On-line Non-transferable Signatures Revisited[⋆]

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

[1] Research Center for Information Security, AIST, 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: signatures, on-line non-transferability, standard model.

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.

[⋆] The full version of this paper is available at `http://eprint.iacr.org/2009/406`
[⋆⋆] This author is supported by a JSPS scholarship.

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.* [5] 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 [4] 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 and designated confirmer schemes have been proposed, e.g. see [3,7,12,14,2,8].

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 [13] pointed out that almost all[1] 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 [13] 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 [13] for further discussion of this). In this

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

paper, we adopt the same general approach as [13], assume verifiers are equipped with public/private key pairs, and will furthermore explicitly model verifier key registration in our security model[2].

In [13], 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 `ind-cpa` secure public key encryption and `uf-cma` secure signatures. 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[3].

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 [13] (see Section 3 for details). Unlike [13], 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.

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

[2] Note that while the security definitions in [13] 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.

[3] The defined disavow protocol in [13] is non-interactive and provides a publicly verifiable proof of invalidity.

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 [19].

Related Work. Jakobsson *et al.* [10] 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.* [17] who introduced universal designated verifier signatures which allow any user 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, and most of the recently proposed schemes will in fact 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.* [11] and Huang *et al.* [9] provide non-interactive proofs which are simulatable by the verifier, and hence avoid the problem of on-line attacks. Furthermore, Monnerat *et al.* [15] proposed an undeniable signature scheme with interactive 2-move confirm and disavow protocols. While the used definition of non-transferability in [15] 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 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 probabilistic polynomial time (PPT) algorithms:

- `Setup` which, given a security parameter 1^k, returns the public parameters *par*.

- $\texttt{KeyGen}_S$, $\texttt{KeyGen}_C$, and $\texttt{KeyGen}_V$ 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.
- $\texttt{CSetup}$ 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.
- $\texttt{CKeyValid}$ which, on input par, pk_S, pk_C, and $pk_{C,S}$, outputs either $\texttt{accept}$ or $\texttt{reject}$.
- $(\texttt{Sign}, \texttt{Receive})$ which is a pair of interactive algorithms with common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$. $\texttt{Sign}$ is run by the signer and is given sk_S as private input, and $\texttt{Receive}$ is run by the verifier. At the end of the interaction, both $\texttt{Sign}$ and $\texttt{Receive}$ will output a signature, σ_S and σ_R, and $\texttt{Receive}$ will in addition output either $\texttt{accept}$ or $\texttt{reject}$.
- $\texttt{Convert}$ which, on input par, pk_S, m, σ, and $sk_{C,S}$, returns a verification token tk_σ.
- $\texttt{TkVerify}$ which, on input par, pk_S, pk_C, $pk_{C,S}$, m, σ, and tk_σ, returns either $\texttt{accept}$ or $\texttt{reject}$.
- $(\texttt{Confirm}, \texttt{V}_C)$ which is a pair of interactive algorithms with common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma)$. $\texttt{Confirm}$ is run by the confirmer and is given $sk_{C,S}$ as private input, and $\texttt{V}_C$ is run by the verifier. At the end of the interaction, $\texttt{V}_C$ outputs either $\texttt{accept}$ or $\texttt{reject}$.
- $(\texttt{Disavow}, \texttt{V}_D)$ which is also a pair of interactive algorithms. Input for $\texttt{Disavow}$ and $\texttt{V}_D$ is exactly as in $(\texttt{Confirm}, \texttt{V}_C)$ above, and the output of $\texttt{V}_D$ is either $\texttt{accept}$ or $\texttt{reject}$.

Like Liskov and Micali [13], 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 $\texttt{CSetup}$. 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 [13] in that we explicitly define a key validation algorithm $\texttt{CKeyValid}$ for signer specific confirmer keys[4], and introduce $(\texttt{Confirm}, V_C)$ 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 [13].

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

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

[4] This algorithm is required by our extended security model. More specifically, it is required to define key unforgeability (see Section 3).

We will use the notation $\{\texttt{Sign}(sk_S) \leftrightarrow \texttt{Receive}\}(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$ to denote the interaction between Sign and Receive on the common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$ and private input sk_S to the Sign algorithm. To shorten this notation, we will sometimes use $PK = (pk_S, pk_C, pk_{C,S}, pk_V)$ to represent the public keys. We furthermore use $(\sigma_S, (\sigma_R, z)) \leftarrow \{\texttt{Sign}(sk_S) \leftrightarrow \texttt{Receive}\}(par, PK, m)$ 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. We require that for all honestly generated public parameters par and key pairs (pk_S, sk_C), (pk_C, sk_C), (pk_V, sk_V), and $(pk_{C,S}, sk_{C,S})$, that $z \leftarrow \texttt{CKeyValid}(par, pk_C, pk_S, pk_{C,S})$ yields $z = \texttt{accept}$ and that, for all messages m, the signature generation $(\sigma_S, (\sigma_R, z_R)) \leftarrow \{\texttt{Sign}(sk_S) \leftrightarrow \texttt{Receive}\}(par, PK, m)$, where $PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$, yields that $z_R = \texttt{accept}$ and $\sigma_R = \sigma_S$, that $\texttt{Valid}(par, pk_S, pk_C, pk_{C,S}, sk_{C,S}, m, \sigma) = \texttt{true}$ and that $z_C \leftarrow \{\texttt{Confirm}(sk_{C,S}) \leftrightarrow \texttt{V}_C\}(par, PK, m, \sigma)$ results in $z_C = \texttt{accept}$. Furthermore, for all (m', σ') such that $\texttt{Valid}(par, pk_S, pk_C, pk_{C,S}, sk_{C,S}, m', \sigma') = \texttt{false}$, we require that $z_D \leftarrow_2 \{\texttt{Disavow}(sk_{C,S}) \leftrightarrow \texttt{V}_D\}(par, PK, m', \sigma')$ yields $z_D = \texttt{accept}$.

3 Security Model

An ONS scheme is required to satisfy the security notions *unforgeability*, *key unforgeability*, *soundness*, *non-repudiation* and *non-transferability* to be considered secure. However, before we can formally define these security notions, we require a scheme to define the verifier simulation algorithms $\texttt{SimSign}(par, PK, m, sk_V)$, $\texttt{SimCon}(par, PK, m, \sigma, sk_V)$ and $\texttt{SimDis}(par, PK, m, \sigma, sk_V)$ which simulates the Sign, Confirm and Disavow algorithms, respectively. 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. 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. Our notion of unforgeability requires 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 $\texttt{Exp}^{\texttt{uf-cma}}_{N,\mathcal{A}}$ shown in Figure 1. 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 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 $\mathtt{KeyGen}_V$.

- $\mathcal{O}_{VKeyReg}$: 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 $\perp$ 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 $\perp$ 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 $(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$ 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 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 algorithm $\mathtt{SimDis}(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma, sk_V)$.

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

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 [13] 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 $\mathtt{Exp}^{\mathtt{key\text{-}uf}}_{N,\mathcal{A}}(1^k)$ shown in Figure 1. 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 above, and the remaining oracles are defined as follows:

- $\mathcal{O}_{CSetup}$: given pk_S, this oracle runs $(pk_{C,S}, sk_{C,S}) \leftarrow \mathtt{CSetup}(par, pk_S, sk_C)$, stores $(pk_S, pk_{C,S}, sk_{C,S})$ 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}, and returns $\perp$ if no such tuple is found. Otherwise, the oracle returns $tk_\sigma \leftarrow \mathtt{Convert}(par, pk_S, m, \sigma, sk_{C,S})$.
- $\mathcal{O}_{Con}$: 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 $\perp$. Otherwise, the oracle interacts with $\mathcal{A}$ by running $\mathtt{Confirm}$ with common input $(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma)$ and secret input $sk_{C,S}$.

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

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

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

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

– $\mathcal{O}_{Dis}$: given the same input as $\mathcal{O}_{Con}$, this oracle returns $\perp$ to $\mathcal{A}$ if there is no tuple $(pk_S, pk_{C,S}, sk_{C,S})$ 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 secret input $sk_{C,S}$.

Definition 2. *An ONS scheme N is said to be* key unforgeable *if no PPT algorithm $\mathcal{A}$ with non-negligible advantage* $\mathtt{Adv}^{key\text{-}uf}_{N,\mathcal{A}}(k) = \Pr[\mathtt{Exp}^{key\text{-}uf}_{N,\mathcal{A}}(1^k) = 1]$ *exists.*

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 [13] in that we allow the adversary a negligible success probability whereas [13] requires the success probability to be zero. However, we highlight that [13] 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 $\mathtt{Exp}^{\mathtt{non\text{-}rep}}_{N,\mathcal{A}}(1^k)$ shown in Figure 1. 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.

```
$\mathtt{Exp}_{N,\mathcal{A}}^{\mathtt{snd\text{-}sign}}(1^k)$
    $par \leftarrow \mathtt{Setup}(1^k)$; $(pk_C, sk_C) \leftarrow \mathtt{KeyGen}_C(par)$
    $(pk_V, sk_V) \leftarrow \mathtt{KeyGen}_V(par)$
    $(pk_S, m, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_C, sk_C, pk_V)$
    $(pk_{C,S}, sk_{C,S}) \leftarrow \mathtt{CSetup}(par, pk_S, sk_C)$
    $PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$
    $(\sigma, z_1) \leftarrow_2 \{\mathcal{A}^{\mathcal{O}}(st, pk_{C,S}, sk_{C,S}) \leftrightarrow \mathtt{Receive}(par, PK, m)\}$
    $z_2 \leftarrow \mathtt{Valid}(par, pk_S, pk_C, pk_{C,S}, m, \sigma, sk_{C,S})$
    if $z_1 = \mathtt{accept} \wedge z_2 = \mathtt{reject}$ output 1
    else output 0

$\mathtt{Exp}_{N,\mathcal{A}}^{\mathtt{snd\text{-}conf}}(1^k)$
    $par \leftarrow \mathtt{Setup}(1^k)$; $(pk_V, sk_V) \leftarrow \mathtt{KeyGen}_V(par)$
    $(pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, pk_V)$
    $PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$
    $z_1 \leftarrow_2 \{\mathcal{A}^{\mathcal{O}}(st) \leftrightarrow \mathtt{V}_D(par, PK, m, \sigma)\}$
    $z_2 \leftarrow_2 \{\mathcal{A}^{\mathcal{O}}(st) \leftrightarrow \mathtt{V}_C(par, PK, m, \sigma)\}$
    $z_3 \leftarrow \mathtt{TkVerify}(par, pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma)$
    if $z_1 = \mathtt{accept} \wedge (z_2 = \mathtt{accept} \vee z_3 = \mathtt{accept})$ output 1
    else output 0
```

Fig. 2. Soundness security experiments

Definition 3. *An ONS scheme N is said to provide* non-repudiation *if no PPT algorithm $\mathcal{A}$ with non-negligible advantage* $\mathtt{Adv}_{N,\mathcal{A}}^{non\text{-}rep}(k) = \Pr[\mathtt{Exp}_{N,\mathcal{A}}^{non\text{-}rep}(1^k) = 1]$ *exists.*

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

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 [13]. 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 above.

Definition 5. *An ONS scheme is said to provide* confirmer soundness *if no PPT algorithm $\mathcal{A}$ with non-negligible advantage* $\mathtt{Adv}_{N,\mathcal{A}}^{snd\text{-}conf}(k) = \Pr[\mathtt{Exp}_{N,\mathcal{A}}^{snd\text{-}conf}(1^k) = 1]$ *exists.*

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 3. In the experiment, $\mathcal{A}$ has

```
$\mathtt{Exp}_{N,\mathcal{A}}^{\mathtt{non\text{-}trans}}(1^k)$
  $L_{VKeyReg} \leftarrow \{\}$; $par \leftarrow \mathtt{Setup}(1^k)$; $(pk_S, sk_S) \leftarrow \mathtt{KeyGen}_S(par)$
  $(pk_C, sk_C) \leftarrow \mathtt{KeyGen}_C(par)$; $(pk_{C,S}, sk_{C,S}) \leftarrow \mathtt{CSetup}(par, pk_S, sk_C)$
  $(pk_V, sk_V) \leftarrow \mathtt{KeyGen}_V(par)$; $PK \leftarrow (pk_S, pk_C, pk_{C,S}, pk_V)$
  $(m, st) \leftarrow \mathcal{A}^{\mathcal{O}}(par, PK, sk_V)$
  $b \leftarrow \{0,1\}$
  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

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

4 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. Formal definitions of standard signatures, sigma protocols and trapdoor commitments can be found in the full version.

4.1 Core Confirmer Signature Scheme

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

– `CS.Sign`: given par, pk_C, m, and sk_S, this algorithm returns a signature σ.
– `CS.Convert`: given par, pk_S, m, σ and sk_C, this algorithm returns a verification token tk_σ.
– `CS.TkVerify`: given par, pk_S, pk_C, m, σ and tk_σ, this algorithm returns either `accept` or `reject`.

Both `CS.Convert` and `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 `CS.Valid` as

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

We require that a scheme is *correct* i.e. for all $par \leftarrow \texttt{CS.Setup}(1^k)$, $(pk_S, sk_S) \leftarrow \texttt{CS.KeyGen}_S(par)$, $(pk_C, sk_C) \leftarrow \texttt{CS.KeyGen}_V(par)$, all messages m and all $\sigma \leftarrow \texttt{CS.Sign}(par, pk_C, m, sk_S)$, we require that $z \leftarrow \texttt{CS.Valid}(par, pk_S, pk_C, m, \sigma, sk_C)$ yields $z = \texttt{accept}$. 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 $(pk_C, sk_C) \in \{\texttt{CS.KeyGen}_C(par)\}$ where $\{\texttt{CS.KeyGen}_C(par)\}$ denotes the set of all possible confirmer key pairs generated by $\texttt{CS.KeyGen}_C$[5].

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 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 $\texttt{Exp}^{\texttt{cs-uf-cma}}_{CS,\mathcal{A}}(1^k)$ shown in Figure 4. 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 $\sigma \leftarrow \texttt{CS.Sign}(par, pk_C, m, sk_S)$, adds (pk_C, m, σ) to L_{CSSign}, and returns σ.

Definition 7. *A core confirmer signature scheme CS is said to be* unforgeable *if no PPT algorithm $\mathcal{A}$ with non-negligible advantage $\texttt{Adv}^{\textit{cs-uf-cma}}_{CS,\mathcal{A}}(k) = \Pr[\texttt{Exp}^{\textit{cs-uf-cma}}_{CS,\mathcal{A}}(1^k) = 1]$ exists.*

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 $\texttt{Exp}^{\texttt{cs-inv-cma}}_{CS,\mathcal{A}}$ shown in Figure 4. 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 the verification token $tk_\sigma \leftarrow \texttt{CS.Convert}(par, pk_S, pk_C, m, \sigma, sk_C)$.

Note that $\mathcal{O}_{Convert}$ 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 8. *A core confirmer signature scheme CS is said to be* invisible *if there exists no PPT algorithm $\mathcal{A}$ with non-negligible advantage $\texttt{Adv}^{\textit{cs-inv-cma}}_{CS,\mathcal{A}}(k) = |\Pr[\texttt{Exp}^{\textit{cs-inv-cma}}_{CS,\mathcal{A}}(1^k) = 1] - \frac{1}{2}|$ exists.*

[5] This property is needed to prove confirmer soundness (Theorem 15) of the construction presented in Section 4.3.

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

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

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

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

Lastly, we consider token soundness 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 $\texttt{Exp}^{\texttt{cs-tk-snd}}_{CS,\mathcal{A}}$ shown in Figure 4. In the figure, $\{\texttt{CS.KeyGen}_C(par)\}$ denotes the set of all possible key pairs generated by $\texttt{CS.KeyGen}_C$.

Definition 9. *A core confirmer signature scheme CS is said to provide* token soundness *if there exists no PPT algorithm $\mathcal{A}$ with non-negligible advantage* $\texttt{Adv}^{cs\text{-}tk\text{-}snd}_{CS,\mathcal{A}}(k) = \Pr[\texttt{Exp}^{cs\text{-}tk\text{-}snd}_{CS,\mathcal{A}}(1^k) = 1]$.

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.

$$\Sigma_S\{(x, (sk_S, r)) : (pk_S, sk_S) \in \{\texttt{CS.KeyGen}_S(par)\} \wedge \\ \sigma = \texttt{CS.Sign}(par, pk_C, m, sk_S; r)\}$$

$$\Sigma_C\{(x, sk_C) : (pk_C, sk_C) \in \{\texttt{CS.KeyGen}_C(par)\} \wedge \\ \texttt{CS.Valid}(par, pk_S, pk_C, m, \sigma, sk_C) = \texttt{accept}\}$$

$$\overline{\Sigma}_C\{(x, sk_C) : (pk_C, sk_C) \in \{\texttt{CS.KeyGen}_C(par)\} \wedge \\ \texttt{CS.Valid}(par, pk_S, pk_C, m, \sigma, sk_C) = \texttt{false}\}$$

In the above, 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 assume that the challenge space of Σ_S, Σ_C and $\overline{\Sigma}_C$ are of the same size.

4.2 On-line Non-transferable Protocols

Our construction of on-line non-transferable protocols is based on a simple and intuitive approach inspired by the construction of designated verifier proofs by Jakobsson *et al.* [10] and is furthermore closely related to the construction of efficient zero-knowledge proofs in the auxiliary string model [6]. More specifically, we modify the compatible 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 $com \leftarrow \texttt{Comm}(ck, a, r)$ for random r, and 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 $com = \texttt{Comm}(ck, a, r)$, 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 $\texttt{NT}(ck)\text{-}\Sigma$ 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 `SimSign`, `SimCon` and `SimDis` in many of the security notions defined in Section 3, 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 a scheme T defined by

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

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 $(com, aux) \leftarrow \texttt{TdComm}$, 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 $r \leftarrow \texttt{TdOpen}(aux, w, td)$ such that $com = \texttt{Comm}(ck, w, r)$. 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 10. *A trapdoor commitment scheme T is said to be* binding under selective trapdoor openings *if no PPT algorithm $\mathcal{A}$ with non-negligible advantage* $\texttt{Adv}^{bind}_{T,\mathcal{A}}(k)$ *exists.*

Pedersen's commitment scheme [16] 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 the security of ordinary signatures [18]. In the full version, we recall this scheme and prove it binding under selective trapdoor openings.

4.3 Combined Scheme

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

- $\texttt{Setup}(1^k)$: Compute $par_S \leftarrow \texttt{S.Setup}(1^k)$ and $par_{CS} \leftarrow \texttt{CS.Setup}(1^k)$, 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 $\texttt{CS.Sign}$ algorithm.
- $\texttt{KeyGen}_S(par)$: Return $(pk_S, sk_S) \leftarrow \texttt{CS.KeyGen}_S(par_{CS})$.
- $\texttt{KeyGen}_C(par)$: Return $(pk_C, sk_C) \leftarrow \texttt{S.KeyGen}(par_S)$.
- $\texttt{KeyGen}_V(par)$: Return $(pk_V, sk_V) \leftarrow \texttt{T.G}(1^k)$.
- $\texttt{CSetup}(par, pk_S, sk_C)$: Compute the key pair $(pk'_C, sk'_C) \leftarrow \texttt{CS.KeyGen}_C(par_{CS})$ and the signature $\delta \leftarrow \texttt{S.Sign}(par_S, \text{“}pk_S||pk'_C\text{”}, sk_C)$, and return $pk_{C,S} \leftarrow (pk'_C, \delta)$ and $sk_{C,S} \leftarrow sk'_C$.
- $\texttt{CKeyValid}(par, pk_S, pk_C, pk_{C,S})$ Let $pk_{C,S} = (pk'_C, \delta)$ and return the result of the verification $\texttt{S.Verify}(par_S, pk_C, \text{“}pk_S||pk'_C\text{”}, \delta)$.
- $(\texttt{Sign}, \texttt{Receive})$: The common input is $(par, pk_S, pk_C, pk_{C,S}, pk_V, m)$ where $pk_{C,S} = (pk'_C, \delta)$ and the signer is given sk_S as private input. The signer picks $r \in \mathcal{R}$ and computes $\sigma \leftarrow \texttt{CS.Sign}(par_{CS}, pk'_C, sk_S, pk_C||pk_{C,S}||m; r)$. Then the signer sends σ to the verifier, and interacts with the verifier in the protocol $\texttt{NT}(pk_V)\text{-}\Sigma_S$ using $(par, pk_S, pk'_C, pk_C||pk_{C,S}||m, \sigma)$ as common input and (sk_S, r) as secret input[6].

[6] Note that this construction of the sign protocol is slightly more flexible than required by the definition in Section 2 in that a signer is able to re-confirm a signature by running $\texttt{NT}(pk_V)\text{-}\Sigma_S$. This, however, requires the signer to remember the randomness used to construct the signature.

- $\texttt{Convert}(par, pk_S, m, \sigma, sk_{C,S})$: Return the core confirmer signature verification token $tk_\sigma \leftarrow \texttt{CS.Convert}(par_{CS}, pk_S, m, \sigma, sk_{C,S})$.
- $\texttt{TkVerify}(par, pk_S, pk_C, pk_{C,S}, m, \sigma, tk_\sigma)$: Firstly, verify the validity of $pk_{C,S}$ by computing $z \leftarrow \texttt{CKeyValid}(par, pk_S, pk_C, pk_{C,S})$, and return $\texttt{reject}$ if $z = \texttt{reject}$. Otherwise, let $pk_{C,S} = (pk'_C, \delta)$ and return the output of $\texttt{CS.TkVerify}(par_{CS}, pk_S, pk'_C, pk_C||pk_{C,S}||m, \sigma, tk_\sigma)$.
- $(\texttt{Confirm}, V_C)$: The common input is given by $(par, pk_S, pk_C, pk_{C,S}, pk_V, m, \sigma)$ 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 $(par_S, pk_S, pk'_C, pk_C||pk_{C,S}||m, \sigma)$.
- $(\texttt{Disavow}, V_D)$: Having the same input as in $(\texttt{Confirm}, V_C)$, 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 common input $(par_S, pk_S, pk'_C, pk_C||pk_{C,S}||m, \sigma)$ and private input $sk_{C,S}$ to the confirmer.

Security. We will now state the theorems showing that the above constructed ONS scheme satisfies the security definitions given in Section 3 assuming the underlying primitives are secure. Due to space limitation, the proofs are not included here, but can be found in the full version.

Theorem 11. *Assume that CS is unforgeable, that Σ_S is honest verifier zero-knowledge and has special soundness, and that T is perfectly hiding and binding under selective trapdoor openings. Then the above ONS scheme N is unforgeable.*

Theorem 12. *Assume that S is strongly unforgeable. Then the above ONS scheme N has key unforgeability.*

Theorem 13. *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.*

Theorem 14. *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.*

Theorem 15. *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.*

Theorem 16. *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.*

4.4 Concrete Instantiation

To instantiate the above construction, we can use the strongly unforgeable signature scheme by Boneh *et al.* [1] and the double trapdoor Pedersen commitment scheme mentioned above. In the full version, we define and prove secure a core confirmer signature scheme and compatible sigma protocols to complete the instantiation. The scheme is essentially based on a linear encryption of the first component of a Waters signature [19] combined with the technique of Boneh *et al.* [1] to obtain a strongly unforgeable scheme. More specifically, the public/private key pairs of the signer and confirmer is given by $(pk_S, sk_S) = ((g^\alpha, g_2, h, F), \alpha)$ and $(pk_C, sk_C) = ((u, v), (x, y))$ where F is a Waters hash function and $u^x = v^y = g$, and a signature is of the form $\sigma = (u^a, v^b, g^{a+b+r}, g_2^\alpha H(M)^r, s)$ where $M = g^t h^s$, $t = H(pk_C || u^a || v^b || g^{a+b+r} || m)$, and H is a collision resistant hash function. The scheme is invisible assuming the decisional linear problem is hard, and is strongly unforgeable assume the discrete logarithm problem is hard, H is collision resistant and Waters signatures are (weakly) unforgeable. Furthermore, the structure of the scheme allows the compatible sigma protocols to be implemented using well-know techniques for proving equality and inequality of discrete logarithms. We refer the reader to the full version for the details.

5 Comparison

The generic construction by Liskov and Micali [13] 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, provides short signatures compared to the $\mathcal{O}(k)$ size signatures of [13] which furthermore allows a more efficient sign protocol, and lastly, all interactive protocols 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 [13]. We note, however, that our scheme requires large public keys whereas [13] can be instantiated to provide compact public keys, and that the non-interactive disavow protocol of [13] allows perfect non-repudiation whereas our scheme only achieves computational non-repudiation.

References

1. Boneh, D., Shen, E., Waters, B.: Strongly Unforgeable Signatures Based on Computational Diffie-Hellman. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 229–240. Springer, Heidelberg (2006)

2. Camenisch, J., Michels, M.: Confirmer signature schemes secure against adaptive adversaries. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 243–258. Springer, Heidelberg (2000)
3. Chaum, D.: Zero-knowledge undeniable signatures. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 458–464. Springer, Heidelberg (1991)
4. Chaum, D.: Designated Confirmer Signatures. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 86–91. Springer, Heidelberg (1995)
5. Chaum, D., van Antwerpen, H.: Undeniable Signatures. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 212–216. Springer, Heidelberg (1990)
6. Damgård, I.: Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 418–430. Springer, Heidelberg (2000)
7. Galbraith, S.D., Mao, W., Paterson, K.G.: RSA-Based Undeniable Signatures for General Moduli. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 200–217. Springer, Heidelberg (2002)
8. Gentry, C., Molnar, D., Ramzan, Z.: Efficient Designated Confirmer Signatures Without Random Oracles or General Zero-Knowledge Proofs. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 662–681. Springer, Heidelberg (2005)
9. Huang, X., Mu, Y., Susilo, W., Wu, W.: Provably Secure Pairing-Based Convertible Undeniable Signature with Short Signature Length. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 367–391. Springer, Heidelberg (2007)
10. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated Verifier Proofs and Their Applications. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 143–154. Springer, Heidelberg (1996)
11. Kudla, C., Paterson, K.G.: Non-interactive Designated Verifier Proofs and Undeniable Signatures. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 136–154. Springer, Heidelberg (2005)
12. Kurosawa, K., Heng, S.-H.: 3-Move Undeniable Signature Scheme. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 181–197. Springer, Heidelberg (2005)
13. Liskov, M., Micali, S.: Online-Untransferable Signatures. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 248–267. Springer, Heidelberg (2008)
14. Michels, M., Stadler, M.: Generic Constructions for Secure and Efficient Confirmer Signature Schemes. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 406–421. Springer, Heidelberg (1998)
15. Monnerat, J., Vaudenay, S.: Short 2-Move Undeniable Signatures. In: Nguyên, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 19–36. Springer, Heidelberg (2006)
16. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
17. Steinfeld, R., Bull, L., Wang, H., Pieprzyk, J.: Universal Designated-Verifier Signatures. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 523–542. Springer, Heidelberg (2003)
18. Teranishi, I., Oyama, T., Ogata, W.: General Conversion for Obtaining Strongly Existentially Unforgeable Signatures. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 191–205. Springer, Heidelberg (2006)
19. Waters, B.: Efficient Identity-Based Encryption Without Random Oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)