address this, Chaum [9] introduced designated confirmer signatures in which a third party, the confirmer, can interact with a verifier to confirm or disavow a signature on behalf of the signer. Furthermore, the confirmer can, in the case of a dispute, extract a publicly verifiable signature (of the signer) from a valid designated confirmer signature. Since their introduction, a number of undeniable schemes [8, 16, 15, 25, 21, 22] and designated confirmer schemes [27, 4, 18, 17, 36] have been proposed.

Off-line and On-line Non-transferability. An important security notion for these types of signature schemes is non-transferability. Intuitively, non-transferability guarantees that once a verifier has verified a signature and is convinced about its validity, he cannot transfer this conviction to a third party. This is achieved by ensuring that a verifier is able to simulate a transcript of the interaction with the signer/confirmer i.e. any "evidence" of validity obtained through the interaction, could have been generated by the verifier himself. A scheme providing this property is said to be *off-line non-transferable*. However, Liskov and Micali [26] pointed out that almost all[3] previous schemes relying on interactive protocols to provide off-line non-transferability are vulnerable to *on-line* attacks, i.e. an attacker who is present *while* the verifier interacts with a signer/confirmer might be able to determine the validity of a signature by influencing messages sent by the verifier. A scheme preventing these types of attacks is said to be *on-line non-transferable* and is constructed by enabling the verifier to *interactively* simulate the interaction with a signer/confirmer. To preserve soundness of the scheme, only the verifier should be able to simulate a proof, and to facilitate this, Liskov and Micali [26] assumed the verifier holds a public/private key pair i.e. to simulate the interaction between the signer/confirmer and a verifier, the private key of the verifier is required. Note that this approach to on-line non-transferability requires that the verifier knows the private key corresponding to his public key to maintain security. More specifically, if it is possible for a verifier to convince a third party that he does not know his private key (e.g. by generating his public key by applying a hash function to a random seed $pk_V = H(x)$, and then presenting x to the third party), the scheme will no longer provide on-line non-transferability. To prevent this type of malicious behavior, verifier key registration is required i.e. a verifier should prove knowledge of his private key when registering his public key (see [26] for further discussion of this). In this paper, we adopt the same general approach as [26], assume verifiers are equipped with public/private key pairs, and will furthermore explicitly model verifier key registration in our security model[4].

In [26], Liskov and Micali illustrated the feasibility of constructing an on-line non-transferable signature scheme under the above described assumption of verifier key registration. More specifically, they proposed a generic construction based on an `ind-cpa` secure public key encryption scheme and a `uf-cma` secure signature scheme. The resulting scheme provides on-line non-transferability of an interactive sign protocol through which the signer both constructs and proves validity of a signature. Furthermore, the scheme supports the use of confirmers and is proved secure in the standard model. However, to achieve on-line non-transferability, a signer has to be willing to issue "fake" signatures to anyone requesting them. This is essential since a verifier will not be able to simulate the sign protocol without the ability to ask the signer for fake signatures. This drawback limits the practical applicability of the scheme. Furthermore, the functionality and security guarantees of the confirmer are somewhat limited. More specifically, a confirmer can disavow but not confirm the validity of a signature, and neither off-line nor on-line non-transferability are considered for the disavow protocol[5].

Our Contribution. In this paper, we address many of the limitations of the approach by Liskov and Micali. Firstly, we extend the security model to model not only the on-line non-transferability of the sign protocol, but also of the confirm and disavow protocols executed by the confirmer. Furthermore, we introduce two additional security notions, confirmer soundness and key unforgeability, required by the added ability of the confirmer to confirm signatures and to prevent attacks related to the forgery of signer specific confirmer keys which are used both in our construction and in [26] (see Section 4 for details). Unlike [26], our security model also allows the signer to make use of multiple confirmers and ensures unforgeability even against malicious confirmers, which will guarantee security in a more realistic usage scenario.

---

[3] See *Related Work* below for a few exceptions in the random oracle model

[4] Note that while the security definitions in [26] does not explicitly describe verifier key registration, this *is* a requirement to ensure basic security, and we argue that our security models are fundamentally the same.

[5] The defined disavow protocol in [26] is non-interactive and provides a publicly verifiable proof of invalidity

We then propose a new general approach to the construction of on-line non-transferable signatures. More specifically, we show how a simple *core confirmer signature scheme*, which essentially implements the non-interactive functionality of an on-line non-transferable signature scheme, can be extended to a fully secure scheme with the additional use of ordinary signatures, sigma protocols, and trapdoor commitments with an enhanced binding property. Based on this approach, we propose a concrete instantiation which is provably secure in the standard model assuming the computational Diffie-Hellman problem and the decisional linear problem are hard.

Compared to the approach taken by Liskov and Micali, our scheme has several advantages. Besides implementing additional confirmer functionality and providing security in our extended security model, our scheme allows a verifier to independently simulate the sign, confirm and disavow protocols, and does not require the signer to issue "fake" signatures to maintain security. Lastly, our concrete instantiation provides efficient protocols and short signatures consisting of four group elements and an integer, whereas the scheme by Liskov and Micali requires signatures consisting of more than $3k$ encryptions, where k is the security parameter. However, we note that our concrete scheme requires large public keys due to the use of the techniques by Waters [35].

Related Work. Jakobsson *et al.* [20] introduced an alternative approach to limiting the verifiability of signatures with their proposal of designated verifier signatures in which only a specific verifier chosen by the signer will be convinced about the validity of a signature. This concept was extended by Steinfeld *et al.* [32] who introduced universal designated verifier signatures which allow any user (i.e. not only the signer) to convert a publicly verifiable signature into a designated verifier signature for a chosen verifier. Since this type of schemes do not rely on interactive protocols for signature confirmation, on-line attacks are not a concern. However, these schemes do not provide a mechanism to determine the validity of a (converted) signature in a dispute. In fact, most of the proposed concrete schemes (e.g. [24, 33, 23]) enable the designated verifier to construct signatures which are perfectly indistinguishable from signatures constructed by the signer. Hence, unlike undeniable and designated confirmer signatures, non-repudiation cannot be enforced in these schemes which make them unsuitable for a number of applications.

A few existing schemes, which are provably secure in the random oracle model, implicitly provide protection against on-line attacks. For example, the undeniable signature schemes by Kudla *et al.* [21] and Huang *et al.* [19] provide non-interactive proofs which are simulatable by the verifier, and hence avoid the problem of on-line attacks. Note that an undeniable signature scheme with non-interactive proofs is different from a designated verifier signature scheme in that a signature is independent of the verifier(s) and that the signer is able to disavow a signature. Furthermore, Monnerat *et al.* [29] proposed an undeniable signature scheme which uses interactive 2-move confirm and disavow protocols and requires verifiers to hold a public/private key pair. While the used definition of non-transferability in [29] only guarantees transcript simulatability (i.e. defines off-line non-transferability), the concrete scheme allows a verifier to use his private key to simulate proofs interactively, and hence the scheme provides on-line non-transferability. However, we emphasize that all of the above schemes are only provable secure in the random oracle model, and that the schemes furthermore do not support the use of confirmers to ensure non-repudiation if the signer becomes off-line or refuses to cooperate.

2 Preliminaries

In this section, we will define the computational problems underlying our concrete instantiation of a on-line non-transferable signature scheme, as well as a introduce the basic primitives which will be used in our general construction.

Negligible function. A function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ is said to be *negligible* if for all $c > 0$ there exists an k_c such that for all $k > k_c$ $\epsilon(k) < 1/k^c$.

The discrete logarithm problem. Let $\mathcal{G}$ be a group generator which given a security parameter k, outputs a group $\mathbb{G}$ of prime order p, where $2^k < p < 2^{k+1}$, and a generator g. The advantage of an adversary against the discrete logarithm problem is defined as follows:

$$\mathtt{Adv}_{\mathcal{G},\mathcal{A}}^{\mathtt{dl}}(k) = \Pr[(\mathbb{G},p,g) \leftarrow \mathcal{G}(k); h \leftarrow \mathbb{G}; x \leftarrow \mathcal{A}(\mathbb{G},p,g,h) : g^x = h]$$

Definition 1 *The discrete logarithm problem is said to be* hard *with respect to $\mathcal{G}$, if all polynomial time adversaries $\mathcal{A}$ have negligible advantage $\mathtt{Adv}_{\mathcal{G},\mathcal{A}}^{dl}(k)$.*

The decisional linear problem. This decisional problem was introduced by Boneh *et al.* in [1], and is defined follows. Let the advantage of an adversary $\mathcal{A}$ against the decisional linear problem with respect to a group generator $\mathcal{G}$ be given as:

$$\mathtt{Adv}_{\mathcal{G},\mathcal{A}}^{\mathtt{dlin}}(k) = |\Pr[(\mathbb{G},p,g) \leftarrow \mathcal{G}(k); u,v \leftarrow \mathbb{G}; a,b \leftarrow \mathbb{Z}_p : \mathcal{A}(\mathbb{G},p,g,u^a,v^b,g^{a+b}) = 1]-$$
$$\Pr[(\mathbb{G},p,g) \leftarrow \mathcal{G}(k); u,v \leftarrow \mathbb{G}; a,b,c \leftarrow \mathbb{Z}_p : \mathcal{A}(\mathbb{G},p,g,u^a,v^b,g^c) = 1]|$$

Definition 2 *The decisional linear problem is said to be* hard *with respect to $\mathcal{G}$ if all polynomial time algorithms $\mathcal{A}$ have negligible advantage $\mathtt{Adv}_{\mathcal{G},\mathcal{A}}^{dlin}(k)$.*

In [1], the decisional linear problem is shown to be computationally infeasible in the generic bilinear group model.

Collision resistant hash function. Let $\mathcal{H} = \{H_\nu : \{0,1\}^* \rightarrow \{0,1\}^{|\nu|}\}$ be a hash function family indexed by a key $\nu \in \{0,1\}^*$. The advantage of an adversary $\mathcal{A}$ against the collision resistance of $\mathcal{H}$ is defined as:

$$\mathtt{Adv}_{\mathcal{H},\mathcal{A}}^{\mathtt{col}}(k) = \Pr[\nu \leftarrow \{0,1\}^k; (m,m') \leftarrow \mathcal{A}(\nu) : H_\nu(m) = H_\nu(m') \wedge m \neq m']$$

Definition 3 *A hash function family $\mathcal{H}$ is said to be collision resistant if all polynomial time algorithms $\mathcal{A}$ have negligible advantage $\mathtt{Adv}_{\mathcal{H},\mathcal{A}}^{col}(k)$ against $\mathcal{H}$.*

Signatures. A standard signature scheme is given by the following four algorithms: `Setup` which takes as input a security parameter 1^k, and returns a set of public parameters *par*; `KeyGen` which takes as input *par*, and returns a public/private key pair (pk, sk); `Sign` which takes as input *par*, a private key *sk* and a message *m*, and returns a signature σ; and lastly `Verify` which takes as input *par*, a public key *pk*, a message *m* and a signature σ, and returns $\top$ if σ is a valid signature on *m* under the public key *pk* and $\bot$ otherwise.

We require that a signature scheme is *correct* i.e. for all $par \leftarrow \mathtt{Setup}(1^k)$, all $(pk, sk) \leftarrow \mathtt{KG}(par)$, and all messages *m*, it is required that $\mathtt{SVer}(par, pk, m, \mathtt{Sign}(par, sk, m)) = \top$.

Strong unforgeability against a chosen message attack (`suf-cma`) for a signature scheme S is defined as follows: Let the advantage of an adversary $\mathcal{A}$ against S be given by

$$\mathtt{Adv}_{S,\mathcal{A}}^{\mathtt{uf-cma}}(k) = \Pr[par \leftarrow \mathtt{Setup}(1^k); (pk,sk) \leftarrow \mathtt{KeyGen}(par); (m^*,\sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{sig}}(par,pk) :$$
$$(m^*,\sigma^*) \notin \{(m_1,\sigma_1),\ldots,(m_q,\sigma_q)\} \wedge \mathtt{Verify}(par,pk,m^*,\sigma^*) = \top]$$

where $\mathcal{O}_{sig}$ is a sign oracle which given m_i returns $\sigma_i \leftarrow \mathtt{Sign}(par, sk, m_i)$, q is the number of queries made by $\mathcal{A}$ to $\mathcal{O}_{sig}$, and $\{(m_1, \sigma_1), \ldots, (m_q, \sigma_q)\}$ is the list of messages/signature pairs which $\mathcal{A}$ obtained from $\mathcal{O}_{sig}$.

Definition 4 *A signature scheme S is said to be `suf-cma` secure if all polynomial time adversaries $\mathcal{A}$ have negligible advantage $\mathtt{Adv}_{S,\mathcal{A}}^{uf-cma}(k)$ against S.*

Weak unforgeability against a chosen message attack (`wuf-cma`) is defined exactly as above, except that the requirement that $(m^*, \sigma^*) \notin \{(m_1, \sigma_1), \ldots, (m_q, \sigma_q)\}$ is relaxed to $m^* \notin \{m_1, \ldots, m_q\}$ i.e. for $\mathcal{A}$ to successfully attack the scheme, it is required that m^* was not previously submitted to the sign oracle.

We will now recall a well-known signature scheme by Waters [35]. This scheme will play an important role in the construction of our concrete instantiation of a on-line non-transferable signature scheme.

- **Setup**: Pick a group $\mathbb{G}$ of primer order p and equipped with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Furthermore, pick generator g of $\mathbb{G}$ and return the parameters $par \leftarrow (\mathbb{G}, p, g, e)$.
- **KeyGen** : Given par, pick $\alpha \leftarrow \mathbb{Z}_p$ and $h \leftarrow \mathbb{G}$, and set $g' \leftarrow g^\alpha$. Furthermore, pick $u_0, \ldots, u_n \leftarrow \mathbb{G}$, and for a message $m \in \{0,1\}^n$, define $F(m) = u_0 \prod_{i=1}^n u_i^{m_i}$ where m_i is the ith bit of m. Finally set the public key to $pk \leftarrow (g', h, u_0, \ldots, u_n)$ and the private key to $sk \leftarrow \alpha$. Return (pk, sk).
- **Sign** : Given input (par, sk, m), where $sk = \alpha$, pick $r \leftarrow \mathbb{Z}_p$, compute $\sigma_1 \leftarrow g^r$ and $\sigma_2 \leftarrow h^\alpha F(m)^r$, and return the signature $\sigma = (\sigma_1, \sigma_2)$.
- **Ver** : Given par, a public key $pk = (g', h, u_0, \ldots, u_n)$, a message m and a signature $\sigma = (\sigma_1, \sigma_2)$, return **accept** if $e(g, \sigma_2) = e(g', h)e(\sigma_1, F(m))$.

In [35], the above signature scheme is shown to be `wuf-cma` secure given that the computational Diffie-Hellman problem is hard in $\mathbb{G}$.

Sigma protocols. A sigma protocol for a binary relation R is a 3-move protocol between a prover and a verifier. Both prover and verifier receive a common input x, but the prover receives a witness w such that $(x, w) \in R$ as an additional private input. In the first move of the protocol, the prover sends a "commitment" message a, in the second move, the verifier sends a random "challenge" message c, and in the final move, the prover sends a "response" message z. Given the response message, the verifier either accepts or rejects the proof. We use the notation $\Sigma\{(x, w) : R(x, w) = 1\}$ to denote a sigma protocol for the relation R with common input x and witness w. A sigma protocol is required to have two security properties:

- *Special honest verifier zero-knowledge:* There exists a simulation algorithm $\mathtt{Sim}_\Sigma$ that given input x and a challenge message c, outputs an accepting transcript $(a, c, z) \leftarrow \mathtt{Sim}_\Sigma(x, c)$. We require that the distribution of the simulated (a, c, z) is perfectly indistinguishable from the distribution of the transcripts of a real interaction.
- *Special soundness:* There exists an algorithm $\mathtt{WExt}_\Sigma$ that, given two accepting transcripts, (a, c, z) and (a, c', z'), for input x which have the same commitment message a but different challenge messages $c \neq c'$, can extract a witness w such that $(x, w) \in R$.

Trapdoor commitment schemes. A trapdoor commitment scheme $T = \{\mathtt{G}, \mathtt{Comm}, \mathtt{TdComm}, \mathtt{TdOpen}\}$ is given by a generation algorithm $\mathtt{G}$ which, given a security parameter 1^k, returns a commitment key ck and a trapdoor td; a deterministic commitment algorithm $\mathtt{Comm}$ which, given ck, a value $w \in \mathcal{W}$ and randomness $r \in \mathcal{R}$, returns a commitment com on w (an opening of com is simply (w, r), and a verifier checks that $com = \mathtt{Comm}(ck, w, r)$); a trapdoor commitment algorithm $\mathtt{TdComm}$ that, given ck, returns a commitment com' and auxiliary information aux such that the trapdoor opening algorithm $\mathtt{TdOpen}$, given aux, any value w' and the trapdoor td, returns r' such that $com' = \mathtt{Comm}(ck, w', r')$. We consider the following security properties for a trapdoor commitment scheme:

- *Computational binding:* For $(ck, td) \leftarrow \mathtt{G}(1^k)$, the probability that any computationally bounded adversary given ck can compute (w, r, w', r') such that $w \neq w'$ and $\mathtt{Comm}(ck, w, r) = \mathtt{Comm}(ck, w', r)$, is negligible in the security parameter k.
- *Perfect hiding:* For $(ck, td) \leftarrow \mathtt{G}(1^k)$, random $r, r' \leftarrow \mathcal{R}$, and for any $w, w' \in \mathcal{W}$, the commitments $\mathtt{Comm}(ck, w, r)$ and $\mathtt{Comm}(ck, w', r')$ are distributed identically.
- *(Perfect) trapdoor property:* For $(ck, td) \leftarrow \mathtt{G}(1^k)$, any $w \in \mathcal{W}$, an honestly computed commitment $com \leftarrow \mathtt{Comm}(ck, w, r)$ where $r \leftarrow \mathcal{R}$, and a commitment computed using the trapdoor $(com', aux) \leftarrow \mathtt{TdComm}(ck)$ and $r' \leftarrow \mathtt{TdOpen}(aux, w, td)$, the values (com, r) and (com', r') are distributed identically.

Note that a perfect trapdoor property implies perfect hiding, since an honestly computed commitment is indistinguishable from a commitment computed using $\mathtt{TdComm}$, and the latter can be opened to any value.

3 On-line Non-transferable Signatures

An on-line non-transferable signature (ONS) scheme involves a signer S, a confirmer C, and a verifier V, and is given by the following algorithms

- Setup which, given a security parameter 1^k, returns the public parameters par.
- KeyGen_S, KeyGen_C, and KeyGen_V which, given par, return public/private key pairs (pk_S, sk_S), (pk_C, sk_C), and (pk_V, sk_V) for a signer, a confirmer, and a verifier, respectively.
- CSetup which on input par, sk_C and pk_S, returns a signer specific public/private confirmer key pair $(pk_{C,S}, sk_{C,S})$. This algorithm is run once by the confirmer for each signer S, and the confirmer stores $sk_{C,S}$ for later use. The public key $pk_{C,S}$ is given to the signer who is to use this when constructing signatures with confirmer C.
- CKeyValid which, on input par, pk_S, pk_C, and $pk_{C,S}$, outputs either accept or reject.
- $(\text{Sign}, \text{Receive})$ which is a pair of interactive algorithms with common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$. Sign is run by the signer and is given sk_S as private input, and Receive is run by the verifier. At the end of the interaction, both Sign and Receive will output a signature, σ_S and σ_R, and Receive will in addition output either accept or reject.
- Convert which, on input par, pk_S, $sk_{C,S}$, m, and σ, returns a verification token tk_σ.
- TkVerify which, on input par, pk_S, pk_C, $pk_{C,S}$, m, σ, and tk_σ, returns either accept or reject.
- $(\text{Confirm}, \text{V}_C)$ which is a pair of interactive algorithms with common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma)$. Confirm is run by the confirmer and is given $sk_{C,S}$ as private input, and V_C is run by the verifier. At the end of the interaction, V_C outputs either accept or reject.
- $(\text{Disavow}, \text{V}_D)$ which is also a pair of interactive algorithms. Input for Disavow and V_D is exactly as in $(\text{Confirm}, \text{V}_C)$ above, and the output of V_D is either accept or reject.

Like Liskov and Micali [26], we require that before signer S makes use of a confirmer C, he will approach C to obtain a signer specific confirmer key $pk_{C,S}$ which C generates by running CSetup. This process can be seen as a registration procedure in which the confirmer agrees to act as a confirmer for this specific signer. Note that this does not require a confidential channel between the signer and confirmer. Our definition differs slightly from that of [26] in that we explicitly define a key validation algorithm CKeyValid for signer specific confirmer keys[6], and introduce $(\text{Confirm}, V_C)$ to allow C to confirm signatures. Furthermore, we do not include a fake signature algorithm which is required to maintain the security of the scheme in [26]. Lastly, our definition allows a confirmer to *convert* a signature as opposed to *extract* an ordinary publicly verifiable signature of the signer. While a conversion of a message/signature pair (m, σ) will produce a token tk_σ which will allow anyone to verify the validity of (m, σ), an extraction will produce another signature σ' (of the signer) which is publicly verifiable, but does not necessarily provide any confirmation of the validity of the original signature σ. Hence, conversion essentially provides the same amount of information as the confirm protocol, whereas extraction only guarantees that the signer at some point signed the message m.

Using the above defined algorithms, a confirmer can verify a signature by first computing a verification token using Convert and then verifying the signature using TkVerify. To simplify notation, we define an algorithm Valid which performs these two steps:

- Valid: given the input $(par, pk_S, pk_C, pk_{C,S}, m, \sigma, sk_{C,S})$, compute the verification token $tk_\sigma \leftarrow$ $\text{Convert}(par, pk_S, m, \sigma, sk_{C,S})$ and return $\text{TkVerify}(par, pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma)$.

We will use the notation $\{\text{Sign}(sk_S) \leftrightarrow \text{Receive}\}(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$ to denote the interaction between Sign and Receive on common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$ and private input sk_S to the Sign algorithm. To shorten the common input, we will sometimes use $PK = (pk_S, pk_C, pk_{C,S}, pk_V)$ to represent the public keys i.e. the above common input might be written as (par, PK, m). We furthermore use $(\sigma_S, (\sigma_R, z)) \leftarrow \{\text{Sign}(sk_S) \leftrightarrow \text{Receive}\}(par, PK, m)$

---

[6] This algorithm is required by our extended security model. More specifically, it is required to define key unforgeability (see Section 4).

to denote the output of Sign and Receive, respectively, and use $(\sigma_R, z) \leftarrow_2 \{\texttt{Sign}(sk_S) \leftrightarrow \texttt{Receive}\}(par, PK, m)$ when we are only considering the output of Receive. Similar notation is used for the confirm and disavow protocols.

Correctness. Intuitively, correctness simply requires that if all parties behave honestly, the outcome of all algorithms and protocols are as expected. More specifically, correctness requires that for all $par \leftarrow \texttt{Setup}(1^k)$, $(pk_S, sk_S) \leftarrow \texttt{KeyGen}_S(par)$, $(pk_C, sk_C) \leftarrow \texttt{KeyGen}_C(par)$, $(pk_{C,S}, sk_{C,S}) \leftarrow \texttt{CSetup}(par, pk_S, sk_C)$, $(pk_V, sk_V) \leftarrow \texttt{KeyGen}_V(par)$, $PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$, all messages m, and all $(\sigma_S, (\sigma_R, z_R)) \leftarrow_2 \{\texttt{Sign}(sk_S) \leftrightarrow \texttt{Receive}\}(par, PK, m)$, that

- $\texttt{CKeyValid}(par, pk_S, pk_C, pk_{C,S}) = \texttt{accept}$
- $z_R = \texttt{accept}$ and $\sigma_R = \sigma_S$
- $\texttt{Valid}(par, pk_S, pk_C, pk_{C,S}, sk_{C,S}, m, \sigma) = \texttt{accept}$
- $z_C \leftarrow \{\texttt{Confirm}(sk_{C,S}) \leftrightarrow V_C\}(par, PK, m, \sigma)$ yields $z_C = \texttt{accept}$

Furthermore, for all (m', σ') such that $\texttt{Valid}(par, pk_S, pk_C, pk_{C,S}, sk_{C,S}, m', \sigma') = \texttt{reject}$, we require that $z_D \leftarrow_2 \{\texttt{Disavow}(sk_{C,S}) \leftrightarrow V_D\}(par, PK, m', \sigma')$ yields $z_D = \texttt{accept}$.

4 Security Model

An ONS scheme has to satisfy a number of security requirements to be considered secure. These are *unforgeability*, *key unforgeability*, *soundness*, *non-repudiation* and *non-transferability*, which will be defined in the following. However, before we can formally define these security notions, we require a ONS scheme to implement the following verifier simulation algorithms:

- $\texttt{SimSign}(par, pk_S, pk_C, pk_{C,S}, pk_V, m, sk_V)$: Simulates the Sign algorithm.
- $\texttt{SimCon}(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma, sk_V)$: Simulates the Confirm algorithm.
- $\texttt{SimDis}(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma, sk_V)$: Simulates the Disavow algorithm.

While these algorithms are not part of the basic functionality of an ONS scheme, they must be defined to ensure that a verifier can simulate the interactive protocols of the scheme as required by the non-transferability notion defined below (note that all simulation algorithms require the private key of the verifier as input). Furthermore, since an adversary might observe the execution of these algorithms while attempting to mount attacks against other security properties of the scheme, we must provide the adversary with oracle access to these algorithms in the relevant security definitions.

Unforgeability. Intuitively, unforgeability guarantees that only the signer should be able to produce valid signatures. We define a strong notion of unforgeability requiring that, even for a maliciously chosen confirmer key, an adversary with oracle access to an honest signer cannot produce a new message/signature pair and convince a verifier about the validity of this pair, either by interacting with the verifier in the confirm protocol or by producing a token such that TkVerify outputs accept. Our definition allows the adversary to obtain signatures using any confirmer key, and thereby ensures security in a scenario where a signer makes use of multiple potentially malicious confirmers. In comparison, the unforgeability notion defined by Liskov and Micali only considers a signer using a single honest confirmer. Formally, we define unforgeability of an ONS scheme N via the experiment $\texttt{Exp}_{N,\mathcal{A}}^{\texttt{uf-cma}}$ shown in Figure 4. In the experiment, $\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathcal{O}_{VKeyReg}, \mathcal{O}_{Sign}, \mathcal{O}_{SimSign}, \mathcal{O}_{SimCon}, \mathcal{O}_{SimDis}\}$ which are defined as below. The oracle $\mathcal{O}_{VKeyReg}$ implements verifier key registration and maintains a list, $L_{VKeyReg}$, of registered keys. It is assumed that it can be verified that a key pair (pk_V, sk_V) is valid i.e. that (pk_V, sk_V) lies in the range of $\texttt{KeyGen}_V$[7].

---

[7] Note that this can be achieved for any scheme by including the randomness used to generate (pk_V, sk_V) in the private key sk_V.

$\mathrm{Exp}_{S,\mathcal{A}}^{\mathtt{uf\text{-}cma}}(1^k)$
$\quad L_{Sign} \leftarrow \{\}; L_{VKeyReg} \leftarrow \{\}$
$\quad par \leftarrow \mathtt{Setup}(1^k)$
$\quad (pk_S, sk_S) \leftarrow \mathtt{KeyGen}_S(par)$
$\quad (pk_V, sk_V) \leftarrow \mathtt{KeyGen}_V(par)$
$\quad (pk_C, pk_{C,S}, m, \sigma, tk_\sigma, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_S, pk_V)$
$\quad PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$
$\quad z \leftarrow_2 \{\mathcal{A}^{\mathcal{O}}(st) \leftrightarrow V_C(par, PK, m, \sigma)\}$
$\quad z' \leftarrow \mathtt{TkVerify}(par, pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma)$
$\quad \text{if } (pk_C, pk_{C,S}, m, \sigma) \notin L_{Sign} \wedge$
$\quad\quad (z = \mathtt{accept} \vee z' = \mathtt{accept})$
$\quad\quad \text{output } 1$
$\quad \text{else output } 0$

$\mathrm{Exp}_{S,\mathcal{A}}^{\mathtt{key\text{-}uf}}(1^k)$
$\quad L_{CSetup} \leftarrow \{\}; L_{VKeyReg} \leftarrow \{\}$
$\quad par \leftarrow \mathtt{Setup}(1^k)$
$\quad (pk_C, sk_C) \leftarrow \mathtt{KeyGen}_S(par)$
$\quad (pk_S, pk_{C,S}) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_C)$
$\quad z \leftarrow \mathtt{CKeyValid}(par, pk_S, pk_C, pk_{C,S})$
$\quad \text{if } (pk_S, pk_{C,S}, *) \notin L_{CSetup} \wedge z = \mathtt{accept}$
$\quad\quad \text{output } 1$
$\quad \text{else output } 0$

$\mathrm{Exp}_{S,\mathcal{A}}^{\mathtt{non\text{-}rep}}(1^k)$
$\quad par \leftarrow \mathtt{Setup}(1^k)$
$\quad (pk_V, sk_V) \leftarrow \mathtt{KeyGen}_V(par)$
$\quad (pk_S, pk_C, pk_{C,S}, m, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_V)$
$\quad PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$
$\quad (st', (\sigma, z_1)) \leftarrow \{\mathcal{A}^{\mathcal{O}}(st) \leftrightarrow \mathtt{Receive}(par, PK, m)\}$
$\quad z_2 \leftarrow_2 \{\mathcal{A}^{\mathcal{O}}(st') \leftrightarrow \mathtt{V}_D(par, PK, m, \sigma)\}$
$\quad \text{if } z_1 = z_2 = \mathtt{accept}$
$\quad\quad \text{output } 1$
$\quad \text{else output } 0$

Fig. 1. Unforgeability, key unforgeability and non-repudiation security experiments

- $\mathcal{O}_{VKeyReg}$: given (pk_V, sk_V), this oracle stores (pk_V, sk_V) in the list $L_{VKeyReg}$ and returns $\top$ to $\mathcal{A}$ if (pk_V, sk_V) is a valid key pair. Otherwise, the oracle returns $\bot$ to $\mathcal{A}$. In the following, if a query to an oracle involves a verifier key pk_V, it is assumed that $\mathcal{A}$ has previously submitted pk_V to this oracle as part of a valid key pair. If this is not the case, the relevant oracle will return $\bot$ to $\mathcal{A}$.
- $\mathcal{O}_{Sign}$: given input $(pk_C, pk_{C,S}, pk_V, m)$, this oracle interacts with $\mathcal{A}$ by running $\mathtt{Sign}$ with common input $(pk_S, pk_C, pk_{C,S}, pk_V, m)$ and secret input sk_S. Local output of $\mathtt{Sign}$ will be a signature σ, and $(pk_C, pk_{C,S}, m, \sigma)$ is added to L_{Sign}.
- $\mathcal{O}_{SimSign}$: given input pk_S, pk_C, $pk_{C,S}$, and m, this oracle interact with $\mathcal{A}$ by running the simulation algorithm $\mathtt{SimSign}(par, pk_S, pk_C, pk_{C,S}, m, sk_V)$.
- $\mathcal{O}_{SimCon}$: given input pk_S, pk_C, $pk_{C,S}$, m and σ, this oracle interacts with $\mathcal{A}$ by running the simulation algorithm $\mathtt{SimCon}(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma, sk_V)$.
- $\mathcal{O}_{SimDis}$: given the same input as $\mathcal{O}_{SimCon}$, this oracle interacts with $\mathcal{A}$ by running the simulation algorithm $\mathtt{SimDis}(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma, sk_V)$.

Definition 5 *An ONS scheme N is said to be* unforgeable*, if the exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage* $\mathtt{Adv}_{N,\mathcal{A}}^{uf\text{-}cma}(k) = \Pr[\mathrm{Exp}_{N,\mathcal{A}}^{uf\text{-}cma}(1^k) = 1]$.

Key unforgeability. The use of the confirmer setup, $\mathtt{CSetup}$, warrants additional security requirements. Key unforgeability requires that an adversary without access to the private confirmer key, cannot produce a new valid signer specific confirmer key i.e. a new key which is accepted by $\mathtt{CKeyValid}$. The security model in [26] does not have a similar security requirement and does in fact not rule out the possibility that a signer is able to forge a signer specific confirmer key and then use this forged key in the sign protocol. This would leave the confirmer unable to either confirm, disavow or convert the signature. However, such concerns are eliminated by explicitly requiring key unforgeability. Formally, key unforgeability of an ONS scheme N is defined via the experiment $\mathrm{Exp}_{N,\mathcal{A}}^{\mathtt{key\text{-}uf}}(1^k)$ shown in Figure 4. In the experiment, $\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathcal{O}_{VKeyReg}, \mathcal{O}_{CSetup}, \mathcal{O}_{Convert}, \mathcal{O}_{Con}, \mathcal{O}_{Dis}\}$ where $\mathcal{O}_{VKeyReg}$ is defined as in the unforgeability experiment, and the remaining oracles are defined as follows:

$\text{Exp}_{N,\mathcal{A}}^{\text{snd-sign}}(1^k)$

$\quad par \leftarrow \text{Setup}(1^k)$

$\quad (pk_C, sk_C) \leftarrow \text{KeyGen}_C(par)$

$\quad (pk_V, sk_V) \leftarrow \text{KeyGen}_V(par)$

$\quad (pk_S, m, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_C, sk_C, pk_V)$

$\quad (pk_{C,S}, sk_{C,S}) \leftarrow \text{CSetup}(par, pk_S, sk_C)$

$\quad PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$

$\quad (\sigma, z_1) \leftarrow_2 \{\mathcal{A}^{\mathcal{O}}(st, pk_{C,S}, sk_{C,S}) \leftrightarrow$

$\qquad\qquad\qquad \text{Receive}(par, PK, m^*)\}$

$\quad z_2 \leftarrow \text{Valid}(par, pk_S^*, pk_C, pk_{C,S}, m^*, \sigma, sk_{C,S})$

$\quad \text{if } z_1 = \text{accept} \wedge z_2 = \text{reject}$

$\qquad \text{output } 1$

$\quad \text{else output } 0$

$\text{Exp}_{N,\mathcal{A}}^{\text{snd-conf}}(1^k)$

$\quad par \leftarrow \text{Setup}(1^k)$

$\quad (pk_V, sk_V) \leftarrow \text{KeyGen}_V(par)$

$\quad (pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_V)$

$\quad PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$

$\quad z_1 \leftarrow_2 \{\mathcal{A}^{\mathcal{O}}(st) \leftrightarrow \text{V}_D(par, PK, m, \sigma)\}$

$\quad z_2 \leftarrow_2 \{\mathcal{A}^{\mathcal{O}}(st) \leftrightarrow \text{V}_C(par, PK, m, \sigma)\}$

$\quad z_3 \leftarrow \text{TkVerify}(par, pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma)$

$\quad \text{if } z_1 = \text{accept} \wedge (z_2 = \text{accept} \vee z_3 = \text{accept})$

$\qquad \text{output } 1$

$\quad \text{else output } 0$

Fig. 2. Soundness security experiments.

- $\mathcal{O}_{CSetup}$: given pk_S, this oracle runs $(pk_{C,S}, sk_{C,S}) \leftarrow \text{CSetup}(par, pk_S, sk_C)$, stores the tuple $(pk_S, pk_{C,S}, sk_{C,S})$ in L_{CSetup}, and returns $pk_{C,S}$ to $\mathcal{A}$.
- $\mathcal{O}_{Convert}$: given $pk_S, pk_{C,S}, m$, and σ, this oracle searches for a matching tuple $(pk_S, pk_{C,S}, sk_{C,S})$ in L_{CSetup}. If no such tuple is found, the oracle returns $\bot$ to $\mathcal{A}$. Otherwise, the oracle returns $tk \leftarrow \text{Convert}(par, pk_S, pk_C, pk_{C,S}, sk_{C,S}, m, \sigma)$.
- $\mathcal{O}_{Con}$: given $pk_S, pk_{C,S}, pk_V, m$, and σ, the oracle searches for a tuple $(pk_S, pk_{C,S}, sk_{C,S})$ in L_{CSetup}. If no such tuple is found the oracle returns $\bot$. Otherwise, the oracle interacts with $\mathcal{A}$ running Confirm with the common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma)$ and private input $sk_{C,S}$.
- $\mathcal{O}_{Dis}$: given the same input as $\mathcal{O}_{Con}$, this oracle returns $\bot$ to $\mathcal{A}$ if the tuple $(pk_S, pk_{C,S}, sk_{C,S})$ is not found in L_{CSetup}. Otherwise, the oracle interacts with $\mathcal{A}$ by running Disavow with common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma)$ and private input $sk_{C,S}$.

Definition 6 *An ONS scheme N is said to be* key unforgeable *if there exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage* $\text{Adv}_{N,\mathcal{A}}^{key\text{-}uf}(k) = \Pr[\text{Exp}_{N,\mathcal{A}}^{key\text{-}uf}(1^k) = 1]$.

Non-repudiation. Informally, non-repudiation requires that, even if a malicious signer and confirmer collude, it is not possible for the signer to make an honest verifier accept a message/signature pair as valid in the sign protocol, while the confirmer is able to disavow the validity of the message/signature pair. Our definition of non-repudiation is slightly weaker than the definition given in [26] in that we allow the adversary a negligible success probability whereas [26] requires the success probability to be zero. However, we highlight that [26] makes use of a non-interactive disavow protocol which is both off-line and on-line transferable which allows the slightly stronger non-repudiation property, whereas our constructions will rely on interactive non-transferable protocols with negligible soundness error.

We define non-repudiation of an ONS scheme N via the experiment $\text{Exp}_{N,\mathcal{A}}^{\text{non-rep}}(1^k)$ shown in Figure 4. In the experiment, $\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathcal{O}_{SimSign}, \mathcal{O}_{SimCon}, \mathcal{O}_{SimDis}\}$ which are defined as above.

Definition 7 *An ONS scheme N is said to provide* non-repudiation *if there exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage* $\text{Adv}_{N,\mathcal{A}}^{non\text{-}rep}(k) = \Pr[\text{Exp}_{N,\mathcal{A}}^{non\text{-}rep}(1^k) = 1]$.

Soundness. We consider two soundness notions – *signer soundness* and *confirmer soundness*. The first notion, signer soundness, guarantees that a signer cannot make a verifier accept a message/signature pair as valid through the sign protocol without the confirmer being able to confirm the validity of this pair as well as compute a verification token showing validity. Our definition guarantees signer soundness even if the confirmer is corrupted since the adversary is allowed to access the private confirmer key, and implies the soundness notion by Liskov and Micali which only grants the adversary access to the public confirmer key. Formally, we define signer soundness

of an ONS scheme N via the experiment $\mathtt{Exp}_{N,\mathcal{A}}^{\mathtt{snd\text{-}sign}}(1^k)$ shown in Figure 2. In the experiment, $\mathcal{A}$ will have access to the oracles $\mathcal{O} = \{\mathcal{O}_{SimSign}, \mathcal{O}_{SimCon}, \mathcal{O}_{SimDis}\}$ defined as above.

Definition 8 *An ONS scheme N is said to provide* signer soundness *if there exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage* $\mathtt{Adv}_{N,\mathcal{A}}^{snd\text{-}sign}(k) = \Pr[\mathtt{Exp}_{N,\mathcal{A}}^{snd\text{-}sign}(1^k) = 1]$.

The second notion, confirmer soundness, guarantees that even if both signer and confirmer key is maliciously generated, a confirmer cannot produce a message/signature pair which he can successfully disavow by completing the disavow protocol, while still being able to confirm validity of this pair, either by completing the confirm protocol or by producing a token that will make `TkVerify` output `accept`. Since Liskov and Micali do not consider the confirmer's ability to confirm a signature, an equivalent security notion is not defined in [26]. We define confirmer soundness of an ONS scheme N via the experiment $\mathtt{Exp}_{N,\mathcal{A}}^{\mathtt{snd\text{-}conf}}(1^k)$ shown in Figure 2. In the experiment, $\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathcal{O}_{SimSign}, \mathcal{O}_{SimCon}, \mathcal{O}_{SimDis}\}$ defined as in the non-repudiation experiment.

Definition 9 *An ONS scheme is said to provide* confirmer soundness *if there exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage* $\mathtt{Adv}_{N,\mathcal{A}}^{snd\text{-}conf}(k) = \Pr[\mathtt{Exp}_{N,\mathcal{A}}^{snd\text{-}conf}(1^k) = 1]$.

On-line Non-transferability. Intuitively, on-line non-transferability of a protocol requires that an adversary cannot distinguish between a real execution of the protocol and a simulated execution by the verifier. Note that since we are considering the on-line non-transferability of both the sign, confirm and disavow protocols, a verifier must be able to provide a consistent response, even if the adversary first obtains a signature through the (simulated) sign protocol and later try to re-confirm the validity through the (simulated) confirm protocol. We define a single non-transferability notion covering the non-transferability of all three interactive protocols. More specifically, we require that an adversary cannot distinguish between a scenario in which he obtains a valid signature through the sign protocol, confirms the validity through the confirm protocol, and then interacts in the simulated disavow protocol, and a scenario in which he obtains a signature through the simulated sign protocol, confirms the validity through the simulated confirm protocol, and then interacts in the disavow protocol. Our non-transferability notion implies a similar type of non-transferability of the sign protocol as defined by Liskov and Micali, but does not involve fake signature generation. Formally, we define on-line non-transferability of a ONS scheme N via the experiment $\mathtt{Exp}_{N,\mathcal{A}}^{\mathtt{non\text{-}trans}}(1^k)$ shown in Figure 4. In the experiment, $\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathcal{O}_{VKeyReg}, \mathcal{O}_{CSetup}, \mathcal{O}_{Sign}, \mathcal{O}_{Convert}, \mathcal{O}_{Con}, \mathcal{O}_{Dis}\}$ defined as in the unforgeability and the key unforgeability experiments. The oracles $\mathcal{O}'$ are defined exactly as $\mathcal{O}$, except that $\mathcal{O}_{Convert}$ will not respond to the query consisting of the challenge $(pk_S, pk_{C,S})$, m^* and σ^*, and $\mathcal{O}_{Con}$ and $\mathcal{O}_{Dis}$ will not respond to queries on the challenge $(pk_S, pk_{C,S})$, m^*, σ^* and any pk_V. Note that $\mathcal{O}_{Sign}$ allows the adversary to obtain signatures under any confirmer key, and that $\mathcal{O}_{Convert}$, $\mathcal{O}_{Con}$ and $\mathcal{O}_{Dis}$ accepts any signer key. This ensures security in a scenario where multiple confirmers service multiple signers. Note also that the adversary is given the private key of the verifier. This will ensure that even if the verifier is compromised, the non-transferability is still maintained.

Definition 10 *An ONS scheme N is said to be* on-line non-transferable *if there exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage* $\mathtt{Adv}_{N,\mathcal{A}}^{non\text{-}trans}(k) = |\Pr[\mathtt{Exp}_{N,\mathcal{A}}^{non\text{-}trans}(1^k) = 1] - \frac{1}{2}|$.

5 Construction of an ONS scheme

In this section we will present a construction of an ONS scheme based on four simpler building blocks: a standard signature scheme, a core confirmer signature scheme, sigma protocols, and a trapdoor commitment scheme with an enhanced binding property. In the following, we will formally define a core confirmer signature scheme as well as the needed security requirements, motivate and define the enhanced binding property of a trapdoor commitment scheme, and finally show how the above mentioned primitives can be combined into a secure ONS scheme.

$$\mathbf{Exp}_{N,\mathcal{A}}^{\text{non-trans}}(1^k)$$

$L_{VKeyReg} \leftarrow \{\}$
$par \leftarrow \mathtt{Setup}(1^k)$
$(pk_S, sk_S) \leftarrow \mathtt{KeyGen}_S(par)$
$(pk_C, sk_C) \leftarrow \mathtt{KeyGen}_C(par)$
$(pk_{C,S}, sk_{C,S}) \leftarrow \mathtt{CSetup}(par, pk_S, sk_C)$
$(pk_V, sk_V) \leftarrow \mathtt{KeyGen}_V(par)$
$PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$
$(m^*, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, PK, sk_V)$
$b \leftarrow \{0,1\}$
if $b = 0$
   $(\sigma^*, st') \leftarrow \{\mathtt{Sign}(par, PK, m^*, sk_S) \leftrightarrow \mathcal{A}(st)\}$
   $st'' \leftarrow_2 \{\mathtt{Confirm}(par, PK, m^*, \sigma^*, sk_{C,S}) \leftrightarrow \mathcal{A}(st')\}$
   $st''' \leftarrow_2 \{\mathtt{SimDis}(par, PK, m^*, \sigma^*, sk_V) \leftrightarrow \mathcal{A}(st'')\}$
else $(b = 1)$
   $(\sigma^*, st') \leftarrow \{\mathtt{SimSign}(par, PK, m^*, sk_V) \leftrightarrow \mathcal{A}(st)\}$
   $st'' \leftarrow_2 \{\mathtt{SimCon}(par, PK, m^*, \sigma^*, sk_V) \leftrightarrow \mathcal{A}(st')\}$
   $st''' \leftarrow_2 \{\mathtt{Disavow}(par, PK, m^*, \sigma^*, sk_{C,S}) \leftrightarrow \mathcal{A}(st'')\}$
$b' \leftarrow A^{\mathcal{O}'}(st''')$
if $b = b'$ output 1
else output 0

Fig. 3. On-line non-transferability security experiment.

5.1 Core Confirmer Signature Scheme

A core confirmer signature scheme is essentially an ONS scheme without any of the interactive algorithm. More specifically, a core confirmer signature scheme is given by $CS = \{\mathtt{CS.Setup}, \mathtt{CS.KeyGen}_S,$ $\mathtt{CS.KeyGen}_C, \mathtt{CS.Sign}, \mathtt{CS.Convert}, \mathtt{CS.TkVerify}\}$ where the algorithms $\mathtt{CS.Setup}$, $\mathtt{CS.KeyGen}_S$, and $\mathtt{CS.KeyGen}_C$ are defined as in a full ONS scheme, and the $\mathtt{CS.Sign}$, $\mathtt{CS.Convert}$ and $\mathtt{CS.TkVerify}$ algorithms are defined as follows:

- $\mathtt{CS.Sign}$: given par, pk_C, m, and sk_S, this algorithm returns a signature σ.
- $\mathtt{CS.Convert}$: given par, pk_S, m, σ and sk_C, this algorithm returns a token tk_σ.
- $\mathtt{CS.TkVerify}$: given par, pk_S, pk_C, m, σ and tk_σ, this algorithm returns either $\mathtt{accept}$ or $\mathtt{reject}$.

Both $\mathtt{CS.Convert}$ and $\mathtt{CS.TkVerify}$ are assumed to be deterministic. Note that all algorithms are non-interactive and that no specific confirmer keys $pk_{C,S}$ or verifier keys pk_V are required. Like for an ONS scheme, we define an algorithm $\mathtt{CS.Valid}$ as

- $\mathtt{CS.Valid}$: given par, pk_S, pk_C, m, σ and sk_C, this algorithms computes the verification token $tk_\sigma \leftarrow \mathtt{CS.Convert}(par, pk_S, m, \sigma, sk_C)$ and returns the output of the verification algorithm $\mathtt{CS.TkVerify}(par, pk_S, pk_C, m, \sigma, tk_\sigma)$.

We require that a core confirmer signature scheme is *correct* i.e. we require that for all $par \leftarrow \mathtt{CS.Setup}(1^k)$, $(pk_S, sk_S) \leftarrow \mathtt{CS.KeyGen}_S(par)$, $(pk_C, sk_C) \leftarrow \mathtt{CS.KeyGen}_V(par)$, all messages m and all $\sigma \leftarrow \mathtt{CS.Sign}(par, pk_C, m, sk_S)$, that $\mathtt{CS.Valid}(par, pk_S, pk_C, m, \sigma, sk_C)$ yields $\mathtt{accept}$.

Furthermore, we require that a core confirmer signature scheme has *unique* private confirmer keys i.e. for any pk_C, there exists at most one sk_C such that $(pk_C, sk_C) \in \{\mathtt{CS.KeyGen}_C(par)\}$ where $\{\mathtt{CS.KeyGen}_C(par)\}$ denotes the set of all possible confirmer key pairs generated by $\mathtt{CS.KeyGen}_C$.[8]

Security Requirements. For a core confirmer signature scheme to be secure, we require that the scheme provides *unforgeability*, *invisibility* and *token soundness*. However, due to the reduced

---

[8] This property is needed to prove confirmer soundness (Theorem 20) of the construction presented in Section 5.3.

$$\mathrm{Exp}_{CS,\mathcal{A}}^{\texttt{cs-uf-cma}}(1^k)$$
$\quad L_{CSSign} \leftarrow \{\}$
$\quad par \leftarrow \texttt{CS.Setup}(1^k)$
$\quad (pk_S, sk_S) \leftarrow \texttt{CS.KeyGen}_S(par)$
$\quad (pk_C^*, m^*, \sigma^*, tk^*) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_S)$
$\quad z \leftarrow \texttt{CS.TkVerify}(par, pk_S, pk_C^*, \sigma^*, m^*, tk^*)$
$\quad$ if $(pk_C^*, m^*, \sigma^*) \notin L_{CSSign} \wedge z = \texttt{accept}$
$\quad\quad$ output 1
$\quad$ else output 0

$$\mathrm{Exp}_{CS,\mathcal{A}}^{\texttt{cs-inv-cma}}(1^k)$$
$\quad par \leftarrow \texttt{CS.Setup}(1^k)$
$\quad (pk_S, sk_S) \leftarrow \texttt{CS.KeyGen}_S(par)$
$\quad (pk_C, sk_C) \leftarrow \texttt{CS.KeyGen}_C(par)$
$\quad (m^*, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_S, pk_C)$
$\quad b \leftarrow \{0,1\}$
$\quad$ if $b = 0$ $\sigma^* \leftarrow \mathcal{S}$
$\quad$ else $\sigma^* \leftarrow \texttt{CS.Sign}(par, pk_C, sk_S, m)$
$\quad b' \leftarrow \mathcal{A}^{\mathcal{O}}(st, \sigma^*)$
$\quad$ if $b = b'$ output 1
$\quad$ else output 0

$$\mathrm{Exp}_{CS,\mathcal{A}}^{\texttt{cs-tk-snd}}(1^k)$$
$\quad par \leftarrow \texttt{CS.Setup}(1^k)$
$\quad (pk_S^*, pk_C^*, sk^*, m^*, \sigma^*, tk_\sigma^*) \leftarrow \mathcal{A}(par)$
$\quad z_1 \leftarrow \texttt{CS.TkVerify}(par, pk_S^*, pk_C^*, \sigma^*, m^*, tk_\sigma^*)$
$\quad z_2 \leftarrow \texttt{CS.Valid}(par, pk_S^*, pk_C^*, \sigma^*, m^*, sk_C^*)$
$\quad$ if $(pk_C^*, sk^*) \in \{\texttt{CS.KeyGen}_C(par)\} \wedge z_1 = \texttt{accept} \wedge z_2 = \texttt{reject}$
$\quad\quad$ output 1
$\quad$ else output 0

Fig. 4. Unforgeability, invisibility and token soundness experiments for a core confirmer signature scheme.

functionality of a core confirmer signature scheme, these definitions will be much simpler compared to the security definitions of a full ONS scheme.

We define unforgeability of a core confirmer signature scheme CS via the experiment $\mathrm{Exp}_{CS,\mathcal{A}}^{\texttt{cs-uf-cma}}(1^k)$ shown in Figure 5.1. In the experiment, $\mathcal{A}$ has access to the oracle $\mathcal{O}_{Sign}$ defined as follows:

- $\mathcal{O}_{Sign}$: given pk_C, and m, this oracle computes the signature $\sigma \leftarrow \texttt{CS.Sign}(par, pk_C, m, sk_S)$, adds (pk_C, m, σ) to L_{CSSign}, and returns σ.

Definition 11 *A core confirmer signature scheme CS is said to be* unforgeable *if there exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage* $\mathrm{Adv}_{CS,\mathcal{A}}^{cs-uf-cma}(k) = \Pr[\mathrm{Exp}_{CS,\mathcal{A}}^{cs-uf-cma}(1^k) = 1]$.

Invisibility of a core confirmer signature scheme CS, which captures the property that valid signatures cannot be distinguished from random elements of the signature space, is defined via the experiment $\mathrm{Exp}_{CS,\mathcal{A}}^{\texttt{cs-inv-cma}}$ shown in Figure 5.1. In the experiment, $\mathcal{S}$ denotes the signature space of the scheme, and $\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathcal{O}_{Sign}, \mathcal{O}_{Convert}\}$ where $\mathcal{O}_{Sign}$ is defined as in the above unforgeability experiment, and $\mathcal{O}_{Convert}$ is defined as follows:

- $\mathcal{O}_{Convert}$: given m and σ, this oracle returns $tk \leftarrow \texttt{CS.Convert}(par, pk_S, pk_C, m, \sigma, sk_C)$.

Note that this oracle will only convert signatures from the signer pk_S and is not required to work for maliciously generated public signer keys for which the adversary might know the corresponding private key. Hence, intuitively, this security requirement only requires the scheme to be secure in a "single user" setting in which a confirmer only services a single signer. This weaker requirement is important for the security proof of our concrete construction.

Definition 12 *A core confirmer signature scheme CS is said to be* invisible *if there exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage* $\mathrm{Adv}_{CS,\mathcal{A}}^{cs-inv-cma}(k) = |\Pr[\mathrm{Exp}_{CS,\mathcal{A}}^{cs-inv-cma}(1^k) = 1] - \frac{1}{2}|$.

Lastly, we consider token soundness [31] for a core confirmer signature scheme which intuitively captures the property that an accepting verification token cannot be constructed for an invalid signature. Formally, we define token soundness of a scheme CS via the experiment $\mathrm{Exp}_{CS,\mathcal{A}}^{\texttt{cs-tk-snd}}$ shown in Figure 5.1. In the figure, $\{\texttt{CS.KeyGen}_C(par)\}$ denotes the set of all possible key pairs generated by $\texttt{CS.KeyGen}_C$.

Definition 13 *A core confirmer signature scheme CS is said to have* token soundness *if there exists no polynomial time algorithm* $\mathcal{A}$ *with non-negligible advantage* $\mathtt{Adv}^{cs-tk-snd}_{CS,\mathcal{A}}(k) = \Pr[\mathtt{Exp}^{cs-tk-snd}_{CS,\mathcal{A}}(1^k) = 1]$.

Compatible Sigma Protocols. In our full ONS scheme, we will base the construction of on-line non-transferable protocols on sigma protocols. For this purpose, we require that a set of sigma protocols compatible with the core confirmer signature scheme exists. More specifically, we say that a triple of sigma protocols, Σ_S, Σ_C and $\overline{\Sigma}_C$, and a core confirmer signature scheme CS are compatible if the sigma protocols are defined for the common input $x = (par, pk_S, pk_C, m, \sigma)$ and the following relations.

- $\Sigma_S\{(x, (sk_S, r)) : (pk_S, sk_S) \in \{\mathtt{CS.KeyGen}_S(par)\} \wedge \sigma = \mathtt{CS.Sign}(par, pk_C, m, sk_S; r)\}$
- $\Sigma_C\{(x, sk_C) : (pk_C, sk_C) \in \{\mathtt{CS.KeyGen}_C(par)\} \wedge \mathtt{CS.Valid}(par, pk_S, pk_C, m, \sigma, sk_C) = \mathtt{accept}\}$
- $\overline{\Sigma}_C\{(x, sk_C) : (pk_C, sk_C) \in \{\mathtt{CS.KeyGen}_C(par)\} \wedge \mathtt{CS.Valid}(par, pk_S, pk_C, m, \sigma, sk_C) = \mathtt{reject}\}$

Recall that we use the notation $\Sigma\{(x, w) : R(x, w) = 1\}$ to denote the sigma protocol Σ for relation R with common input x and witness w. For simplicity, we furthermore assume that the challenge space of Σ_S, Σ_C and $\overline{\Sigma}_C$ are of the same size.

5.2 On-line Non-transferable Protocols

As mentioned above, our construction of on-line non-transferable protocols is based on extensions of the compatible sigma protocols for a core confirmer signature scheme. We use a simple and intuitive approach inspired by the construction of designated verifier proofs by Jakobsson *et al.* [20] which is furthermore closely related to the construction of efficient zero-knowledge proofs in the auxiliary string model [14] and the recent construction of deniable authentication for signatures [28]. More specifically, we modify the sigma protocols using a trapdoor commitment scheme, and let a prover and a verifier interact as follows:

1. The prover computes the first message a of the sigma protocol and the commitment $com \leftarrow \mathtt{Comm}(ck, a, r)$ for random r, and then sends com to the verifier.
2. The verifier then sends a random challenge c to the prover.
3. The prover computes the last message z of the sigma protocol and sends the opening (a, r) together with z to the verifier. The verifier checks that $com = \mathtt{Comm}(ck, a, r)$, and accepts if (a, c, z) is an accepting transcript of the sigma protocol.

The commitment key and trapdoor (ck, td) will be used as the public/private key pair (pk_V, sk_V) of the verifier. Hence, using sk_V, the verifier will be able to open com to any message a of his choice, and can therefor postpone generating a until after the challenge c is revealed which allows him to simulate the proof interactively. We use the notation $\mathtt{NT}(ck)\text{-}\Sigma$ to denote the non-transferable protocol obtained by modifying the sigma protocol Σ as described above using the commitment key ck.

However, the above approach is not sufficient for proving our constructions secure. Essentially the problem is that an adversary can request to interact with $\mathtt{SimSign}$, $\mathtt{SimCon}$ and $\mathtt{SimDis}$ in many of the security notions defined in Section 4, choosing any message or message/signature pair (valid or invalid) as input. This type of query can be difficult to handle for a simulator not knowing the trapdoor of the commitment scheme, whereas a simulator knowing the trapdoor might not gain sufficient information from an adversary breaking the security of the scheme. To address this problem, we introduce a commitment scheme with a stronger binding property. Specifically, we consider the advantage of an adversary $\mathcal{A}$ against binding property of a commitment scheme T to be defined as

$$\mathtt{Adv}^{bind}_{T,\mathcal{A}}(k) = \Pr[(ck, td) \leftarrow \mathtt{G}(1^k); (w, r, w', r') \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_o}(ck) :$$
$$w \neq w' \wedge \mathtt{Comm}(ck, w, r) = \mathtt{Comm}(ck, w', r')]$$

where $\mathcal{A}$ has access to a commit and an open oracle, $\mathcal{O}_c$ and $\mathcal{O}_o$, which behave as follows: upon request, $\mathcal{O}_c$ computes $(com, aux) \leftarrow \mathtt{TdComm}$, stores aux and returns com to $\mathcal{A}$. Given a commitment com returned by $\mathcal{O}_c$ and a value w, $\mathcal{O}_o$ retrieves the corresponding aux and returns $r \leftarrow \mathtt{TdOpen}(aux, w, td)$ such that $com = \mathtt{Comm}(ck, w, r)$. The adversary is only allowed to query $\mathcal{O}_o$ with a commitment com obtained from O_c, and is not allowed to make more than one query to $\mathcal{O}_o$ for a given commitment com.

Definition 14 *A trapdoor commitment scheme T is said to be* binding under selective trapdoor openings *if there exists no polynomial time algorithm $\mathcal{A}$ with non-negligible advantage $\mathtt{Adv}_{T,\mathcal{A}}^{bind}$.*

Pedersen's commitment scheme [30] can be shown to be binding under selective trapdoor openings assuming the one more discrete logarithm problem is hard. However, to obtain a commitment scheme which can be shown secure only assuming the ordinary discrete logarithm problem is hard, we can make use of the "double trapdoor" extension also used to strengthen signatures [34] and improve on-line/off-line signatures [3, 7]. Below, we recall this scheme.

- G: Given 1^k, pick a group $\mathbb{G}$ of prime order p such that $2^k < p < 2^{k+1}$, and furthermore pick a generator $g \leftarrow \mathbb{G}$ and random $x_1, x_2 \leftarrow \mathbb{Z}_p$. Compute $h_1 \leftarrow g^{x_1}$ and $h_2 \leftarrow g^{x_2}$, and set the public parameters to $ck \leftarrow (\mathbb{G}, p, g, h_1, h_2)$ and the trapdoor to $td \leftarrow (x_1, x_2)$.
- Comm: Given ck and a value $w \in \mathbb{Z}_p$, pick random $r_1, r_2 \leftarrow \mathbb{Z}_p$ and compute a commitment to w as $com \leftarrow g^w h_1^{r_1} h_2^{r_2}$. (The opening of com is (w, r_1, r_2).)
- TdComm: Pick random $r \leftarrow \mathbb{Z}_p$ and set $com \leftarrow g^r$ and $aux \leftarrow r$.
- TdOpen: Given com, a corresponding aux, the trapdoor $td = (x_1, x_2)$ and a value w, pick random $r_1 \leftarrow \mathbb{Z}_p$, compute $r_2 \leftarrow (aux - w - x_1 r_1)/x_2$, and return the opening (w, r_1, r_2).

Theorem 15 *Assume the discrete logarithm problem is hard in $\mathbb{G}$. Then the above commitment scheme is binding under selective trapdoor openings. More specifically, there exists a polynomial time algorithm $\mathcal{B}$ such that for any polynomial time adversary $\mathcal{A}$, $\mathtt{Adv}_{T,\mathcal{A}}^{bind}(k) \leq \mathtt{Adv}_{\mathbb{G},\mathcal{B}}^{dl}(k)/2$.*

The proof of the above theorem can be found in Appendix A. It is relatively easy to verify that the scheme is also perfectly hiding and provides a perfect trapdoor property.

5.3 Combined Scheme

We will now show how to combine the above mentioned primitives into a full ONS scheme. More specifically, let $CS = \{\mathtt{CS.Setup}, \mathtt{CS.KeyGen}_S, \mathtt{CS.KeyGen}_C, \mathtt{CS.Sign}, \mathtt{CS.Convert}, \mathtt{CS.TkVerify}\}$ be a core confirmer signature scheme with compatible sigma protocols Σ_S, Σ_C and $\overline{\Sigma}_C$, let $T = \{\mathtt{T.G}, \mathtt{T.Comm}, \mathtt{T.TdComm}, \mathtt{T.TdOpen}\}$ be a trapdoor commitment scheme, and let $S = \{\mathtt{S.Setup}, \mathtt{S.KeyGen}, \mathtt{S.Sign}, \mathtt{S.Verify}\}$ be an ordinary signature scheme. We construct an ONS scheme N as follows:

- $\mathtt{Setup}(1^k)$: Compute $par_S \leftarrow \mathtt{S.Setup}(1^k)$ and $par_{CS} \leftarrow \mathtt{CS.Setup}(1^k)$, and return the parameters $par \leftarrow (par_S, par_{CS})$. It is assumed that par_{CS} include a description of the randomness space $\mathcal{R}$ used by the $\mathtt{CS.Sign}$ algorithm.
- $\mathtt{KeyGen}_S(par)$: Return $(pk_S, sk_S) \leftarrow \mathtt{CS.KeyGen}_S(par_{CS})$.
- $\mathtt{KeyGen}_C(par)$: Return $(pk_C, sk_C) \leftarrow \mathtt{S.KeyGen}(par_S)$.
- $\mathtt{KeyGen}_V(par)$: Return $(pk_V, sk_V) \leftarrow \mathtt{T.G}(1^k)$.
- $\mathtt{CSetup}(par, pk_S, sk_C)$: Compute the key pair $(pk_C', sk_C') \leftarrow \mathtt{CS.KeyGen}_C(par_{CS})$ and the signature $\delta \leftarrow \mathtt{S.Sign}(par_S, \text{``}pk_S||pk_C'\text{''}, sk_C)$, and return $pk_{C,S} \leftarrow (pk_C', \delta)$ and $sk_{C,S} \leftarrow sk_C'$.
- $\mathtt{CKeyValid}(par, pk_S, pk_C, pk_{C,S})$ Let $pk_{C,S} = (pk_C', \delta)$ and return the result of the verification $\mathtt{S.Verify}(par_S, pk_C, \text{``}pk_S||pk_C'\text{''}, \delta)$.
- $(\mathtt{Sign}, \mathtt{Receive})$: The common input is given by $(par, pk_S, pk_C, pk_{C,S}, m, pk_V)$ where $pk_{C,S} = (pk_C', \delta)$ and the signer is given sk_S as private input.
  Firstly, the signer picks $r \in \mathcal{R}$ and computes $\sigma \leftarrow \mathtt{CS.Sign}(par_{CS}, pk_C', sk_S, pk_C||pk_{C,S}||m; r)$. Then the signer sends σ to the verifier, and interacts with the verifier in the protocol $\mathtt{NT}(pk_V)$-Σ_S using $(par, pk_S, pk_C', pk_C||pk_{C,S}||m, \sigma)$ as common input and (sk_S, r) as secret input. [9]

---

[9] Note that this construction of the sign protocol is slightly more flexible than required by the definition in Section 3 in that a signer is able to re-confirm a signature by running $\mathtt{NT}(pk_V)$-Σ_S. This, however, requires the signer to remember the randomness used to construct the signature.

- Convert$(par, pk_S, m, \sigma, sk_{C,S})$: Return $tk_\sigma \leftarrow \texttt{CS.Convert}(par_{CS}, pk_S, m, \sigma, sk_{C,S})$.
- TkVerify$(par, pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma)$: Firstly, verify the validity of $pk_{C,S}$ by computing $z \leftarrow \texttt{CKeyValid}(par, pk_S, pk_C, pk_{C,S})$ defined above. If $z = \texttt{accept}$, let $pk_{C,S} = (pk'_C, \delta)$ and return $\texttt{CS.TkVerify}(par_{CS}, pk_S, pk'_C, pk_C || pk_{C,S} || m, \sigma, tk_\sigma)$. Otherwise, return $\texttt{reject}$.
- (Confirm$, V_C)$: The common input is given by $(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$ where $pk_{C,S} = (pk'_C, \delta)$ and the signer is given $sk_{C,S}$ as private input. Firstly, the verifier checks validity of $pk_{C,S}$ by running $\texttt{CKeyValid}(par, pk_S, pk_C, pk_{C,S})$, and aborts if the output is $\texttt{reject}$. The confirmer then interacts with the verifier in the protocol $\texttt{NT}(pk_V)\text{-}\Sigma_C$ with private input $sk_{C,S}$ and common input $(par_S, pk_S, pk'_C, pk_C || pk_{C,S} || m, \sigma)$.
- (Disavow$, V_D)$: Having the same common input as in (Confirm$, V_C$), the verifier firstly checks if $\texttt{CKeyValid}(par, pk_S, pk_C, pk_{C,S}) = \texttt{accept}$, and aborts if this is not the case. The verifier and signer then interact in the protocol $\texttt{NT}(pk_V)\text{-}\overline{\Sigma}_C$ with private input $sk_{C,S}$ to the confirmer and common input $(par_S, pk_S, pk'_C, pk_C || pk_{C,S} || m, \sigma)$.

Security. We will now prove that the above constructed ONS scheme satisfies the security definitions given in Section 4 assuming the underlying primitives are secure. However, firstly we must define the simulation algorithms required to achieve security. $\texttt{SimSign}$, $\texttt{SimCon}$ and $\texttt{SimDis}$ are constructed as follows:

- SimSign$(par, pk_S, pk_C, pk_{C,S}, pk_V, m, sk_V)$
   1. Pick random $\sigma \leftarrow \mathcal{S}$ and send σ to the verifier.
   2. Run $(com, aux) \leftarrow \texttt{TdCom}(pk_V)$ and send com to the verifier.
   3. When a challenge c is received from the verifier, let $pk_{C,S} = (pk'_C, \sigma)$ and compute $(a, z) \leftarrow \texttt{Sim}_{\Sigma_S}(par, pk_S, pk'_C, pk_C || pk_{C,S} || m, \sigma, c)$ and $r \leftarrow \texttt{TdOpen}(aux, a, sk_V)$, and send (a, z, r) to the verifier.
- SimCon and SimDis with input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma, sk_V)$ are both implemented by step 2 and 3 above in which $\texttt{Sim}_{\Sigma_S}$ is replaced with $\texttt{Sim}_{\Sigma_C}$ and $\texttt{Sim}_{\overline{\Sigma}_C}$, respectively.

Theorem 16 *Assume that CS is unforgeable, that Σ_S is honest verifier zero-knowledge and has special soundness, and that T provides a perfect trapdoor property and is binding under selective trapdoor openings. Then the above ONS scheme N is unforgeable. More specifically, there exist polynomial time algorithms $\mathcal{B}$, $\mathcal{B}'$ and $\mathcal{B}''$ such that for any polynomial time adversary $\mathcal{A}$,*

$$\texttt{Adv}_{N,\mathcal{A}}^{uf\text{-}cma} \leq \sqrt{4\texttt{Adv}_{CS,\mathcal{B}}^{cs\text{-}uf\text{-}cma} + 1/p^2} + \texttt{Adv}_{T,\mathcal{B}'}^{bind} + \texttt{Adv}_{CS,\mathcal{B}''}^{cs\text{-}uf\text{-}cma} + 1/p,$$

where p is the size of the challenge space of Σ_S.

The proof can be found in Appendix B.1.

Theorem 17 *Assume the signature scheme S is strongly unforgeable. Then the above ONS scheme N has key unforgeability. More specifically, there exists a polynomial time algorithm $\mathcal{B}$ such that for any polynomial time adversary $\mathcal{A}$, $\texttt{Adv}_{N,\mathcal{A}}^{uf\text{-}cma} \leq \texttt{Adv}_{S,\mathcal{B}}^{suf\text{-}cma}$.*

This follows directly from the construction since a specific confirmer key $pk_{C,S}$ consists of a public key pk'_C of the underlying core confirmer signature scheme and a signature δ on $pk_S || pk'_C$ i.e. producing a new valid $pk_{C,S}$ for a different $pk_S || pk'_C$ pair corresponds directly to forging a new message/signature pair of the underlying signature scheme. We omit the details here.

Theorem 18 *Assume that Σ_S and $\overline{\Sigma}_C$ have special soundness, and that T is binding under selective trapdoor openings. Then the above ONS scheme N provides non-repudiation. More specifically, there exists a polynomial time algorithm $\mathcal{B}$ such that for any polynomial time adversary $\mathcal{A}$, $\texttt{Adv}_{N,\mathcal{A}}^{non\text{-}rep} \leq 2/p + 2\texttt{Adv}_{T,\mathcal{B}}^{bind}$, where p is the size of the challenge space of Σ_S and $\overline{\Sigma}_C$.*

The proof can be found in Appendix B.2.

The following two theorems shows the soundness properties of the above proposed ONS scheme. The proof of the first theorem follows a very similar pattern to the proof of Theorem 18, and can be derived from this. We leave out the details of this proof. The proof of the second theorem can be found in Appendix B.3.

Theorem 19 *Assume that Σ_S have special soundness, and that T is binding under selective trapdoor openings. Then the above ONS scheme N provides signer soundness. More specifically, there exists a polynomial time algorithm $\mathcal{B}$ such that for any polynomial time adversary $\mathcal{A}$, $\mathtt{Adv}_{N,\mathcal{A}}^{snd-sign} \leq 2/p + \mathtt{Adv}_{T,\mathcal{B}}^{bind}$, where p is the size of the challenge space of Σ_S.*

Theorem 20 *Assume that CS has unique private confirmer keys and provides token soundness, that Σ_C and $\overline{\Sigma}_C$ have special soundness, and that T is binding under selective trapdoor openings. Then the above ONS scheme N provides confirmer soundness. More specifically, there exists polynomial time algorithms $\mathcal{B}$ and $\mathcal{B}'$ such that for any polynomial time adversary $\mathcal{A}$,*

$$\mathtt{Adv}_{N,\mathcal{A}}^{snd-conf} \leq \sqrt{4\mathtt{Adv}_{CS,\mathcal{A}}^{cs-tk-snd} + 1/p^2} + 3\mathtt{Adv}_{T,\mathcal{B}'}^{bind} + 3/p,$$

where p is the size of the challenge space of Σ_C and $\overline{\Sigma}_C$.

Lastly, we consider the non-transferability of the proposed ONS scheme. The proof of the following theorem can be found in Appendix B.4.

Theorem 21 *Assume that CS is invisible, that Σ_S, Σ_C and $\overline{\Sigma}_C$ are honest verifier zero-knowledge, and that T provides a perfect trapdoor property. Then the above ONS scheme N provides on-line non-transferability. More specifically, there exists a polynomial time algorithm $\mathcal{B}$ such that for any polynomial time adversary $\mathcal{A}$, $\mathtt{Adv}_{N,\mathcal{A}}^{non-trans} \leq \mathtt{Adv}_{CS,\mathcal{B}}^{cs-inv-cma}$.*

6 Concrete Instantiation

To instantiate the construction presented in Section 5.3, we can use the strongly unforgeable signature scheme by Boneh *et al.* [2] and the double trapdoor Pedersen commitment scheme shown in Section 5.2. To complete the instantiation, we will define and prove secure a core confirmer signature scheme and compatible sigma protocols.

6.1 Concrete Core Confirmer Signature Scheme

Our scheme is essentially based on a linear encryption of the first component of a Waters signature [35] combined with the technique of Boneh *et al.* [2] to obtain a strongly unforgeable scheme. This will result in a strongly unforgeable and invisible scheme with a structure that allows the compatible sigma protocols to be implemented using well-know techniques for proving knowledge of discrete logarithms. Our scheme CS is defined below.

- **Setup**: Given 1^k, choose bilinear groups $(\mathbb{G}, \mathbb{G}_T)$ of prime order $2^k < p < 2^{k+1}$, a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and generator g of $\mathbb{G}$. Furthermore, choose a collision resistant hash function family $\mathcal{H} = \{H_\nu : \{0,1\}^* \to \{0,1\}^{|\nu|}\}$ indexed by a key $\nu \in \{0,1\}^k$ (note that since $2^{|\nu|} < p$, the output of H_ν can be viewed as an element of $\mathbb{Z}_p$). Return $par = (\mathbb{G}, \mathbb{G}_T, e, p, g, \mathcal{H})$.
- **KeyGen**$_S(par)$: Pick $\alpha \leftarrow \mathbb{Z}_p$, set $g_1 \leftarrow g^\alpha$, and pick $g_2, h \leftarrow \mathbb{G}$. Furthermore, pick $u_0, \dots, u_n \leftarrow \mathbb{Z}_p$, and define $F(m) = u_0 \prod_{i=1}^n u_i^{m_i}$ where m_i is the ith bit of m. Finally pick a hash key $\nu \in \{0,1\}^k$ and return $pk_S = (q, g_1, g_2, h, u_0, \dots, u_n)$ and $sk_S = \alpha$.
- **KeyGen**$_C(par)$: Pick $x, y \leftarrow \mathbb{Z}_p$ and set $u \leftarrow g^{x^{-1}}$ and $v \leftarrow g^{y^{-1}}$. Return $pk_C = (u, v)$ and $sk_C = (x, y)$.
- **Sign** : Given input (par, pk_C, m, sk_S), where $pk_C = (u, v)$ and $sk_S = \alpha$, pick $a, b, r, s \leftarrow \mathbb{Z}_p$, compute $t \leftarrow H_\nu(pk_C||u^a||v^b||g^{r+a+b}||m)$ and $M = g^t h^s$, and return the signature $\sigma \leftarrow (u^a, v^b, g^{r+a+b}, g_2^\alpha F(M)^r, s)$.
- **Convert**$(par, pk_S, m, \sigma, sk_C)$: Let $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, s)$ and $sk_C = (x, y)$. If the equation $e(g, \sigma_4) = e(g_1, g_2)e(\sigma_3/(\sigma_1^x \sigma_2^y), F(M))$ holds where $M = g^t h^s$ and $t = H_\nu(pk_C||\sigma_1||\sigma_2||m)$, return the token $tk_\sigma = (\sigma_1^x, \sigma_2^y)$. Otherwise return $\perp$.
- **TkVerify**$(par, pk_S, pk_C, m, \sigma, tk_\sigma)$: Let $pk_S = (q, g_1, g_2, h, u_0, \dots, u_n)$, $pk_C = (u, v)$, $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, s)$ and $tk_\sigma = (tk_1, tk_2)$. Return **accept** if the equations $e(u, tk_1) = e(\sigma_1, g)$, $e(v, tk_2) = e(\sigma_2, g)$ and $e(g, \sigma_4) = e(g_1, g_2)e(\sigma_3/(tk_1 tk_2), F(M))$ holds where $M = g^t h^s$ and $t = H_\nu(pk_C||\sigma_1||\sigma_2||\sigma_3||m)$.

Security The unforgeability of our core confirmer signature scheme rests on the unforgeability of the signature scheme by Waters [35], defined in Section 2.

Theorem 22 *Assume that Waters' signature scheme W is unforgeable, $\mathcal{H}$ is a collision resistant hash function family, and the discrete logarithm problem is computationally hard with respect to the group generator $\mathcal{G}$ implicitly defined by* `Setup`. *Then the above core confirmer signature scheme CS is unforgeable. More specifically, there exists polynomial time algorithms, $\mathcal{B}$, $\mathcal{B}'$ and $\mathcal{B}''$ such that for any polynomial time adversary $\mathcal{A}$, $\mathtt{Adv}_{CS,\mathcal{A}}^{cs-uf-cma}(k) \leq \mathtt{Adv}_{W,\mathcal{B}}^{wuf-cma}(k) + \mathtt{Adv}_{\mathcal{H},\mathcal{B}'}^{col}(k) + \mathtt{Adv}_{\mathcal{G},\mathcal{B}''}^{dl}(k)$.*

The proof of the above theorem follows a very similar strategy to the proof of strong unforgeability of the signature scheme by Boneh *et al.* [2] based on Waters' signature scheme, and we leave out the details here. Note that Waters' signature scheme is proved unforgeable assuming the computational Diffie-Hellman problem is hard [35], and that collision resistant hash functions can be constructed using this assumption as well [13] i.e. the unforgeability of the above core confirmer signature scheme can be reduced to the computational Diffie-Hellman problem only.

The proofs of the following two theorems, showing invisibility and token soundness of the above core confirmer signature scheme, can be found in Appendix C.1 and C.2, respectively.

Theorem 23 *Assume that the above core confirmer signature scheme CS is unforgeable and that the decisional linear problem is hard with respect to the group generator $\mathcal{G}$ implicitly defined by* `Setup`. *Then CS is invisible. More specifically, there exists polynomial time algorithms $\mathcal{B}$ and $\mathcal{B}'$ such that for any polynomial time adversary $\mathcal{A}$, $\mathtt{Adv}_{CS,\mathcal{A}}^{cs-inv-cma}(k) \leq \mathtt{Adv}_{CS,\mathcal{B}}^{cs-uf-cma}(k) + \mathtt{Adv}_{\mathcal{G},\mathcal{B}'}^{dlin}(k)$.*

Theorem 24 *The above core confirmer signature scheme has token soundness.*

Compatible sigma protocols. To be able to use the above core confirmer signature scheme in the construction of a concrete ONS scheme, we need to provide compatible sigma protocols Σ_S, Σ_C and $\overline{\Sigma}_C$. Let $pk_S = (k, g_1, g_2, h, u_0, \ldots, u_n)$, $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, s)$, and let $(a, b, r) \in \mathbb{Z}_p^3$ be the randomness used to construct σ. Then, for the concrete core confirmer signature scheme, Σ_S can be described as follows

$$\Sigma_S\{((par, pk_S, pk_C, m, \sigma), (\alpha, a, b, r)) : g^\alpha = g_1 \wedge u^a = \sigma_1 \wedge v^b = \sigma_2 \wedge g^{a+b+r} = \sigma_3 \wedge g_2^\alpha F(M)^r = \sigma_4\}$$

where $M = g^t h^s$ and $t = H_\nu(pk_C||\sigma_1||\sigma_2||\sigma_3||m)$. This sigma protocol can be implemented using standard and well-known techniques for proving knowledge of discrete logarithms (e.g. see [6]).

The protocol Σ_C can be described as follows.

$$\Sigma_C\{((par, pk_S, pk_C, m, \sigma), (x, y)) : u^x = g \wedge v^y = g \wedge$$
$$e(\sigma_1, F(M))^x e(\sigma_2, F(M))^y = e(g_1, g_2)e(\sigma_3, F(M))/e(g, \sigma_4)\}$$

where $M = g^t h^s$ and $t = H_\nu(pk_C||\sigma_1||\sigma_2||\sigma_3||m)$. Note that if σ is converted using $sk_C = (x, y)$, the token will be of the form $tk_\sigma = (tk_1, tk_2) = (\sigma_1^x, \sigma_2^y)$ and the first two equations will guarantee that $e(u, tk_1) = e(g, \sigma_1)$ and $e(v, tk_2) = e(g, \sigma_2)$. Furthermore, the last equation above can be re-written as $e(tk_1 tk_2, F(M)) = e(g_1, g_2)e(\sigma_3, F(M))/e(g, \sigma_4)$. Hence, if the above protocol is sound, all the equations of `TkVerify` are guaranteed to hold when the conversion is done using $sk_C = (x, y)$ i.e. `Valid`$(par, pk_S, pk_C, m, \sigma, sk_C) = $ `accept`. Like Σ_S, this sigma protocol can be implemented using standard protocols for proving knowledge of discrete logarithms.

Lastly, the protocol $\overline{\Sigma}_C$ can be described as follows.

$$\Sigma_C\{((par, pk_S, pk_C, m, \sigma), (x, y)) : u^x = g \wedge v^y = g \wedge$$
$$e(\sigma_1, F(M))^x e(\sigma_2, F(M))^y \neq e(g_1, g_2)e(\sigma_3, F(M))/e(g, \sigma_4)\}$$

where $M = g^t h^s$ and $t = H_\nu(pk_C||\sigma_1||\sigma_2||\sigma_3||m)$. As above, the soundness of the above protocol guarantees that `TkVerify`$(par, pk_S, pk_C, m, \sigma, tk_\sigma) = $ `reject` if tk_σ is generated using sk_C i.e. `Valid`$(par, pk_S, pk_C, m, \sigma, sk_C) = $ `reject`. To implement the protocol, we make use of a technique by Camenish and Shoup [5] for proving inequality between discrete logarithms. More specifically, let $e_1 = e(\sigma_1, F(M))$, $e_2 = e(\sigma_2, F(M))$ and $e_3 = e(g_1, g_2)e(\sigma_3, F(M))/e(g, \sigma_4)$. A prover and verifier then interact as follows

– The prover picks random $r \leftarrow \mathbb{Z}_p$, computes $C \leftarrow (e_1^x e_2^y / e_3)^r$ and sends C to the verifier.
– The prover and verifier then interact in the protocol $\Sigma\{(\alpha, \beta, \gamma) : u^\alpha g^{-\gamma} = 1 \wedge v^\beta g^{-\gamma} = 1 \wedge e_1^\alpha e_2^\beta e_3^{-\gamma} = C\}$ where $\alpha = xr$, $\beta = yr$ and $\gamma = r$.

The value C can be sent together with the first message of the sigma protocol in the second step to obtain a 3-move protocol. To see that the above protocol is special honest verifier zero-knowledge and has special soundness, similar arguments to [5] can be used, and we refer the reader to [5] for the details.

7 Comparison

The generic construction by Liskov and Micali [26] provides many instantiation options due to their use of standard primitives, whereas our approach relies on special building blocks. However, our concrete instantiation has several advantages compared to any instantiation of the scheme by Liskov and Micali, both in terms of functionality, efficiency and security. More specifically, our scheme allows a confirmer to both confirm and disavow signatures whereas [26] only allows the latter, and our scheme furthermore provides publicly verifiably conversion of signatures as opposed to signature extraction. Secondly, our scheme provides short signatures compared to the $\mathcal{O}(k)$ size signatures of [26] which results in the additional advantage of a more efficient sign protocol in which the verifier obtains a signature from the signer. However, we note that our scheme requires large public keys due to the use of Waters signatures, whereas [26] can be instantiated to provide compact public keys. Lastly, all interactive protocols of our scheme are on-line non-transferable, security is guaranteed when multiple (potentially malicious) confirmers are used, and the signer is not required to engage in any "fake" signing protocols to maintain security. As mentioned in the introduction, these security properties are not enjoyed by [26]. However, it should be noted that the non-interactive disavow protocol of [26] allows perfect non-repudiation whereas our scheme only achieves computational non-repudiation.

References

1. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew K. Franklin, editor, *CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.
2. Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational diffie-hellman. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2006.
3. Emmanuel Bresson, Dario Catalano, and Rosario Gennaro. Improved on-line/off-line threshold signatures. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2007.
4. Jan Camenisch and Markus Michels. Confirmer signature schemes secure against adaptive adversaries. In *EUROCRYPT*, pages 243–258, 2000.
5. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2003.
6. Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Institute for Theoretical Computer Science, ETH Zurich, March 1997.
7. Dario Catalano, Mario Di Raimondo, Dario Fiore, and Rosario Gennaro. Off-line/on-line signatures: Theoretical aspects and experimental results. In Cramer [12], pages 101–120.
8. David Chaum. Zero-knowledge undeniable signatures. In *EUROCRYPT*, volume 473 of *LNCS*, pages 458–464. Springer, 1990.
9. David Chaum. Designated confirmer signatures. In *EUROCRYPT*, pages 86–91, 1994.
10. David Chaum and Hans Van Antwerpen. Undeniable signatures. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 212–216. Springer, 1989.
11. Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

12. Ronald Cramer, editor. *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography. Proceedings*, volume 4939 of *Lecture Notes in Computer Science*. Springer, 2008.
13. Ivan Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, volume 304 of *LNCS*, pages 203–216. Springer, 1987.
14. Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT*, pages 418–430, 2000.
15. Steven D. Galbraith and Wenbo Mao. Invisibility and anonymity of undeniable and confirmer signatures. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2003.
16. Steven D. Galbraith, Wenbo Mao, and Kenneth G. Paterson. Rsa-based undeniable signatures for general moduli. In Bart Preneel, editor, *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 2002.
17. Craig Gentry, David Molnar, and Zulfikar Ramzan. Efficient designated confirmer signatures without random oracles or general zero-knowledge proofs. In Bimal Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 662–681. Springer, 2005.
18. Shafi Goldwasser and Erez Waisbard. Transformation of digital signature schemes into designated confirmer signature schemes. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 77–100. Springer, 2004.
19. Xinyi Huang, Yi Mu, Willy Susilo, and Wei Wu. Provably secure pairing-based convertible undeniable signature with short signature length. In Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto, editors, *Pairing*, volume 4575 of *Lecture Notes in Computer Science*, pages 367–391. Springer, 2007.
20. Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In *EUROCRYPT*, pages 143–154, 1996.
21. Caroline Kudla and Kenneth G. Paterson. Non-interactive designated verifier proofs and undeniable signatures. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 136–154. Springer, 2005.
22. Kaoru Kurosawa and Swee-Huay Heng. 3-move undeniable signature scheme. In Cramer [11], pages 181–197.
23. Fabien Laguillaumie, Benoît Libert, and Jean-Jacques Quisquater. Universal designated verifier signatures without random oracles or non-black box assumptions. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2006.
24. Fabien Laguillaumie and Damien Vergnaud. Designated verifier signatures: Anonymity and efficient construction from any bilinear map. In Carlo Blundo and Stelvio Cimato, editors, *SCN*, volume 3352 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2004.
25. Benoît Libert and Jean-Jacques Quisquater. Identity based undeniable signatures. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 112–125. Springer, 2004.
26. Moses Liskov and Silvio Micali. Online-untransferable signatures. In Cramer [12], pages 248–267.
27. Markus Michels and Markus Stadler. Generic constructions for secure and efficient confirmer signature schemes. In *EUROCRYPT*, pages 406–421, 1998.
28. Jean Monnerat, Sylvain Pasini, and Serge Vaudenay. Efficient deniable authentication for signatures. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 272–291, 2009.
29. Jean Monnerat and Serge Vaudenay. Short 2-move undeniable signatures. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2006.
30. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.
31. Jacob C. N. Schuldt and Kanta Matsuura. An efficient convertible undeniable signature scheme with delegatable verification. In Jin Kwak, Robert H. Deng, Yoojae Won, and Guilin Wang, editors, *ISPEC*, volume 6047 of *Lecture Notes in Computer Science*, pages 276–293. Springer, 2010.
32. Ron Steinfeld, Laurence Bull, Huaxiong Wang, and Josef Pieprzyk. Universal designated-verifier signatures. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 523–542. Springer, 2003.
33. Ron Steinfeld, Huaxiong Wang, and Josef Pieprzyk. Efficient extension of standard schnorr/rsa signatures into universal designated-verifier signatures. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2004.

34. Isamu Teranishi, Takuro Oyama, and Wakaha Ogata. General conversion for obtaining strongly existentially unforgeable signatures. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2006.
35. Brent Waters. Efficient identity-based encryption without random oracles. In Cramer [11], pages 114–127.
36. Douglas Wikström. Designated confirmer signatures revisited. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 342–361. Springer, 2007.

A Proof of Theorem 15

Proof. Given an adversary $\mathcal{A}$ that breaks the binding under selective trapdoor openings of the commitment scheme, we construct an algorithm $\mathcal{B}$ that solves the discrete logarithm problem.

$\mathcal{B}$ is given the group $\mathbb{G}$ of order p, a generator g, and an element h, and is required to compute the discrete logarithm $x = \log_g h$. Firstly, $\mathcal{B}$ flips a random coin b. If $b = 0$, $\mathcal{B}$ randomly picks $x_1 \leftarrow \mathbb{Z}_p$ and sets $h_1 \leftarrow g^{x_1}$ and $h_2 \leftarrow h$. Otherwise, $\mathcal{B}$ randomly picks $x_2 \leftarrow \mathbb{Z}_p$ and sets $h_1 \leftarrow h$ and $h_2 \leftarrow g^{x_2}$. Then $\mathcal{B}$ runs $\mathcal{A}$ with input $ck = (\mathbb{G}, p, g, h_1, h_2)$.

If $\mathcal{A}$ makes a commit query to $\mathcal{O}_c$, $\mathcal{B}$ responds by randomly picking $r_g, r_h \leftarrow \mathbb{Z}_p$ and returning the commitment $c \leftarrow g^{r_g} h^{r_h}$. Note that c is distributed as an ordinary commitment. The values (r_g, r_h) are stored for later use.

If $\mathcal{A}$ makes an open query to $\mathcal{O}_o$ with value w and a commitment $c = g^{r_g} h^{r_h}$ obtained through a previous commit query, $\mathcal{B}$ responds as follows: Firstly, $\mathcal{B}$ retrieves (r_g, r_h) for c (note that this is always possible since c must be obtained through $\mathcal{O}_c$). Then if $b = 0$, $\mathcal{B}$ computes $r_1 \leftarrow (r_g - w)/x_1$, sets $r_2 \leftarrow r_h$ and returns (w, r_1, r_2). Otherwise, if $b = 1$, $\mathcal{B}$ sets $r_1 \leftarrow r_h$, computes $r_2 \leftarrow (r_g - w)/x_2$, and returns (w, r_1, r_2). Note that since (r_g, r_h) was picked uniformly at random, (r_1, r_2) will also be uniformly distributed in $\mathbb{Z}_p$ in both cases.

Eventually $\mathcal{A}$ outputs $(w, r_1, r_2, w', r'_1, r'_2)$ such that $g^w h_1^{r_1} h_2^{r_2} = g^{w'} h_1^{r'_1} h_2^{r'_2}$ and $w \neq w'$. This implies that either $r_1 \neq r'_1$ or $r_2 \neq r'_2$. In the first case, $\mathcal{B}$ can recover the discrete logarithm of h_1 w.r.t. g as $x_1 \leftarrow (w - w' + x_2(r_2 - r'_2))/(r'_1 - r_1)$ assuming $b = 1$ (i.e. we have $h_2 = g^{x_2}$). In the second case, $\mathcal{B}$ can recover the discrete logarithm of h_2 w.r.t. g as $x_2 \leftarrow (w - w' + x_1(r_1 - r'_1))/(r'_2 - r_2)$ assuming $b = 0$ (i.e. we have $h_1 = g^{x_1}$). Since b is completely hidden from $\mathcal{A}$, $\mathcal{B}$ will recover the discrete logarithm of h with probability $1/2$ and hence solve the given discrete logarithm problem. □

B ONS Scheme Security Proofs

B.1 Proof of Theorem 16

Proof. We prove the theorem by considering the following sequence of games:

$Game_0$ This is the original $\texttt{Exp}_{N,\mathcal{A}}^{\texttt{uf-cma}}(1^k)$ experiment.

$Game_1$ In this game we change how $\mathcal{A}$'s sign queries are handled. More specifically, when interacting with $\mathcal{A}$ in the $\texttt{NT}(pk_V)\text{-}\Sigma_S$ protocol, the trapdoor functionality of T is used as follows: Firstly, the private key sk_V corresponding to pk_V is retrieved from $L_{VKeyReg}$. Then, instead of sending $com \leftarrow \texttt{T.Comm}(pk_V, a; r)$ where a is the first message of Σ_S to $\mathcal{A}$, we compute $(com', aux) \leftarrow \texttt{T.TdCom}(pk_V)$, store aux, and send com' to $\mathcal{A}$. When a challenge c is received from $\mathcal{A}$, we compute $r' \leftarrow \texttt{T.TdOpen}(aux, a, sk_V)$, and return (a, r', z) where z is the last message of Σ_S. Note that since a is not needed until $\mathcal{A}$ submits his challenge c, the computation of a can be delayed until this happens.

$Game_2$ In this game, we replace the honestly generated messages of Σ_S with the simulated messages output by $\texttt{Sim}_{\Sigma_S}$ i.e. when a challenge c is received from $\mathcal{A}$, we compute the messages $(a', z') \leftarrow \texttt{Sim}_{\Sigma_S}((par, pk_S, pk'_C, pk_C || pk_{C,S} || m, \sigma), c)$ and $r' \leftarrow \texttt{T.TdOpen}(aux, a', sk_V)$, and returns (a', r', z') to $\mathcal{A}$.

Let $Succ_i$ be the event that $\mathcal{A}$ wins in $Game_i$. Furthermore, let $Forge_1$ be the event that $\mathcal{A}$ wins by successfully completing the confirm protocols in $Game_2$ (i.e. $z = \texttt{accept}$) and let $Forge_2$ be the event that $\mathcal{A}$ wins by producing a token tk^* such that the output of $\texttt{TkVerify}$ is accept in $Game_2$ (i.e. $z' = \texttt{accept}$). Note that if $\mathcal{A}$ wins in $Game_2$, either $Forge_1$ or $Forge_2$ must occur i.e. $\Pr[Succ_2] \leq \Pr[Forge_1] + \Pr[Forge_2]$. The advantage of $\mathcal{A}$ can be expressed as follows:

$$\Pr[Succ_0] \leq |\Pr[Succ_0] - \Pr[Succ_1]| + |\Pr[Succ_1] - \Pr[Succ_2]| + \Pr[Forge_1] + \Pr[Forge_2]$$

Claim 25 $\Pr[Succ_0] = \Pr[Succ_1]$

Proof. The only difference between $Game_0$ and $Game_1$ is how the commitments in $\mathcal{A}$'s sign queries are computed; in $Game_0$, these are honestly computed using the commitment key pk_V, whereas in $Game_1$, the trapdoor functionality of T is used in combination with the trapdoor sk_V retrieved from $L_{VKeyReg}$. Note that it is assumed (pk_V, sk_V) was previously submitted to $\mathcal{O}_{VKeyReg}$ and that (pk_V, sk_V) is a valid key pair. Since T is assumed to provide a perfect trapdoor property, the distribution of honestly computed commitments and openings is identical to the distribution of commitments and openings computed using the trapdoor functionality, and the view of $\mathcal{A}$ must be identical in $Game_0$ and $Game_1$. Hence, the success probability of $\mathcal{A}$ must also be identical in $Game_0$ and $Game_1$. $\square$

Claim 26 $\Pr[Succ_1] = \Pr[Succ_2]$

Proof. The only difference between $Game_1$ and $Game_2$ is that in the last message of the sign protocol in $\mathcal{A}$'s sign queries, $\mathcal{A}$ will receive messages from an honest run of Σ_S in $Game_1$, whereas in $Game_2$, $\mathcal{A}$ will receive the simulated messages computed by $\texttt{Sim}_{\Sigma_S}$. Since Σ_S is assumed to be honest verifier zero knowledge, the distribution of these messages will be identical, and hence, $\mathcal{A}$'s view and success probability in $Game_1$ and $Game_2$ will be identical. $\square$

Claim 27 $\Pr[Forge_1] \leq \sqrt{4\texttt{Adv}_{CS,\mathcal{B}}^{cs-uf-cma} + 1/p^2} + \texttt{Adv}_{T,\mathcal{B}'}^{bind} + 1/p$

Proof. Assume an adversary $\mathcal{A}$ with advantage $\epsilon_\mathcal{A} = \Pr[Forge_1]$ exists i.e. $\mathcal{A}$ wins in $Game_2$ by completing the confirm protocol with probability $\epsilon_\mathcal{A}$. From $\mathcal{A}$, we will construct either an algorithm $\mathcal{B}$ that breaks the unforgeability of the underlying core confirmer signature scheme, or an algorithm $\mathcal{B}$' which breaks the binding under selective trapdoor openings property of T.

$\mathcal{B}$ is constructed as follows: Firstly, $\mathcal{B}$ receives parameters par_{CS} and a public signer key pk_S for the core confirmer signature scheme, then generates $par_S \leftarrow \texttt{S.Setup}(1^k)$ and $(pk_V, sk_V) \leftarrow \texttt{T.G}(1^k)$, sets $par \leftarrow (par_S, par_{CS})$, and runs $\mathcal{A}$ with input (par, pk_S, pk_V). $\mathcal{B}$ answer $\mathcal{A}$'s oracle quires as follows

- $\mathcal{O}_{Sign}$: given $(pk_C, pk_{C,S}, pk_V, m)$, $\mathcal{B}$ submits $pk_C||pk_{C,S}||m$ to his own signing oracle and obtains a signature σ on $pk_C||pk_{C,S}||m$ under pk_S. $\mathcal{B}$ then sends σ to $\mathcal{A}$ and interacts with $\mathcal{A}$ as described in $Game_2$ using the trapdoor sk_V and simulated messages generated with $\texttt{Sim}_{\Sigma_S}$.
- $\mathcal{O}_{SimSign}$, $\mathcal{O}_{SimCon}$ and $\mathcal{O}_{SimDis}$: $\mathcal{B}$ interacts with $\mathcal{A}$ as an honest unforgeability challenger using his knowledge of sk_V.

At some point, $\mathcal{A}$ will output $(pk_C^*, pk_{C,S}^*, m^*, \sigma^*)$ and interact with $\mathcal{B}$ in the confirm protocol. Upon completion of the protocol, $\mathcal{B}$ obtains a transcript (com, c, a, z, r) of the $\texttt{NT}(pk_V)$-Σ_C protocol. At that point, $\mathcal{B}$ rewinds $\mathcal{A}$ to the point where $\mathcal{A}$ sent the message com to $\mathcal{B}$, and then replays $\mathcal{A}$ with a new randomly chosen challenge value c' as a response to com. Since we assume that $\mathcal{A}$ will successfully complete the protocol with probability $\epsilon_\mathcal{A}$, a standard rewinding argument shows that with probability $\epsilon_\mathcal{A}^2/4 - \epsilon_\mathcal{A}/2p$ where p is the size of the challenge space, $\mathcal{A}$ will successfully complete both runs of the protocol using the same random tape. More specifically, following [14], consider a matrix in which each row corresponds to the random tape used by $\mathcal{A}$, and each column corresponds to a challenge value c. Let an entry in the matrix be "1" if $\mathcal{A}$ successfully completes the protocol with the corresponding random tape and challenge value, and "0" otherwise. Since $\mathcal{A}$ is assumed to succeed with probability $\epsilon_\mathcal{A}$, the average number of ones in a row must be $\epsilon_\mathcal{A} p$ where

p is the size of the challenge space. Define a row to be heavy if it contains more than $\epsilon_{\mathcal{A}} p/2$ ones. It follows that at least half of all ones in the matrix must be in heavy rows i.e. with probability $\epsilon_{\mathcal{A}}/2$, $\mathcal{A}$ will successfully complete the protocol and the random tape used by $\mathcal{A}$ will correspond to a heavy row. In this case, the row will contain at least $\epsilon_{\mathcal{A}} p/2 - 1$ other challenge values for which $\mathcal{A}$ will successfully complete the protocol i.e. if we rewind $\mathcal{A}$ using the same random tape and provide $\mathcal{A}$ with a different randomly chosen challenge value c', with probability $\epsilon_{\mathcal{A}}/2 - 1/p$, $\mathcal{A}$ completes the second run of the protocol successfully. Hence, with probability $\epsilon_{\mathcal{A}}^2/4 - \epsilon_{\mathcal{A}}/2p$, $\mathcal{B}$ will obtain two transcripts, (com, c, a, z, r) and (com', c', a', z', r'), where $com = com'$.

Now, if $a \neq a'$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ computes $sk'_C \leftarrow \mathtt{WExt}_{\Sigma_C}(a, c, c', z, z')$ that, by the definition of Σ_C, must satisfy $\mathtt{CS.Valid}(par_{CS}, pk_S, pk'_C, pk^*_C || pk^*_{C,S} || m^*, \sigma^*, sk'_C) = \mathtt{accept}$ where $pk^*_{C,S} = (pk'_C, \delta)$ i.e. generating the token $tk_{\sigma^*} \leftarrow \mathtt{CS.Convert}(par_{CS}, pk_S, pk'_C, pk^*_C || pk^*_{C,S} || m^*, \sigma^*, sk'_C)$ and checking validity by running $\mathtt{CS.TkVerify}(par_{CS}, pk_S, pk'_C, pk^*_C || pk^*_{C,S} || m^*, \sigma^*, tk_{\sigma^*})$ will yield $\mathtt{accept}$. Lastly, $\mathcal{B}$ returns $(pk^*_C || pk^*_{C,S} || m^*, \sigma^*, tk^*_\sigma)$ as his own forgery. Note that, since it is required that $(pk^*_C, pk^*_{C,S}, m^*, \sigma^*) \notin L_{Sign}$, $\mathcal{B}$ will not have received σ^* as a response to one of his own sign queries on $pk^*_C || pk^*_{C,S} || m^*$, and $\mathcal{B}$'s forgery will be a valid forgery. Conditioned on $\mathcal{A}$ not causing $\mathcal{B}$ to abort, the advantage of $\mathcal{B}$ is $\mathtt{Adv}^{\mathtt{cs-uf-cma}}_{CS,\mathcal{B}} \geq \epsilon_{\mathcal{A}}^2/4 - \epsilon_{\mathcal{A}}/2p$.

Let *Abort* be the event that $\mathcal{B}$ aborts in the above simulation. We have that

$$\Pr[Forge_1] \leq \Pr[Forge_1 | \overline{Abort}] + \Pr[Abort]$$

From the above construction of $\mathcal{B}$, we conclude that $\Pr[Forge_1 | \overline{Abort}] \leq \sqrt{4\mathtt{Adv}^{\mathtt{cs-uf-cma}}_{CS,\mathcal{B}} + 1/p^2} + 1/p$. To complete the proof of the claim, we will show that if the event *Abort* happens, we can construct an algorithm $\mathcal{B}'$ which breaks the binding under selective trapdoor openings property of the commitment scheme T.

We construct the algorithm $\mathcal{B}'$ as follows: Firstly, $\mathcal{B}'$ receives a commitment key $ck = pk_V$, generates parameters $par_S \leftarrow \mathtt{S.Setup}_S(1^k)$ and $par_{CS} \leftarrow \mathtt{CS.Setup}_{CS}(1^k)$, sets $par \leftarrow (par_S, par_{CS})$, compute $(pk_S, sk_S) \leftarrow \mathtt{S.KeyGen}_S(par_{CS})$, and runs $\mathcal{A}$ with input (par, pk_S, pk_V). $\mathcal{B}'$ answers $\mathcal{A}$'s queries as follows

- $\mathcal{O}_{Sign}$: given $(pk_C, pk_{C,S}, m)$ where $pk_{C,S} = (pk'_C, \delta)$, $\mathcal{B}'$ uses his knowledge of sk_S to construct a valid signature $\sigma \leftarrow \mathtt{CS.Sign}(par_{CS}, pk'_C, pk_C || pk_{C,S} || m, sk_S)$ which is sent to $\mathcal{A}$. Then, $\mathcal{B}'$ uses his oracles $\mathcal{O}_c$ and $\mathcal{O}_o$ to interact with $\mathcal{A}$ in the simulated sign protocol as described in $Game_2$. More specifically, $\mathcal{B}'$ interacts with $\mathcal{A}$ as follows:
  1. $\mathcal{B}'$ queries $\mathcal{O}_c$, obtains com, and sends com to $\mathcal{A}$.
  2. When a challenge c is received from $\mathcal{A}$, $\mathcal{B}'$ obtains a simulated transcript of Σ_S by running $(a, z) \leftarrow \mathtt{Sim}_{\Sigma_S}(par_{CS}, pk_S, pk'_C, pk_C || pk_{C,S} || m, \sigma, c)$, submits (com, a) to $\mathcal{O}_o$ to obtain r such that $com = \mathtt{T.Comm}(pk_V, a; r)$, and lastly sends (a, z, r) to $\mathcal{A}$.
- $\mathcal{O}_{SimSign}$: given $(pk_C, pk_{C,S}, m)$, $\mathcal{B}'$ picks a random signature $\sigma \leftarrow \mathcal{S}$, sends σ to $\mathcal{A}$, and interacts with $\mathcal{A}$ as described in step 1 and 2 of the simulation of $\mathcal{O}_{Sign}$ above.
- $\mathcal{O}_{SimCon}$, $\mathcal{O}_{SimDis}$: given $(pk_S, pk_C, pk_{C,S})$, $\mathcal{B}'$ simulates these oracles by executing step 1 and 2 from the simulation of $\mathcal{O}_{Sign}$.

At some point, $\mathcal{A}$ outputs $(pk^*_C, pk^*_{C,S}, m^*, \sigma^*)$ and interacts with $\mathcal{B}'$ in the confirm protocol. Like above, $\mathcal{B}'$ first completes the protocol, then rewinds and re-plays $\mathcal{A}$ to obtain two transcripts (com, c, a, z, r) and (com', c', a', z', r') of $\mathtt{NT}(pk_V)\text{-}\Sigma_C$ where $com = com'$. Note that $\mathcal{A}$'s view will distributed identically to the view when interacting with $\mathcal{B}$. If *Abort* occurs (i.e. $a \neq a'$), $\mathcal{B}'$ returns (com, a, r, a', r') as an attack against the binding property of T. Note that if (com, c, a, z, r) and (com', c', a', z', r') are accepting transcripts, $\mathtt{T.Comm}(pk_V, a, r) = com = com' = \mathtt{T.Comm}(pk_V, a', r')$ i.e. $\mathcal{B}'$ successfully attacks the binding under selective openings property of T.

From the above construction of $\mathcal{B}'$ we conclude that $\Pr[Abort] \leq \mathtt{Adv}^{\mathtt{bind}}_{\mathcal{B}',T}$. Combining this with the above yields the result stated in the claim. $\square$

Claim 28 $\Pr[Forge_2] \leq \mathtt{Adv}^{\mathtt{cs-uf-cma}}_{CS,\mathcal{B}}$

Proof. Assume there exists an algorithm $\mathcal{A}$ with success probability $\epsilon_{\mathcal{A}} = \Pr[Forge_2]$. Using $\mathcal{A}$, we construct an algorithm $\mathcal{B}$ that breaks the unforgeability of CS with probability $\epsilon_{\mathcal{A}}$.

$\mathcal{B}$ receives parameters par_{CS} and a public key pk_S, generates $par_S \leftarrow \mathtt{S.Setup}_S(1^k)$, $par \leftarrow (par_S, par_{CS})$, and $(pk_V, sk_V) \leftarrow \mathtt{T.G}_V(1^k)$, and runs $\mathcal{A}$ with input (par, pk_S, pk_V). $\mathcal{B}$ then simulates the oracles $\mathcal{O}_{Sign}$, $\mathcal{O}_{SimSign}$, $\mathcal{O}_{SimCon}$ and $\mathcal{O}_{SimDis}$ as in the first part of Claim 27. Lastly, $\mathcal{A}$ outputs a tuple $(pk_C^*, pk_{C,S}^*, m^*, \sigma^*, tk^*)$ where $pk_{C,S}^* = (pk_C', \delta)$. If $Forge_2$ occurs, we must have that $\mathtt{CS.TkVerify}(par_{CS}, pk_S, pk_C', pk_C^* || pk_{C,S}^* || m^*, \sigma^*, tk^*) = \mathtt{accept}$ and that σ^* was not returned as a response to a sign query on $(pk_C^*, pk_{C,S}^*, m^*)$. Hence, $\mathcal{B}$ returns $(pk_C', pk_C^* || pk_{C,S}^* || m^*, \sigma^*, tk_\sigma^*)$ which will be a valid forgery of CS with probability $\epsilon_{\mathcal{A}}$. □

The theorem follows by the combination of the above claims. □

B.2 Proof of Theorem 18

Proof. Assume there exists an adversary $\mathcal{A}$ with success probability $\epsilon_{\mathcal{A}}$ in the non-repudiation experiment. We will shown that, using $\mathcal{A}$, an adversary against the binding property of the trapdoor commitment scheme can be constructed. To this end, first consider the following simulator $\mathcal{B}$ interacting with $\mathcal{A}$ in the non-repudiation experiment.

Firstly, $\mathcal{B}$ generates $par_S \leftarrow \mathtt{S.Setup}(1^k)$, $par_{CS} \leftarrow \mathtt{CS.Setup}(1^k)$, $par \leftarrow (par_S, par_{CS})$ and $(pk_V, sk_V) \leftarrow \mathtt{T.G}(1^k)$ as an honest challenger. Then $\mathcal{B}$ runs $\mathcal{A}$ with input (par, pk_V) and responds honestly to $\mathcal{A}$'s queries to the $\mathcal{O}_{SimSign}$, $\mathcal{O}_{SimCon}$ and $\mathcal{O}_{SimDis}$ oracles using sk_V. At some point, $\mathcal{A}$ outputs $(pk_S, pk_C, pk_{C,S}, m)$, where $pk_{C,S} = (pk_C', \delta)$, and then interacts with $\mathcal{B}$ in the sign protocol i.e. $\mathcal{A}$ will send a signature σ to $\mathcal{B}$, and then interact with $\mathcal{B}$ in the $\mathtt{NT}(pk_V)\text{-}\Sigma_S$ protocol. With probability at least $\epsilon_{\mathcal{A}}$, $\mathcal{A}$ successfully completes $\mathtt{NT}(pk_V)\text{-}\Sigma_S$. At this point, $\mathcal{B}$ rewinds and replays $\mathcal{A}$, but provide a different randomly chosen challenge value in the $\mathtt{NT}(pk_V)\text{-}\Sigma_S$ protocol. As in the proof of Claim 27, a standard rewinding argument shows that, with probability at least $\epsilon_{\mathcal{A}}^2/4 - \epsilon_{\mathcal{A}}/2p$, $\mathcal{A}$ will successfully complete the protocol with the new challenge value, and $\mathcal{B}$ obtains two accepting transcripts (com, a, c, z, r) and (com', a', c', z', r') of $\mathtt{NT}(pk_V)\text{-}\Sigma_S$ where $com = com'$ (note also that the signature σ sent in the first move of the protocol will be the same for both runs). If $a \neq a'$, $\mathcal{B}$ will abort. This event will be denoted $Abort_1$. Otherwise, by the special soundness of Σ_S, $\mathcal{B}$ can compute $(sk_S, r) \leftarrow \mathtt{WExt}_{\Sigma_S}(a, c, c', z, z')$ such that $(pk_S, sk_S) \in \{\mathtt{CS.KeyGen}_S(par_{CS})\}$ and $\sigma = \mathtt{CS.Sign}(par_{CS}, pk_C', pk_C || pk_{C,S} || m, sk_S; r)$.

After completing the sign protocol, $\mathcal{A}$ interacts with $\mathcal{B}$ in the disavow protocol by running $\mathtt{NT}(pk_V)\text{-}\overline{\Sigma}_C$. Given that $\mathcal{A}$ successfully completed the sign protocol, $\mathcal{A}$ will successfully complete $\mathtt{NT}(pk_V)\text{-}\overline{\Sigma}_C$ with probability at least $\epsilon_{\mathcal{A}}$ since it is assumed $\mathcal{A}$ will succeed in wining the entire security experiment with probability $\epsilon_{\mathcal{A}}$. Note also that the distribution of $\mathcal{A}$'s view before executing $\mathtt{NT}(pk_V)\text{-}\overline{\Sigma}_C$ is independent of whether $\mathcal{A}$ was rewinded in the $\mathtt{NT}(pk_V)\text{-}\Sigma_S$ protocol or not, since, in either case, a uniformly random challenge value is provided. Upon completion of $\mathtt{NT}(pk_V)\text{-}\overline{\Sigma}_C$, $\mathcal{B}$ rewinds and replays $\mathcal{A}$, but provides a different randomly chosen challenge value in the second run. Like above, with probability $\epsilon_{\mathcal{A}}^2/4 - \epsilon_{\mathcal{A}}/2p$, $\mathcal{B}$ obtains two accepting transcript, (com, a, c, z, r) and (com', a', c', z', r'), of $\mathtt{NT}(pk_V)\text{-}\overline{\Sigma}_C$ where $com = com'$. Again, if $a \neq a'$, $\mathcal{B}$ aborts. This event will be denoted $Abort_2$. Otherwise, by the special soundness property of $\overline{\Sigma}_C$, $\mathcal{B}$ can compute $sk_C' \leftarrow \mathtt{WExt}_{\overline{\Sigma}_C}(a, c, c', z, z')$ such that $(pk_C', sk_C') \in \{\mathtt{CS.KeyGen}_C(par_{CS})\}$ and $\mathtt{CS.Valid}(par_{CS}, pk_S, pk_C', m, \sigma, sk_C') = \mathtt{reject}$.

Assuming that neither $Abort_1$ nor $Abort_2$ occur and that $\epsilon_{\mathcal{A}} > 2/p$, $\mathcal{B}$ will, with non-zero probability, obtain σ, sk_S, r and sk_C' with the above stated properties. However, this directly lead to a contradiction. More specifically, the correctness of CS implies that since $(pk_S, sk_S) \in \{\mathtt{CS.KeyGen}_S(par_{CS})\}$, $(pk_C', sk_C') \in \{\mathtt{CS.KeyGen}_C(par_{CS})\}$ and $\sigma = \mathtt{CS.Sign}(par_{CS}, pk_C', m, sk_S; r)$, then we must have that $\mathtt{CS.Valid}(par_{CS}, pk_S, pk_C', m, \sigma, sk_C') = \mathtt{accept}$. This contradicts the above established property that $\mathtt{CS.Valid}(par_{CS}, pk_S, pk_C', m, \sigma, sk_C') = \mathtt{reject}$. We conclude that conditioned on neither $Abort_1$ nor $Abort_2$ occurring, we must have $\epsilon_{\mathcal{A}} \leq 2/p$.

Let $Succ$ denote the event that $\mathcal{A}$ succeeds in the non-repudiation experiment. We have that

$$\mathtt{Adv}_{N,\mathcal{A}}^{\mathtt{non\text{-}rep}} = \Pr[Succ] \leq \Pr[Succ | \overline{Abort_1} \wedge \overline{Abort_2}] + \Pr[Abort_1] + \Pr[Abort_2]$$

From the above we conclude that $\Pr[Succ|\overline{Abort_1} \wedge \overline{Abort_2}] \leq 2/p$. By constructing simulators almost identical to the simulator $\mathcal{B}$' in the proof of Claim 27, we will obtain $\Pr[Abort_1] \leq \mathtt{Adv}^{\mathtt{bind}}_{T,\mathcal{B}'}$ and $\Pr[Abort_2] \leq \mathtt{Adv}^{\mathtt{bind}}_{T,\mathcal{B}'}$. Hence, for all algorithms $\mathcal{A}$, we must have $\Pr[Succ] \leq 2/p + 2\mathtt{Adv}^{\mathtt{bind}}_{T,\mathcal{B}'}$ i.e. assuming T is binding under selective openings, the advantage of $\mathcal{A}$ must be negligible. This completes the proof. $\square$

B.3 Proof of Theorem 20

Proof. Assume there exists a successful adversary $\mathcal{A}$ against the confirmer soundness of the constructed scheme. Using $\mathcal{A}$, we will construct algorithms breaking either the token soundness of the core confirmer signature scheme or the enhanced binding property of the commitment scheme.

Firstly, consider the following events: let $Succ_1$ denote that $\mathcal{A}$ succeeds in the confirmer soundness experiment by completing both the confirm and disavow protocols (i.e. $z_1 = z_2 = \mathtt{accept}$), and let $Succ_2$ denote the event that $\mathcal{A}$ succeeds by completing the disavow protocol and producing a valid verification token (i.e. $z_1 = z_3 = \mathtt{accept}$). Then we must have

$$\mathtt{Adv}^{\mathtt{snd\text{-}conf}}_{N,\mathcal{A}} \leq \Pr[Succ_1] + \Pr[Succ_2]$$

Claim 29 $\Pr[Succ_1] \leq 2/p + 2\mathtt{Adv}^{bind}_{T,\mathcal{B}'}$

Proof. Let $\mathcal{A}$ be an adversary such that $\Pr[Succ_1] = \epsilon_{\mathcal{A}}$. Consider the following simulator $\mathcal{B}$ interacting with $\mathcal{A}$ in the confirmer soundness experiment: $\mathcal{B}$ generates $par_S \leftarrow \mathtt{S.Setup}(1^k)$, $par_{CS} \leftarrow \mathtt{CS.Setup}(1^k)$, $par \leftarrow (par_S, par_{CS})$, and $(pk_V, sk_V) \leftarrow \mathtt{T.G}(1^k)$, and runs $\mathcal{A}$ with input (par, pk_V). When $\mathcal{A}$ queries one of the oracles $\mathcal{O}_{SimSign}$, $\mathcal{O}_{SimCon}$ or $\mathcal{O}_{SimDis}$, $\mathcal{B}$ responds honestly using sk_V. At some point, $\mathcal{A}$ will output $(pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma)$ where $pk_{C,S} = (pk'_C, \delta)$ and interact with $\mathcal{B}$ in the disavow protocol i.e. $\mathtt{NT}(pk_V) - \overline{\Sigma}_C$. With probability at least ϵ, $\mathcal{A}$ will successfully complete the protocol, and at this point, $\mathcal{B}$ rewinds and replays $\mathcal{A}$, but provides $\mathcal{A}$ with a new randomly chosen challenge value in $\mathtt{NT}(pk_V) - \overline{\Sigma}_C$. As in the proof of Claim 27, $\mathcal{B}$ will obtain two accepting transcripts, (com, a, c, z, r) and (com', a', c', z', r') where $com = com'$, with probability at least $\epsilon - \mathcal{A}^2/4 - \epsilon_{\mathcal{A}}/2p$. If $a \neq a'$, $\mathcal{B}$ will abort. Let this event be denoted $Abort_1$. Otherwise, $\mathcal{B}$ uses the special soundness of $\overline{\Sigma}_C$ to compute $sk'_C \leftarrow \mathtt{Ext}_{\overline{\Sigma}_C}(a, c, c', z, z')$ such that $(pk'_C, sk'_C) \in \{\mathtt{CS.KeyGen}_C(par)$ and $\mathtt{CS.Valid}(par_{CS}, pk_S, pk'_C, m, \sigma, sk'_C) = \mathtt{reject}$.

After completing the disavow protocol, $\mathcal{A}$ will interact with $\mathcal{B}$ in the confirm protocol i.e. $\mathtt{NT}(pk_V) - \Sigma_C$. Note that $\mathcal{A}$'s view is independent of whether the previous disavow protocol was rewinded or not, and that $\mathcal{A}$ will successfully complete the confirm protocol with probability at least $\epsilon_{\mathcal{A}}$. Upon completion, $\mathcal{B}$ will rewind $\mathcal{A}$ and provide him with a new randomly chosen challenge value in $\mathtt{NT}(pk_V) - \Sigma_C$. As above, with probability at least $\epsilon_{\mathcal{A}}^2/4 - \epsilon_{\mathcal{A}}/2p$, $\mathcal{B}$ obtains two accepting transcripts, (com, a, c, z, r) and (com', a', c', z', r') where $com = com'$, and will abort if $a \neq a'$. The latter event will be denoted $Abort_2$. If $\mathcal{B}$ does not abort, he will use the special soundness of Σ_C to compute $sk''_C \leftarrow \mathtt{Ext}_{\Sigma_C}(a, c, c', z, z')$ such that $(pk'_c, sk''_C) \in \{\mathtt{CS.KeyGen}_C(par_{CS})\}$ and $\mathtt{CS.Valid}(par_{CS}, pk_S, pk'_C, m, \sigma, sk''_C) = \mathtt{accept}$.

Assuming that neither $Abort_1$ nor $Abort_2$ occur, and that $\epsilon_{\mathcal{A}} > 2/p$, $\mathcal{B}$ will, with non-zero probability, obtain sk'_C and sk''_C with the above described properties. However, since the core confirmer signature scheme has unique private confirmer keys, we must have $sk'_C = sk''_C$. This directly leads to a contradiction since the deterministic $\mathtt{CS.Valid}$ cannot both output $\mathtt{accept}$ and $\mathtt{reject}$ given the same input. We conclude that conditioned on neither $Abort_1$ nor $Abort_2$ occurring, we must have $\epsilon \leq 2/p$. More specifically, we have

$$\Pr[Succ_1] \leq \Pr[Succ_1|\overline{Abort_1} \wedge \overline{Abort_2}] + \Pr[Abort_1] + \Pr[Abort_2]$$

and, from the above, we have $\Pr[Succ_1|\overline{Abort_1} \wedge \overline{Abort_2}] \leq 2/p$. Using a simulator similar to $\mathcal{B}$' in the proof of Claim 27, we can furthermore show that $\Pr[Abort_1] \leq \mathtt{Adv}^{\mathtt{bind}}_{T,\mathcal{B}'}$ and $\Pr[Abort_2] \leq \mathtt{Adv}^{\mathtt{bind}}_{T,\mathcal{B}'}$. Hence, the claim follows. $\square$

Claim 30 $\Pr[Succ_2] \leq \sqrt{4\mathtt{Adv}^{cs\text{-}tk\text{-}snd}_{CS,B} + 1/p^2} + \mathtt{Adv}^{bind}_{T,\mathcal{B}'} + 1/p$

Proof. Let $\mathcal{A}$ be an adversary such that $\Pr[Succ_2] = \epsilon_{\mathcal{A}}$. Consider the following simulator $\mathcal{B}$ interacting with $\mathcal{A}$ in the confirmer soundness experiment while attempting to break the token soundness of the core confirmer signature scheme.

$\mathcal{B}$ receives par_{CS}, generates $par_S \leftarrow \mathtt{S.Setup}(1^k)$, $par \leftarrow (par_S, par_{CS})$ and $(pk_V, sk_V) \leftarrow \mathtt{T.G}(1^k)$, and runs $\mathcal{A}$ with input (par, pk_V). Exactly as in the above claim, $\mathcal{B}$ responds to $\mathcal{A}$'s queries to $\mathcal{O}_{SimSign}$, $\mathcal{O}_{SimCon}$ and $\mathcal{O}_{SimDis}$ by using sk_V, and will interact with $\mathcal{A}$ in $\mathtt{NT}(pk_V)-\overline{\Sigma}_C$ when $\mathcal{A}$ outputs $(pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma)$ where $pk_{C,S} = (pk'_C, \delta)$. Note that $\mathcal{A}$ will successfully complete the protocol and $\mathtt{CS.TkVerify}(par_{CS}, pk_S, pk'_C, m, \sigma, tk_\sigma) = \mathtt{accept}$ with probability $\epsilon_{\mathcal{A}}$. Upon completion of the protocol, $\mathcal{B}$ rewinds and replays $\mathcal{A}$ with a new randomly chosen challenge value in $\mathtt{NT}(pk_V)-\overline{\Sigma}_C$. As above, with probability at least $\epsilon_{\mathcal{A}}^2/4 - \epsilon_{\mathcal{A}}/2p$, $\mathcal{B}$ obtains two accepting transcripts, (com, a, c, z, r) and (com', a', c', z', r') where $com = com'$, and will abort if $a \neq a'$. If $\mathcal{B}$ does not abort, he will extract $sk'_C \leftarrow \mathtt{Ext}_{\overline{\Sigma}_C}(a, c, c', z, z')$ such that $(pk'_C, sk'_C) \in \{\mathtt{CS.KeyGen}_C(par_{CS})\}$ and $\mathtt{CS.Valid}(par_{CS}, pk_S, pk'_C, m, \sigma, sk'_C) = \mathtt{reject}$. Lastly, $\mathcal{B}$ outputs $(pk_S, pk'_C, sk'_C, m, \sigma, tk_\sigma)$ as his own attack against the token soundness of the core confirmer signature scheme. Note that this will be a valid attack due to the properties of $(pk_S, pk'_C, sk'_C, m, \sigma, tk_\sigma)$ outlined above. Conditioned on $\mathcal{B}$ not aborting, we have $\mathtt{Adv}_{CS,\mathcal{B}}^{\mathtt{cs-tk-snd}} \geq \epsilon_{\mathcal{A}}^2/4 - \epsilon_{\mathcal{A}}/2p$.

Let *Abort* denote the event that $\mathcal{B}$ aborts. We have that

$$\Pr[Succ_2] \leq \Pr[Succ_2|\overline{Abort}] + \Pr[Abort]$$

From the above, we conclude that $\Pr[Succ_2|\overline{Abort}] \leq \sqrt{4\mathtt{Adv}_{CS,\mathcal{B}}^{\mathtt{cs-tk-snd}} + 1/p^2} + 1/p$. Lastly, by constructing a simulator similar to $\mathcal{B}'$ in the proof of Claim 27, we can show that $\Pr[Abort] \leq \mathtt{Adv}_{T,\mathcal{B}'}^{\mathtt{bind}}$. $\qquad\square$

The theorem follows from combining the above claims. $\qquad\square$

B.4 Proof of Theorem 21

Proof. In the following, we let $\mathtt{Exp}_{N,\mathcal{A}}^{\mathtt{non\text{-}trans}\text{-}b}$ denote the experiment in which the challenge bit b is chosen. We prove the theorem by considering the following sequence of games:

$Game_0$ This game corresponds to the $\mathtt{Exp}_{S,\mathcal{A}}^{\mathtt{non\text{-}trans}\text{-}0}$ experiment.

$Game_1$ In this game, we change how the interaction with $\mathcal{A}$ in the sign, confirm and disavow oracle queries is done. More specifically, the protocols $\mathtt{NT}(pk_V)\text{-}\Sigma_S$, $\mathtt{NT}(pk_V)\text{-}\Sigma_C$ and $\mathtt{NT}(pk_V)\text{-}\overline{\Sigma}_C$ are changed using the trapdoor property of T as follows: Firstly, the private key sk_V corresponding to pk_V is retrieved from $L_{VKeyReg}$. Then, instead of sending $com \leftarrow \mathtt{T.Comm}(pk_V, a; r)$ to $\mathcal{A}$, where a is the first message of Σ_S, Σ_C or $\overline{\Sigma}_C$ (depending on the query type), we compute $(com', aux) \leftarrow \mathtt{T.TdCom}(pk_V)$, store aux, and send com' to $\mathcal{A}$. When a challenge c is received from $\mathcal{A}$, we compute $r' \leftarrow \mathtt{T.TdOpen}(aux, a, sk_V)$, and return (a, r', z) where z is the last message of Σ_S, Σ_C or $\overline{\Sigma}_C$. Note that since a is not needed until $\mathcal{A}$ submits his challenge c, the computation of a can be delayed until this happens.

$Game_2$ In this game, we replace the honestly generated message of Σ_S, Σ_C and $\overline{\Sigma}_C$ in the sign, confirm and disavow queries with simulated messages output by $\mathtt{Sim}_{\Sigma_S}$, $\mathtt{Sim}_{\Sigma_C}$ and $\mathtt{Sim}_{\overline{\Sigma}_C}$ e.g. when a challenge c is received from $\mathcal{A}$ in a sign query, we compute $(a', z') \leftarrow \mathtt{Sim}_{\Sigma_S}((par, pk_S, pk_{C,S}, pk_C||pk_{C,S}||m, \sigma), c)$ and $r' \leftarrow \mathtt{TdOpen}(aux, a', sk_V)$, and return (a', r', z') to $\mathcal{A}$.

$Game_3$ In this game, the challenge sign and confirm protocols are modified using the trapdoor property of T in the same way the sign and confirm oracle queries was modified in $Game_1$.

$Game_4$ In this game, the challenge sign and confirm protocols are modified using the simulated messages from $\mathtt{Sim}_{\Sigma_S}$ and $\mathtt{Sim}_{\Sigma_{sk_C}}$ in the same way the sign and confirm oracle queries was modified in $Game_2$.

$Game_5$ In this game, the challenge signature σ^* is picked at random from the signature space $\mathcal{S}$ instead of being honestly generated using the $\mathtt{CS.Sign}$ algorithm of the core confirmer signature scheme.

$Game_6$ In this game, the challenge (simulated) disavow protocol is modified by replacing the simulated messages generated using $\mathtt{Sim}_{\overline{\Sigma}_C}$ with the honestly generated messages of $\overline{\Sigma}_C$. Furthermore, the first message a of $\overline{\Sigma}_C$ is generated at the time the challenger sends the first message of the disavow protocol.

$Game_7$ In this game, we again modify the challenge disavow protocol, this time by replacing the trapdoor commitment generated by $\mathtt{T.TdComm}(pk_V)$ with an honest commitment to the message a introduced in $Game_6$ generated by $\mathtt{T.Comm}(pk_V, a; r)$.

$Game_8$ **and** $Game_9$ In these games, the changes introduces in $Game_2$ and $Game_1$ are reversed. Note that the changes regarding the challenge query introduced in $Game_3$, $Game_4$, $Game_6$ and $Game_7$ remains, and that $Game_9$ corresponds to the experiment $\mathtt{Exp}_{N,\mathcal{A}}^{\mathtt{non-trans-1}}$.

Let E_i denote the event that $\mathcal{A}$ outputs 1 in game i. The advantage of $\mathcal{A}$ can be expressed as

$$\mathtt{Adv}_{N,\mathcal{A}}^{\mathtt{non-trans}} = \frac{1}{2}|\Pr[E_0] - \Pr[E_7]| \leq \frac{1}{2}\sum_{i=0}^{8}|\Pr[E_i] - \Pr[E_{i+1}]|$$

Claim 31 $\Pr[E_0] = \Pr[E_1]$, $\Pr[E_2] = \Pr[E_3]$, $\Pr[E_6] = \Pr[E_7]$ *and* $\Pr[E_8] = \Pr[E_9]$

This follows from the perfect trapdoor property of the commitment scheme T. The proofs of these statements are almost identical to the proof of Claim 25, and we leave out the details.

Claim 32 $\Pr[E_1] = \Pr[E_2]$, $\Pr[E_3] = \Pr[E_4]$, $\Pr[E_5] = \Pr[E_6]$ *and* $\Pr[E_7] = \Pr[E_8]$

This follows from the special honest verifier zero-knowledge property of Σ_S, Σ_C and $\overline{\Sigma}_C$. The proofs of these statements are almost identical to the proof of Claim 26, and we leave out the details.

Claim 33 $|\Pr[E_4] - \Pr[E_5]| \leq 2\mathtt{Adv}_{CS,\mathcal{B}}^{cs-inv-cma}$

Proof. Assume there exists an adversary $\mathcal{A}$ that outputs 1 in $Game_4$ with a different probability than in $Game_5$, and let the difference be denoted $\epsilon_{\mathcal{A}} = |\Pr[E_4] - \Pr[E_5]|$. Using $\mathcal{A}$, we construct a distinguisher $\mathcal{B}$ that breaks the invisibility of the core confirmer signature scheme with probability $\epsilon_{\mathcal{A}}/2$.

Firstly, $\mathcal{B}$ is given parameters par_{CS}, a public signer key pk_S, and a public confirmer key pk'_C of the core confirmer signature scheme. $\mathcal{B}$ then generates $par_S \leftarrow \mathtt{S.Setup}(1^k)$, sets $par \leftarrow (par_S, par_{CS})$, and computes the key pairs $(pk_C, sk_C) \leftarrow \mathtt{S.KeyGen}(par_S)$ and $(pk_V, sk_V) \leftarrow \mathtt{T.G}(1^k)$. Lastly, $\mathcal{B}$ computes $\delta \leftarrow \mathtt{S.Sign}(par_S, pk_S||pk'_C, sk_C)$, sets $pk_{C,S} \leftarrow (pk'_C, \delta)$, and runs $\mathcal{A}$ with input $(par, pk_S, pk_C, pk_{C,S}, pk_V, sk_V)$.

While running, $\mathcal{A}$ can make confirmer setup, sign, convert and disavow oracle queries, which $\mathcal{B}$ answers as follows:

- $\mathcal{O}_{CSetup}$: Given pk_S, $\mathcal{B}$ runs $(pk'_C, sk'_C) \leftarrow \mathtt{CS.KeyGen}_C(par_{CS})$, constructs the signature $\delta \leftarrow \mathtt{S.Sign}(par_S, pk_S||pk'_C, sk_C)$, sets $pk_{C,S} \leftarrow (pk_C, \delta)$ and $sk_{C,S} \leftarrow sk'_C$, stores $(pk_{C,S}, sk_{C,S})$ in L_{CSetup}, and finally returns $pk_{C,S}$ to $\mathcal{A}$.
- $\mathcal{O}_{Sign}$: Same as in the proof of Claim 27 above. Note that $\mathcal{B}$ has access to a (non-interactive) signing oracle and that the sign protocol is simulated using sk_V in $Game_4/Game_5$.
- $\mathcal{O}_{Convert}$: Given $(pk_S, pk_{C,S}, m, \sigma)$ where $pk_{C,S} = (pk'_C, \delta)$, if pk_S and $pk_{C,S}$ are identical to the challenge keys given to $\mathcal{A}$, $\mathcal{B}$ submits $(pk_C||pk_{C,S}||m, \sigma)$ to his own convert oracle (note that this conversion oracle only works for the challenge keys (pk_S, pk'_C) given given to $\mathcal{B}$). The obtained token tk_σ is then returned to $\mathcal{A}$. Otherwise, $\mathcal{B}$ searches for a tuple $(pk_S, pk_{C,S}, sk_{C,S})$ in L_{CSetup}, and if such a tuple is found, $\mathcal{B}$ computes and returns $tk_\sigma \leftarrow \mathtt{CS.Convert}(par_{CS}, pk_S, pk'_C, pk_C||pk_{C,S}||m, \sigma, sk_{C,S})$ (note that $sk_{C,S} = sk'_C$). If no such tuple is found, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$.

– $\mathcal{O}_{Con}$: Given $(pk_S, pk_{C,S}, pk_V, m, \sigma)$ where $pk_{C,S} = (pk'_C, \delta)$, if pk_S and $pk_{C,S}$ are identical to the challenge keys given to $\mathcal{A}$, $\mathcal{B}$ submits $((pk_C||pk_{C,S}||m, \sigma)$ to his own convert oracle to obtain tk_σ. Otherwise, $\mathcal{B}$ searches for a tuple $(pk_S, pk_{C,S}, sk_{C,S})$ in L_{CSetup}, and computes the token $tk_\sigma \leftarrow \mathtt{CS.Convert}(par_{CS}, pk_S, pk'_C, pk_C||pk_{C,S}||m, \sigma, sk_{C,S})$ if such a tuple is found. Then, if $\mathtt{CS.TkVerify}(par_{CS}, pk_S, pk'_C, pk_C||pk_{C,S}||m, \sigma, tk_\sigma) = \mathtt{accept}$, $\mathcal{B}$ retrieves (pk_V, sk_V) from $L_{VKeyReg}$ and interacts with $\mathcal{A}$ in the (simulated) confirm protocol using sk_V as described in $Game_4/Game_5$. If no matching tuple $(pk_S, pk_{C,S}, sk_{C,S})$ is found or $\mathtt{CS.TkVerify}(par_{CS}, pk_S, pk_{C,S}, m, \sigma, tk_\sigma) = \mathtt{reject}$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$.

– $\mathcal{O}_{Dis}$: $\mathcal{B}$ answers these queries in a similar way to confirm queries, except that the disavow protocols is simulated for signatures which are found to be invalid.

At some point, $\mathcal{A}$ outputs a challenge message m^*. $\mathcal{B}$ forwards the message $pk_C||pk_{C,S}||m^*$ as his own challenge message, and obtains a signature σ^* which is send to $\mathcal{A}$. Then $\mathcal{B}$ interacts with $\mathcal{A}$ in the simulated sign protocol using sk_V as described in $Game_4/Game_5$. After this challenge query, $\mathcal{A}$ can ask additional confirmer setup, sign, convert, confirm and disavow queries which $\mathcal{B}$ answers as above. Note that $\mathcal{A}$ is not allowed to submit the challenge keys, message and signature, $(pk_S, pk_{C,S}, m^*, \sigma^*)$ to the convert, confirm or disavow oracles. Lastly, $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as his own answer in the invisibility game of the core confirmer signature scheme.

From the above description of $\mathcal{B}$, it should be clear that if the challenge signature given to $\mathcal{B}$ is a valid honestly generated signature, $\mathcal{B}$ provides $\mathcal{A}$ with a perfect simulation of $Game_4$. On the other hand, if the challenge signature given to $\mathcal{B}$ is picked at random from the signature space of the core confirmer signature scheme, then $\mathcal{B}$ will provide $\mathcal{A}$ with a perfect simulation of $Game_5$. Letting β be the bit chosen by $\mathcal{B}$'s challenger and β' be the bit output by $\mathcal{B}$, the advantage of $\mathcal{B}$ can be expressed as

$$
\begin{aligned}
\mathtt{Adv}^{\mathtt{cs\text{-}inv\text{-}cma}}_{CS,\mathcal{B}} &= \Pr[\beta' = \beta] - 1/2 \\
&= \Pr[\beta' = 1|\beta = 1]\Pr[\beta = 1] + \Pr[\beta' = 0|\beta = 0]\Pr[\beta = 0] - 1/2 \\
&= 1/2(\Pr[\beta' = 1|\beta = 1] - \Pr[\beta' = 1|\beta = 0]) \\
&= 1/2(\Pr[b' = 1|\beta = 1] - \Pr[b' = 1|\beta = 0]) \\
&= 1/2(\Pr[E_4] - \Pr[E_5])
\end{aligned}
$$

The above claim follows directly from this. □

The theorem follows by the combination of the above claims. □

C Core Confirmer Signature Scheme Security Proofs

C.1 Proof of Theorem 22

Proof. To prove the theorem, we consider the following two games:

$Game_0$ This is the original invisibility game.
$Game_1$ In this game, the response to convert queries is changed as follows. If $\mathcal{A}$ submits a tuple (m, σ) such that σ was not obtained by submitting the challenge confirmer key pk_C and the message m to the sign oracle, we return $\perp$ to $\mathcal{A}$ i.e. the signature is treated as invalid.

Let $Succ_i$ be the event that $\mathcal{A}$ wins in $Game_i$. The advantage of $\mathcal{A}$ in the invisibility game can be expressed as

$$
\mathtt{Adv}^{\mathtt{cs\text{-}inv\text{-}cma}}_{CS,\mathcal{A}} = |\Pr[Succ_0] - 1/2| \leq |\Pr[Succ_0] - \Pr[Succ_1]| + |\Pr[Succ_1] - 1/2|
$$

Claim 34 $|\Pr[Succ_0] - \Pr[Succ_1]| < \mathtt{Adv}^{cs\text{-}uf\text{-}cma}_{CS,\mathcal{B}}$

Proof. Firstly, note that $Game_0$ and $Game_1$ are identical unless $\mathcal{A}$ makes a convert query on at least one signature tuple (m, σ) which is valid wrt. the challenge keys (pk_S, pk_C) and which was not obtained by submitting (pk_C, m) to the sign oracle. Hence, if there is a difference in the output of $\mathcal{A}$ in $Game_0$ and $Game_1$, $\mathcal{A}$ must make such a query. We show the claim by a proof by contradiction, and assume there exists an adversary $\mathcal{A}$ which makes one of these queries. Using $\mathcal{A}$, we construct an algorithm $\mathcal{B}$ that breaks the unforgeability of the core confirmer signature scheme as follows.

$\mathcal{B}$ receives parameters par and a public signer key pk_S, computes $(pk_C, sk_C) \leftarrow \texttt{KeyGen}_C(par)$, and runs $\mathcal{A}$ with input (par, pk_S, pk_C). While $\mathcal{A}$ is running, $\mathcal{A}$ can query his sign and convert oracles which $\mathcal{B}$ simulates as follows.

- $\mathcal{O}_{Sign}$: Given (pk_C, m), $\mathcal{B}$ simply forwards (pk_C, m) to his own signing oracle, and returns the obtained signature σ to $\mathcal{A}$.
- $\mathcal{O}_{Convert}$: Given (m, σ), $\mathcal{B}$ computes $tk_\sigma \leftarrow \texttt{Convert}(par, pk_S, pk_C, m, \sigma, sk_C)$. If σ was not obtained through a sign query on (pk_C, m) and $\texttt{TkVerify}(par, pk_S, pk_C, m, \sigma, tk_\sigma) = \texttt{accept}$, $\mathcal{B}$ halts with the output $(pk_C, m, \sigma, tk_\sigma)$. Otherwise, $\mathcal{B}$ returns tk_σ to $\mathcal{A}$.

At some point, $\mathcal{A}$ outputs a challenge message m^*. $\mathcal{B}$ flips a random coin $b \leftarrow \{0, 1\}$, and if $b = 0$, $\mathcal{B}$ submits (pk_C, m^*) to his own signing oracle to obtain σ^*. Otherwise, if $b = 1$, $\mathcal{B}$ randomly picks $\sigma^* \leftarrow \mathcal{S}$. Lastly, $\mathcal{B}$ sends σ^* to $\mathcal{A}$. After this challenge query, $\mathcal{A}$ can ask additional sign and convert queries, which are answered as above.

From the above description of $\mathcal{B}$, it is clear that $\mathcal{B}$ breaks the unforgeability of the core confirmer signature scheme whenever $\mathcal{A}$ submits a convert query of the above described form. This, combined with the observation that $Game_0$ and $Game_1$ are identical unless such a query is made, yields the claimed result. $\square$

Claim 35 $|\Pr[Succ_1] - 1/2| < \texttt{Adv}_{\mathcal{G}, \mathcal{B}}^{dlin}$

Proof. We show the claim by contradiction. Assume that an adversary $\mathcal{A}$ with non-negligible advantage $\epsilon_\mathcal{A} = |\Pr[Succ_1] - 1/2|$ in $Game_1$ exists. Using $\mathcal{A}$, we will construct an algorithm $\mathcal{B}$ which solves the decisional linear problem with respect to the group generator $\mathcal{G}$ (which is implicitly defined by the $\texttt{Setup}$ algorithm) with probability $\epsilon_\mathcal{A}$.

Firstly, $\mathcal{B}$ is given a group description $(\mathbb{G}, \mathbb{G}_T, e, p)$ and an instance $(u, v, h, u^x, v^y, h^c) \in \mathbb{G}^6$ of the decisional linear problem in $\mathbb{G}$. To generate a set of parameters, $\mathcal{B}$ picks a hash function family $\mathcal{H} = \{H_\nu : \{0, 1\}^* \to \{0, 1\}^{|\nu|}\}$, sets $g \leftarrow h$ and $par \leftarrow (\mathbb{G}, \mathbb{G}_T, e, p, g, \mathcal{H})$. To generate a public/private signing key pair, $\mathcal{B}$ picks a hash key $\nu \leftarrow \{0, 1\}^k$, $\alpha \leftarrow \mathbb{Z}_p$, $g_2, h', u_0, \ldots, u_n, \leftarrow \mathbb{G}$ and sets $g_1 \leftarrow g^\alpha$, $pk_S \leftarrow (k, g_1, g_2, h', u_0, \ldots, u_n)$ and $sk_S \leftarrow \alpha$. As a public confirmer key, $\mathcal{B}$ uses $pk_C \leftarrow (u, v)$. Finally, $\mathcal{B}$ runs $\mathcal{A}$ with input (par, pk_S, pk_C). While running, $\mathcal{A}$ can ask sign and convert queries which $\mathcal{B}$ answers as follows.

- $\mathcal{O}_{Sign}$: Given (pk_C, m), $\mathcal{B}$ simply runs $\sigma \leftarrow \texttt{Sign}(par, pk_S, pk_C, m, sk_S)$ but remembers the random choices $a, b, r \in \mathbb{Z}_p$ used to construct the signature, and stores (m, σ, a, b, r) for later use. $\mathcal{B}$ then returns σ to $\mathcal{A}$.
- $\mathcal{O}_{Convert}$: Given (m, σ) where $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, s)$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if σ was not returned to $\mathcal{A}$ as a response to a query on (pk_C, m) (note that this response is exactly as described in $Game_1$). Otherwise, $\mathcal{B}$ recalls the random choices $a, b, r \in \mathbb{Z}_p$ used to construct σ, and returns the token $tk_\sigma = (g^a, g^b)$. Note that $(g^a, g^b) = (u^{x'a}, v^{y'b}) = (\sigma_1^{x'}, \sigma_2^{y'})$ where the tuple (x', y') is the (unknown) private confirmer key corresponding to pk_C.

At some point, $\mathcal{A}$ outputs a challenge message m^*. $\mathcal{B}$ constructs a challenge signature by picking $r, s \leftarrow \mathbb{Z}_p$ and setting $\sigma^* \leftarrow (u^x, v^y, h^c h^r, g_2^\alpha F(M)^r)$ where $M = g^t(h')^s$, $t = H\nu(pk_C||u^x||v^y||h^c h^r||m)$, and (u^a, v^b, h^c) are the elements received in the decisional linear problem. Note that if c is random, then σ^* will be distributed as a random element in $\mathbb{G}^4 \times \mathbb{Z}_p$, whereas if $c = a + b$, σ^* will be a valid signature on m^* since $h^c h^r = g^{a+b+r}$. After the challenge query, $\mathcal{A}$ can ask additional sign and convert queries which $\mathcal{B}$ answers as above. When $\mathcal{A}$ halts with output b', $\mathcal{B}$ simply forwards b' as his own final output and halts.

From the above description of $\mathcal{B}$, it should be clear that $\mathcal{B}$ will solve the decisional linear problem whenever $\mathcal{A}$ correctly guesses whether the challenge signature is honestly generated or a random element of the signature space, and that $\mathcal{B}$ will have advantage $\epsilon_{\mathcal{A}}$. Hence, the claim follows. □

The theorem follows from the combination of the above claims. □

C.2 Proof of Theorem 24

Proof. Assume there exists an adversary $\mathcal{A}$ which has a non-zero advantage in the token soundness experiment, and let $(pk_S, pk_C, sk_C, m, \sigma, tk_\sigma)$ be the output of $\mathcal{A}$ where $pk_C = (u, v)$, $sk_C = (x, y)$, $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, s)$ and $tk_\sigma = (tk_1, tk_2)$.

Since it is required that $(pk_C, sk_C) \in \{\texttt{KeyGen}_C(par)\}$, we must have that $u^x = g$ and $v^y = g$. The requirement that $\texttt{TkVerify}(par, pk_S, pk_C, m, \sigma, tk_\sigma) = \texttt{accept}$ means that $e(u, tk_1) = e(\sigma_1, g)$ and $e(v, tk_2) = e(\sigma_2, g)$ which, by the properties of the pairing, implies that $tk_1 = \sigma_1^x$ and $tk_2 = \sigma_2^y$.

On the other hand, the requirement that $\texttt{Valid}(par, pk_S, pk_C, m, \sigma, sk_C) = \texttt{reject}$ means that if the token $tk_\sigma' \leftarrow \texttt{Convert}(par, pk_S, m, \sigma, sk_C)$ is computed, the output of $\texttt{TkVerify}(par, pk_S, pk_C, m, \sigma, tk_\sigma')$ is $\texttt{reject}$. However, since $sk_C = (x, y)$, we must have that $tk_\sigma' = (\sigma_1^x, \sigma_2^y) = tk_\sigma$. Since $\texttt{TkVerify}$ is deterministic, this contradicts the above which requires that the output of $\texttt{TKVerify}(par, pk_S, pk_C, m, \sigma, tk_\sigma)$ is $\texttt{accept}$. Hence, we conclude that $\mathcal{A}$ cannot exist. □