Online-Untransferable Signatures

Moses Liskov[1] and Silvio Micali[2]

[1] Computer Science Department,
The College of William and Mary
Williamsburg, VA 23187, USA
mliskov@cs.wm.edu
[2] CSAIL,
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
silvio@csail.mit.edu

Abstract. Non-transferability of digital signatures is an important security concern, traditionally achieved via interactive verification protocols. Such protocols, however, are vulnerable to "online transfer attacks" — i.e., attacks mounted during the protocols' executions.

In this paper, we show how to guarantee *online untransferability* of signatures, via a reasonable public-key infrastructure and general assumptions, without random oracles. Our untransferable signatures are as efficient as prior ones that provably provide weaker types of untransferability.

1 Introduction

Berkeley wishes to make a signed job offer to Alice. In this scenario, the ability of Alice to show the signature to others is a negative for Berkeley: for instance, Alice could use that ability to leverage a better offer from another university (e.g., Stanford). The transferability of digital signatures is indeed a well recognized concern. The focus of this paper is to make digital signatures as untransferable as possible. Let us start by recalling prior solutions to this problem.

A first solution to the problem of transferability in signatures was offered by Chaum and Van Antwerpen[10]. Their "undeniable signatures" cannot ever be verified without the signer's cooperation. The idea, therefore, is that the signature recipient should be unable to transfer a signature to a third party, because the signer would refuse to interact with that party.

Undeniable signatures, however, suffer from another drawback: the signer can effectively repudiate even a valid signature by refusing to cooperate. In the job offer scenario, this allows Berkeley to escape from the contract, leaving Alice no recourse.

Jakobsson, Sako, and Impagliazzo proposed *designated verifier signatures* [19], in which only a particular party, chosen by the signer, can verify without the signer's help. The untransferability of this solution is effectively the same as in undeniable signatures. Recent extensions allow a signature holder other than the original signer to designate a separate verifier [26,1]. However, this still falls short

R. Cramer (Ed.): PKC 2008, LNCS 4939, pp. 248–267, 2008.

of the solution we need for the job offer scenario: the designated verifier will have to be the recipient, since she must be convinced of the validity of the signature, but this implies that the recipient will be unable to establish its validity in court, should the signer be uncooperative.

Designated Confirmer Signatures. A solution to these repudiation problems has been provided by Chaum [9]. In a designated confirmer signature scheme (or DCS scheme for short), there are three parties: the signer, the recipient, and a trusted party called the *designated confirmer*. The idea is that the signer will produce a signature of an arbitrary message m in a way such that the recipient can be convinced of the validity of the signature in an interactive protocol. If such a signature of m is valid then either the confirmer or the signer will be able to prove its validity with respect to m, and will also be able to deny its validity with respect to any other message. Further, the confirmer and the signer should each be able to transform a valid signature of a message m into a traditional signature of m that can be verified by (and transferred to) anyone. Valuable variants of DCS schemes have been provided by Okamoto [22], Michels and Stadler [20], Camenisch and Michels [5], Goldwasser and Waisbard [16], Monnerat and Vaudenay [21], and Gentry, Molnar, and Ramzan [15].

The problem of online transferability. In prior solutions, the key to achieving untransferability is to make signature verification an interactive process between the sender and the receiver. Such untransferability, however, is guaranteed only after a certain time, namely, when the protocol completes. When Berkeley uses a DCS scheme to sign Alice's job offer, she will be unable to convince Stanford that the DCS signature she received was valid, but assuming that Alice attempts to convince Stanford of the validity of the signature only *after* completing the verification protocol. We call such an attack an *offline transfer*. On the other hand, Alice and Stanford could be actively communicating *during* the protocol Alice engages in with Berkeley, in which case, Alice may attempt to convince Stanford interactively of the validity of the signature. We call such an attack an *online transfer*.

All prior solutions to the problem of preventing transfer of signatures are vulnerable to online transfer.[1] Consider the following common paradigm to guarantee untransferability. The sender produces a signature σ of the message m and encrypts σ under a public encryption key to obtain a ciphertext c. Then, to prove that the signature is valid, the sender provides a zero-knowledge proof that c is an encryption of a valid signature of m. Here, it is then apparent that if Alice merely acts as a passive conduit for a conversation that is really taking place between Berkeley and Stanford, Stanford necessarily will be convinced that the job offer is genuine, because the legitimate recipient ought to be convinced, and there is no difference between a transferee and the legitimate recipient in this attack. Notice that even replacing the general zero-knowledge

[1] Steinfeld et al. [26] show how to prevent the transfer of a proof of ownership of a signature in a way that is online-untransferable. However, there is no confirmer, so the recipient of such a proof cannot trust that it will not be repudiated.

proof with a stronger form of zero-knowledge (e.g. nonmalleable [14] or resettable zero-knowledge proofs [8]) does not appear to help. Whether relying on zero-knowledge proofs, or using some other type of protocol, if the (illegitimate) transferee and the (legitimate) recipient of the signature cannot be meaningfully distinguished, this attack remains viable.

Our Model. In our job offer example, Alice is merely an entity who makes some random choices in order to be convinced by Berkeley that the signature is valid. To prevent online transferability, therefore, we put forward a reasonable model that ensures some kind of distinction between Alice and Stanford.

The distinction we propose to build upon is this: although Alice and Stanford may be colluding at present, if they are separate entities at all, Alice and Stanford will not have been colluding at some point in the past. We thus plan to solve the online untransferability problem via a model that in essence *forces* part of the signature process to take place when Alice and Stanford are not colluding.

Our model is very simple. It consists of a public-key infrastructure (PKI) in which not only signers and confirmers have registered public keys, but signature verifiers have them as well. In a variant of our model—guaranteeing a stronger version of online untransferability— verifiers can register their public keys only after providing a proof of knowledge of their corresponding secret keys.

First of all, our basic model is very *reasonable:* in fact, some form of PKI is necessary for signatures to be meaningful. In addition, even our variant model is realistic. Indeed, PKI requiring proofs of knowledge of secret keys have been considered in the past and proved to possess many attractive properties. In particular, plaintext-aware encryption can be realized without random oracles [17] in this model, resettable zero-knowledge can be achieved [8], and this model is favorable for universally composable security [2]. In addition, Steinfeld et al. rely on this same model in order to establish proofs of signature knowledge that cannot be transferred. Thus, not only are such PKIs feasibly implemented, but actually have many independent and valid reasons to be used.

There must be a crucial point in time in the past at which Alice was honest (or at least, that she was not colluding with Stanford). It is by leveraging this past point that online untransferability can be guaranteed in the present. Our model assumes that the time at which Alice registers her key is such a time in the past. This provides a meaningful version of online untransferability.

Our Solution. Our model provides only a framework in which online untransferability is *plausible*. It is, however, quite far from guaranteeing the existence of a solution, let alone a *reasonably efficient* solution.

The high-level structure of our solution (like that of Gentry et al. [15]) is that the signer produces (1) an encryption c that specifies the message m, (2) a zero-knowledge proof that c specifies m appropriately, and (3) a signature of c, along with certain elements of the proof transcript.

It is a crucial property for the security of our solution that the proof in point 2 is a full-fledged zero-knowledge proof, and that can be simulated without rewinding. Due to the result of [8], this guarantees that this proof is concurrently zero-knowledge. While this does not in itself imply online untransferability, it will imply

that we do not need to worry whether our separately proven online untransferability will apply in a concurrent setting. (The concurrent setting is very natural, where we may imagine multiple verifiers, multiple third parties, et cetera.)

Most prior protocols for achieving (just) offline untransferability achieved a reasonable level of efficiency by either relying on unusually strong assumptions (e.g. the random oracle assumption) [5,20], or provided weaker security by relying on protocols that are not fully zero-knowledge (e.g. the solution of Goldwasser and Waisbard [16] at TCC 2004). A second crucial property of our solution is that it be reasonably efficient for the provable security it delivers, and does not rely on the random oracle assumption.

Most known constructions of DCS schemes fall under the following paradigm. To create a confirmer signature, the signer creates a traditional signature σ, and encrypts it under the confirmer's public encryption key to produce c. Thus, the extraction requirement is guaranteed by the fact that both the signer and the confirmer can produce σ. To verify that c is a designated confirmer signature of m, one proves in zero-knowledge that c is indeed an encryption of a valid traditional signature of m relative to the proper public keys. Similarly, zero-knowledge proofs are used to disavow invalid signatures.

Goldwasser and Waisbard [16] were the first to give practical and efficient schemes in the plain model by using strong witness-hiding proofs instead of fully zero-knowledge ones. This achieves a weaker, but reasonable, level of security: in their scheme, transfer is only prevented when the transferee is honest. Gentry, Molnar, and Ramzan [15] give a practical and efficient scheme based on the Paillier cryptosystem [23], and their proofs of confirmation and disavowal are fully zero-knowledge, so they prevent all offline transfer.

One drawback of our solution is that, in order to prevent the transfer of signatures, the signer must be willing to issue invalid signatures to anyone. This is, however, in the signer's interest as the signer is the one being protected by the untransferability properties.

Our Results. We give a secure and efficient scheme similar to a designated confirmer signature scheme under general assumptions, without random oracles or general zero-knowledge proofs for secure designated confirmer signatures. If we assume the recipient and the third party (the transferee) were not conspiring at the time the recipient registers his or her public key, even online transfer cannot occur in our scheme. However this assumption is not critical in any other way; all other properties can be proven without it, including the impossibility of offline transfer.

2 Definitions

2.1 Intuitive Description

There are three players, the signer S, the confirmer C, and the recipient R. Before any signatures are issued, there is a setup phase in which all three parties generate public keys, PK_S, PK_C and PK_R respectively, that are assumed to

be known to all parties (or are certified by a PKI). Each party also generates a secret key: SK_S, SK_C, and SK_R, respectively.

It is assumed that in any algorithm or protocol, each party has their own secret key and all public keys as inputs. It is further assumed that 1^k is an input to all parties in all algorithms or protocols, where k is a system-wide security parameter.

An *online-untransferable signature scheme with confirmer* consists of the following protocols:

- $\mathsf{KeyGen}_S, \mathsf{KeyGen}_C, \mathsf{KeyGen}_R$ are algorithms for generating the public and private keys of each party.
- Setup. This is an algorithm run by the confirmer, once per signer, in which the confirmer produces an additional public key $PK_{C,S}$ which is to be used by the signer S in creating designated confirmer signatures for confirmer C, and a secret key $SK_{C,S}$ which the confirmer remembers for use later.[2]
- Sign. This is an interactive protocol between the signer and the recipient on common input a message m. At the end of the protocol, the recipient outputs an online-untransferable signature σ and either accepts or rejects, while the signer outputs an online-untransferable signature σ'.
- Disavow. This in an interactive protocol between the confirmer and the recipient, in which the confirmer proves that the given signature σ is not a valid one.
- $\mathsf{Extract}_C, \mathsf{Extract}_S$. This is a non-interactive algorithm in which the confirmer or signer, respectively, on input an online-untransferable signature σ, outputs an extracted signature σ^*.
- ExVerify. This is a non-interactive algorithm that can be performed by any party, given the public keys, on input an extracted signature σ^* that either accepts or rejects that signature.
- FakeSign. In order to prove the impossibility of online transfer, the simulator will need an invalid but valid-looking signature from the signer; that created and given to the simulator in this protocol.[3] At the end of the protocol, the signer outputs an online untransferable signature σ, and the simulator outputs an online-untransferable signature σ'.

The security requirements, informally, are the following:

Completeness: When all players are honest, the online untransferable signature produced by the recipient in Sign will be valid (that is, a valid extracted signature

[2] This algorithm is not one included traditionally, but its addition is reasonable: we expect that it will be performed offline, just after key generation. We can avoid having this additional setup step if we make the stronger assumption that identity-based encryption [3] exists.

[3] It may seem strange to describe this algorithm as part of the scheme: it is only to be used in the proofs of non-transferability, and need never be run in practice. However, it is important that the signer be willing to engage in it, because the third party *must believe* that the signer would. Because of this, it is important to include it in the description of the scheme, because the signer's willingness to engage in this protocol should not affect any other security properties of the scheme.

can be extracted from it). Also, the signatures produced by the signer in FakeSign will not be valid, and the recipient will accept in Disavow on such a signature.

Soundness: No dishonest signer can succeed in making the honest recipient accept in Sign unless the resulting designated confirmer signature is valid (that is, can be successfully extracted by the designated confirmer.)

Non-repudiation: No dishonest confirmer can succeed in making the honest recipient accept in Disavow on a valid (extractable) signature.

Unforgeability: No adversary with the ability to engage in any of the above protocols with the honest confirmer and the honest signer (including FakeSign) in any role and on any common input, can produce either a valid online-untransferable signature σ' on a message the adversary never requested a signature of, nor a valid extracted signature $\sigma^{*\prime}$, on a message the adversary didn't first request a signature, and later request extraction.

Online untransferability: Sign can be simulated in such a way that is indistinguishable from a real interaction to any distinguisher, so long as the adversary does not request an extraction of that signature. The simulator is assumed to have access to the secret key of the recipient, and may engage in FakeSign with the signer, but must engage in the Sign protocol interactively with the distinguisher.

Offline untransferability: There is a simulator that can produce a view indistinguishable from that of the dishonest recipient in the Sign protocol with the real signer, so long as the adversary never requests an extraction of the result of that protocol. The simulator is assumed to be able to engage in the FakeSign protocol with the signer.

2.2 Notation

When S is a finite set, the notation $x \leftarrow S$ refers to x being chosen uniformly at random from S. When M denotes a randomized algorithm, $x \leftarrow M(i)$ refers to x being determined by a random execution of M on input i. When we write $x_1 \leftarrow D_1; x_2 \leftarrow D_2(x_1); \ldots, x_r \leftarrow D_r(x_1, \ldots, x_{r-1})$ we refer to the probability distribution on $\{x_1, \ldots, x_r\}$ determined by first assigning x_1 according to D_1, then assigning x_2 according to D_2 on input x_1, et cetera.

When M is a two-party protocol, $(x_A, x_B) \leftarrow M^{A,B}(i_A; i_B; i)$ refers to an assignment where M is executed between parties A and B, where A's private input is i_A, B's private input is i_B, and i is the common input, and where x_A becomes the output of A, and x_B becomes the output of B. We omit the inclusion of A's secret key in i_A, B's secret key in i_B, and all public keys in i; we write only $M^{A,B}(i)$ to indicate that no unusual secret inputs are required. We use $x \leftarrow_b M^{A,B}(i_A; i_B; i)$ to refer to an assignment where x becomes x_A if $b = 1$ and x becomes x_B if $b = 2$; that is, b specifies which party's output is to be denoted by x.

When M is a two-party protocol, and P is one of the parties that participates in M, the notation $M^{P,\cdot}$ or $M^{\cdot,P}$ refers to the set of interactive Turing machines run by the honest party P in their execution of M. Thus, when an adversary is

said to have oracle access to $M^{\cdot,P}$, this means the adversary has oracle access to all the Turing machines used by P during honest execution of M, where P has all ordinary inputs (the public keys of all parties, the security parameter 1^k, and P's own private keys), however, the adversary has control of all other inputs.

Similarly, when M is an algorithm, M^P (where P is the party that runs algorithm M) is that algorithm with the standard inputs of P specified, that is, all public keys, the security parameter, and P's own secret key. When M is an algorithm, we denote by $M(i;r)$ that we run M on input i with randomness r. When r is not previously specified, it is assumed to be chosen at random and remembered.

Honest parties are assumed to be state-preserving interactive Turing machines. Adversaries are assumed to be state-preserving oracle Turing machines. We write $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_r\}$ to indicate a single oracle that can be used to query any of the sub-oracles $\mathcal{O}_1, \ldots, \mathcal{O}_r$.

We use the symbol ν to designate a *negligible function*. A function is negligible if, for any $c > 0$, $\nu(k) < k^{-c}$ for all sufficiently large k.

2.3 Formal Definitions

Online-untransferable signatures. An online untransferable signature scheme is a tuple of several algorithms and two-party protocols:

1. $\mathsf{KeyGen}_S, \mathsf{KeyGen}_R, \mathsf{KeyGen}_C, \mathsf{Setup}, \mathsf{ExVerify}, \mathsf{Extract}_S$, and $\mathsf{Extract}_C$ are algorithms,
2. Sign and $\mathsf{FakeSign}$ are two-party protocols run between the signer (the first party) and a recipient (second party).
3. $\mathsf{Disavow}$ is a two-party protocol run between the confirmer (first party) and a recipient (second party).

Such algorithms and two-party protocols constitute a secure online-untransferable signature scheme if the following properties hold:

Efficiency: All algorithms, and all defined behavior for honest parties in two-party protocols, are probabilistic polynomial-time.

Completeness: If keys are generated honestly and setup is performed honestly, and the signing protocol is performed between honest parties, the result will be an online-untransferable signature that produces a valid extracted signature under both $\mathsf{Extract}^C$ and $\mathsf{Extract}^S$. If the $\mathsf{FakeSign}$ protocol is performed between an honest S and an honest R, the result is an online-untransferable signature that will be disavowed by $\mathsf{Disavow}$. Formally,

$\forall m,$
$\Pr[\ (PK_S, SK_S) \leftarrow \mathsf{KeyGen}_S(1^k); (PK_R, SK_R) \leftarrow \mathsf{KeyGen}_R(1^k);$
$(PK_C, SK_C) \leftarrow \mathsf{KeyGen}_C(1^k); (PK_{S,C}, SK_{S,C}) \leftarrow \mathsf{Setup}^C(PK_S);$
$(\sigma, x) \leftarrow_2 \mathsf{Sign}^{S,R}(m);$
$\sigma' \leftarrow_2 \mathsf{FakeSign}^{S,R}(m);$

$y \leftarrow \mathsf{ExVerify}(\mathsf{Extract}_S(\sigma)); z \leftarrow \mathsf{ExVerify}(\mathsf{Extract}_C(\sigma));$
$w \leftarrow_2 \mathsf{Disavow}^{C,R}(\sigma') :$
$x = y = z = w = \text{accept}] = 1$

Soundness: For all S' with oracle access to all of the algorithms and two party protocols run by parties C and R, and for all σ, if PK_C and PK_R are generated according to KeyGen_C and KeyGen_R, the probability that σ is not a valid signature, but S' succeeds in making the recipient accept and output σ in Sign is negligible. Formally, let $\mathcal{O} = \{\mathsf{Sign}^{\cdot,R}, \mathsf{Setup}^C, \mathsf{Disavow}^{\cdot,R}, \mathsf{FakeSign}^{\cdot,R}, \mathsf{Disavow}^{C,\cdot}, \mathsf{Extract}_C^C\}$. Then,

$\forall A$ oracle PPT, $\exists \nu\ \forall k$
$\Pr[\ (PK_R, SK_R) \leftarrow \mathsf{KeyGen}_R(1^k); (PK_C, SK_C) \leftarrow \mathsf{KeyGen}_C(1^k);$
$PK_S \leftarrow A^{\mathcal{O}}(PK_R, PK_C, 1^k); (PK_{S,C}, SK_{S,C}) \leftarrow \mathsf{Setup}^C(PK_S);$
$m \leftarrow A^{\mathcal{O}}; (\sigma, x) \leftarrow_2 \mathsf{Sign}^{A^{\mathcal{O}},R}(m);$
$z \leftarrow \mathsf{ExVerify}(\mathsf{Extract}_C^C(\sigma), m) :$
$x = \text{accept} \wedge z \neq \text{accept}\ ] < \nu(k)$

Non-Repudiation: The weakest possible notion here is that it should be hard for a dishonest signer and a dishonest confirmer to conspire to create a valid online-untransferable signature that could be successfully disavowed. We will use a stronger formulation, namely, that no such signatures exist for validly generated keys.

$\forall C'$ PPT adversary, $\forall (PK_C, SK_C) \in \mathsf{KeyGen}_C, (PK_S, SK_S) \in \mathsf{KeyGen}_S,$
$(PK_{C,S}, SK_{C,S}) \in \mathsf{Setup}, m, \sigma,$
$\Pr[\ x \leftarrow \mathsf{ExVerify}(\mathsf{Extract}_C^C(\sigma), m); y \leftarrow_2 \mathsf{Disavow}^{C',R}(SK_C, SK_S, SK_{C,S}; ; \sigma) :$
$x = y = \text{accept}] = 0$

Unforgeability: For all adversaries with oracle access to all algorithms run by all honest parties, if keys are generated honestly, cannot succeed in either (1) producing a valid signature σ on a message he never requested a signature of, or (2) producing a valid extracted signature σ^* on a message he never requested a signature of and then later requested extraction of. Formally, let $\mathcal{O} = \{\mathsf{Sign}^{\cdot,R}, \mathsf{FakeSign}^{\cdot,R}, \mathsf{FakeSign}^{S,\cdot}, \mathsf{Disavow}^{\cdot,R}, \mathsf{Extract}_S, \mathsf{Extract}_C, \mathsf{Setup}^C\}$. Then

$\forall A$ oracle PPT, $\forall p\ \exists \nu\ \forall k$
$\Pr[\ (PK_S, SK_S) \leftarrow \mathsf{KeyGen}_S(1^k); (PK_R, SK_R) \leftarrow \mathsf{KeyGen}_R(1^k);$
$(PK_C, SK_C) \leftarrow \mathsf{KeyGen}_C(1^k); (PK_{S,C}, SK_{S,C}) \leftarrow \mathsf{Setup}^C(PK_S);$
$m_1 \leftarrow A^{\mathcal{O}}(PK_S, PK_C, PK_R, PK_{S,C}); (\sigma_1, \omega_1) \leftarrow \mathsf{Sign}^{S,A^{\mathcal{O}}}(m_1); \ldots;$
$m_{p(k)} \leftarrow A^{\mathcal{O}}(\sigma_{p(k)-1}); (\sigma_{p(k)}, \omega_{p(k)}) \leftarrow \mathsf{Sign}^{S,A^{\mathcal{O}}}(m_{p(k)});$
$(m, \sigma) \leftarrow A^{\mathcal{O}}(\sigma_{p(k)}); y \leftarrow \mathsf{ExVerify}(\mathsf{Extract}_C^C(\sigma), m); z \leftarrow \mathsf{ExVerify}(\sigma, m) :$
$z = \text{accept}$, and if $m = m_i$ then A did not query $\mathsf{Extract}$ on σ_i, or
$y = \text{accept}$, but $m \notin \{m_1, \ldots, m_{p(k)}\}] < \nu(k)$

Online Untransferability: For all adversaries with oracle access to all algorithms run by all honest parties, if keys for C, S, and R are generated honestly, then the adversary cannot distinguish between interacting with the real signer in Sign about a chosen message m and interacting with a simulator with access only to $\mathsf{FakeSign}$ on m, so long as the adversary never requests $\mathsf{Extract}$ or $\mathsf{Disavow}$ be run on the resulting signature. Let $\mathcal{O} = \{\mathsf{Sign}^{\cdot,R}, \mathsf{Sign}^{S,\cdot}, \mathsf{FakeSign}^{\cdot,R}, \mathsf{FakeSign}^{S,\cdot}, \mathsf{Extract}_S^S, \mathsf{Disavow}^{\cdot,R}, \mathsf{Disavow}^{C,\cdot}, \mathsf{Extract}_C^C, \mathsf{Setup}^C\}$, and let $\mathcal{O}_\sigma$ be $\mathcal{O}$ except where the $\mathsf{Disavow}$, $\mathsf{Extract}_C$, $\mathsf{Extract}_S$ oracles will not operate if given σ as input. Then:

$\forall A$ oracle PPT, $\exists \mathsf{Sim}\ \exists \nu\ \forall k$

$$
\begin{aligned}
|\Pr[\ & (PK_S, SK_S) \leftarrow \mathsf{KeyGen}_S(1^k); (PK_R, SK_R) \leftarrow \mathsf{KeyGen}_R(1^k); \\
& (PK_C, SK_C) \leftarrow \mathsf{KeyGen}_C(1^k); (PK_{S,C}, SK_{S,C}) \leftarrow \mathsf{Setup}^C(PK_S); \\
& m \leftarrow A^{\mathcal{O}}(PK_S, PK_R, PK_C, PK_{S,C}, SK_R); (\sigma, \omega) \leftarrow \mathsf{Sign}^{S,A}(m); \\
& b \leftarrow A^{\mathcal{O}_\sigma} : b = 1] - \\
\Pr[\ & (PK_S, SK_S) \leftarrow \mathsf{KeyGen}_S(1^k); (PK_R, SK_R) \leftarrow \mathsf{KeyGen}_R(1^k); \\
& (PK_C, SK_C) \leftarrow \mathsf{KeyGen}_C(1^k); (PK_{S,C}, SK_{S,C}) \leftarrow \mathsf{Setup}^C(PK_S); \\
& m \leftarrow A^{\mathcal{O}}(PK_S, PK_R, PK_C, PK_{S,C}, SK_R); \\
& (\sigma, \omega) \leftarrow \mathsf{Sign}^{\mathsf{Sim}^{\mathsf{FakeSign}^S}, A}(SK_R; -; m); b \leftarrow A^{\mathcal{O}_\sigma} : b = 1]| < \nu(k)
\end{aligned}
$$

Offline Untransferability: For all dishonest recipients R' and for all adversaries with oracle access to all algorithms run by all honest parties, there is a simulator Sim such that if keys for C and S are generated honestly, the adversary cannot distinguish between R' after interacting with the signer in Sign and Sim after interacting only with the signer in $\mathsf{FakeSign}$. Let $\mathcal{O}$ and $\mathcal{O}_\sigma$ be as in the online untransferability definition. Then:

$\forall A$ oracle PPT, $\forall R'$ PPT $\exists \mathsf{Sim}$ oracle PPT $\ \exists \nu\ \forall k$

$$
\begin{aligned}
\Pr[\ & (PK_S, SK_S) \leftarrow \mathsf{KeyGen}_S(1^k); (PK_C, SK_C) \leftarrow \mathsf{KeyGen}_C(1^k); \\
& (PK_{S,C}, SK_{S,C}) \leftarrow \mathsf{Setup}^C(PK_S); \\
& (m, PK_R, \alpha) \leftarrow A^{\mathcal{O}}; x_0 \leftarrow_2 \mathsf{Sign}^{S,R'}(SK_S; \alpha; m); \\
& x_1 \leftarrow \mathsf{Sim}^{\{R', \mathsf{FakeSign}^S\}}(\alpha, m); b \leftarrow \{0,1\} \\
& b' \leftarrow A^{\mathcal{O}_{\sigma_b}}(\sigma_b, \omega_b) : b' = b] < 1/2 + \nu(k)
\end{aligned}
$$

There are two main differences between the online and offline definitions for untransferability. First, in the online untransferability definition, there is only the simulator Sim and the adversary A; the adversary in this case is meant to model both the recipient and the third party. In the offline untransferability definition, there are two adversaries: the dishonest recipient R' and the third party, represented by A. A receives output either from the real signing protocol or from the simulator, but cannot interact in those protocols directly. The second difference is that in the online untransferability definition, the simulator knows the receiver's secret key SK_R (this models the notion that the recipient is aware of his own key), whereas the simulator is not given this information in the offline definition. Naturally, if the recipient is required to perform a proof of

knowledge of SK_R during key registration, then online untransferability implies offline untransferability.

3 Our Construction

Our construction is fairly complex, so to help the reader understand it, we present our ideas incrementally.

As a first idea, we imagine that to make a designated confirmer signature on message m, the signer will create k random pairs of strings α_i, β_i such that $\alpha_i \oplus \beta_i = m$, and encrypt these values in the confirmer's encryption key to obtain $a_i = E_{PK_C}(\alpha_i)$ and $b_i = E_{PK_C}(\beta_i)$. The signer will then sign m along with $a_1, b_1, \ldots, a_k, b_k$; the signature is considered valid so long as σ is valid, and some pair a_i, b_i decrypt to values that XOR to m.

The recipient can verify on his own that σ is valid, but the signer and recipient must engage in a protocol for the recipient to be convinced that some pair decrypts to values that XOR to m. In order to accomplish this, the recipient first sends a commitment to a challenge string CH. The signer responds with $(\sigma, a_1, \ldots, b_k)$. The recipient checks σ and responds by opening CH. The signer then "opens" the encryption of a_i or b_i, depending on the ith bit of CH. Note that if none of the pairs actually decrypt to a pair that XOR to m, the probability that the signer will be able to succeed is 2^{-k}.

To extract a (valid) signature, the signer can simply decommit some pair of encryptions; this, along with the signature and the α_i value, is proof to anyone that the signature is valid.

Offline untransferability. In order to provide deniability, the signer must be willing to freely give out signatures that are valid *in format* but not valid *in content.* That is, the signer should provide a signature to any recipient on any m and any sequence of pairs $a_1, b_1, \ldots, a_k, b_k$ so long as (1) the signer obtains the decommitments of each a_i and b_i, and (2) the decryptions of each pair actually do not XOR to m. Thus, the mere signature of the signer proves nothing. Given this, a simulator can be constructed for the proof system: this fake signature service can be used to make the recipient reveal CH; once it is revealed, the simulator can rewind and use the fake signature service again to obtain an invalid signature for which the challenge CH can be answered.

Confirmer extractability. Another issue we must resolve is how the confirmer will actually extract plain signatures. In the current scheme, the confirmer will be able to decrypt all the pairs, but this does not necessarily imply that the confirmer will be able to *decommit* them.

In order to handle this, we modify our scheme. We ask that the signer assist the confirmer by encrypting the randomness used in producing a_i, b_i and including this in the signature. Now, if the signer is honest, the confirmer will be able to give the same decommitment information the signer would. If the signer is not honest, and does not properly include the decommitment information, the

confirmer can always reveal his decryption secret key as an alternative form of decommitment. However, this is obviously not an ideal solution since it ruins the confirmer's keys. To fix this problem, we modify the setting so that the confirmer's true public key is a signing key, and the confirmer creates a different encryption key pair for each signer, and signs the public encryption key along with the signer's public key. The signature assures the recipient that the signer is using the correct key. Now, if the signer doesn't help the confirmer extract signatures in the normal way, C can reveal this secret encryption key: in effect, C is still able to extract signatures, but S's assurance that signatures cannot be transferred is lost. Of course, S has no one else to blame, since S was the one who was dishonest.

This mechanism allows the confirmer not only to extract, but also to disavow signatures (disavowal is necessary because of the FakeSign protocol the signer provides) by decrypting all the pairs, or by revealing the secret key.

Reconfirmation. In designated confirmer signature schemes, a Verify protocol and a Disavow is typically provided both for the signer and for the confirmer, in order to prove the validity or invalidity of a designated confirmer signature to the recipient. Verify is often given as a separate protocol from Sign, in order to establish, initially, the validity of a signature. Here, though, the proof of validity is part of the Sign protocol, so Verify would be unneeded initially, and later, there would be no need for the recipient to reconfirm the validity of a signature already established as valid. We insist that the recipient sign the messages they send during the initial proof of validity, so that it will be clear that a given online-untransferable signature was produced after a proof was provided to the recipient.

Online untransferability. The difference between online untransferability and offline is that in an attempt at online transfer, the dishonest recipient interacts concurrently with both the signer and some third party. To prevent online transfer, we will need to assume that the recipient knows their secret key. As part of key generation for a recipient, the recipient generates an encryption key pair, that will be used for the initial commitment to CH. If we assume that the simulator knows the corresponding secret key, the simulator can determine CH *without rewinding*, which is what allows us to simulate signing in the presence of an actively interacting adversary.

3.1 The Scheme

Now, we will give the full specification of our scheme. We assume the existence of a secure (IND-CPA), perfectly-faithful[4], checkable[5] public-key cryptosystem (G, E, D) and a secure (CMA) signature scheme (KeyGen, Sig, Ver).

[4] That is, decryption inverts encryption with probability 1.

[5] That is, there is a simple check given PK and SK to determine whether SK could be generated along with PK. It is easy to make any cryptosystem checkable, by simply including the randomness used in key generation in the secret key.

The specification of the algorithms and protocols for our scheme are as follows:

- KeyGen_S: Generate a signature key pair (PK_S, SK_S) using KeyGen.
- KeyGen_C: Generate a signature key pair (PK_C, SK_C) using KeyGen.
- KeyGen_R: The recipient uses G to generate a key pair (PK_R^E, SK_R^E), and uses KeyGen to generate a signature pair PK_R^{sig}, SK_R^{sig}. The recipient's public key $PK_R = (PK_R^E, PK_R^{sig})$ and the recipient's secret key is $SK_R = (SK_R^E, SK_R^{sig})$.
- Setup: The confirmer generates an encryption key pair $(PK_{C,S}^E, SK_{C,S}^E)$ from G, and creates a signature σ_0 on the pair $(PK_S, PK_{C,S}^E)$ using SK_C. The key $PK_{C,S}$ consists of the triple $(\sigma_0, PK_S, PK_{C,S}^E)$, while $SK_{C,S} = SK_{C,S}^E$.
  Sign: The protocol runs in the following steps:
  1. The recipient generates a uniform random string CH of length k and sends $e = E_{PK_R^E}(CH; r)$ to the signer and remembers r for later use.
  2. The signer generates k uniform random strings $\alpha_1, \ldots, \alpha_k$ each of the same length as the message m. The signer produces $3k$ encryptions under the encryption key of $PK_{C,S}$: $a_i = E_{PK_{C,S}^E}(\alpha_i; r_i^0)$, $b_i = E_{PK_{C,S}^E}(\alpha_i \oplus m; r_i^1)$, and $c_i = E_{PK_{C,S}^E}(r_i^0 || r_i^1)$. The signer then sends the public key $PK_{C,S}$, and for each i, α_i, a_i, b_i, c_i to the recipient. The signer remembers r_i^0, r_i^1 for later use.
  3. The recipient checks that $PK_{C,S}$ contains a valid signature σ_0; if not, the recipient rejects. Otherwise, the recipient sends CH and r, as well as a signature σ_R on the tuple $(m, CH, PK_{C,S}, \alpha_1, \ldots, c_k)$ under SK_R^{sig}.
  4. The signer checks validity of the signature σ_R, and checks that $e = E_{PK_R^E}(CH; r)$; if either check fails, the signer aborts. The signer then sends $r_i^{CH_i}$ for each bit CH_i of the challenge string, to the recipient, along with a signature σ on $(m, CH, PK_{C,S}, PK_R, \alpha_1, \ldots, c_k, \sigma_R)$.
  5. The recipient checks, for each i such that $CH_i = 0$, that r_i^0 provided by S, used to encrypt α_i under $PK_{C,S}$, gives a_i. The recipient then checks, for each i such that $CH_i = 1$, that r_i^1 provided by S, used to encrypt $\alpha_i \oplus m$ under $PK_{C,S}$, gives b_i. The recipient then checks that the signature σ is a valid one. If all these checks are successful, the recipient accepts and outputs σ along with $(m, CH, PK_{C,S}, PK_R, \alpha_1, \ldots, c_k, \sigma_R)$, otherwise the recipient rejects.

At this point we pause to make a couple of remarks, which simplify the task of describing the remaining parts of the scheme. In our scheme, an online-untransferable signature is a signature σ on $m_\sigma = (m, CH, PK_{C,S}, PK_R, \alpha_1, \ldots, c_k, \sigma_R)$. Three things can be checked about σ and m_σ based only on public information, namely:

1. σ should be a valid signature under PK_S, and PK_S should be specified as part of the signature σ_0 in $PK_{C,S}$.
2. The signature σ_0 in $PK_{C,S}$ should be valid.
3. The signature σ_R should be a valid on $(m, CH, PK_{C,S}, \alpha_1, \ldots, c_k)$, checked with the verifying key PK_R^{sig} of PK_R.

For simplicity, we say that an online-untransferable signature σ is *format-valid* if all these checks are passed, and we assume that any of the below methods halt with an error if given a format-invalid online-untransferable signature.

- $\mathsf{Extract}_S$: On input a valid online-untransferable signature σ, m_σ the signer reveals $\sigma^* = (\sigma, i, r_i^0, r_i^1, \epsilon)$ for an arbitrary i.[6] Given an invalid online-untransferable signature, the signer rejects.
- $\mathsf{Extract}_C$: On input a format-valid online-untransferable signature σ, m_σ, the confirmer decrypts $a_1, b_1, \ldots, a_k, b_k$ under $SK_{C,S}$, and finds some i such that $D_{SK^E_{C,S}}(a_i) \oplus D_{SK^E_{C,S}}(b_i) = m$, and rejects if there is no such i. The confirmer then finds r_i^0 and r_i^1 by computing $D_{SK^E_{C,S}}(c_i)$, and checks to see if $E_{PK^E_{C,S}}(\alpha_i; r_i^0) = a_i$ and $E_{PK^E_{C,S}}(\alpha_i \oplus m; r_i^1) = b_i$. If so, the confirmer publishes $\sigma^* = (\sigma, m_\sigma, i, r_i^0, r_i^1, \epsilon)$. If not, the confirmer publishes $\sigma^* = (\sigma, m_\sigma, i, \epsilon, \epsilon, SK_{C,S})$.
- ExVerify: On input a quadruple $(\sigma, i, r^0, r^1, SK)$, we first check that σ is format-valid, and then check that either $E_{PK_{C,S}}(\alpha_i; r^0) = a_i$ and $E_{PK_{C,S}}(\alpha_i \oplus m; r^1) = b_i$ or that if this is not the case, that SK is the secret key associated with $PK_{C,S}$ and that $D_{SK}(a_i) \oplus D_{SK}(b_i) = m$.[7]
- Disavow: On input σ an invalid but format-valid untransferable signature, the confirmer decrypts each c_i to obtain r_i^0, r_i^1, and checks that for all i, $E_{PK^E_{C,S}}(D_{SK^E_{C,S}}(a_i)); r_i^0) = a_i$ and $E_{PK^E_{C,S}}\ (D_{SK^E_{C,S}}\ (b_i)); r_i^1) = b_i$. If so, the confirmer reveals $D_{SK^E_{C,S}}(a_i), D^E_{SK_{C,S}}(b_i), r_i^0$, and r_i^1 for each i. If not, but $D_{SK^E_{C,S}}(a_i) \oplus D_{SK^E_{C,S}}(b_i) \neq m$ for any i, the confirmer reveals $SK^E_{C,S}$. The recipient checks that no pair XORs to make m. Then, in the former case, the recipient checks that the r values properly decommit all of the a_i and b_is; in the latter case, the recipient checks that $SK^E_{C,S}$ is the secret key relating to $PK^E_{C,S}$.
- FakeSign: The recipient first asks the signer to provide $PK_{C,S}$, and then
  1. Chooses any $\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k$,
  2. Chooses random strings r_i^0, r_i^1, r_i^2 for each $1 \leq i \leq k$,
  3. Computes $a_i = E_{PK_{C,S}}(\alpha_i; r_i^0)$, $b_i = E_{PK_{C,S}}(\beta_i; r_i^1)$, and $c_i = E_{PK_{C,S}}(r_i^0 || r_i^1; r_i^2)$ for each i,
  4. Computes $\sigma_R = \mathsf{Sig}_{SK_R}(m, CH, PK_{C,S}, \alpha_1, a_1, b_1, c_1, \ldots, \alpha_k, a_k, b_k, c_k)$,

  and sends $m, CH, PK_{C,S}, PK_R, \sigma_R$ and for each i, $\alpha_i, \beta_i, a_i, b_i, c_i, r_i^0, r_i^1$, and r_i^2 to the signer. The signer then checks that r_i^0, r_i^1 and r_i^2 properly decommit a_i, b_i, c_i to α_i, β_i, and $r_i^0 || r_i^1$, respectively, and that for each i, $\alpha_i \oplus \beta_i \neq m$. If so, the signer produces a signature of $(m, CH, PK_{C,S}, PK_R, \alpha_1, \ldots, c_k, \sigma_R)$ and if it is format-valid, sends it to the recipient.

[6] In order to perform this function, the signer will need to be able to remember, for each online untransferable signature it issues, what the random strings r_i^0 and r_i^1 are that it used. This can be simplified by generating r_i^b according to a pseudorandom function with fixed seed and input, say, c.

[7] For simplicity, we imagine that the decryption key $SK_{C,S}$ includes the randomness used to generate the key pair $(PK_{C,S}, SK_{C,S})$, so checking that SK is the secret key associated with $PK_{C,S}$ involves regenerating the key pair from the same randomness.

We can now state our main result.

Theorem 1. ($\mathsf{KeyGen}_S, \mathsf{KeyGen}_R, \mathsf{KeyGen}_C, \mathsf{Setup}, \mathsf{ExVerify}, \mathsf{Extract}_S, \mathsf{Extract}_C, \mathsf{Sign}, \mathsf{FakeSign}, \mathsf{Disavow}$) *is an online-untransferable signature scheme.*

Proof. It should be clear that all the algorithms involved in our construction are efficient, and that our scheme satisfies completeness and non-repudiation.

Soundness: In order for adversary to succeed, the adversary must be able, with non-negligible probability, make the recipient accept in Sign but output a result that is not confirmed as valid by the confirmer. Such an adversary must either make the recipient accept with a σ_0 not produced by the confirmer, or make the recipient accept with a σ_0 produced by the confirmer, for an invalid signature. If the former occurs with non-negligible probability, the adversary can be used in a simple reduction to forge signatures relative to PK_C.

Otherwise, the adversary can make the recipient accept an invalid signature but with a σ_0 produced by the confirmer, with non-negligible probability.[8] If this is the case, note that if the signer sends encrypted pairs for an invalid signature, there is at most one string CH for which the adversary can send a satisfactory response in step 4. When the adversary sends its step 2 message, however, CH has not yet been revealed. We can therefore use the adversary's choice of encrypted pairs to break the security of encryption under PK_R^E.

The reduction is simple: we choose two random messages to distinguish, CH and CH', and obtain the encryption of one or the other under the key we are to break. We run the adversary in its attack, using this key as PK_R^E, and generating all other keys normally. In the sign protocol, we send the challenge ciphertext in step 1. In step 2, if the adversary responds with $PK_{C,S}$ that was produced by the confirmer (so we are aware of $SK_{C,S}^E$), we decrypt all the pairs and determine if the ultimate signature would be invalid. If the adversary uses an unexpected $PK_{C,S}$, or if the ultimate signature would be valid, we flip a coin. Otherwise, we determine if there exists a challenge string CH'' for which the adversary could give an answer in step 4; if so, and $CH'' = CH$, we output 0, otherwise, we output a random bit. It can be readily verified that a successful adversary results in a successful attack against PK_R^E.

Unforgeability: Note that any valid untransferable signature or extracted signature must first be a format-valid signature, and thus, a signature under the signer's key. The adversary cannot produce a forgery with an original signature (i.e. not issued by the signer) except with negligible probability, by the existential unforgeability of the signature scheme. Similarly, the adversary cannot reuse a signature issued in $\mathsf{FakeSign}$ because such a signature will never be valid. The remaining case is where the adversary is able to extract a signature obtained via Sign.

There are two ways this can happen: either the adversary produces and reveals $SK_{C,S}^E$ or it can demonstrate the decryption for both parts of one of the pairs. If

[8] Note that in this case the pairs must *be* invalid: otherwise, the confirmer would succeed in extracting a signature, since the confirmer knows $SK_{C,S}^E$.

the former happens with non-negligible probability, there are two sub-cases: either the adversary manages to query $\mathsf{Extract}_C$ on an input that will cause $SK^E_{C,S}$ to be revealed, or the adversary produces $SK^E_{C,S}$ without that information.

Note that any signature produced by the signer in Sign will not result in the key being revealed by $\mathsf{Extract}_C$. Similarly, a signature produced in FakeSign will not result in any answer from $\mathsf{Extract}_C$ since the signature will be invalid. Thus, if the adversary obtains $SK^E_{C,S}$ from $\mathsf{Extract}_C$, the adversary must have produced a signature never created by the signer. If this happens with non-negligible probability, a simple reduction shows that the existential unforgeability of the signature scheme is violated.

If the adversary outputs $SK^E_{C,S}$ but not via $\mathsf{Extract}_C$, we can attack the encryption scheme under key $PK^E_{C,S}$. We can run the adversary in its attack without knowing $SK^E_{C,S}$ ourselves (and, without ever needing to decrypt with it); if the adversary can determine the correct $SK^E_{C,S}$ in such a circumstance, we can easily decipher messages.

If the adversary reveals both parts of one of the pairs, we can make a reduction to break the security of the encryption scheme. The proof is a hybrid argument. First we argue that the adversary cannot distinguish between a normal setting in which for one random instance of the Sign protocol, the adversary never requests extraction of that signature, and one in which for one random instance of the Sign protocol, all the c_i values are encryptions of random values unrelated to the randomness used in encrypting a_i and b_i, and the adversary never requests extraction of that signature. If not, we can distinguish between the encryption of two random messages.

Given that the adversary cannot distinguish between these two scenarios, we can make a reduction directly. The reduction works by choosing one instance of Sign at random, giving encryptions of unrelated random values for all the c_i, and choosing one element of one pair at random and substituting an unknown challenge ciphertext there: either the proper encryption of α_i or β_i, or a distinct random value. If the unknown ciphertext encrypts the correct value, the adversary has a non-negligible chance of revealing it, in which case we discover the value. If the adversary does not reveal the ciphertext we hope for, we simply output a random guess. The adversary cannot reveal the ciphertext to be other than it is, so the non-negligible advantage we obtain is not offset at all.

Thus, if the adversary can break unforgeability, he can either break encryption under $PK^E_{C,S}$ or he can forge signatures under PK_S.

Online untransferability: We show how a simulator, with the secret information of the recipient (including the secret decryption key SK^E_R, which is part of the recipient's secret key), and working with the signer, can make a transcript computationally indistinguishable from one obtained in Sign on message m, but for which the signature is invalid. The simulator works as follows:

1. The simulator initiates FakeSign with the signer and obtains $PK_{C,S}$.
2. On input m and e, the simulator decrypts e to obtain CH. The simulator then generates $2k$ random strings $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k$ such that for

all i, $\alpha_i \oplus \beta_i \neq m$. The simulator then computes a_i and b_i as follows: If $CH_i = 0$ then $a_i = E_{PK^E_{C,S}}(\alpha_i; r_i^0)$ and $b_i = E_{PK^E_{C,S}}(\beta_i; r_i^1)$. If $CH_i = 1$ then $a_i = E_{PK^E_{C,S}}(\beta_i \oplus m; r_i^0)$ and $b_i = E_{PK^E_{C,S}}(\alpha_i \oplus m; r_i^1)$. The simulator then generates $c_i = E_{PK^E_{C,S}}(r_i^0 || r_i^1; r_i^2)$ and sends $PK^E_{C,S}$, and for each i, α_i, a_i, b_i, c_i to the dishonest recipient.

3. If the recipient responds with a decommitment of e to the string CH and a signature σ_R, the simulator checks the decommitment and σ_R, and if they are both valid, sends $m, CH, PK_{C,S}, PK_R, \sigma_R$, and for each i the values $\alpha_i', \beta_i', a_i, b_i, c_i, r_i^0, r_i^1, r_i^2$ to the signer in FakeSign, where if $CH_i = 0$, $\alpha_i' = \alpha_i$ and $\beta_i' = \beta_i$, and where $\alpha_i' = \beta_i \oplus m$ and $\beta_i' = \alpha_i \oplus m$ if $CH_i = 1$. When the signer responds with σ, the simulator sends the recipient with $r_i^{CH_i}$ for each i along with σ.

The only distinction between the messages generated by the simulator and the messages generated in the real signing protocol is that in the simulated messages, each pair (a_i, b_i) do not represent encryptions of two plaintexts that XOR to m, whereas in the real protocol, they do. A simple reduction proves that distinguishing these transcripts implies the ability to break encryption under $PK^E_{C,S}$. It should be clear this simulator is efficient.

Offline untransferability: We show how a simulator, with the ability to rewind the dishonest recipient, can produce a view indistinguishable from the one the dishonest recipient produces with the signer in Sign. The simulator works as follows:

1. Initiate FakeSign with the signer and obtain $PK_{C,S}$.
2. Run the Sign protocol honestly until the beginning of step 4. If the recipient sends an invalid decommitment CH, r, abort to the recipient, and output what it outputs.
3. Rewind to step 2 of Sign. Pick new random strings α_i for each i, and if $CH_i = 0$ we let $\beta_i \neq \alpha_i \oplus m$ be random and let $a_i = E_{PK^E_{C,S}}(\alpha_i; r_i^0)$ and $b_i = E_{PK^E_{C,S}}(\beta_i; r_i^1)$ and compute c_i normally. If $CH_i = 1$ we let $\alpha_i' \neq \alpha_i$ be random and let $a_i = E_{PK^E_{C,S}}(\alpha_i'; r_i^0)$ and $b_i = E_{PK^E_{C,S}}(\alpha_i \oplus m; r_i^1)$ and compute c_i normally. Send $PK_{C,S}$ and for each i, α_i, a_i, b_i, c_i to the recipient; remember r_i^0, r_i^1, r_i^2 for use later.
4. If the recipient sends back an invalid decommitment, go back to (simulator) step 3 and try again with new random values. Otherwise, if the recipient sends a valid decommitment but an invalid signature, abort to the recipient and output what it outputs. If the recipient sends a valid decommitment and a valid signature, send $m, CH, PK_{C,S}, PK_R, \sigma_R$, and for each i, send $\alpha_i, \beta_i, a_i, b_i, c_i, r_i^0, r_i^1$, and r_i^2 to the signer in FakeSign and obtain σ. Send σ to the recipient and output what it outputs.

The main point, again, is that the adversary should not be able to distinguish between being given pairs that decrypt to values that XOR to m from being

given pairs that do not, so long as all the decrypted responses are as expected. This is important for two reasons: first of all, it makes the views computationally indistinguishable, and second, it guarantees that the simulator runs in expected polynomial time.

Let p be the probability that the dishonest recipient reveals a proper decommitment in step 3 of Sign with the real signer. Since the Simulator does exactly as the signer does until that decommitment is given, the simulator has a probability p of producing a recipient output in which a proper decommitment is given. (Note that if one is given, the simulator will continue to try its steps 3 and 4 until the recipient decommits properly in one of them. Given this as a precondition, the outputs are computationally indistinguishable, since the only difference is whether the pairs decrypt properly or not.

With probability $1 - p$, the signer (or the simulator) encounters an invalid decommitment from the recipient. Given this as a precondition, the outputs are identical, since the simulator and signer act exactly the same.

It only remains to prove that the simulator runs in expected polynomial time. We can consider the probability p^0 that the recipient decommits properly when interacting with the signer, and the probability p^1 that the recipient decommits properly when interacting with the simulator's further attempts. If it is likely that an e is chosen such that p^0 is significantly larger than p^1, this leads directly to an attack on the encryption system. If not, then for all e without this property, the expected number of attempts taken by the simulator is $1 + \frac{p^0|_e}{p^1|_e} = 1 + \frac{p^0|_e}{p^0|_e + \nu} \leq 2$. The probability that an e is chosen without this property is negligible; therefore, with all but negligible probability, the simulator runs in expected polynomial time.

4 Analysis

4.1 Efficiency and Assumptions

In our Sign protocol, the signer must compute $3k$ encryptions and a signature, and must check a signature and an encryption. The recipient must check k encryptions and a signature, and produce one encryption and two signatures. Thus, each party computes $O(k)$ cryptographic operations. The signing protocol is four rounds. In our Disavow protocol, each party must compute a similar amount but the protocol is non-interactive, and we have no need for Verify protocols. The remaining protocols are similarly efficient but less important.

Only the schemes of Camenisch and Shoup [6] and Gentry, Molnar, and Ramzan [15] attain a confirmation protocol with $O(1)$ operations but both of those results were based on specific computational assumptions, and all prior schemes require at least $O(k)$ operations for disavowal.

The security of our scheme is based on the security of (1) the underlying signature scheme, and (2) the underlying encryption scheme. These assumptions are minimal, as such a scheme obviously implies signatures, and it is known the designated confirmer signatures imply public-key encryption [22].

4.2 Model and Variants

In addition, we make an assumption for the online security case that the recipient knows SK_R. The most natural way to ensure this assumption is to force the recipient to prove knowledge of SK_R when key registration takes place. This simplifies things significantly, because then the simulator for *offline* untransferability can extract the secret key, so online untransferability implies offline untransferability. However, requiring proofs of knowledge at key registration is burdensome.

The other way to deal with this assumption is to not require a proof of knowledge but simply to assume the recipient knows their own key. This is fairly reasonable, since we imagine "piggybacking" on an already existing PKI, in which the recipient probably already needs to know his or her key. In this scenario, we still guarantee online untransferability for any recipient that is honest during key registration (this reflects the likely case in the job offer scenario: Berkeley may believe that Alice was honest initially, although she might have become tempted later on.) Recipients that are dishonest during key registration may circumvent this, but as we show, this still does not allow them to perform offline transfer attacks.

The recipient's *signature* key does not serve the most important function. It is used to sign σ_R, which exists to satisfy the recipient should they ever want to reconfirm an online-untransferable signature without extracting it, a property that prior schemes have. If this requirement is unnecessary, the recipient's signature key can be dropped entirely.

For simplicity of presentation, we assume that the confirmer generates a separate encryption key for each signer. This may be objectionable, as it increases the interaction necessary for the scheme to proceed. However, if we are willing to assume the existence of identity-based encryption schemes, we can do away with the extra step. Instead, then, PK_C will be a master key for an IBE scheme, and $PK^E_{C,S}$ will be defined as the public key associated with identity S. By the security properties of IBE schemes, the encryption remains secure for other identities, even when the secret keys for certain identities are revealed (thus, one dishonest signer will not "ruin" the security for any honest signer).

Finally, it may seem to be a drawback of our scheme that the signer must be willing to engage in the FakeSign protocol on request. This does put a potential burden on the signer, but the ability of others to engage in the FakeSign protocol with the signer is only useful in the proofs of security: thus, offering this service will have little drawback, and will ensure untransferability.

5 Conclusion

Designated confirmer signatures were designed to be a solution that balances untransferability of signatures with accountability of the signer. Much of the work done on designated confirmer signatures was concerned with enhancing its efficiency and reducing the assumptions involved. However, even the original definitions were somewhat lacking in terms of online transferability. Fortunately,

the notion of public keys comes to our rescue: by assuming the existence of an established public key, we can push the window of opportunity for collusion between the recipient and the third party back in time: now, instead of colluding only during the actual signature protocol, they must have been colluding ever since the recipient's key was registered.

We have shown how to attain this level of online untransferability, while at the same time giving a protocol that is efficient, does not rely on the random oracle assumption, and uses general cryptographic assumptions.

We wish to thank Jens Groth, David Molnar, and the anonymous reviewers for their valuable input.

References

1. Baek, J., Safavi-Naini, R., Susilo, W.: Universal Designated Verifier Signature Proof. In: Roy [25], pp. 644–661
2. Barak, B., Canetti, R., Nielsen, J., Pass, R.: Universally Composable Protocols with Relaxed Set-up Assumptions. In: 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004), pp. 186–195 (2004)
3. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. SIAM Journal on Computing 32(3), 586–615 (2003)
4. Boneh, D. (ed.): CRYPTO 2003. LNCS, vol. 2729. Springer, Heidelberg (2003)
5. Camenisch, J., Michels, M.: Confirmer Signature Schemes Secure Against Adaptive Adversaries. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 14–18. Springer, Heidelberg (2000)
6. Camenisch, J., Shoup, V.: Practical Verifiable Encryption and Decryption of Discrete Logarithms. In: Boneh [4], pp. 126–144
7. Camenisch, J., Lysyanskaya, A.: An Identity Escrow Scheme with Appointed Verifiers. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 388–407. Springer, Heidelberg (2001)
8. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable Zero-Knowledge. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (May 21–23, 2000)
9. Chaum, D.: Designated Confirmer Signatures. In: De Santis [12], pp. 86–91
10. Chaum, D., Van Antwerpen, H.: Undeniable Signatures. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 20–24. Springer, Heidelberg (1990)
11. Davies, D.W. (ed.): EUROCRYPT 1991. LNCS, vol. 547, pp. 8–11. Springer, Heidelberg (1991)
12. De Santis, A. (ed.): EUROCRYPT 1994. LNCS, vol. 950, pp. 9–12. Springer, Heidelberg (1995)
13. Desmedt, Y., Yung, M.: Weaknesses of Undeniable Signature Schemes (Extended Abstract). In: Davies [11], pp. 205–220
14. Dolev, D., Dwork, C., Naor, M.: Nonmalleable Cryptography. SIAM Journal on Computing 30(2), 391–437 (2000)
15. Gentry, C., Molnar, D., Ramzan, Z.: Efficient Designated Confirmer Signatures Without Random Oracles or General Zero-Knowledge Proofs. In: Roy [25], pp. 662–681.
16. Goldwasser, S., Waisbard, E.: Transformation of Digital Signature Schemes into Designated Confirmer Signature Schemes. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 77–100. Springer, Heidelberg (2004)

17. Herzog, J., Liskov, M., Micali, S.: Plaintext Awareness via Key Registration. In: Boneh [4], pp. 548–564
18. Jakobsson, M.: Blackmailing Using Undeniable Signatures. In: De Santis [12], pp. 425–427
19. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated Verifier Proofs and their Applications. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 12–16. Springer, Heidelberg (1996)
20. 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)
21. Monnerat, J., Vaudenay, S.: Chaum's Designated Confirmer Signature Revisited. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 20–23. Springer, Heidelberg (2005)
22. Okamoto, T.: Designated Confirmer Signatures and Public-Key Encryption are Equivalent. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 21–25. Springer, Heidelberg (1994)
23. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residue Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 2–6. Springer, Heidelberg (1999)
24. Pedersen, T.P.: A Threshold Cryptosystem without a Trusted Party (Extended Abstract). In: Davies [11], pp. 522–526
25. Roy, B. (ed.): ASIACRYPT 2005. LNCS, vol. 3788, pp. 4–8. Springer, Heidelberg (2005)
26. 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)