Universal Witness Signatures

Chen Qian, Mehdi Tibouchi, Rémi Géraud

Universal Witness Signatures

Chen Qian[1], Mehdi Tibouchi[2], and Rémi Géraud[3]

[1] Univ Rennes
chen.qian@irisa.fr
[2] NTT Secure Platform Laboratories
tibouchi.mehdi@lab.ntt.co.jp
[3] École normale supérieure
remi.geraud@ens.fr

Abstract. A lot of research has been devoted to the problem of defining and constructing signature schemes with various delegation properties. They mostly fit into two families. On the one hand, there are signature schemes that allow the delegation of *signing rights*, such as (hierarchical) identity-based signatures, attribute-based signatures, functional signatures, etc. On the other hand, there are *malleable* signature schemes, which make it possible to derive, from a signature on some message, new signatures on related messages without owning the secret key. This includes redactable signatures, set-homomorphic signatures, signatures on formulas of propositional logic, etc.

In this paper, we set out to unify those various delegatable signatures in a new primitive called *universal witness signatures* (UWS), which subsumes previous schemes into a simple and easy to use definition, and captures in some sense the most general notion of (unary) delegation. We also give several constructions based on a range of cryptographic assumptions (from one-way functions alone to SNARKs and obfuscation) and achieving various levels of security, privacy and succinctness.

Keywords: Digital signatures, Delegation, Malleable signatures, Functional signatures, Identity-based cryptography, Obfuscation, SNARKs.

1 Introduction

Many signature schemes in the literature have delegatability properties either for keys or for messages.

The first family includes notions like (hierarchical) identity-based [Sha84, GS02, CHYC04, KN08] and attribute-based [LAS+10, MPR11, OT14] signatures, in which a master key authority grants signing rights to users, who may in turn be able to delegate those rights to lower-level signers. It also includes a slightly different class of schemes where the owner of the master secret key can sign all messages in the message space, and can delegate signing rights on restricted families of messages (again, possibly with recursive delegation): functional signatures [BGI14, BMS16] and policy-based signatures [BF14] are examples of such schemes.

The second family of schemes is that of malleable signature schemes, as defined e.g. by Ahn et al. [ABC+15], Attrapadung et al. [ALP12] and Chase et al. [CKLM14]: in those schemes, it is possible, given a signature on some message, to publicly derive signatures on certainly related messages. Specific examples include content extraction signatures [SBZ02], redactable and sanitizable signatures [JMSW02, ACdT05], some variants of set-homomorphic and network coding signatures [JMSW02, BFKW09] (where users sign sets, resp. vector spaces, and those signatures can be delegated to subsets or subspaces) and more. A different example and one of the original motivations of this work is Naccache's notion of signatures on formulas of propositional logic [Nac10], which makes it possible to derive a signature on a propositional formula Q from a signature on P whenever $P \Rightarrow Q$.

Main idea of this work. The goal of this paper is to unify all of the schemes above into a single very general and versatile primitive, which we call *universal witness signatures* (UWS), and to propose concrete instantiations of that primitive achieving good security and privacy properties.

Our first observation is that delegation of keys and delegation of messages are really two sides of the same coin, which can be unified by regarding a *signature* on a message m by an identity A as really the same object as a *key* associated with the sub-identity (A, m) of A. The operation of signing messages simply becomes a special case of key

delegation. Then, we can obtain in some sense the most general notion of signature scheme with (unary) delegation by saying that the set of identities is endowed with an essentially arbitrary pre-order relation $\leq$, and that given a key SK_A on an identity A, we are able to derive another key SK_B on any identity B such that $B \leq A$. Unforgeability is then defined in the obvious way: roughly speaking, after obtaining keys SK_{A_i} on various identities A_i (possibly derived from higher-level identities), an adversary is unable to construct a valid key SK_B for an identity B that doesn't satisfy $B \leq A_i$ for any i.

The type of delegation functionality achieved by such a scheme is simply determined by the pre-order relation $\leq$. For example, regular (non-delegatable) signatures are obtained by choosing a set of identities equal to the message space together with a special identity $*$, such that $m \leq *$ for any message m, and no other relationship exists. Then, SK_* is the secret key in the traditional sense, and it can be used to derive keys SK_m playing the role of signatures. If non-trivial relationships exist between the messages m themselves, we get a malleable signature scheme instead. For instance, we get redactable signatures if messages are ordered in such a way that $m' \leq m$ if and only if m' is obtained from m by replacing some of the text in m by blanks. And we obtain identity-based signatures with a larger set of identities, still containing a special identity $*$ associated with the master key authority, but also identities id_i associated with the various users of the system, and identities (id_i, m) for each pair of a user and a message. The order relation is then given by $\mathsf{id}_i \leq *$ for all i, and $(\mathsf{id}_i, m) \leq \mathsf{id}_i$ for all i and all messages. If additional nontrivial relationships exist between the id_i's, we essentially get hierarchical identity-based signatures.

We would like to support a really general class of pre-order relations $\leq$, to support for example the signatures on propositional formulas mentioned earlier, which are malleable signatures on messages (formulas) ordered by logical implication. However, the delegation algorithm that lets us publicly derive SK_B from SK_A when $B \leq A$ should certainly be able to efficiently test whether the relation $B \leq A$ actually holds. This is not possible directly for a relation like logical implication. For general NP relations, however, it does become possible if the delegation algorithm also receives as input a witness w of $B \leq A$ (for logical implication, for example, it would be a proof that $A \Rightarrow B$).

Our contributions. Along the lines sketched above, our first contribution is to define the notion of universal witness signature (UWS) scheme with respect to an arbitrary NP pre-order relation $\leq$ with a greatest element $*$. Such a scheme consists of only two algorithms:

$\mathsf{Setup}(1^\lambda)$: returns a key SK_* on the special identity $*$, as well as some public parameters PP;

$\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w)$: checks that SK_A is a valid key for the identity A and that w is a valid witness of $B \leq A$. If so, returns a fresh key SK_B for the identity B. Otherwise, returns $\perp$.

We do not actually need a separate algorithm for signature verification: to check whether SK_A is a valid key on A, we can simply try to delegate A to itself (since $\leq$ is a pre-order, there is a trivial witness $w_{A \leq A}$ for $A \leq A$), and test whether the $\mathsf{Delegate}$ algorithm returns something or just $\perp$.

Security for a UWS scheme is defined as the unforgeability notion described in the previous paragraph. As usual, we distinguish between selective security (in which the adversary has to choose in advance the identity on which it will try to forge) and adaptive security. We also identify two other desirable properties of a UWS scheme: the privacy notion of context-hiding UWS, which says that the delegation path used to obtain a given key is computationally hidden, and the notion of succinctness, which says that the size of a key SK_A is bounded only in terms of the size of A and the security parameter, independently of the delegation path.

We show that universal witness signatures are sufficient to obtain many earlier schemes appearing the literature, including HIBS, redactable signatures, functional signatures and Naccache's propositional signatures (for which our concrete constructions provide the first complete instantiations, to the best of our knowledge).

And finally, we give several constructions of UWS based on a range of assumptions, and achieving various subsets of our desirable properties. First, we show that one-way functions alone are enough to obtain adaptively secure UWS for arbitrary NP pre-order relations. The approach is similar to the one-way function-based construction of functional signatures [BGI14]. As in the work of Boyle et al., however, the resulting scheme is neither context-hiding nor succinct. Then, we prove that virtual black-box obfuscation [BGI+12] (for a well-defined program depending on the pre-order relation under consideration) provides a very simple construction of secure, succinct, context-hiding UWS. This construction ticks all of our boxes, but it is of course based on a very strong assumption: in fact, Barak et al.

showed that VBB is unachievable for general circuits. There *could* exist a virtual black-box obfuscator for our specific program of interest (and in fact, candidate constructions of indistinguishability obfuscation conjecturally satisfy that property), but this is rather speculative. We therefore try to achieve similarly strong properties based on somewhat more reasonable assumptions. We give two such constructions: one based on SNARKs [Kil92, BCI+13, GGPR13] (actually, proof-carrying data [CT10, BCCT13, BCTV14]), which is secure and succinct but achieves a somewhat weaker form of privacy than the context-hiding property; and another based on indistinguishability obfuscation [BGI+12, GGH+13, SW14, GMM+16], which is succinct and context-hiding but which we only prove selectively secure. Both of those constructions suffer from a limitation on delegation depth: it can be an arbitrary polynomial in the security parameter but fixed at Setup time. Moreover, our iO-based construction only applies to order relations rather than general pre-orders.

2 Universal Witness Signatures

Our notion of universal witness signature generalizes many existing signature schemes with delegation or malleability, such as homomorphic signatures [JMSW02], functional signatures [BGI14] and hierarchical identity-based signatures [CHYC04]. It supports delegation hierarchies defined by arbitrary NP pre-order relations, which, to the best of our knowledge, makes it more general than all previously proposed such schemes. And yet, its syntax is quite simple, consisting of only two algorithms.

Formal definition. Now we give a formal definition of our scheme. Note that the terms *identity* and *message* are synonymous in our setting, and can be used interchangeably. Similarly, the *signing key* for an identity can equivalently be seen as a *signature* on the corresponding message.

Definition 1 (Universal Witness Signatures). *A* universal witness signature scheme *(*UWS *for short) for an* NP *pre-order relation* $\leq$ *with a greatest element* $*$ *consists of two probabilistic polynomial-time algorithms:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{PP}, \mathsf{SK}_*)$: *This algorithm takes as input a security parameter* λ*, produces a master signing key* SK_**, keeps it secret and outputs public parameters* PP.
- $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w) \to \mathsf{SK}_B$: *This function takes as input the public parameters, a signing key for identity* A*, an identity* B*, and a witness* w*. If* w *is a valid witness of the* NP *statement* $B \leq A$ *and* SK_A *is a valid signing key for identity* A*, then the algorithm outputs a signing key for identity* B*, otherwise it outputs* $\perp$.

We note that there exists a trivial witness for $A \leq A$*. For simplicity's sake, we also define a third algorithm* Verify*, which is merely a specialization of* Delegate*:*

- $\mathsf{Verify}(\mathsf{PP}, \mathsf{SK}_A, A) \to \{\mathsf{True}, \mathsf{False}\}$: *This algorithm takes a public parameter, an identity* A *and a signing key* SK_A*. We have* $\mathsf{Verify}(\mathsf{PP}, \mathsf{SK}_A, A) = \mathsf{False}$ *if and and only if* $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, A, w_{A \leq A})$ *returns* $\perp$*, where* $w_{A \leq A}$ *is the trivial witness for the reflexive property* $A \leq A$.

Additional properties of UWS**.** We now propose three desirable properties for a UWS scheme. Intuitively, we will say that a UWS scheme is *secure* (i.e. unforgeable) if a polynomially-bounded adversary allowed to make arbitrary delegation queries cannot come up with a valid signing key on an identity A^* that isn't reachable by delegation from any of the signing keys she obtained.

We will say that the scheme is *context-hiding* if a signing key does not reveal the delegation path used to derive it. For example, a signing key SK_A on A obtained from a long delegation path is indistinguishable from a signing key for the same identity delegated directly from the master key SK_*.

Finally, we will say that the scheme is *succinct* when the size of a signing key is bounded depending only on the size of the associated identity (and not on the length of the delegation path used to derive it).

These properties can be captured formally as follows.

Definition 2 (Correctness). Correctness *of an* UWS *scheme states that for all* SK_A*, if*

$$\mathsf{UWS.Verify}(\mathsf{PP}, \mathsf{SK}_A, A) \text{ outputs } \mathsf{True} \text{ and } w_{B \leq A} \text{ is a witness of } B \leq A,$$

then the signing key $\mathsf{SK}_B \leftarrow \mathsf{UWS.Delegate}(\mathsf{PP}, A, B, w_{B \leq A})$ *is not equal to* $\perp$*, and* $\mathsf{UWS.Verify}(\mathsf{PP}, \mathsf{SK}_B, B)$ *outputs* True.

Definition 3 (Selective security of UWS schemes). *The* selective security *of an* UWS *scheme is captured by the advantage of an adversary $\mathcal{A}$ in the following security game against a challenger $\mathcal{C}$:*

1. *$\mathcal{A}$ chooses a target identity A^*.*
2. *$\mathcal{C}$ runs the algorithm $\mathsf{Setup}(1^\lambda)$ to get SK_* and PP. Then it keeps the master signing key SK_* secret and sends the public parameter PP to $\mathcal{A}$, together with a tag t_{SK_*} referring to SK_*.*
3. *$\mathcal{C}$ initializes an associative list $\mathcal{H}$ in which the challenger maintains a set of tuples (t, A, SK) where t is a tag for the signing key generated by the challenger, A is an identity, and SK is a signing key for the identity A. The associative initially consists of just the tuple $(t_{\mathsf{SK}_*}, *, \mathsf{SK}_*)$.*
4. *$\mathcal{A}$ can submit two types of queries:*
   Delegate queries: *Assume that $\mathcal{C}$ generated the signing key SK_A on an identity A, and that $\mathcal{A}$ received the corresponding tag t_{SK_A}. For any identity B for which $\mathcal{A}$ knows a witness $w_{B \leq A}$ of $B \leq A$, $\mathcal{A}$ can instruct $\mathcal{C}$ to execute $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B \leq A})$. The challenger will then generate a new unique tag t_{SK_B} on the resulting signing key SK_B, and add the tuple $(t_{\mathsf{SK}_B}, B, \mathsf{SK}_B)$ to its associative list $\mathcal{H}$, sending back the tag t_{SK_B} to $\mathcal{A}$.*
   Reveal queries: *Using the corresponding tag, $\mathcal{A}$ can ask $\mathcal{C}$ to reveal any previously generated signing key SK_A provided that the associated identity A does not satisfy $A^* \leq A$.*
5. *After polynomially many queries of the type above, $\mathcal{A}$ outputs a candidate forgery SK_{A^*}, and wins if and only if*

$$\mathsf{Verify}(\mathsf{PP}, \mathsf{SK}_{A^*}, A^*) = \mathsf{True}.$$

A UWS *scheme is* selectively secure *if and only if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the previous game is negligible.*

We also consider the *adaptive security* of UWS schemes, which is similarly defined, with the notable exception that the adversary does not announce in advance the identity A^* on which she will forge, but can choose it adaptively instead (with the condition that it does not satisfy $A^* \leq A$ for any of the identities A associated with revealed signatures).

Definition 4 (Context-hiding). *A* UWS *scheme is* context-hiding *if, for every tuple $(\mathsf{SK}_1, \mathsf{SK}_2, A_1, A_2, w_1, w_2)$ and every identity B such that*

$$\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_1, A_1, B, w_1) \to \mathsf{SK}'_1 \neq \bot$$
$$\textit{and}$$
$$\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_2, A_2, B, w_2) \to \mathsf{SK}'_2 \neq \bot,$$

the distributions of $(\mathsf{PP}, B, \mathsf{SK}'_1)$ and $(\mathsf{PP}, B, \mathsf{SK}'_2)$ are statistically close.

Definition 5 (Succinctness). *An* UWS *scheme is* succinct *if there exists a polynomial p such that the size of any signature SK_A is bounded by $p(\lambda, |A|)$.*

3 Applications: From UWS to Other Primitives

Our universal witness signature scheme can be considered as a generalization of many existing malleable signature schemes. To showcase this, we use our UWS scheme to instanciate several well-known signature schemes, and some more original ones, namely: functional signatures and propositional signatures. A construction of hierarchical identity-based signatures and redactable signatures is also given in Appendix B. To the best of our knowledge, this is the first time that a construction for propositional signatures appears in the literature.

Functional signatures. Functional signatures, introduced by Boyle et al. [BGI14], are a particularly wide-ranging generalization of identity-based signatures in which the key authority can generate signing keys sk_f associated to functions f, such that the owner of sk_f can sign exactly those messages that are in the image of f. Moreover, to sign m in the image of f, the owner of sk_f needs a witness to this fact, namely a preimage of m under f. In this section, we show how we can easily obtain functional signatures based on universal witness signatures.

Definition 6 (Functional Signature). *The* functional signature *for a message space $\mathcal{M}$ and a function family $\mathcal{F} = \{f : \mathcal{D}_f \to \mathcal{M}\}$ is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *which is specified as follows:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{msk}, \mathsf{mvk})$*: the setup algorithm takes a security parameter λ and it returns a master signing key* msk *and a master verification key* mvk*. Then it keeps the master signing key secret* msk *and publishes the master verification key* mvk*.*
- $\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$ *: the key generation algorithm takes a master signing key* msk *and a function f to specify which one will be allowed to sign the messages, then it outputs a corresponding signing key.*
- $\mathsf{Sign}(f, \mathsf{sk}_f, m) \to (f(m), \sigma_{f(m)})$*: the signing algorithm takes a function f and the corresponding signing key sk_f as input and produces $f(m)$ and the signature $\sigma_{f(m)}$of $f(m)$*
- $\mathsf{Verify}(\mathsf{mvk}, \sigma_m, m) \to \{\mathsf{True}, \mathsf{False}\}$*: the verification algorithm takes a signature-message pair (m, σ_m) and the master verification key* mvk*. The algorithm outputs* True *if σ_m is a valid signature of m, otherwise outputs* False*.*

Functional signatures from UWS*.* Consider the order $\mathcal{O}_\mathcal{F}$ corresponding to the function family $\mathcal{F}$:

- Let $\mathcal{M}$ be the message space. The order $\mathcal{O}_\mathcal{F}$ is an order on the set $\{*\} \cup \mathcal{F} \cup \mathcal{M}$
- $*$ is the greatest identity, bigger than all other messages.
- $m \leq f$ when $\exists m' \in \mathcal{M} \wedge m = f(m')$
- There does not exist any other non-trivial order

We note $\leq$ be the previously defined order. Then consider the UWS scheme corresponding to this order. The construction of the functional signature is described in figure Figure 1.

Security of the functional signature scheme. A functional signature scheme is typically expected to verify the following properties:

- *Correctness:* This property expresses the fact that a properly generated signature is verified to be correct. Formally: $\forall f \in \mathcal{F}, \forall m \in \mathcal{D}_f$, $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$, $(f(m), \sigma_{f(m)}) \leftarrow \mathsf{Sign}(f, \mathsf{sk}_f, m)$,
$$\mathsf{Verify}(\mathsf{mvk}, \sigma_{f(m)}, f(m)) = \mathsf{True}.$$
- *Unforgeability:* The unforgeability of the functional signature scheme is defined by the the following security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:
  - $\mathcal{C}$ generates a pair of keys $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Setup}(1^\lambda)$, then publishes the master verification key mvk byt keeps the master signing key msk secret.
  - $\mathcal{C}$ constructs an initially empty associative list $\mathcal{H}$ indexed by $f \in \mathcal{F}$, a key generation oracle $\mathcal{O}_{\mathsf{KeyGen}}$ and a signing oracle $\mathcal{O}_{\mathsf{Sign}}$ as follows:
    - $\mathcal{O}_{\mathsf{KeyGen}}(f)$: If there exists already a value associated to f in the associative list $\mathcal{H}$, then outputs $\mathcal{H}(f)$ directly. Otherwise the oracle uses the KeyGen algorithm to generate the signing key $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ associated to f, then adds the function-signing key pair (f, sk_f) to the associative list.
    - $\mathcal{O}_{\mathsf{Sign}}(f, m)$: If there exists a value associated to f in the associative list $\mathcal{H}$, then uses the signing algorithm to generate a signature $\sigma_{f(m)}$of $f(m)$

$\underline{\mathsf{Verify}(\mathsf{mvk}, \sigma_m, m)}$:
return $\mathsf{UWS.Verify}(\mathsf{PP}, \mathsf{SK}_{f(m)}, f(m))$

$\underline{\mathsf{Setup}(1^\lambda)}$:
$(\mathsf{PP}, \mathsf{SK}_*) \leftarrow \mathsf{UWS.Setup}(1^\lambda)$
return $(\mathsf{SK}_*, \mathsf{PP})$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, f)}$:
return $\mathsf{UWS.Delegate}(\mathsf{PP} = \mathsf{msk}, \mathsf{SK}_*, *, f, \text{“}f \leq *\text{''})$

$\underline{\mathsf{Sign}(f, \mathsf{sk}_f, m)}$:
$\sigma_{f(m)} = \mathsf{UWS.Delegate}(\mathsf{PP}, \mathsf{sk}_f, f, f(m), m, \text{“}f(m) \leq f\text{''})$
return $\big(f(m), \sigma_{f(m)}\big)$

Fig. 1. Functional signatures from UWS.

- $\mathcal{A}$ can query the two oracles $\mathcal{O}_{\mathsf{Sign}}$ and $\mathcal{O}_{\mathsf{KeyGen}}$. $\mathcal{A}$ can also make requests of corresponding value in the associative list $\mathcal{H}$. $\mathcal{A}$ win against the security game if it can produce a message-signature pair (m, σ) such that :
  - $\mathsf{Verify}(\mathsf{mvk}, m, \sigma) = 1$
  - There does not exist m' and f such that $m = f(m')$ and f was sent as a query to the key generation oracle $\mathcal{O}_{\mathsf{KeyGen}}$.
  - There does not exist a function-message pair (f, m') was a query to the signing oracle $\mathcal{O}_{\mathsf{Sign}}$ and $m = f(m')$

Let $\mathcal{A}$ be an adversary against the functional signature constructed using UWS scheme with non-negligible advantage. Then by the definition it can produce a message-signature pair (m, σ) such that:

1. $\mathsf{Verify}(\mathsf{mvk}, m, \sigma) = \mathsf{True}$
2. There does not exist m' and f such that $m = f(m')$ and f was sent as a query to the key generation oracle $\mathcal{O}_{\mathsf{KeyGen}}$.
3. The message m was not sent as a query to the signing oracle $\mathcal{O}_{\mathsf{Sign}}$.

Condition 1 implies that $\mathsf{Verify}(\mathsf{PP}, \sigma, m) = \mathsf{True}$. Condition 2 implies that for all (f, m') which verifies that $f(m') = m$, sk_f has never been revealed. Then the third condition implies that σ_m has not been revealed. As in the specific order corresponding to the functional signature, the only elements bigger than m are m itself, $*$, and $\{f \mid \exists m' \in \mathcal{M}.f(m') = m\}$. With the conditions 2 and 3, the signatures of these identities have never been revealed, but $\mathcal{A}$ can produce a valid signature for m which break the existential unforgeability of the underlying UWS scheme.

Propositional signatures. In an invited talk at CRYPTO and CHES 2010, Naccache [Nac10] introduced the new notion of propositional signatures, for which he suggested a number of real-world applications such as contract-signing. Propositional signatures are signatures on formulas of propositional calculus, which are homomorphic with respect to logical implication. In other words, given a signature on a propositional formula P, one should be able to publicly derive a signature on any Q such that $P \Rightarrow Q$. Since the satisfiablity of propositional formulas cannot be decided efficiently without auxiliary information, the derivation algorithm should also take as input a witness of $P \Rightarrow Q$, i.e. a proof of Q assuming P.

To the best of our knowledge, no construction of propositional signatures has been proposed so far. However, it is easy to see that they are, again, easily obtained from UWS.

Security of propositional signatures. Formally, a propositional signature scheme is a triple $(\mathcal{G}, \mathcal{D}, \mathcal{V})$ of efficient algorithms for key generation: $\mathcal{G}(1^\lambda) \to (\mathsf{mvk}, \sigma_{\mathsf{False}})$, signature derivation: $\mathcal{D}(\mathsf{mvk}, \sigma_P, P, Q, \pi) \to \sigma_Q$, and verification: $\mathcal{V}(\mathsf{mvk}, \sigma_P, P) \to \mathsf{True}/\mathsf{False}$. The signature σ_{False} on the false proposition plays the role of master secret key, due to *ex falso quodlibet*. Correctness states that if σ_P is a valid signature on proposition P (in the sense that $\mathcal{V}(\mathsf{mvk}, \sigma_P, P)$ evaluates to True) and π is a valid proof of $P \Rightarrow Q$, then $\mathcal{D}(\mathsf{mvk}, \sigma_P, P, Q, \pi)$ a valid signature σ_Q on Q. Unforgeability says that after obtaining signatures on propositions P_i of his choice, an efficient adversary cannot produce a valid signature on a prosition Q such that none of the P_i's implies Q.

Propositional signatures from UWS. Clearly, the UWS scheme associated with the corresponding set of propositional formulas endowed with the NP preorder relation given by logical implication (where witnesses are proofs) exactly gives a propositional signature scheme.

In fact, our definition of security captures for UWS captures a slightly stronger security model, where unforgeability still holds when the adversary can make unrestricted delegation queries on messages he cannot see, so as to control delegation paths.

4 Construction of UWS

We now explain how to realise UWS. The first construction requires only the existence of one-way function, but is neither context-hiding nor succinct. These properties can be obtained at the cost of introducing new assumptions such as the existence of proof-carrying data (for succinctness) or obfuscators (for context-hiding).

4.1 Construction from One-Way Functions

Firstly, we propose a construction of the UWS scheme based only on the existence of one-way functions. The existence of one-way function is a very weak and basic assumption of cryptography. But this very basic construction does not verify many other properties than adaptive security like succinctness or context-hiding.

In this construction, we use a "certificate of computation" approach: Every signature is provided with a certificate of the delegation path. And an identity A can provide a certificate of the identity B if and only if $B \leq A$. For example if for the generation of the signature of the identity A, we have passed the identities $*, id_1, id_2, \ldots, id_n, A$. Each identity produced a signing-verification key pair $(\mathsf{sk}_{id_i}, \mathsf{vk}_{id_i})$ using the key generation of a signature scheme, and $\sigma_{\mathsf{vk}_{id_i}}$ is a signature of $(id_i, w_{id_i \leq id_{i-1}}, \mathsf{vk}_{id_i})$[4] produced using the signing key sk_{i-1} The signature of the identity A is represented by

$$\mathsf{SK}_A = \big(\mathsf{sk}_A, \big[\,(\mathsf{vk}_A, A, w_{A\leq n}, \sigma_{\mathsf{vk}_A}), \big(\mathsf{vk}_{id_n}, id_n, w_{id_n \leq id_{n-1}}, \sigma_{\mathsf{vk}_{id_n}}\big), \ldots, \ldots, (\mathsf{vk}_*, *, w_{*\leq *}, \sigma_{\mathsf{vk}_*})\,\big]\big)$$

For simplicity we will note "w is a valid witness of $x \leq y$" by "$x \leq_w y$" in the following sections of this paper and we propose the following construction of our UWS scheme using a classical existential unforgeable signature scheme $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$.

$\mathsf{Setup}(1^\lambda)$*:*

- The setup algorithm first takes two pairs of keys $(\mathsf{msk}, \mathsf{mvk}), (\mathsf{vk}_*, \mathsf{sk}_*)$ from the signature's parameters generation algorithm.
- Then it generates a signature σ_{vk_*} of $(*, w_{*\leq *}, \mathsf{vk}_*)$ using the signing key msk, this can be considered as a certificate of the verification key vk_* delivered by the master authority.
- The certificate c_* of the identity $*$ is $[(\mathsf{vk}_*, *, w_{*\leq *}, \sigma_{\mathsf{vk}_*})]$. The signing key (signature) SK_* of the identity $*$ in our UWS scheme is (sk_*, c_*) and the public parameter PP will be mvk.

$\mathsf{Delegate}(\mathsf{mvk}, \mathsf{SK}_A, A, B, w)$*:*

- The signing key SK_A has the form (sk_A, c_A), where the certificate c_A is a list

$$[(\mathsf{vk}_A, A, w_A, \sigma_A), \ldots, (\mathsf{vk}_*, *, w_{*\leq *}, \sigma_{\mathsf{vk}_*})].$$

- The delegation algorithm first verifies that the certificate c_A is valid, by checking that each σ_j is a valid signature on $(id_j, w_j, \mathsf{vk}_j)$ with respect to the verification key vk_{j-1}. It also checks that the witnesses are valid: $B \leq_w A \leq_{w_A} id_n \ldots$ and that vk_A is a correct public key for sk_A by signing a random message and verifying it.
- If all these verification steps succeed, a fresh key pair $(\mathsf{sk}_B, \mathsf{vk}_B)$ for Sig is generated, together with a signature σ_B on $(B, w_{B\leq A}, \mathsf{vk}_B)$ using the siging key sk_A The algorithm then computes an extended certificate c_B by prepending $(\mathsf{vk}_B, B, w_B, \sigma_B)$ to c_A, and returns the signature SK_B as (sk_B, c_B).

This construction is summarized in Figure 2.

Theorem 1. *The construction of* UWS *based on the one-way function is correct.*

Proof. If we have $\mathsf{Verify}(\mathsf{mvk}, \mathsf{SK}_A, A) = \mathsf{True}$ and $B \leq_w A$, then $\mathsf{SK}_B = (\mathsf{sk}_B, c_B)$ with $c_B = [(\mathsf{vk}_B, B, w, \sigma_B), c_A]$, and by construction, we have that $(\mathsf{vk}_B, \mathsf{sk}_B)$ is a valid signature key pair, and σ_B is a valid signature of $(B, w_{B\leq A}, \mathsf{vk}_B)$. By the verification above and hypothesis of SK_A is valid, c_B is also a valid certificat and $\mathsf{Verify}(\mathsf{mvk}, \mathsf{SK}_B, B)$ outputs True. □

The proof that this construction is adaptively secure is given in Appendix C.

[4] This is represented as a message $id_i || w_{id_i \leq id_{i-1}} || \mathsf{vk}_{id_i}$.

Fig. 2. Construction of UWS from one way functions.

$\underline{\mathsf{Setup}(1^\lambda)}$:
$(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$
$(\mathsf{sk}_*, \mathsf{vk}_*) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$
$\sigma_{\mathsf{vk}_*} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, (*, w_{*\leq *}, \mathsf{vk}_*))$
$c_* \leftarrow [(vk_*, *, w_{*\leq *}, \sigma_{vk_*})]$
return $(\mathsf{mvk}, (\mathsf{sk}_*, c_*))$

$\underline{\mathsf{Delegate}(\mathsf{mvk}, \mathsf{SK}_A, A, B, w)}$:
$sk_A, c_A \leftarrow \mathsf{SK}_A$
$\left\{(\mathsf{vk}_{id_i}, i, w_i, \sigma_{\mathsf{vk}_{id_i}})\right\} \leftarrow c_A$
Check certificate and witnesses:
  Assert $\mathsf{Sig.Verify}(\mathsf{vk}_{j-1}, (id_j, w_j, \mathsf{vk}_j), \sigma_j)$ for all j
  Assert $B \leq_w A \leq_{w_{A\leq id_n}} id_n, \ldots, * \leq_{w_{*\leq *}} *$
Check public key:
  $m \xleftarrow{\$} random(1^\lambda)$
  $\sigma_m \leftarrow \mathsf{Sig.Sign}(sk_A, m)$
  Assert $\mathsf{Sig.Verify}(vk_A, m, \sigma_m)$
Generate new key pair and certificate:
  $(\mathsf{sk}_B, \mathsf{vk}_B) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$
  $\sigma_{\mathsf{vk}_B} \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_A, (B, w_{B\leq A}, \mathsf{vk}_B))$
  Concatenate $(\mathsf{vk}_B, B, w_{B\leq A}, \sigma_{\mathsf{vk}_B})$ with c_A:
  $c_B \leftarrow (\mathsf{vk}_B, B, w_{B\leq A}, \sigma_{\mathsf{vk}_B})||c_A$
return $\mathsf{SK}_B = (\mathsf{sk}_B, c_B))$

4.2 Succinct Construction from Proof-Carrying Data

The previous construction based on one-way functions is clearly neither context-hiding nor succinct. In this section, we give a construction based on SNARKs, which is adaptively secure and succinct; we also provide arguments suggesting that it should satisfy some form of context-hiding property at least when the SNARK is zero-knowledge. As a downside, we are limited to a polynomial number of delegation steps: the maximum delegation depth must be fixed at setup time, as a polynomial in the security parameter. Our construction use the following theorems proposed by Bitansky et al. [BCCT13].

Theorem 2 (SNARK Recursive Composition Theorem). *There exists an efficient transformation* RecComp *algorithm such that, for every publicly-verifiable SNARK* $(\mathcal{G}_{SNARK}, \mathcal{P}_{SNARK}, \mathcal{V}_{SNARK})$*, the 3-tuple algorithms*

$$(\mathcal{G}, \mathcal{P}, \mathcal{V}) = \mathsf{RecComp}(\mathcal{G}_{SNARK}, \mathcal{P}_{SNARK}, \mathcal{V}_{SNARK})$$

is a publicly-verifiable PCD system for every constant-depth compliance predicate.

Theorem 3 (PCD Depth-Reduction Theorem). *Let $\mathcal{H} = \{\mathcal{H}_K\}_{k\in\mathcal{N}}$ be a collision-resistant hash-function family. There exists an efficient transformation $\mathrm{DepthRed}_{\mathcal{H}}$ with the following properties:*

- *Correctness: If $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a PCD system for constant-depth compliance predicates, then $(\mathcal{G}', \mathcal{P}', \mathcal{V}') = \mathrm{DepthRed}_{\mathcal{H}}(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a path PCD for polynomial-depth compliance predicates.*
- *Verifiability Properties: If $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is publicly verifiable then so is $(\mathcal{G}', \mathcal{P}', \mathcal{V}')$*
- *Efficiency: There exists a polynomial p such that the (time and space) efficiency of $(\mathcal{G}', \mathcal{P}', \mathcal{V}')$ is the same as that of $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ up to the multiplicative factor $p(k)$*

For the construction, we follow the same line of thinking as in Section 4.1. We consider an existential unforgeable signature scheme $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, with master signing key msk and master verification key mvk. Let us first define the distributed computation transcript $T = (G, \mathsf{linp}, data)$ and the corresponding $\mathcal{C}$-compliance $\mathcal{C}(z_{\text{out}}, \mathsf{linp}, z_{\text{in}})$ as follows:

- $G = (V, E)$: The graph of the distributed computation transcript, with V labeled by the identity and $(A, B) \in E$ labeled by the tuple $(\mathsf{VK}_B, \mathsf{SK}_B, w_{B\leq A}, \sigma_{\mathsf{VK}_B})$.
- linp: The local input of the vertices will be the identity corresponding to this vertex.
- z_{out}: The data of the edges are the labels of the edges, specifically $z_{\text{out}} = (\mathsf{VK}_B, \mathsf{SK}_B, w_{B\leq A}, \sigma_{\mathsf{VK}_B})$.
- $\mathcal{C}(z_{\text{out}}, \mathsf{linp}, z_{\text{in}})$: parse z_{out} as $(\mathsf{VK}_B, \mathsf{SK}_B, w_{B\leq A}, \sigma_{\mathsf{VK}_B})$, and suppose $z_{\text{in}} = (\mathsf{VK}_A, \mathsf{SK}_A, w_{A\leq C}, \sigma_{\mathsf{VK}_A})$ with C the predecessor of A. The algorithm $\mathcal{C}(z_{\text{out}}, \mathsf{linp}, z_{\text{in}})$ outputs True if

- $\mathsf{Sig.Verify}(\mathsf{VK}_A, (\mathsf{linp}(B), w_{B\leq A}, \mathsf{VK}_B), \sigma_{\mathsf{VK}_A}) = \mathsf{True}$
- $B \leq_{w_{B\leq A}} A$
- For a random message m, we have $\mathsf{Sig.Verify}(\mathsf{VK}_B, m, \mathsf{Sig.Sign}(\mathsf{SK}_B, m)) = \mathsf{True}$.

Let us consider the PCD scheme $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ corresponding to the distributed computation transcript described as above. Then we have all the building blocks for our universal witness signature.

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{PP}, \mathsf{SK}_*)$: Let $(\mathsf{mvk}, \mathsf{msk}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$, then use the generation algorithm of the proof-carrying data to get $\mathsf{PP} = \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda)$. Let $z_* = (\mathsf{mvk}, \mathsf{msk}, \text{“}* \leq *\text{”}, \sigma_{\mathsf{VK}_*})$, we compute $\pi_* \leftarrow \mathcal{P}(\mathsf{crs}, \bot, \bot, z_*, *)$ and $\sigma_{\mathsf{VK}_*} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, (*, \text{“}* \leq *\text{”}, \mathsf{mvk}))$. Then output $\mathsf{SK}_* = (z_*, \pi_*)$ and $\mathsf{VK}_* = \mathsf{mvk}$.
- $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B\leq A}) \to \mathsf{SK}_B$:
  - Parse SK_A as (z_A, π_A) with $z_A = (\mathsf{VK}_A, \mathsf{SK}_A, w_{A\leq C}, \sigma_{\mathsf{VK}_A})$.
  - We first check whether π_A is valid proof of the fact that z_A is consistent with the $\mathcal{C}$-compliance transcript. If the test fails then the algorithm ouputs $\bot$
  - If the check suceeds, use the underlying signature scheme's generation algorithm to get $(\mathsf{SK}_B, \mathsf{VK}_B) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$, and use the sign algorithm to get $\sigma_{\mathsf{VK}_B} = \mathsf{Sig.Sign}(\mathsf{SK}_A, (B, w_{A\leq B}, \mathsf{VK}_B))$. Let $z_B = (\mathsf{VK}_B, \mathsf{SK}_B, w_{B\leq A}, \sigma_{\mathsf{VK}_B})$.
  - Finally use the proof algorithm $\mathcal{P}$ to generate a proof of z_B: $\pi_B \leftarrow \mathcal{V}(\mathsf{crs}, z_A, \pi_A, z_B, B)$ which is a proof of the fact that z_B is consistent with the $\mathcal{C}$-compliance transcript. The algorithm outputs $\mathsf{SK}_B = (z_B, \pi_B)$.

This construction is summarized in Figure 3.

Theorem 4. *The construction of* UWS *scheme based on the Proof-Carrying Data is correct.*

Proof. If SK_B is produced by the Delegate algorithm on SK_A, and SK_A is a valid signature of A, by the correctness of the PCD scheme, we have π_B is a valid proof of the fact that z_B is consistent with the $\mathcal{C}$-compliance transcript. This assures that the new signature SK_B will pass the verification algorithm which means $\mathsf{Verify}(\mathsf{PP}, \mathsf{SK}_B, B)$ will outputs True. □

We give also proof of the selective security and succinctness of this construction in Appendix D.

4.3 Construction from Indistinguishability Obfuscation

The SNARK based construction from the previous section has some limitations: a polynomial bound on delegation depth, and no rigorously proved context-hiding property. A very simple construction without either of these shortcomings (thus achieving all the desired properties of universal witness signatures) can be obtained from virtual black-box obfuscation: we describe that construction in Appendix E. Admittedly, however, virtual black-box obfuscation is an onerous assumption: it is known to be unachievable for general circuits in the standard model [BGI+12]. In this section, we provide a satisfactory construction using a somewhat more realistic flavor of obfuscation (indistinguishability obfuscation) together with puncturable pseudorandom functions (as is usual for iO-based constructions). Both notions are formally recalled in Appendix A.

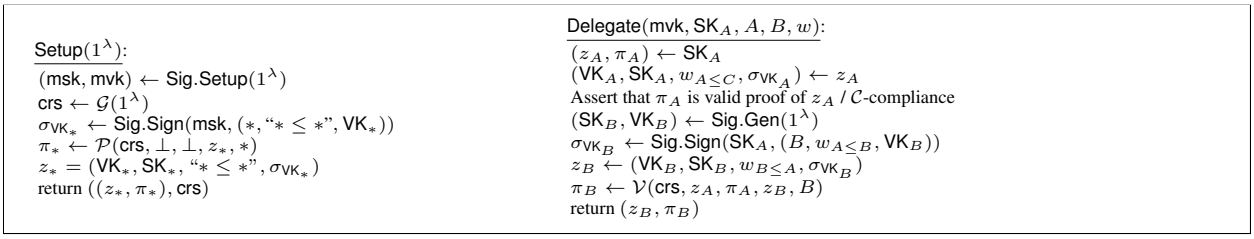

Fig. 3. Construction of UWS from proof-carrying data.

Fig. 4. The CheckSign algorithm.

$\mathsf{CheckSign}[K](\mathsf{SK}_A, A, B, w)$:
if $f(\mathcal{F}(K, A)) == f(\mathsf{SK}_A) \wedge A >_w B$ then
  return $\mathsf{SK}_B = \mathcal{F}(K, B)$
else if $f(\mathcal{F}(K, A)) == f(\mathsf{SK}_A) \wedge A =_w B$ then
  return $\mathsf{SK}_B := \mathsf{SK}_A$
else return $\perp$

Our construction of UWS from punctured PRFs and iO achieves correctness, context-hiding, and succinctness. However, besides the reliance on iO, which is a strong assumption, the maximum delegation level must again be chosen at setup time, and there is a limitation of the type of preorder relations we support. More precisely, our construction applies to *order* relations ($\leq$ is anti-symmetric) and moreover, any given element has at most polynomially many elements greater than itself. We will use the following three primitives in our construction:

1. Let $C^\flat = i\mathcal{O}(C)$ be the indistinguishability obfuscation of the circuit C;
2. Let $(\mathcal{G}, \mathcal{F})$ be a punctured PRF scheme in which $\mathcal{G}$ generates the system parameters and $\mathcal{F}$ is the keyed PRF;
3. Let f be an injective length-doubling PRG.

Here is our construction:

- $\mathsf{Setup}(1^\lambda)$:
  - We generate the PRF key K from $\mathcal{G}(1^\lambda)$ and compute the indistinguishability obfuscation $\mathsf{CheckSign}^\flat$ of the algorithm CheckSign.
  - The master signing key (signature of $*$) SK_* is the PRF value $\mathcal{F}(K, *)$ of $*$ and the public parameter PP is the obfuscated circuit $\mathsf{CheckSign}^\flat$.
- $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B \leq A})$:
  - $\mathsf{CheckSign}^\flat(\mathsf{SK}_A, A, B, w_{B \leq A})$ which is an indistinguishability obfuscation of the program CheckSign described Figure 4.

Theorem 5. *The $i\mathcal{O}$-based construction is correct, succinct, and context-hiding.*

Proof. All three properties rely on the underlying indistinguishability obfuscation's properties; we denote this obfuscator $i\mathcal{O}$. By construction, a valid signature SK_A of identity A is equivalent to the fact that $\mathsf{SK}_A = \mathcal{F}(K, A)$. Very roughly, it follows that:

- **Correctness** If SK_B is produced by the algorithm Delegate, then $\mathsf{SK}_B = \mathcal{F}(K, B)$. We have $\mathsf{Verify}(\mathsf{PP}, \mathsf{SK}_B, B) = \mathsf{True}$.
- **Succinctness and Context-hiding** $\mathsf{SK}_A = \mathcal{F}(K, A)$ is independent of the delegation path, and its output is that of a pseudorandom function. Hence the UWS scheme is succinct and context-hiding.

A security proof and extended proofs of sunccinctness and the context-hiding property are provided in Appendix F. □

5 Conclusion

In this paper, we have introduced a very general notion of delegatable signature scheme: universal witness signatures (UWS). We have formally defined the security properties that UWS should ideally satisfy, and provided four different constructions based on a range of assumptions from the existence of one-way functions to virtual black-box obfuscation, and achieving some or all of these security properties. Those constructions can be used to instantiate a number of other primitives, and provide, in particular, the first instantiations of propositional signatures, a notion with numerous interesting applications.

Each of our constructions has some limitations, however. Constructing secure, succinct and context-hiding UWS with unbouded delegation depth based on relatively weak assumptions is left as a challenging open problem. In addition, our work did not tackle the problem of anonymity for UWS-like schemes, and we only considered *unary* delegation. Those problems are also worth investigating in future work.

References

ABC+15. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. *Journal of Cryptology*, 28(2):351–395, April 2015.

ACdT05. Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS 2005*, volume 3679 of *LNCS*, pages 159–177. Springer, Heidelberg, September 2005.

ALP12. Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 367–385. Springer, Heidelberg, December 2012.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

BCI+13. Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

BCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014.

BF14. Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 520–537. Springer, Heidelberg, March 2014.

BFKW09. Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87. Springer, Heidelberg, March 2009.

BGI+12. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012.

BGI14. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

BMS16. Michael Backes, Sebastian Meiser, and Dominique Schröder. Delegatable functional signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 357–386. Springer, Heidelberg, March 2016.

CHYC04. Sherman S. M. Chow, Lucas Chi Kwong Hui, Siu-Ming Yiu, and K. P. Chow. Secure hierarchical identity based signature and its application. In Javier López, Sihan Qing, and Eiji Okamoto, editors, *ICICS 04*, volume 3269 of *LNCS*, pages 480–494. Springer, Heidelberg, October 2004.

CKLM14. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. In *CSF*, pages 199–213. IEEE Computer Society, 2014.

CT10. Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS 2010*, pages 310–331. Tsinghua University Press, January 2010.

GGH+13. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

GMM+16. Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 241–268. Springer, Heidelberg, October / November 2016.

GS02. Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 548–566. Springer, Heidelberg, December 2002.

JMSW02. Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Heidelberg, February 2002.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

KN08. Eike Kiltz and Gregory Neven. Identity-based signatures. In Marc Joye and Gregory Neven, editors, *Identity-Based Cryptography*, volume 2 of *Cryptology and Information Security Series*, pages 31–44. IOS Press, 2008.

LAS+10. Jin Li, Man Ho Au, Willy Susilo, Dongqing Xie, and Kui Ren. Attribute-based signature and its applications. In Dengguo Feng, David A. Basin, and Peng Liu, editors, *ASIACCS 10*, pages 60–69. ACM Press, April 2010.

MPR11. Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 376–392. Springer, Heidelberg, February 2011.
Nac10. David Naccache. Is theoretical cryptography any good in practice? CRYPTO & CHES 2010 invited talk, 2010.
OT14. Tatsuaki Okamoto and Katsuyuki Takashima. Efficient attribute-based signatures for non-monotone predicates in the standard model. *IEEE Trans. Cloud Computing*, 2(4):409–421, 2014.
SBZ02. Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In Kwangjo Kim, editor, *ICISC 01*, volume 2288 of *LNCS*, pages 285–304. Springer, Heidelberg, December 2002.
Sha84. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.
SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

A Some Definitions of Required Primitives

In this section, we recall the formal definitions of SNARKs, proof-carrying data, and notions related to obfuscation. All these notions are used in this paper to construct our Universal Witness Signature scheme.

A.1 SNARK Proof Systems

Succinct non-interactive arguments of knowledge, or SNARKs for short, are powerful proof systems that we use in particular to instanciate proof-carrying data constructions. To achieve this we define the SNARK proof system for the universal language on Random-Access Machines.

Definition 7 (Universal relation and Language). *The* universal relation *is the set $\mathcal{R}_U$ of instance-witness pairs $(y, w) = ((M, x, t), w)$, where $|y|, |w| \leq t$ and M is a random-access machine, such that M accepts (x, w) after at most t steps. We denote by $\mathcal{L}_U$ the* universal language *corresponding to the universal relation $\mathcal{R}_U$.*

For any constant $c > 0$, we denote by $\mathcal{R}_c \subset \mathcal{R}_U$ the subset of $\mathcal{R}_U$ consisting of pairs $(y, w) = \big((M, x, t), w\big)$ such that $t \leq |x|^c$ (and hence the running time of M on input (x, w) is polynomially bounded).

We will define a SNARK proof system for NP using those relations $\mathcal{R}_c$. Such a proof system is a triple $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ of three algorihms operating as follows:

$\mathcal{G}(1^\lambda)$**:** on input of the security parameter λ, outputs a common reference string crs and verification state τ;
$\mathcal{P}(\mathsf{crs}, y, w)$**:** produces a proof π of the statement $y = (M, x, t)$ if w is a valid witness;
$\mathcal{V}(c, \tau, y, \pi)$**:** deterministically verifies the proof π, provided that $y = (M, x, t)$ satisfies $t \leq |x|^c$.

The proof system is a SNARK if it verifies the following properties.

Definition 8 (SNARK for NP). *A* SNARK for NP *is a triple $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ as above statisfying the following properties.*

- ***Completeness**: For all large enough λ, all constants $c > 0$, and every pair $(y, w) = ((M, x, t), w) \in \mathcal{R}_c$, we have*

$$\Pr\left[\mathcal{V}(c, \tau, y, \pi) = 1 \;\middle|\; (\mathsf{crs}, \tau) \leftarrow \mathcal{G}(1^\lambda);\ \mathcal{P}(\mathsf{crs}, y, w)\right] = 1.$$

- ***Adaptive proof of knowledge**: For every polynomial size prover $\mathcal{P}^*$, there exists a polynomial size extractor $\mathcal{E}_{\mathcal{P}^*}$ and a negligible function ε such that for all λ, all constants $c > 0$ and every auxiliary input string $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, we have*

$$\Pr\left[\begin{matrix}\mathcal{V}(c, \tau, y, \pi) = 1 \\ \text{and } (y, w) \notin \mathcal{R}_c\end{matrix} \;\middle|\; \begin{matrix}(\mathsf{crs}, \tau) \leftarrow \mathcal{G}(1^\lambda) \\ (y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}, z) \\ w \leftarrow \mathcal{E}_{\mathcal{P}^*}(\mathsf{crs}, z)\end{matrix}\right] \leq \varepsilon(\lambda).$$

- ***Full succinctness**: There exists a universal polynomial p such that, for all large enough λ, all $c > 0$ and every instance $y = (M, x, t)$ such that $t \leq |x|^c$, we have that:*
  - *the generator $\mathcal{G}(1^\lambda)$ runs in time $p(\lambda)$;*
  - *the prover $\mathcal{P}(\mathsf{crs}, y, w)$ runs in time $p(\lambda + |M| + |x| + t)$;*
  - *the verifier $\mathcal{V}(c, \tau, y, \pi)$ runs in time $p(\lambda + |M| + |x|)$ (and independent of c); and*
  - *an honestly generated proof π has length $|\pi| \leq p(\lambda)$.*

A.2 Proof-Carrying Data

Proof Carrying Data (PCD), introduced by Chiesa and Tromer, is a cryptographic mechanism for ensuring that a given property is maintained at every step of a computation, typically in a distributed setting. The property of interest is specified as a compliance predicate, and every modification of the data comes accompanied with a proof that the data, and any operation leading to its current contents, satisfies the compliance predicate.

In this section we give a formal definition of PCD adapted from Bitansky et al. [BCCT13].

Definition 9 (Distributed computation transcript). *A* distributed computation transcript *is a tuple* $T = (G, \mathsf{linp}, data)$ *with* $G = (V, E)$ *a directed acyclic graph,* $\mathsf{linp} : V \to \{0,1\}^*$ *a label function for vertices and* $data : E \to \{0,1\}^*$ *a label function for edges. We require that* $\mathsf{linp}(v) = \perp$ *for all* v *which are sources or sinks. The output of* T*, denoted* $out(T)$*, is equal to* $data(\tilde{u}, \tilde{v})$ *which is the lexicographical first edge such that* $\tilde{v}$ *is a sink.*

Definition 10 (Proof-Carrying transcript). *A* proof-carrying transcript *is a pair* (T, π) *with* T *a distributed computation transcript and* $\pi : E \to \{0,1\}^*$ *an edge label function.*

Definition 11 (Compliance predicate). *A* compliance predicate $\mathcal{C}$ *is a polynomial-time computable predicate for nodes of the distributed computation transcript. We denote it* $\mathcal{C}(z_{out}; \mathsf{linp}, z_{in})$*, where* z_{in} *is some input,* z_{out} *is the (alledged) output, and the node's label is* linp*.*

Given a distributed computation transcript DCT, we say that node n *in DCT, with inputs* z_{in} *and local input* linp*, is* $\mathcal{C}$*-compliant if* $\mathcal{C}(z_{out}, \mathsf{linp}, z_{in})$ *holds for every output* z_{out} *of* n*. We say that DCT is* $\mathcal{C}$*-compliant if every node in the graph is* $\mathcal{C}$*-compliant. We say that a string* z *is* $\mathcal{C}$*-compliant if there exists a* $\mathcal{C}$*-compliant distributed computation transcript containing an edge labeled* z*.*

Definition 12 (Distributed-computation generator). *A* distributed-computation generator *is an algorithm* $S(\mathcal{C}, \sigma, PCT)$ *which takes a* $\mathcal{C}$*-compliance, a reference string* σ *and a computation transcript. Then at every time step, it chooses to do one of the following actions*

- *Add a new unlabeled vertex to the computation transcript. Then the algorithm outputs a tuple* $(\text{"add unlabeled vertex"}, x, \perp)$*, with* x *the new vertex.*
- *Label an unlabeled vertex. Then it outputs* $(\text{"label vertex"}, x, y)$ *with* $x \in V$ *neither a source nor a sink and* $\mathsf{linp}(x) = \perp$*, and* y *is the new label of the vertex.*
- *Add a new labeled edge. Then it outputs* $(\text{"add labeled edge"}, x, y)$ *with* $x \notin E$ *and* y *the label of the edge.*

We introduce in Algorithm A.1 the $\mathsf{ProofGen}(\mathcal{C}, \sigma, S, \mathcal{P})$ procedure, which describes an interactive protocol with $\mathcal{C}$ a compliance predicate, σ a reference string, S a distributed-computation generator (not necessarily efficient) and $\mathcal{P}_{\mathcal{C}}$ a PCD prover w.r.t. $\mathcal{C}$, the PCD prover $\mathcal{P}_{\mathcal{C}}$ interacts with the distributed-computation generator $\mathcal{S}$ such that when $\mathcal{S}$ chooses to add a labeled edge $\mathcal{P}_{\mathcal{C}}$ produces a proof for the $\mathcal{C}$-compliance of the new message and add this new proof as the proof label of the edge.

Using the above $\mathsf{ProofGen}$ algorithm, we can give the formal definition of the Proof-Carrying Data scheme.

Definition 13 (Proof-carrying data scheme). *A* proof-carrying data (PCD) scheme *is a triple of algorithms* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *with* $\mathcal{G}$ *is probabilistic and* $\mathcal{P}$ *and* $\mathcal{V}$ *are deterministic.*

A proof-carrying data system *for a class of compliance predicates* C *is a triple of algorithm* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *that works as follows:*

- $\mathcal{G}(1^\lambda) \to \mathsf{crs}$*, on input the security parameter* λ*, outputs a common reference string* crs*.*
- $\mathcal{P}_{\mathcal{C}}(\mathsf{crs}, \pi_i, z_i.z_o, \mathsf{linp}) \to \pi_o$ *for a compliance predicate* $\mathcal{C} \in C$ *which takes as input a common reference string* crs*, inputs* z_i *with corresponding proofs* π_i*, a local input* linp*, and an output* z_o*. The algorithm produces a proof* π_o *for the fact that* z_o *is consistent with some* $\mathcal{C}$*-compliance transcript such that* $\mathcal{C} \in C$*.*
- $\mathcal{V}_{\mathcal{C}}(\mathsf{crs}, z_o, \pi_o) \to \{\mathsf{True}, \mathsf{False}\}$ *for a compliance predicate* $\mathcal{C} \in C$ *takes as input a common reference string* crs*, an output* z_o *with corresponding proof* π_o*, the algorithm returns* True *if* π_o *is a valid proof of the fact that* z_o *is consistent with some* $\mathcal{C}$*-compliance transcript, otherwise it returns* False*.*

And there exists a negligible function $\varepsilon(\cdot)$ *such that* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *satisfies:*

Algorithm A.1: $\mathsf{ProofGen}(\mathcal{C}, \sigma, S, \mathcal{P})$

Result: z_o, π_o, T

Set T and PCT to be "empty transcript" (with $T = (G, \mathsf{linp}, data)$ and $PCT = (T, proof)$ with $G = (V, E) = (\emptyset, \emptyset)$);
while *S doesn't halt and outputs a message-proof pair* (z_o, π_o) **do**
  $(b, x, y) \leftarrow S(\mathcal{C}, \sigma, PCT)$;
  if $b =$ *"add unlabeled vertex"* **then**
    Set $V := V \cup \{x\}$ and $\mathsf{linp}(x) := \perp$
  else if $b =$ *"label vertex"* **then**
    Set $\mathsf{linp}(x) := y$
  else if $b =$ *"add labeled edge"* **then**
    Parse $(v, w) := x$ with $(v, w) \in V^2$;
    $E := E \cup \{(v, w)\}$;
    $data(v, w) := y$;
    if v *is a source* **then**
      set $\pi := \perp$
    else
      $\pi := \mathcal{P}_{\mathcal{C}}(\sigma, data(v, w), \mathsf{linp}(v), inputs(v), inproofs(v))$ with $inputs(v) := (data(u_1, v), \ldots, data(u_c, v)), inproofs(v) := (proof(u_1, v), \ldots, proof(u_c, v))$ and $(u_1, \ldots, u_c) := parents(v)$.;
      $proof(v, w) := \pi$
    end
  end
end
$Outputs(z_o, \pi_o, T)$

– ***Completeness****: For every compliance predicate $\mathcal{C} \in C$ and (possibly unbounded) distributed computation generator S,*

$$\Pr\left[\begin{array}{c} T \text{ is } B\text{-bounded} \\ \mathcal{C}(T) = 1 \\ \mathcal{V}(\sigma, z_o, \pi_o) = \mathsf{False} \end{array} \middle| \begin{array}{c} \sigma \leftarrow \mathcal{G}(1^\lambda, B) \\ (z_o, \pi_o, T) \leftarrow \mathsf{ProofGen}(\mathcal{C}, \sigma, S, \mathcal{P}) \end{array}\right] \leq \varepsilon(\lambda)$$

– ***Proof of Knowledge****: For every polynomial-size prover $\mathcal{P}^*$ there exists a polynomial-size extractor $\mathcal{E}_{\mathcal{P}^*}$ such that for every compliance $\mathcal{C} \in C$, every large enough security parameter $k \in \mathbb{N}$, every auxiliary input $z_{in} \in \{0,1\}^{poly(k)}$, and every time bound $B \in \mathbb{N}$.*

$$\Pr\left[\begin{array}{c} \mathcal{V}(\sigma, z_{out}, \pi) = \mathsf{True} \\ \Rightarrow (out(T) = z_{out} \wedge \mathcal{C}(T) = 1) \end{array} \middle| \begin{array}{c} \sigma \leftarrow \mathcal{G}(1^\lambda, B) \\ (z_{out}, \pi) \leftarrow \mathcal{P}^*(\sigma, z_{in}) \\ T \leftarrow \mathcal{E}_{\mathcal{P}^*}(\sigma, z_{in}) \end{array}\right] \geq 1 - \varepsilon(\lambda)$$

A.3 Punctured Pseudorandom Functions and Obfuscation

Punctured pseudorandom functions (punctured PRFs), first introduced by Boyle et al. [BGI14], have been extensively used [SW14] with the obfuscators to construct provably secure cryptographic schemes. In our case, we combine punctured PRFs with an indistinguishability obfuscation to get the UWS scheme.

Definition 14 (Punctured Pseudorandom Function). *A puncturable family of PRFs is a triple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{F})$ corresponding to generation of initial parameters, getting punctured keys and applying the PRF. These three algorithms have to verify that there exists a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions.*

– ***Functionality preserved under puncturing****: For every PPT adversary A such that $A(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$, then for all $x \in \{0,1\}^{n(\lambda)}$ where $x \notin S$, we have that:*

$$\Pr\left[\mathcal{F}(K, x) = \mathcal{F}(K_S, x) : K \leftarrow \mathcal{G}(1^\lambda), K_S = \mathcal{P}(K, S)\right] = 1$$

– ***Pseudorandomness at punctured points****: We consider an experiment with a PPT adversary (A, D) in which $A(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$ and a state σ, and the challenger then generates a PRF key $K \leftarrow \mathcal{G}(1^\lambda)$, punctures it on the set S: $K_S \leftarrow \mathcal{P}(K, S)$, and then sends σ, S, K_S to the distinguisher D together with either the list of evaluations $\mathcal{F}(K, x)$ of the PRF on the elements x of S (where the x's are sorted in lexicographic order, say), or a list of uniformly random strings of the same lengths as those evaluations. The goal of the adversary is to distinguish those two cases, and* pseudorandomness at punctured points *says that the probability of success is negligible.*

Obfuscation of programs is a powerful notion in cryptography, as it enables many attractive protocols [SW14].

As proved by Barak et al. [BGI+12], the ideal virtual black-box notion can not be achieved for all polynomial size circuits (also it can in principle be achieved for a large class of specific circuits nonetheless). As a result, Barak et al. also introduced weaker notions of obfuscation which do not suffer from the same impossibility result, including in particular *indistinguishability obfuscation* ($i\mathcal{O}$) which is defined as follows, and for which concrete candidates have been proposed since then—we can in particular mention the seminal construction of Garg et al. [GGH+13] from multilinear maps.

Definition 15 (Indistinguishability obfuscation). *A uniform PPT machine $i\mathcal{O}$ is called an* indistinguishability obfuscation *($i\mathcal{O}$) for a circuit class $\{\mathcal{C}_\lambda\}$, if it verifies the two following properties:*

1. ***Functionality Preserving****: For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that*

$$\Pr\left[C'(x) = C(x) | C' \leftarrow i\mathcal{O}(\lambda, C)\right] = 1$$

2. ***Indistinguishability of obfuscation****: For any (not necessarily uniform) PPT distinguisher $(Samp, D)$, there exists a negligible function $\varepsilon(\cdot)$ such that the following conditions holds: if for all security parameters $\lambda \in \mathbb{N}$.*

$$\Pr\left[\forall x. C_0(x) = C_1(x) | (C_0, C_1, \tau) \leftarrow Samp(1^\lambda)\right] \geq 1 - \varepsilon(\lambda)$$

then

$$\begin{aligned} &|\Pr\left[D(\sigma, i\mathcal{O}(\tau, C_0)) = 1 \mid (C_0, C_1, \tau) \leftarrow Samp(1^k)\right] \\ &- \Pr\left[D(\sigma, i\mathcal{O}(\tau, C_1)) = 1 \mid (C_0, C_1, \tau) \leftarrow Samp(1^k)\right]| \leq \varepsilon(\lambda) \end{aligned}$$

B Additional Primitives from UWS

B.1 Hierarchical Identity-Based Signatures

Hierarchical identity-based signatures (HIBS) were first introduced by Chow et al. [CHYC04]. In that hierarchical version of identity-based signatures, the Private Key Generators are organized as a tree structure. An identity of depth ℓ, is represented by an ℓ-tuple $\mathsf{id} = (\mathsf{id}_0, \mathsf{id}_1, \ldots \mathsf{id}_{\ell-1})$, and given the signing key $\mathsf{SK}_{\mathsf{id}}$ associated with id, it is possible to extract signing keys $\mathsf{SK}_{\mathsf{id}'}$ on all identities id' of the form $\mathsf{id}' = (\mathsf{id}_0, \ldots, \mathsf{id}_{\ell-1}, \mathsf{id}'_\ell, \ldots)$ (i.e. such that id is a prefix of id').

Definition 16 (HIBS). *A* hierarchical identity-based signature *is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Extract}, \mathsf{Sign}, \mathsf{Verify})$ *which verifies the following specifications:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{PP}, \mathsf{msk})$*: Setup algorithm takes the security parameter λ as input and it outputs public parameters* PP*, and the master signing key* msk*, which is a signing key on the depth zero identity $()$ (which is a prefix of all identities).*
- $\mathsf{Extract}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, \mathsf{id}') \to \mathsf{SK}_{\mathsf{id}'}$*: Given the secret key* $\mathsf{SK}_{\mathsf{id}}$ *on identity* id*, and an identity* id' *such that* id *is a prefix of* id'*, output a signing key* $\mathsf{SK}_{\mathsf{id}'}$ *on* id'*.*
- $\mathsf{Sign}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, m) \to \sigma_m$*: Takes a signing key corresponding to the identity* id *and a message m. Outputs a signature σ_m of the message m.*
- $\mathsf{Verify}(\mathsf{PP}, \sigma_m, m, \mathsf{id}) \to \{\mathsf{True}, \mathsf{False}\}$*: Takes the master verification key, a message m, and its signature σ_m as input. It outputs* True *or* False*.*

These algorithms need to verify also the following standard correctness property: if SK *is a signing key for identity* id *and $\sigma \leftarrow \mathsf{Sign}(\mathsf{SK}, \mathsf{id}, m)$, then we have* $\mathsf{Verify}(\mathsf{PP}, \sigma, m, \mathsf{id}) = \mathsf{True}$*.*

HIBS from UWS. Consider an HIBS scheme with identity space $\mathcal{I}$ and message space $\mathcal{M}$. We can define an order relation $\leq$ on the disjoint union $\mathcal{I} \sqcup (\mathcal{I} \times \mathcal{M})$ as follows: $\mathsf{id}' \leq \mathsf{id}$ if and only if id is a prefix of id'; $(\mathsf{id}', m) \leq ID$ if and only if id is a prefix of id'; and $(\mathsf{id}', m') \leq (\mathsf{id}, m)$ if and only if $m = m'$ and id is a prefix of id'. Then the HIBS scheme can easily be constructed from any universal witness signature scheme for the order relation $\leq$ (note also that since $\leq$ is efficiently computable, witnesses can be omitted). The construction is as described in Figure 5.

Security of HIBS. Chow et al. [CHYC04] give the following game to define the security of HIBS. That model captures what they call existential unforgeability under selective identity, adaptive chosen-message-and-identity attacks.

- The adversary $\mathcal{A}$ outputs an identity id^* which will be used to challenge the security of the HIBS scheme.
- The challenger $\mathcal{C}$ takes a security parameter λ and computes the public parameters PP and the master signing key msk, sending the public parameters PP to the adversary $\mathcal{A}$ and keeping the master signing key msk secret.
- The adversary $\mathcal{A}$ can submit two types of queries:
  Extract: in which $\mathcal{A}$ receives the signing key $\mathsf{SK}_{\mathsf{id}}$ on any identity id of his choosing
  Sign: in which $\mathcal{A}$ chooses a message m and an identity id, and receives a valid signature σ on message m under identity id
- Finally, $\mathcal{A}$ outputs a message m^* and a signature σ^*, and he wins if and only if $\mathcal{V}(\mathsf{PP}, \sigma^*, m^*, \mathsf{id}^*) = \mathsf{True}$.

That security notion is clearly satisfied by our UWS-based construction of HIBS, provided that the UWS scheme is adaptively secure (we in fact achieve the stronger notion of unforgeability under fully adaptive attacks). However, starting from a selectively secure UWS scheme, we may not be able to satisfy the property above, since the UWS definition of selective security would require the adversary to announce both the target identity and the target message of her forgery from the start (since we are effectively forging on (id^*, m^*)).

B.2 Redactable Signatures

Redactable signatures are a type of homomorphic signature scheme originally defined by Johnson, Molnar, Song and Wagner [JMSW02], that allows to redact part of a signed message while preserving signature validity. They were proposed as a way of guaranteeing the authenticity of documents published as part of public disclosure initiatives while taking security and privacy concerns into account. They have also found uses in medical and legal settings.

Let us consider the alphabet $\Sigma = \{0, 1, \#\}$. We define a partial order on Σ with $\# \leq_\Sigma 0$ and $\# \leq_\Sigma 1$, no other non-trivial order in the alphabet. Then this order can induce a partial order in the set Σ^* by point-wise comparison. Which means $x_0, x_1, \ldots, x_n \leq y_0, y_1, \ldots, y_n$ if $\forall i \in \{0, \ldots, n\}. x_i \leq_\Sigma y_i$.

Definition 17 (Redactable signature scheme). *A* redactable signature scheme *is a tuple of algorithms* $(\mathcal{G}, \mathcal{S}, \mathcal{V}, \mathcal{D})$ *with* $(\mathsf{PP}, \mathsf{sk}) \leftarrow \mathcal{G}(1^\lambda)$ *which verifies the following properties:*

- $\mathcal{S}(\mathsf{PP}, \mathsf{sk}, m)$ *outputs a signature* σ_m *on* $m \in \Sigma^*$.
- $\mathcal{D}(\mathsf{PP}, \sigma_m, m, m')$ *takes as input the public parameters, a signature* σ_m *on a message* m*, and another message* $m' \in \Sigma^*$*, and returns a signature* $\sigma_{m'}$ *on* m' *provided that* σ_m *is valid and* $m' \leq m$.
- *The verification algorithm* $\mathcal{V}$ *is subject to the obvious correctness constraints (it accepts validly generated and derived signatures).*

$\underline{\mathsf{Setup}(1^\lambda)}$:
return $\mathsf{UWS.Setup}(1^\lambda)$

$\underline{\mathsf{Sign}(\mathsf{SK}_{\mathsf{id}}, \mathsf{id}, m)}$:
return $\mathsf{UWS.Delegate}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, (\mathsf{id}, m))$

$\underline{\mathsf{Extract}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, \mathsf{id}')}$:
return $\mathsf{UWS.Delegate}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, \mathsf{id}')$

$\underline{\mathsf{Verify}(\mathsf{PP}, \sigma_m, m, \mathsf{id})}$:
return $\mathsf{UWS.Verify}(\mathsf{PP}, \sigma_m, (\mathsf{id}, m))$

Fig. 5. HIBS from UWS.

Redactable signatures from UWS. We will use our UWS scheme to construct the redactable signature scheme. Let us consider a UWS scheme $(\mathsf{Setup}, \mathsf{Delegate})$ with respect to the order $\leq$ on the set $\{*\} \cup \Sigma^*$, where Σ^* is ordered as in the redactable signature scheme, and $*$ is appended as the greatest element. Note that the order $\leq$ is efficiently computable, so that we do not actually need witnesses. We can construct a redactable signature scheme as descibed in Figure 6.

Security of redactable signatures. A redactable signature is said to be secure (i.e. unforgeable) if an efficient adversary allowed to make signature queries on arbitrary messages $m_1, \ldots, m_q \in \Sigma^*$ is not able to produce a valid signature on a new mesage m^* that does not satisfy $m^* \leq m_i$ for any i.

Our construction clearly satisfies that property, provided that the underlying UWS scheme is adaptively secure.

C Adaptive Security of the OWF-Based Construction

Theorem 6. *If the signature scheme* Sig *used in Section 4.1 is adaptively and existentially unforgeable then the OWF-based* UWS *scheme based is also adaptively secure with our definition.*

Proof. We will construct an attacker against the adaptively existential unforgeability of the underlying signature scheme $\mathcal{A}_{\mathsf{Sig}}$ by using an attacker against the existential unforgeability of the UWS scheme $\mathcal{A}_{\mathsf{UWS}}$.

Suppose that we have an adversary $\mathcal{A}_{\mathsf{UWS}}$ which wins the existential unforgeable security game against the UWS scheme with an advantage ε. By definition, it can produce SK_A which is a valid signature of A. Thus SK_A verifies the following properties:

1. $\mathsf{SK}_A = (\mathsf{sk}_A, [(\mathsf{vk}_A, A, w_{A \leq id_n}, \sigma_{\mathsf{vk}_A}),$
   $(\mathsf{vk}_{id_n}, id_n, w_{id_n \leq id_{n-1}}, \sigma_{\mathsf{vk}_{id_n}}), \ldots, (\mathsf{vk}_{id_1}, id_1, w_{id_1 \leq *}, \sigma_{\mathsf{vk}_1}), (\mathsf{vk}_*, *, w_{* \leq *}, \sigma_{\mathsf{vk}_*})])$
2. For $m \leftarrow random(1^\lambda)$ and $\sigma_m \leftarrow \mathsf{Sig.Sign}(m, \mathsf{sk}_A)$, $\mathsf{Sig.Verify}(\mathsf{vk}_A, m, \sigma_m)$ outputs True
3. The identities verifies that $A \leq_{w_{A \leq id_n}} id_n \wedge \cdots \wedge * \leq_{w_{* \leq *}} *$.
4. Each signature in the certificate verifies the following conditions:
   - $\mathsf{Sig.Verify}(\mathsf{mvk}, *||w_{* \leq *}||\mathsf{vk}_*, \sigma_{\mathsf{vk}_*}) \rightarrow \mathsf{True}$
   - $\vdots$
   - $\mathsf{Sig.Verify}(\mathsf{vk}_{id_{n-1}}, n||w_{id_n \leq id_{n-1}}||\mathsf{vk}_{id_n}, \sigma_{\mathsf{vk}_{id_n}}) \rightarrow \mathsf{True}$
   - $\mathsf{Sig.Verify}(\mathsf{vk}_{id_n}, A||w_{A \leq id_n}||\mathsf{vk}_A, \sigma_{\mathsf{vk}_A}) \rightarrow \mathsf{True}$

Suppose that the number of requests of delegation is bounded by a fixed number q and the adversary $\mathcal{A}$ can produce $\mathsf{SK}_m = (\mathsf{sk}_m, c_m)$ corresponding to m where c_m is a certificate of the signature. For example, in the case of SK_A, $c_A = [(\mathsf{vk}_A, A, w_{A \leq id_n}, \sigma_{\mathsf{vk}_A}), \ldots, (\mathsf{vk}_*, *, w_{* \leq *}, \sigma_{\mathsf{vk}_*})]$. We define 2 types of forgeries:

- Type 0 forgery: c_m is a suffix of $c_{m'}$ which is a certificate of m' revealed by the requests of the adversary $\mathcal{A}$.
- Type 1 forgery: c_m is not a suffix of any $c_{m'}$ which is a certificate of m' revealed by the request of the adversary $\mathcal{A}$.

Definition 18 (Valid certificate). *c_x is a* valid certificate *of x if and only if the adversary has required a signature of y and c_x is a suffix of c_y. We note* $\mathrm{maxPref}(c_x)$ *the longest suffix of c_x which is a valid certificate for a certain identity*[5].

[5] This is uniquely defined since suffixes are totally ordered.

$\underline{\mathcal{G}(1^\lambda)}$:	$\underline{\mathcal{V}(\mathsf{PP}, \sigma_m, m)}$:
return $\mathsf{UWS.Setup}(1^\lambda)$	return $\mathsf{UWS.Verify}(\mathsf{PP}, \sigma_m, m)$
$\underline{\mathcal{S}(\mathsf{PP}, \mathsf{sk}, m)}$:	$\underline{\mathcal{D}(\mathsf{PP}, \sigma_m, m, m')}$:
return $\mathsf{UWS.Delegate}(\mathsf{PP}, \mathsf{sk}, *, m)$	return $\mathsf{UWS.Delegate}(\mathsf{PP}, \sigma_m, m, m')$

Fig. 6. Redactable signatures from UWS.

To construct an adversary $\mathcal{A}_{Sig}$, we need simulate the delegation oracle $\mathcal{O}_{\mathsf{Delegate}}$ and the reveal oracle $\mathcal{O}_{Reveal}$ by the functions in the underlying signature scheme.

Let us consider the associative list $\mathcal{H}$ initially empty. Let $q_{\mathcal{O}}$ be the number of queries to the oracle $\mathcal{O}$. We will proof that the attacker $\mathcal{A}_{\mathsf{UWS}}$ with non-negligible advantage ε against its own security game can win the security game of the underlying signature scheme with advantage $\frac{\varepsilon}{2q_{\mathcal{O}_{\mathsf{Delegate}}}+1}$ which is not negligible. We suppose that $\mathcal{A}_{\mathsf{UWS}}$ outputs $\mathsf{SK}_m = (sk_m, c_m)$ as challenge text against the UWS scheme. We choose an integer i in $\{0, \ldots, 2q_{\mathcal{O}_{\mathsf{Delegate}}}\}\}$ and then proceed as in the case i described below:

Case 0*:* In this case, we guess that $\mathcal{A}_{\mathsf{UWS}}$ is a type 1 forgery and $\mathrm{maxPref}(c_m)$ is the empty suffix ε. Firstly we construct a simulator which simulates $\mathcal{O}_{\mathsf{Delegate}}$ and $\mathcal{O}_{\mathsf{Reveal}}$.

- $\mathcal{O}_{\mathsf{Delegate}}$: We construct the delegation function just as it has been defined.
- $\mathcal{O}_{\mathsf{Reveal}}$: Outputs the corresponding value in the associative list.

During the procedure Setup, the simulator follows the same procedure except that for signing $m_* = *||w_{*\leq *}||\mathsf{vk}_*$ we replace the signing algorithm $\mathsf{Sig.Sign}$ by the signing oracle $\mathcal{O}_{\mathsf{msk}}$ using the master signing key msk.

Suppose that there exists an adversary $\mathcal{A}$ which produces an attack text SK' of type 0 forgery against the UWS_{OWF} scheme. Let $(m'_* = (*'||w'_{*\leq *}||\mathsf{vk}'_*), \sigma_{\mathsf{vk}_*})$ be the last element of the certificate of SK'. We submit the message-signature pair $(m'_*, \sigma'_{\mathsf{vk}_*})$ as a challenge text for underlying signature scheme using the master signing key msk. As $(m'_*, \sigma'_{\mathsf{vk}_*})$ is not a suffix of any c_x revealed by the adversary (by the assumption of this case), so the probability of $\mathcal{O}_{sign}$ has evaluated on m'_* is zero. Thus if our guess is correct, we can win the security game against the underlying signature scheme with a non-negligible probability.

Case i *(for* $i \in \{1, \ldots, q_{\mathcal{O}_{\mathsf{Delegate}}}\}$*):* In these cases, we guess that $\mathcal{A}_{\mathsf{UWS}}$ is a type 1 forgery, and it outputs (SK_A, A) with $\mathsf{SK}_A = (\mathsf{sk}_A, c_A)$. In the case i we suppose that in the i-th query's result SK_x we have $c_x = \mathrm{maxPref}(c_A)$.

Then we construct an adversary $\mathcal{A}_{Sig}$ for the underlying signature scheme.

- O_{Delegate} : We follow the construction of the delegation function in the UWS scheme for the first i queries. Then when we use the function $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_x, x, y, w)$, we replace the signing procedure using sk_x by the signing oracle O_{sk_x} until another different sk_x corresponding to x have been generated.
- O_{Reveal} : Outputs the corresponding value in the associative list.

Let $(m, \sigma_m) = ((vk, id, w), \sigma_m)$ be the element in c_A just after $\mathrm{maxPref}(c_A)$. The existence of (m, σ_m) is assured by the fact that $\mathrm{maxPref}(c_A) \neq c_A$ otherwise it is a type 0 forgery. Then we submit it as challenge text for the underlying signature scheme with signing key sk_x. With a negligible probability $\mathcal{O}_{\mathsf{sk}_x}$ has signed the message m (even $\mathcal{A}_{\mathsf{UWS}}$ have required to delegate from x to m, any valid signing key corresponding to the identities smaller than m can never be revealed, otherwise it contradict with the assumption "$c_x = \mathrm{maxPref}(c_A)$", so we can simply ignore these requests), but as SK_A is a valid signature of A. σ_m is a valid signature of m w.r.t the signing key sk_x which means if our guess is correct then we have constructed an adversary $\mathcal{A}_{\mathsf{Sig}}$ who can win the security game against the underlying signature scheme.

Case i *(for* $i \in \{q_{\mathcal{O}_{\mathsf{Delegate}}} + 1, \ldots, 2q_{\mathcal{O}_{\mathsf{Delegate}}}\}$*):* In these cases, we guess that $\mathcal{A}_{\mathsf{UWS}}$ is a type 0 forgery and it outputs (SK_A, A) with $\mathsf{SK}_A = (\mathsf{sk}_A, c_A, c_{\mathsf{sk}_A})$. In the case i, we suppose that in the $(i - q_{\mathcal{O}_{\mathsf{Delegate}}})$-th query's result SK_x we have $c_x = c_A$. Then we construct an adversary $\mathcal{A}_{Sig}$ for the underlying signature scheme.

- $\mathcal{O}_{\mathsf{Delegate}}$: We follow the construction of the delegation function in the UWS scheme for the first $i - q_{\mathcal{O}_{\mathsf{Delegate}}} - 1$ queries. Then when we use the function $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_x, x, y, w)$, we replace the signing procedure using sk_x by the signing oracle $\mathcal{O}_{\mathsf{sk}_x}$.
- $\mathcal{O}_{\mathsf{Reveal}}$: Outputs the corresponding value in the associative list.

Let m be a message randomly generated. We obtain the signature σ_m corresponding to the message m using sk'_x obtained in the SK_A. As m is generated randomly, so the signing oracle $\mathcal{O}_{\mathsf{sk}_x}$ only has a negligible probability to have signed the message m. So with a non-negligible probability (m, σ_m) is a valid challenge text for the security game against the underlying signature scheme. Thus if our guess of this case is correct then we can construct an adversary against the underlying signature scheme.

Put all together: Finally as the $2q_{\mathcal{O}_{\mathsf{Delegate}}} + 1$ types of forgeries recover all the possibilities,[6] our adversary constructed $\mathcal{A}_{\mathsf{Sig}}$ can win at least one of them with advantage ε, and the adversary choose randomly between the $2q_{\mathcal{O}_{\mathsf{Delegate}}} + 1$ cases, then the advantage of $\mathcal{A}_{\mathsf{Sig}}$ against the underlying signature scheme is at least $\frac{\varepsilon}{2q_{\mathcal{O}_{\mathsf{Delegate}}}+1}$, which is still non-negligible if ε is non-negligible. □

D Security Proof of the PCD-Based Construction

Suppose that there exists an adversary $\mathcal{A}_{\text{PCD}}$ with non-negligible advantage against the PCD-based UWS scheme described in Section 4.2. In the selective security game, the adversary first announces the identity id^* that will be targeted. Since as a hypothesis, there are only polynomially many identities, the set $I = \{id \mid \nexists id'.id < id' \wedge id^* \not< id'\}$ also has polynomial size. We ask the $\mathsf{Delegate}_{\text{OWF}}$ oracle to sign all identities in the set I, then we construct a simulator for the $\mathsf{UWS}_{\text{PCD}}$ oracles.

We construct the following simulator S:

- $\mathsf{Delegate}_{\text{PCD}}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B \leq A})$: S verifies that SK_A is a valid signature. Then use the signatures of the identities in I to generate a valid signature for the identity B. Then store it in an associative list[7].
- $\mathsf{Reveal}_{\text{PCD}}(A)$: This algorithm follows the honest procedure of access to the associative list.

Together with the attacker $\mathcal{A}_{\text{PCD}}$, the simulator S can be considered as a prover which provides a valid PCD proof (z_{id^*}, π_{id^*}). Then by the proof of knowledge property there exist a extractor E algorithm which can produce a valid distributed computation transcript $T \leftarrow E(\mathsf{PP})$ verifing that $out(T) = z_{id^*}$ and $\mathcal{C}(T) = \mathsf{True}$. From this transcript we can construct a valid signature $(\mathsf{SK}_{id^*}, c_{id^*})$ for id^* in the underlying signature scheme. This will be a valid attack for the underlying signature scheme in the construction of UWS scheme.

Remind that actually our construction is also selective with the a pre-order without anti-symmetric property, because in our construction, when we delegate to an identity whether there already exists a vertex with the same label, we will always create a new vertex. Thus the distributed computation script corresponding to the pre-order is always acyclic.

E Construction from Virtual Black-Box Obfuscation

Obfuscations of programs is a very powerful cryptographic primitive, from which many attractive protocols can be constructed [SW14].

For our construction, we remind that in previous sections we have constructed two different UWS. But these constructions have not been proven completely context-hiding. In this section, we will use the Virtual Black-Box (VBB) obfuscator to construct the first context-hiding UWS scheme.

Note that constructing a VBB obfuscator for *all* circuits is known to be impossible in the standard model, by a result of Barak et al. [BGI+12]. Nevertheless, this impossibility result doesn't preclude VBB to be achievable for a given, restricted class of circuits (which is the setting in which we use it here); moreover, some existing candidate obfuscators have been shown to achieve VBB for all circuits in idealized models, such as the generic graded encoding model or the weak multilinear map model. Therefore, we regard the use of VBB obfuscation to establish feasibility results in a clean and easy way as quite reasonable: it yields concrete instantiations that are secure in idealized models, can also provide useful insights towards extending these results to weaker forms of obfuscation (like indistinguishability obfuscation) that do not suffer from the same theoretical limitations.

Definition 19 (Virtual Black-Box Obfuscator). *A* virtual black-box obfuscator $\mathcal{O}$ *is a probabilistic algorithm verifying the following three properties:*

- ***Functionality****: For every circuit C, the string $\mathcal{O}(C)$ describes a circuit that given any inputs outputs the same result as C.*
- ***Polynomial Slowdown****: There exists a polynomial p such that $|\mathcal{O}(C)| \leq p(|C|)$.*

[6] Note that they are also all needed, to cover both type 0 and type 1 forgers.
[7] Which may be implemented efficiently as a hash table.

Fig. 7. The $P[\mathsf{msk}]$ algorithm.

$P[\mathsf{msk}](\mathsf{mvk}, \mathsf{SK}_A, A, B, w)$:
if $\mathsf{Sig.Verify}(\mathsf{mvk}, A, \mathsf{SK}_A) \wedge A \geq_w B$ then return $\mathsf{SK}_B = \mathsf{Sig.Sign}(\mathsf{msk}, B)$ else return $\mathsf{SK}_B = \perp$

– ***Virtual Black-Box****: For any PPT adversary $\mathcal{A}$, there exists a PPT algorithm S and a negligible function ε such that for all circuits C:*

$$\left|\Pr\left[\mathcal{A}(\mathcal{O}(C)) = 1\right] = \Pr\left[S^C(1^{|C|}) = 1\right]\right| \leq \epsilon\left(|C|\right)$$

We give our construction as follows. Let $\mathcal{O}$ be a VBB obfuscator and $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ a deterministic existential unforgeable signature scheme. Assume that all identities and witnesses have at most polynomial size with respect to the security parameter. Let us consider the following UWS scheme:

– $\mathsf{Setup}(1^\lambda)$:
  • We generate a pair of signature keys $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$. Then compute the VBB obfuscation $P^\flat = \mathcal{O}(P[\mathsf{msk}])$ of the algorithm $P[\mathsf{msk}]$
  • The master signing key (signature of $*$) SK_* is $\mathsf{Sig.Sign}(\mathsf{msk}, *)$ and the public parameter is $\mathsf{PP} = (P^\flat, \mathsf{mvk})$.
– $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B \leq A})$:
  • $P^\flat(\mathsf{mvk}, \mathsf{SK}_A, A, B, w_{B \leq A})$

  where P is described in Figure 7, in which msk is a constant. This algorithm is doing directly what we want to do, it checks if SK_A is a valid signature of the identity A.

Theorem 7. *The VBB-based construction is correct, succinct and context-hiding.*

Proof.

– **Correctness** Using the functionality of VBB obfuscation, we can see that the valid signature SK_A is equal to $\mathsf{Sig.Sign}(\mathsf{msk}, A)$. Thus if $B \leq_w A$ and $\mathsf{SK}_B = \mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w)$ then wee have $\mathsf{Sig.Verify}(\mathsf{mvk}, B, \mathsf{SK}_B)$ outputs True, which means $\mathsf{Verify}(\mathsf{mvk}, \mathsf{SK}_B, B)$ does not output $\perp$.
– **Succinctness, context-hiding.** A valid signature SK_A equals to $\mathsf{Sig.Sign}(\mathsf{msk}, A)$, then the signature is completely independent from the delegation path, thus the UWS scheme is succinct and context-hiding.

□

The intuition for this construction's security is based on the virtual-black box property of VBB obfuscation. A detailed proof of security is given below.

E.1 Security Proof of the VBB-Based Construction

We simulate the two oracles $\mathcal{O}_{\mathsf{Delegate}}$ and $\mathcal{O}_{\mathsf{Reveal}}$, which will be used by the adversary later on:

– $\mathcal{O}_{\mathsf{Delegate}}(\mathsf{PP}, A, B, w_{B \leq A})$: If $A = *$ or the value corresponding to the identity A in the associative list is $\top$, then it add $(B, \top)$ to the associative list.
– $\mathcal{O}_{\mathsf{Reveal}}(B)$: If the value corresponding to the identity B in the associative list is $\top$, then it outputs $\mathsf{Sig.Sign}(\mathsf{msk}, B)$.

We will present the security proof using hybrids:

– Hybrid 0: This hybrid is the real-world adaptive security game for the adversary $\mathcal{A}$. Denote the advantage of $\mathcal{A}$ by Adv_0
– Hybrid 1: In this hybrid, we replace $\mathsf{PP} = (\mathsf{mvk}, P^\flat)$ by $\mathsf{PP} = \mathsf{mvk}$. and consider the adversary $S^{P(\cdot)}$, which is an adversary with the oracle access to the algorithm P. Denote the advantage of $\mathcal{S}^{\mathcal{P}}$ by Adv_1.

Lemma 1. *There exists a negligible function ε such that with the security parameter λ:*

$$Adv_0 \leq Adv_1 + \varepsilon(\lambda).$$

Proof. By the virtual black-box property which is:

$$\forall \mathcal{A}.\exists \mathcal{S}.\left|\Pr\left[\mathcal{A}(\mathcal{O}(\mathsf{Delegate}))=1\right]-\Pr\left[\mathcal{S}^{\mathsf{Delegate}(\cdot)}\left(1^{|\mathsf{Delegate}|}\right)=1\right]\right| \leq \varepsilon(|\mathsf{Delegate}|)$$

We know that, $Adv_0 \leq Adv_1 + \varepsilon(\lambda)$. □

Lemma 2. *There exists a negligible function ε such that with the security parameter λ:*

$$Adv_1 \leq \varepsilon(\lambda).$$

Proof. By the adaptive security of the signature scheme, there exists a negligible function $\varepsilon(\cdot)$ such that all adversaries $\mathcal{A}$ have at most $\varepsilon(\lambda)$ advantage against the signature scheme.

And from the security game in the hybrid 1, we can see that we can simulate the oracle access to the delegation algorithm by the access of the signing oracle. From a forgery (m, SK_m) of the UWS scheme, we submit it as a forgery of the underlying signature scheme. In the security game, by the definition S^P has never required to get a message-signature pair $(m', \mathsf{SK}_{m'})$ such that $m \leq m'$. Thus (m, SK_m) is a valid forgery for the underlying signature scheme. By the adaptive security of the signature scheme and the fact that S^P is a valid adversary, we have $Adv_1 \leq \varepsilon(\lambda)$. □

Succinctness. The succinctness of the UWS scheme is clear from the construction. A valid signature of the message m in the UWS scheme by the construction in equal to $\mathsf{Sig.Sign}(\mathsf{msk}, m)$. Thus if the underlying signature scheme is succinct. Then the UWS scheme constructed is succinct.

F Security of the Indistinguishability Obfuscation-Based Construction

F.1 Security Proof

We prove the selective security of the constructed signature scheme using the following hybrids.

Hybrid 0 : In this hybrid, we proceed exactly as in the initial selective security game of the UWS scheme.

1. Adversary chooses a message m.
2. Challenger uses the PRF's generation algorithm $\mathcal{G}$ to get the key K for the PRF.
3. Challenger uses the obfuscator $i\mathcal{O}$ to get an indistinguishability obfuscation $\mathsf{CheckSign}^\flat[K]$ of the function $\mathsf{CheckSign}[K]$ and sets it as a public parameter and keep the PRF's key K secret.
4. Adversary can require the Challenger to apply polynomial times the function $\mathsf{CheckSign}^\flat[K](\mathsf{SK}_A, A, B, w)$ and can only require the Challenger to reveal a secret key SK_A of A if $A < m$.
5. Adversary returns a secret key corresponding to the message m.

We topologically sort all the identities $e_1, \ldots, e_\ell$ greater or equal to m in non-increasing order. In particular, $e_\ell = m$, and the number ℓ of such identities is polynomial by assumption.

We also initialize a list L to the empty list, and two associative lists $\mathcal{H}_L$ and $\mathcal{H}'_L$ with key space L, also initially empty. As we advance through the list of hybrids, the list L will be expanded to contain all identities e_i up to a certain index i, and the associative lists will contain bitstrings associated with those identities.

For all $i \in \{1, \ldots, \ell\}$, we define the following four hybrids.

Hybrid i : We add $(e_i, y_i = \mathcal{F}(K_L, e_i))$ to the associative list $\mathcal{H}_L$ and $(e_i, z_i = f(y_i))$ to the associative list $\mathcal{H}'_L$, then we include e_i in the list $L \leftarrow L \cup \{e_i\}$ and we puncture the PRF key K_L at the point e_i where $K_\emptyset = K$ used in the hybrid 0. We also modify the obfuscated circuit as Algorithm F.1:

Algorithm F.1: $\mathsf{CheckSign}[K_L, \mathcal{H}_L](\mathsf{SK}_A, A, B, w)$

Result: SK_B
if $B \in L$ **then**
  if $A == B$ *and* $f(\mathsf{SK}_A) = \mathcal{H}'_L(A)$ **then**
    outputs $\mathsf{SK}_B := \mathsf{SK}_A$
  else if $A >_w B$ *and* $f(\mathsf{SK}_A) = \mathcal{H}'_L(A)$ **then**
    outputs $\mathsf{SK}_B := \mathcal{H}_L(B)$
else if $A \in L$ **then**
  if $A >_w B$ *and* $f(\mathsf{SK}_A) = \mathcal{H}'_L(A)$ **then**
    outputs $\mathsf{SK}_B := \mathcal{F}(K_L, B)$
else
  if $A == B$ *and* $f(\mathcal{F}(K_L, A)) = f(\mathsf{SK}_A)$ **then**
    outputs $\mathsf{SK}_B := \mathsf{SK}_A$
  else if $A >_w B$ *and* $f(\mathcal{F}(K_L, A)) = f(\mathsf{SK}_A)$ **then**
    outputs $\mathsf{SK}_B := \mathcal{F}(K_L, B)$
  else
    outputs $\perp$
  end
end

Hybrid i' : In this hybrid, we replace $y_i = \mathcal{H}_L(e_i)$ by a random $y_i^\star$ in the domain of f, and $z_i = \mathcal{H}'_L(e_i)$ by $z_i^\star = f(y_i^\star)$.

Hybrid i'' : We replace $y_i^\star$ by an independent random element $y_i^{\star\star}$, without modifying $z_i^\star$.

Hybrid i''' : We replace $z_i^\star$ by an independent random element $z_i^{\star\star}$ in the codomain of f, without modifying $y_i^{\star\star}$.

Final hybrid : In the final hybrid which is after hybrid ℓ''', we modify the obfuscated algorithm by Algorithm F.2.

Algorithm F.2: $\mathsf{CheckSign}[K_L, \mathcal{H}_L](\mathsf{SK}_A, A, B, w)$

Result: SK_B
if $A \notin L$ **then**
  if $A == B$ *and* $f(\mathsf{SK}_A) == f(\mathcal{F}(K_L, A))$ **then**
    outputs $\mathsf{SK}_B := \mathsf{SK}_A$
  else if $A >_w B$ *and* $f(\mathsf{SK}_A) == f(\mathcal{F}(K_L, A))$ **then**
    outputs $\mathsf{SK}_B := \mathcal{F}(K_L, B)$
else
  outputs $\perp$
end

Proof. We begin the proof by noticing that in the final hybrid, there is no valid signature of m, so that the probability of the adversary win the security game is 0. Moreover, in all hybrids, the challenger can simulate all valid Reveal queries from the adversary, as he possesses the PRF key K_L punctured on the points L, and none of the identities of valid Reveal queries belong to L.

Thus, to complete the proof, it suffices to show that the hybrids are indistinguishable.

Lemma 3. *If the adversary $\mathcal{A}$ can distinguish hybrid i and the one that precedes it, then $\mathcal{A}$ can break the indistinguishability property of the underlying indistinguishability obfuscator.*

Proof. The only difference between hybrid i and the one that precedes it is the change of the circuit $\mathsf{CheckSign}$ to be obfuscated; namely, the PRF key is punctured at e_i, and evaluations of the PRF at that point are replaced by their values (as program constants) prior to puncturing. As a result the circuits corresponding to both hybrids are functionally equivalent, since each modified evaluation is replaced by its correct value in the previous program. It follows that if $\mathcal{A}$ can distinguish between the two obfuscated programs, it breaks the indistinguishability property of the underlying indistinguishability obfuscator.

Lemma 4. *If the adversary $\mathcal{A}$ can distinguish hybrid i and hybrid i' with non-negligible probability, then $\mathcal{A}$ can break the pseudorandomness of the PRF at punctured points.*

Proof. The only difference between the hybrid i and the hybrid i' is that we replace $\mathcal{H}_L(e_i) = \mathcal{F}(K_{\{e_1,\ldots,e_{i-1}\}}, e_i)$ by a random value, and we only know $K_{\{e_1,\ldots,e_i\}}$, by the pseudorandomness of the PRF at the punctured point e_i, the hybrids i and i' are indistinguishable.

Lemma 5. *If the adversary $\mathcal{A}$ can distinguish hybrid i' and hybrid i'' with non-negligible probability, then $\mathcal{A}$ can break the indistinguishability of the underlying indistinguishability obfuscator.*

Proof. (of Lemma 5) The only points at which the two circuits differ are of the form $(\mathsf{SK}_{e_j}, e_j, e_i, w)$, where $e_j >_w e_i$ and in particular $j < i$ and $f(\mathsf{SK}_{e_j}) = z_j^{\star\star}$. In particular, $z_j^{\star\star}$ must belong to the image of f, which happens with negligible probability since f is length-doubling and $z_j^{\star\star}$ is a random element in the codomain. Thus, with overwhelming probability, the circuits are functionally equivalent. Thus any adversary which can distinguish these two hybrid can break the indistinguishability of the underlying indistinguishability obfuscator.

Lemma 6. *If the adversary $\mathcal{A}$ can distinguish hybrid i'' and hybrid i''' with non-negligible probability, then $\mathcal{A}$ can break the pseudorandomness of the underlying PRG f.*

Proof. The only thing changed between hybrid i'' and hybrid i''', is that we change the output of the PRG by a random value in the codomain, if an adversary $\mathcal{A}$ can distinguish these two hybrids, then $\mathcal{A}$ breaks also the pseudorandomness of the underlying PRG function.

Lemma 7. *If the adversary $\mathcal{A}$ can distinguish the penultimate hybrid ℓ''' and and the final hybrid with non-negligible probability, then $\mathcal{A}$ can break the indistinguishability of the underlying indistinguishability obfuscator.*

Proof. This proof is very similar to the proof of Lemma 5. The only points at which the two circuits differ are of the form $(\mathsf{SK}_{e_j}, e_j, e, w)$, where $e_j \in L$ and in particular $j \leq \ell$ and $f(\mathsf{SK}_{e_j}) = z_j^{\star\star}$. This means $z_j^{\star\star}$ must be an image of f, but since f is length-doubling and $z_j^{\star\star}$ is a random element in the codomain by the modification of the previous hybrids, which can only happen with negligible probability. Thus the two circuits are functionally equivalent with overwhelming probability, and as a result, if the adversary $\mathcal{A}$ can distinguish the two hybrids, it can break the indistinguishability of the underlying indistinguishability obfuscator.

F.2 Context-Hiding Property and Succinctness

Apart from the very basic notion of unforgeability, there are many other properties which are interesting such as context-hiding and succinctness. We will show that our construction verifies such properties.

Remind that the value of the signing key SK_A for the identity A is equal to $\mathcal{F}(K, A)$.

Context-hiding. As defined in the preliminary, no adversary can distinguish between SK_B^1 and SK_B^2 which were generated respectively by $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_{A_1}, A_1, B, w_1)$ and $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_{A_2}, A_2, B, w_2)$, because $\mathsf{SK}_B^1 = \mathsf{SK}_B^2$. Actually they have exactly the same value.

Succinctness. As we mentioned before, a valid signature SK_A for an identity A is equal to $\mathcal{F}(K, A)$. So the length of the signature is bounded by the length of messages in the PRF's range.