Delegateable Signatures

Boaz Barak*

March 4, 2001

Abstract

In this paper we define and construct a new cryptographic tool: a *delegateable signature scheme*. Informally, a delegateable signature scheme is a signature scheme where the owner of the signing key (Alice) can *securely* delegate to another party (Bob) the ability to sign on Alice's behalf on a restricted subset S of the message space. By secure delegation we require both that this other party (Bob) would not be able to generate a signature for a message $m \notin S$ and that the signatures generated by the delegated party (Bob) would be indistinguishable from those generated by the owner of the signature key (Alice). As usual, we also require that any other party will not be able to generate a signature for any message m that it has not seen a signature for. We construct a delegateable signature scheme based on any trapdoor permutation. In constructing this scheme we use a slight variant of non-interactive zero-knowledge proofs of knowledge ([?],[?]) which may be of independent interest. As an example for an application of delegateable signatures, we show that it can be used in order to construct certificates that preserve the user's anonymity and unlinkability.

Keywords: cryptography, digital signatures, delegateable signatures, non-interactive zero-knowledge proofs of knowledge, anonymous unlinkable certificates

*Department of Computer Science, Weizmann Institute of Science, Rehovot, ISRAEL. E-mail: `boaz@wisdom.weizmann.ac.il`

1 Introduction

To motivate the notion of delegateable signatures we shall consider the following (over)-simplified scenario. Suppose that Alice has a bank account, and can perform transactions in this account, by signing the action she wants performed using her private signing key s_{Alice}. The bank can then validate the request against Alice's public verification key v_{Alice}. Suppose now that Alice wishes to allow Bob to perform on her behalf any transaction that involves an amount of at most \$5. Offhand, one can think of three ways she can do this:

1. A first approach would be that whenever Bob wants to perform a transaction t (that involves at most \$5) on Alice's account, Bob would ask Alice to sign the message t using her signing key s_{Alice} and pass the signed message t to the bank. The drawback in this approach is that Bob will need to interact with Alice every time he wants to perform a transaction.

2. The second way is for Alice to give Bob her signing key s_{Alice}. Yet this is clearly insecure, as Bob will be able to perform all sorts of transactions in Alice's account, rather than only the transactions that involve an amount of at most \$5.

3. The third way would be for Alice to sign in advance on a list of messages that contain all of the possible transactions that involve an amount of at most \$5. That is, Alice would sign in advance on all possible messages of the form "transfer \$x from Alice's account to account Y" where x ranges from 0.00 to 5.00 and Y ranges over all possible account numbers. Then, Alice would hand this list over to Bob and any time that Bob wants to perform a transaction he would use the appropriate signed message from the list. The drawback of this approach is that this list may be far too large for Alice to sign each message in it, and for Bob to keep it.

A delegateable signature scheme allows a fourth alternative approach to solving this problem. In a delegateable signature scheme Alice can give Bob a *restricted signing key*. The restricted signing key $s_{Alice,T}$ will allow its holder to sign only messages that fit the *template* T. For example, in our case, T would be the template "transfer \$x from Alice's account to account Y" where $0 \leq x \leq 2$ and Y is any string. Formally, we'll say that a template T is a boolean predicate defined over the set of all strings (i.e. $T : \{0,1\}^* \to \{0,1\}$). We say that a message m *fits the template* T if $T(m) = 1$. In such a scheme we call Alice's key s_{Alice} the *master* signing key (as opposed to a restricted signing key). We require two properties from a delegateable signature scheme:

1. The holder of the master signing key can create a T*-restricted signing key* for any (efficiently computable) function $T : \{0,1\}^* \to \{0,1\}$. We say that s_T is a T-restricted signing key if using s_T one is able to sign any message m that satisfies $T(m) = 1$ and no other messages. Note that this condition implies that a party that did not receive any restricted key s_T cannot sign on any message, and so this condition implies the standard security of signature schemes (security against existential forgery).

2. If T, T' are two (efficiently computable) functions and m is a message such that $T(m) = T'(m) = 1$ then it is infeasible to distinguish between a signature that was generated using s_T and a signature that was generated using $s_{T'}$. Without loss of generality, we can assume that the holder of the master signing key does not use it directly to sign on a message m, but rather generates a T-restricted signing key s_T for some template T such that $T(m) = 1$ and uses s_T to sign on the message. Therefore, this requirement implies that a signature on a message m that is created by the delegator that received a restricted key is indistinguishable

from a signature on the same message m that is created by the holder of the master signing key.

Why require indistinguishable signatures? Consider the above scenario of Alice and Bob. We intend that the restricted key $s_{Alice,T}$ would allow Bob to generate Alice's signatures on any message that fits the template T. Therefore we would like require that a signature on a message m that Bob produces using $s_{Alice,T}$ and Alice's own signature on m are identically distributed. However, for practical purposes we can relax this requirement and only require that these signatures are computationally indistinguishable. It also seems natural to require that a signature given by a delegator on a message m should not reveal the set of messages that this specific delegator is allowed to sign on. Furthermore, the indistinguishable signatures property is essential in some applications of delegateable signatures (in particular it is essential in the application presented in Section 4).

In this paper we provide a formal definition for delegateable signature schemes, and a construction (based on trapdoor permutations) that meets this definition. We also demonstrate how delegateable signatures can be used, by using them in Section 4, in constructing anonymous certificates (For more detail on this application, see Section 4).

Related Work. Delegateable signatures are related to *group signatures*, introduced by Chaum and van Heyst ([**?**]), this relationship and further related work is described in Section 4.3. Our construction uses a variant of non-interactive zero knowledge proofs of knowledge. These were introduced by Rackoff and Simon ([**?**]). For more related work in this subject, see Section 3. Related work in the subject of anonymous certificates is discussed in Section 4.

Organization of this paper. In section 2 we give a formal definition of delegateable signature scheme. We describe our construction of delegateable signatures in Section 3. A major component of this construction are non-interactive zero knowledge proofs of knowledge ([**?**]), which are described in Section 3.1. We then describe several possible extensions to the basic definition of delegateable signatures. A more detailed description, along with a sketch of the modifications needed to our construction in order to obtain these extensions is provided in Appendix B. An example of an application for delegateable signatures is described in Section 4.

1.1 Preliminaries

By standard digital signatures we mean a signature scheme that is secure against adaptive existential forgery attack ([**?**]). A review of this definition is given in Appendix E.

We'll use the standard notion of *boolean circuits*. For a review of the definition of boolean circuits, see Appendix D. We will identify boolean circuits with their canonical representation as strings in $\{0,1\}^*$. The *size* of a circuit C (denoted by $|C|$) is the size of this representation. If C is a circuit with n inputs then we denote by $L(C)$ the following language, $L(C) \stackrel{def}{=} \{x \in \{0,1\}^n | C(X) = 1\}$. For $m \in \{0,1\}^*$, we denote by C_m the canonical circuit of size $10|m|$ such that $L(C_m) = \{m\}$. We assume that any circuit C of n inputs is of size at least $10n$.

An *algorithm* can be either a uniform algorithm, modeled by a Turing machine, or a non-uniform algorithm, modeled by a sequence of circuits for each input length. Our adversaries in this paper will be *polynomial non-uniform algorithms*, which are modeled by a sequence of circuits $\{C_n\}_{n\in\mathbb{N}}$ such that $|C_n| \leq p(n)$ for some polynomial $p(\cdot)$.

An *oracle* algorithm is defined in the standard way (either as a Turing machine with a special oracle query tape, or as a sequence of circuits with oracle gates). If A is an oracle algorithm, and B is an algorithm then by $A^B(x)$ we denote the result of running A on input x with oracle access to B. If A is an algorithm that takes two inputs then by $A(x,\cdot)$ we denote the algorithm that is the result of fixing A's first input to be x.

If D is a distribution then $x \leftarrow D$ means that x is a random variable distributed according to D. If S is a set then $x \leftarrow_{\mathrm{R}} S$ means that x is a random variable distributed according to the uniform distribution on the elements of S.

A *negligible* function is a function that grows slower that the inverse of any polynomial. That is, $\mu : \mathbb{N} \to \mathbb{N}$ is negligible if for any positive polynomial $p(\cdot)$ there exists a number n_0 such that $\mu(n) < \frac{1}{p(n)}$ for all $n > n_0$. We'll sometimes use $negl(\cdot)$ to denote some unspecified negligible function.

The value $\perp$ is a special symbol that denotes "failure"

2 Defining Delegateable Signatures

When defining delegateable signatures, one needs to decide on some way to represent a template. We've decided on representing a template $T : \{0,1\}^* \to \{0,1\}$ as a boolean circuit C that computes T. We identify the template with the circuit that computes it, and so instead of talking about T-restricted keys, we talk about C-restricted keys. We identify circuits with their canonical representation as strings in $\{0,1\}^*$, therefore when we say something like "run the algorithm A with the circuit C as input" we mean that we run A on input the canonical representation of C as a string in $\{0,1\}^*$.

A delegateable signature scheme is composed of four algorithms:

1. The first algorithm is the *key generator* algorithm, denoted KeyGen. The key generator algorithm generates the *master signing and verifying keys.* By the *master signing key* we mean the key that allows signing on *any* message (as well as creation of restricted signing keys). This is in contrast with a *restricted* signing key. The notation we use is $(s_{\mathsf{master}}, v_{\mathsf{master}}) \leftarrow \textsc{KeyGen}(1^n)$.

2. The second algorithm is the *delegation* algorithm, denoted Delegate. This is the algorithm that allows to create a *restricted signing key.* This algorithm takes as input the master signing key s_{master} and a boolean circuit C, the output of Delegate is a *restricted signing key* s_C that allows only to sign messages $m \in \{0,1\}^*$ such that $C(m) = 1$ (or equivalently, $m \in L(C)$). The notation we use is $s_C \leftarrow \textsc{Delegate}(s_{\mathsf{master}}, C)$.

3. The third algorithm is the *signing algorithm*, denoted Sign. This algorithm takes as input a message m and a restricted key s_C. If $C(m) = 1$ then the output of Sign would be a signature α. The notation is $\alpha \leftarrow \textsc{Sign}(s_C, m)$. At first reading, this seems only to describe how one can sign with a *restricted* key and not how one will sign with the *master* key. However, for simplicity of presentation, rather than introducing a fifth algorithm for signing with the master key, we assume that to sign with s_{master} on a message m, one first generates a restricted key s_{C_m} (recall that C_m is the canonical circuit that satisfies $L(C_m) = \{m\}$), and then use s_{C_m} to sign the message.

4. The fourth algorithm is the *verification algorithm* , denoted Verify. The verification algorithm takes as input the master verification key v_{master} , a message m , and an alleged

signature α. It then outputs 1 if and only if α is a valid signature of m with respect to the key v_{master}.

The following definition presents the requirements from the four algorithms described above.

Definition 2.1. A *delegateable signature scheme* is a quadruple of algorithms (KeyGen, Delegate, Sign, Verify) that satisfies the following three requirements:

1. (Validity) For any circuit C and message $m \in \{0,1\}^*$ such that $m \in L(C)$, if we perform the following experiment:

   (a) Generate of master keys: $(s_{\mathsf{master}}, v_{mas}) \leftarrow \textsc{KeyGen}(1^n)$

   (b) Create a *restricted signing key* s_C that allows to sign messages in $L(C)$: $S_C \leftarrow \textsc{Delegate}(s_{\mathsf{master}}, C)$.

   (c) Sign the message m using the restricted key s_c: $\tau \leftarrow \textsc{Sign}(s_C, m)$.

   Then it will always (i.e. with probability 1) hold that $\textsc{Verify}(v_{\mathsf{master}}, m, \tau) = 1$.

2. (Security against existential forgery) We consider an adversary A that is given the master verification key and access to two oracles. The first oracle, when given a circuit C, provides A with a restricted signing key s_C (corresponding to the above verification key). The second oracle, when given a circuit C' and a message m, provides A with a signature on m using $s_{C'}$. Suppose that the restricted keys that A received from its first oracle are $s_{C_1}, \ldots, s_{C_k}$, and the signed messages that A received from its second oracle are $m_1, \ldots, m_j$. We define $\mathcal{M}$ to be the set of messages that S has acquired signatures for, either explicitly (from the second oracle) or implicitly (from the first oracle). That is, $\mathcal{M} \stackrel{def}{=} L(C_1) \cup \cdots \cup L(C_k) \cup \{m_1, \ldots, m_j\}$. We'll require that it is infeasible for A to output a signed message (m, τ), such that $\textsc{Verify}(v_{\mathsf{master}}, m, \tau) = 1$ and $m \notin \mathcal{M}$.

   Actually, we restrict ourselves to adversaries that have access only to the first oracle; As shown in Lemma F.1, this implies security against an adversary that is given access to both oracles.

   The formal requirement would be as follows: for any polynomial non-uniform oracle algorithm A we perform the following experiment:

   (a) generate the keys $(s_{\mathsf{master}}, v_{\mathsf{master}}) \leftarrow \textsc{KeyGen}(1^n)$.

   (b) Run A with oracle to the delegation algorithm w.r.t. the master signature key s_{master}. That is run $A^{\textsc{Delegate}(s_{\mathsf{master}}, \cdot)}$.

   Suppose that the queries that A made to its oracle are $C_1, \ldots, C_k$. We let $\mathcal{M} \stackrel{def}{=} L(C_1) \cup \cdots \cup L(C_k)$. Let (m, τ) be the output of A, we say that A is *successful* if $\textsc{Verify}(v_{\mathsf{master}}, m, \tau) = 1$ and $m \notin \mathcal{M}$. We require that any polynomial non-uniform oracle algorithm A has only negligible probability of succeeding in this experiment.

3. (Indistinguishable Signatures) If $m \in \{0,1\}^*$ is a message and C_1, C_2 are two circuits of the same size such that $m \in L(C_1) \cap L(C_2)$ then we require that it would be infeasible to distinguish between a signature on m that was made using s_{C_1} and a signature on m that was made using s_{C_2}. The actual attack we are considering is even stronger: after having an access to the same oracle as in the previous item (the oracle that on input a circuit C outputs a restricted signing key s_C), the adversary A will output two circuits C_1, C_2 and a message

m such that $C_1(m) = C_2(m) = 1$ and $|C_1| = |C_2|$. We then choose at random one of the two circuits C_b, create a restricted signing key s_{C_b} and use it to sign on the message m. We require that the probability that A manages to guess the which circuit was used is at most $\frac{1}{2}$ (up to a negligible factor).

The formal requirement follows. For any polynomial non-uniform oracle algorithm A we perform the following experiment:

(a) Generate the keys: $(s_{\mathsf{master}}, v_{\mathsf{master}}) \leftarrow \textsc{KeyGen}(1^n)$

(b) Run A with oracle to the delegating algorithm w.r.t. s_{master} (i.e. run A with oracle to $\textsc{Delegate}(s_{\mathsf{master}}, \cdot)$).

(c) At this stage A outputs is (m, C_1, C_2) where $|C_1| = |C_2|$ and $C_1(m) = C_2(m) = 1$.

(d) Choose $b \leftarrow_{\mathrm{R}} \{1, 2\}$.

(e) Generate a restricted signing key s_{C_b}: Let $s_{C_b} \leftarrow \textsc{Delegate}(s_{\mathsf{master}}, C_b)$.

(f) Provide A with $\textsc{Sign}(s_{C_b}, m)$.

(g) Let $b' \in \{0, 1\}$ be the final output of A.

We say that A is *successful* in this experiment if $b = b'$. We require that A is successful with only negligible advantage over half. That is we require that the probability that after performing this experiment b' is equal to b is at most $\frac{1}{2} + negl(n)$.

As mentioned above, properties 2 (security against existential forgery) and 3 (indistinguishable signatures) imply together that the signature scheme is secure against the stronger attack in which the adversary gets access to a signing oracle in addition to the delegating oracle. This is stated and proved in Lemma F.1. The intuition behind the proof is the following: recall that the signing oracle gets as input a circuit C and a message m (such that $m \in L(C)$) and returns a signature on m using the restricted key s_C. We wish to stress that if $m' \in L(C)$ and $m' \neq m$ then as long as the adversary did not get an (explicit or implicit) signature on m', outputting a valid signature on m' will be considered as a success. It is possible to simulate the signing oracle using the delegating oracle. We can do this by requesting a restricted key s_C from the delegating oracle and using it to sign m. The problem is that if $L(C)$ contains a message $m' \neq m$ then outputting a signature on m' is now not considered as success. Yet if $|C'| = |C|$ and $C'(m) = 1$ then the signature indistinguishability property tells us that we can simulate the signing oracle also by requesting a requesting a restricted key $s_{C'}$ from the delegating oracle and using it to sign m. As we can easily construct a circuit C' such that $|C'| = |C|$ and $L(C') = \{m\}$, we'll use such a circuit C' to simulate the signing oracle. As now there is no message $m' \neq m$ such that $m' \in L(C')$ the problem noted above disappears and we're done.

A note on circuit sizes. The fact that the signature indistinguishability requirement only holds with regards to circuits that are of *exactly* the same size, may seem to make Definition 2.1 too weak. However, if in some setting one knows a certain bound s on the sizes of all circuits that will be used for creating restricted keys (say $s = n^2$ where n is the security parameter), then one can easily pad all circuits to size s and so ensure that they are all of the same size.

3 Construction

In this section we describe our construction of a delegateable signature scheme. We shall use two tools in constructing this scheme: a standard signature scheme (see Appendix E for a review of the definition), and a non-interactive zero knowledge proof of knowledge (NIZKPK).

We start by defining non-interactive zero knowledge proofs in general and proofs of knowledge in particular. We then present our construction of delegateable signature schemes.

3.1 Non-interactive Zero-Knowledge Proofs of Knowledge

We start by recalling the notion of non-interactive zero knowledge proofs (NIZK) . Non-interactive zero knowledge proofs were introduced by Blum, Feldman and Micali and constructed by them under specific number theoretical assumptions ([**?**]). Feige, Lapidot and Shamir construct *adaptive multiple proof* non-interactive zero knowledge proofs under the assumption that trapdoor permutations exist ([**?**]). This is the definition we that shall use. Efficient NIZK are constructed by Damgard under the quadratic residuosity assumption ([**?**]) and by Kilian and Petrank under the assumption that trapdoor permutation exist ([**?**]). For more exposition on NIZK, see [**?**], Section 4.10.

Non-interactive zero knowledge proofs of knowledge (NIZKPK) were first defined and constructed by Rackoff and Simon ([**?**]) in a model where all provers get a pair of decryption and encryption keys of a public-key cryptosystem. These keys are generated by a trusted party and the encryption key of each prover is listed in a public directory. Subsequently there has also been some work on defining NIZKPK in the random string model ([**?**]). Our definition of NIZKPK is very similar to one of the definitions presented in [**?**] while our construction is almost identical to the construction of [**?**].

NP relations We say that a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is an **NP** *relation* if there exists a polynomial $p(\cdot)$ such that for any $(x, w) \in R$, $|w| \leq p(|x|)$ and in addition there exists a polynomial time Turing machine for deciding membership in R. We denote by $L(R)$ the following language: $L(R) \stackrel{def}{=} \{x | \exists w \text{ s.t. } (x, w) \in R\}$. We say that $L \in \mathbf{NP}$ if $L = L(R)$ for some **NP** relation R.

The model for non-interactive zero-knowledge proofs is the following: Some trusted party generates a *common reference string* Σ. In the standard versions of NIZK this reference string is random, but this requirement is not really necessary for most applications (and in particular ours) and so we will not make it. Once this reference string is established, any party can prove statements such as "$x \in L$" (where L is some **NP** language) with respect to the reference string Σ. The proofs can be verified by anyone that has access to the string, but yet are zero-knowledge. Basically a non-interactive zero knowledge proof for an **NP** language L (where $L = L(R)$ for some **NP** relation R) is composed of four components:

1. The *generator* (Gen) is the algorithm that generates the common reference string Σ. The generator generates in addition to Σ, some *auxiliary information*, denoted aux_S. In the actual use, the honest user should discard this auxiliary information, but it will be used by the simulator that demonstrates the zero-knowledge property. The notation is $(\Sigma, aux_S) \leftarrow \text{Gen}(1^n)$.

2. The *prover* (Prove) is the algorithm that is used to prove theorems of the type "$x \in L$". It gets as input the common reference string Σ, the theorem to be proved x, and a *witness*

w that satisfies $(x, w) \in R$. It outputs a proof π (that indeed $x \in L$). The notation is $\pi \leftarrow \textsc{Prove}(\Sigma, x, w)$.

3. The *proof verifier* (VerProof) is the algorithm that verifies the proofs. Its inputs are the reference string Σ, a theorem x, and an alleged proof π. It returns 1 if and only if π is a valid proof for x with respect to Σ.

4. The *simulator* (Sim) demonstrates the zero knowledge property of the prover: it should simulate proofs of true theorems without knowing a witness to these theorems. Its inputs are the common reference string Σ, a theorem x (whose proof would be simulated), and the auxiliary information aux_S (generated as item 1) . The notation is $\pi \leftarrow \textsc{Sim}(\Sigma, x, aux_S)$.

For a formal definition of NIZK in this model see Definition C.1

The following theorem is proven[1] in [**?**]:

Theorem 3.1. *Assuming that (non-uniformly secure) trapdoor permutations exists, there exists a non-interactive zero knowledge proof for any language in* **NP**.

For our purposes we actually need a non-interactive zero knowledge *proof of knowledge* (NIZKPK). The definition of non-interactive zero knowledge proof of knowledge, is somewhat simpler than the definition of an *interactive* zero knowledge proof of knowledge (although it is of similar nature). We change the generator Gen, to output in addition to the reference string Σ (and the auxiliary information aux_S), another piece of auxiliary information which we'll denote by aux_E (Notation: $(\Sigma, aux_S, aux_E) \leftarrow \textsc{Gen}(1^n)$). This auxiliary information will be used by a new algorithm: the *extractor* (denoted Ext). The extractor would be an algorithm, that given (in addition to the reference string Σ and the auxiliary information aux_E) any theorem x and a proof π for x that passes verification w.r.t. Σ, will almost always (i.e., with $1 - negl(n)$ probability) output a witness w such that $(x, w) \in R$. For a formal definition see Definition C.2

We have the following theorem:

Theorem 3.2. *Suppose that (adaptive multi-theorem) non-interactive zero-knowledge proofs (NIZK) exist for any language in* **NP**, *and that secure public-key encryption schemes exist, then (adaptive multi-theorem) non-interactive zero-knowledge proofs of knowledge (NIZKPK) exist for any* **NP** *relation.*

The construction is described in Appendix C.1. Although our model is different, the construction and proof are almost identical to the construction and proof of [**?**].

Since the existence of trapdoor permutations implies the existence of both NIZK and secure public-key encryption schemes ([**?**], [**?**]), we have the following corollary:

Corollary 3.3. *If (non-uniformly secure) trapdoor permutations exist then there exists an (adaptive multi-theorem) non-interactive zero knowledge proof of knowledge (NIZKPK) for any* **NP** *relation.*

3.2 Constructing Delegateable Signatures using NIZKPK

We next present a construction of delegateable signatures.

We start by describing the main idea behind the construction.

[1] Actually, because our presentation differs from the presentation of [**?**], this Theorem is only implicit there (See also Section 4.10.3 in [**?**]).

3.2.1 The main idea

Consider the following (flawed) attempt at constructing a delegateable signature scheme using a standard (non-delegateable) signature scheme (GEN$'$,SIGN$'$,VERIFY$'$):

1. The master signing and verifying keys $(s_{\mathsf{master}}, v_{\mathsf{master}})$ would be the signing and verifying keys (s, v) of the standard signature scheme (KEYGEN$'$, SIGN$'$, VERIFY$'$) .

2. To delegate the ability to sign on messages in $L(C)$, the holder of the master signing key will sign on the circuit C. That is, the restricted signing key s_C will be the couple $\langle C, \text{SIGN}'(s, C)\rangle$.

3. The signing algorithm, when given a restricted key $s_C = \langle C, \tau\rangle$ and a message m such that $C(m) = 1$, would simple output s_C.

4. The verifying algorithm, when given a message m and an alleged signature $\langle C, \tau\rangle$ on m, would check that $C(m) = 1$ and $\text{VERIFY}'(v, C, \tau) = 1$.

This scheme does satisfy property 2 (security against existential forgery) of Definition 2.1. This is because a valid signature in this scheme on a message m is a couple $\langle C, \tau\rangle$ where $C(m) = 1$ and τ is a signature using the standard scheme on C. By the existential security of the signature scheme, we know that the only way to get such a signature τ is by receiving it from the holder of the signing key. Therefore, if an adversary A has access to a delegating oracle, and then outputs a message m and a couple $\langle C, \tau\rangle$ such that $C(m) = 1$ and $\text{VERIFY}'(v, C, \tau) = 1$ then we know that with very high probability $(1 - negl(n))$ C must be one of the circuits A queried from its oracle. As $m \in L(C)$ this means that A is not successful.

Yet this scheme obviously *does not* satisfy property 3 (indistinguishable signatures) of Definition 2.1 [2]. This is because a signature using a restricted key s_C contains the description of the circuit C itself, and so clearly if $|C| = |C'|$, $m \in L(C) \cap L(C')$ but C is not equal to C' then one can distinguish between a signature on m using s_C and a signature on m using $s_{C'}$. Our construction solves this problem in the following way: when signing on a message m, the signing algorithm, instead of outputting $\langle C, \tau\rangle$ as the signature, *proves in zero knowledge* that it knows a couple $\langle C, \tau\rangle$ such that $C(m) = 1$ and $\text{VERIFY}'(v, C, \tau) = 1$. As we need this proof to be non-interactive, we use NIZKPK.

3.2.2 The actual construction

Recall that a delegateable signature scheme is composed of four algorithms (KEYGEN, DELEGATE, SIGN, VERIFY). In our construction we'll use a standard (non-delegateable) signature scheme (KEYGEN$'$, SIGN$'$, VERIFY$'$) and a NIZKPK (GEN, PROVE , VERPROOF , SIM , EXT) for some **NP** relation R that will be specified later.

Notation For any strings $x, y \in \{0,1\}^*$ we denote by $x \circ y$ the concatenation of x and y.

Construction 3.4. *A delegateable signature scheme.*

- To generate the *master* signing and verification keys, we'll use the standard signature scheme's key generator to obtain a pair of signing and verification keys (s, v). We'll also use the generator of the NIZKPK to generate a reference string Σ. The master signing key s_{master}

[2] Note that it is also not secure against the stronger attack of Lemma F.1.

will be the signing key s. The master verification key v_{master} will be the concatenation of the verification key v and the reference string Σ. Actually, for technical reasons, we add the master verification key to the master signing key. Formally, this means that the algorithm KeyGen is defined as follows:

$$\textsc{KeyGen}(1^n) \stackrel{def}{=} (\underbrace{s \circ v \circ \Sigma}_{s_{\mathsf{master}}}, \underbrace{v \circ \Sigma}_{v_{\mathsf{master}}})$$

where $(s, v) \leftarrow \textsc{KeyGen}'(1^n)$ and $(\Sigma, aux_S, aux_E) \leftarrow \textsc{Gen}(1^n)$.

- To delegate the ability to sign on a set $L(C)$ where C is some circuit, the holder of the master signing key signs on the circuit C. That is the *restricted signing key* s_C is the concatenation of C and a signature (using s) on C. We'll also add to s_C the public information $v_{\mathsf{master}} = v \circ \Sigma$. Formally, the algorithm Delegate is defined in the following way:

$$\textsc{Delegate}(\underbrace{s \circ v \circ \Sigma}_{s_{\mathsf{master}}}, C) \stackrel{def}{=} \underbrace{C \circ \tau \circ v \circ \Sigma}_{s_C}$$

  where $\tau \leftarrow \textsc{Sign}'(s, C)$.

- If a party has a restricted signing key s_C, and wishes to sign on a message m such that $C(m) = 1$, then this party proves (w.r.t. Σ) that it *knows* a circuit C such that $C(m) = 1$ and a signature on C. That is, the signature on a message m is a proof of knowledge of a circuit C that satisfies $C(m) = 1$ and a string τ such that $\textsc{Verify}'(v, C, \tau) = 1$. We define the following relation R'

$$(\langle m, v\rangle, \langle C, \tau\rangle) \in R' \ \text{ iff } \ C(m) = 1 \wedge \textsc{Verify}'(v, C, \tau) = 1$$

  We see that if we have a NIZKPK proof for R', then any proof in this system that $\langle m, v\rangle \in L(R')$ is actually a proof of knowledge of C and τ that satisfy these conditions[3]. Actually, one needs to make the following technical modification to the relation R' to make it an **NP** relation: In order to make the witness' size bounded by some *fixed* polynomial in the size of the theorem, one needs to add the size of the circuit C (in unary format) to the theorem. We define the modified relation R as follows:

$$(\langle m, v, 1^l\rangle, \langle C, \tau\rangle) \in R \text{ iff } C(m) = 1 \wedge \textsc{Verify}'(v, C, \tau) = 1 \wedge |C| = l$$

  We use a NIZKPK for this relation R. Given that, the formal definition of the algorithm Sign is the following:

$$\textsc{Sign}(\underbrace{C \circ \tau \circ \Sigma}_{s_C}, m) \stackrel{def}{=} \underbrace{\textsc{Prove}(\Sigma, \langle m, v, 1^{|C|}\rangle, \langle C, \tau\rangle) \circ 1^{|C|}}_{\alpha}$$

- To verify the validity of a signature $\alpha = \pi \circ 1^l$ on a message m with respect to the master verification key $v_{\mathsf{master}} = v \circ \Sigma$, one simply verifies that π is a valid proof of knowledge of a circuit C (of size l) such that $C(m) = 1$ and a valid standard signature τ on C w.r.t. the

[3]The reason that the circuit C is part of the witness and not part of the theorem is that we don't want a signature on a message m using s_C to reveal the circuit C.

verification key v (i.e. $\textsc{Verify}'(v, C, \tau) = 1$). Formally we define $\textsc{Verify}$ in the following way:

$$\textsc{Verify}(\underbrace{v \circ \Sigma}_{v_{\mathsf{master}}}, m, \underbrace{\pi \circ 1^l}_{\alpha}) = 1 \text{ iff } \textsc{VerProof}(\Sigma, \langle m, v, 1^l \rangle, \pi) = 1$$

The following Theorem is proved in Appendix A

Theorem 3.5. *The tuple (*$\textsc{KeyGen}$*,*$\textsc{Delegate}$*,*$\textsc{Sign}$*,*$\textsc{Verify}$*) of Construction 3.4 is a delegateable signature scheme.*

An immediate corollary is

Corollary 3.6. *Assuming the existence of trapdoor permutations, there exists a delegateable signature scheme.*

3.3 Extensions.

Our basic construction can be extended in several ways. These are described in more detail in Appendix B:

1. *Relaxing the dependence of the signature's size and the restricted key's size on the size of the circuit.* In the current construction the size of a signature made using a restricted key s_C is polynomial in the size of the circuit C. If we allow the signature scheme to become *interactive* we can reduce this size to a polylogarithmic dependence on the size of C (along with a polynomial dependence on the security parameter). However, with this we lose the *transferability* property of the signature scheme. In some cases we may also be able to reduce the size of the restricted key s_C itself.

2. *Allowing delegation of sets in* $\mathbf{NP}/Poly$. The current construction allows to delegate the ability to sign on any subset L such that $L = L(C)$ for some circuit C. We can modify it to allow delegation of the ability to sign on sets L of the form $L = \{x \in \{0,1\}^* | \exists w \in \{0,1\}^* \text{ s.t. } C(x,w) = 1\}$ where C is some two input circuit.

3. *Discouraging sharing of the restricted key.* In the current setting any party that has received a restricted key s_C can share this key with other parties. This is not always desirable. In a setting where each party p knows some private information SK_p that corresponds to some public information PK_p (e.g., $PK_p = f(SK_p)$ where f is a non-way function) we can modify the definition so that a party p will receive a restricted key $s_{C,p}$ that could only be used by someone that knows SK_p. Note that the delegating party *does not* need to know the secret SK_p.

4. *Making signatures indistinguishable even for the delegating user.* In some situations it may desirable that even the holder of the master signing key will not be able to distinguish between to signatures on the same message made with different restricted keys. To obtain that we will need to assume the existence of a public reference string R that was generated by some trusted party and can be used as a reference for many master keys.

5. *Adding more levels of delegation.* In some applications, it may be desirable to allow a party that received a restricted key s_C, to securely delegate to another party the ability to sign on messages in some subset $L' \subseteq L(C)$. Our construction can be modified to achieve this property.

4 An Application to Anonymous Unlinkable Certificates

In this section we demonstrate the applicability of delegateable signatures by using a delegateable signature scheme for constructing "anonymous unlinkable certificates". In Section 4.1 we'll present the problem that anonymous certificates should solve. In Section 4.2 we describe what anonymous certificates are, how they can be constructed using delegateable signatures, and how this solves the problem described before. In Section 4.3 we briefly describe other ways in which this and similar problems were solved in the past, and how our solution compares to the previous solutions. This section will be relatively informal, and will contain no proofs. As a preliminary we start with a description of standard (non-anonymous) certificates and the setting that we consider.

Standard Certificates. We consider an environment where each participating party p knows a secret SK_p that corresponds to some public information PK_p. We assume that this public information PK_p resides in some common public directory (one may assume that $PK_p = f(SK_p)$ where f is a one-way permutation). In general we identify the party p with the corresponding public information PK_p. Therefore we may sometimes say things like "The party PK_p works for IBM". We also assume the existence of a trusted party which we call the *certificate authority* (CA) [4]. This trusted party knows a secret signing key s_{CA} and all the other parties know the corresponding public verification key v_{CA}. A *certificate* is a signature using s_{CA} on a message of the form "The party PK_p has property X". As the certificate authority is trusted, a certificate of this form can be considered as a *proof* that the party PK_p indeed possesses the property X.

4.1 The Problem

Suppose that Joe is one of the parties in the above setting. That is, Joe knows a secret SK_{Joe} that corresponds to the public information PK_{Joe} that resides in the public directory. Now suppose that nobody will do business with Joe unless he shows a certificate that he is worthy of credit (i.e. a message "The party PK_{Joe} is worthy of credit" that passes verification with respect to the key v_{CA}.) We assume that Joe *is* worthy of credit and so can get a certificate σ to this fact from the certificate authority. Joe has to show this certificate σ *every time* he wants to make a transaction. The problem is that this has an adverse affect on Joe's privacy. For instance, suppose that Jane is another party. As any time Joe interacts with Jane, he has to show the certificate σ, Jane can link together all of Joe's transactions with her and keep them in a database. Joe has no way of stopping Jane from gathering this information on him. How can we provide Joe with a solution?

Pseudonyms. One possible solution would be that Joe could use a "pseudonym". That is, there could be a mechanism that would allow Joe to register in the public database under a phony name. We do not find this solution desirable, because although Jane may not know Joe's real name, she can still link all of Joe's transactions to one another. Furthermore, the linked information (in some cases) may even reveal Joe's identity.

One-time Pseudonyms. The problem with the previous solution was that the same pseudonym was reused over and over again, leading to the ability to link all of Joe's transactions. This does not happen if Joe uses each pseudonym once. That is, whenever Joe wants to perform a transaction, he will generate a pair of secret and public information (SK_{temp}, PK_{temp}) and request the CA to sign on the message "The party PK_{temp} is worthy of credit". When asking for this certificate, Joe

[4] For simplicity of exposition we restrict ourself to the case where there is a single trusted certificate authority.

will identify himself to the certificate authority by proving knowledge of SK_{joe} [5]. As Joe is worthy of credit the CA will provide Joe with a certificate on PK_{temp} and he'll be able to use it. As he will use a different pseudonym for each transaction, Joe's transactions will be unlinkable to one another. The problem with this approach is that it requires that Joe interacts with the CA once per transaction. Even if some of this interaction is done off-line, the CA still has to prepare a new certificate for each transaction that Joe makes, and this may place too high a load on the certificate authority.

4.2 Our Approach to Solving the Problem

The basic solution Our basic solution is to use the idea of one-time pseudonyms, but with a twist. Instead of having Joe interact with the certificate authority whenever he needs a new pseudonym, the CA will delegate to Joe the ability to sign certificates on his own. That is, using delegateable signatures, the CA will give Joe a restricted key s_C that allows him to sign on any message of the form "The party Y is worth of credit". Thus whenever Joe wants to make a transaction with Jane, he will generate a new pair of secret and public information (SK_{temp}, PK_{temp}), use the restricted key to sign on the message "The user PK_{temp} is worthy of credit" and show this certificate to Jane.

Restricting the use of the certificate. One problem with the above approach is that in fact the CA gave Joe the ability to sign a certificate for any party he wants. Now Joe can misuse this ability, and for example, provide certificates for other parties that are not worthy of credit. Not only that this misuse cannot be prevented, even if such a false certificate is discovered, it is impossible to trace it back to Joe. It seems that this makes the whole point of certificates meaningless, as no party will be able to really trust a certificate. As a minimal requirement, we would like to discourage Joe from sharing the restricted key s_C he got from the CA. This can be done by making knowledge of SK_{Joe} necessary for generating any signature using s_C. This is one of the extensions to delegateable signature that are discussed in Section 3.3.

Allowing tracing by a central authority Another approach to discourage Joe from misusing his restricted key, is to add the option that if misuse of a certificate that Joe produced (using the restricted key s_C) is detected, this certificate can be traced back to Joe. Suppose that there is another trusted authority (which may or may not be the same as the CA): to be specific let us assume that this trusted authority is the supreme court. Suppose that the supreme court knows a secret decryption key d_{court} that corresponds to a publicly known encryption key e_{court}. Now instead of giving Joe the ability to sign on any message of the form "The party Y is worthy of credit", the CA can give Joe the ability to sign only on messages of the following form: "The party Y is worthy of credit ; c" where c is a ciphertext that is the encryption of "Joe" using the encryption key e_{court}. The set of messages satisfying this condition can be decided by a *non-deterministic* circuit, and so using the extension described in Section 3.3, we can easily extend our scheme to allow delegation of the ability to sign messages in this set. Any time Joe needs a certificate on a new pseudonym, he will encrypt "Joe" using fresh coins, and so his certificates will still be untraceable to parties other than the CA and the courts.

[5] Note that we are not concerned of the fact that the CA may learn information about Joe's transaction. We will talk later about the scenario in which the certificate authority is not totally trusted.

Allowing revocation by the CA Yet another feature we may want, is allowing the CA to revoke Joe's credit. The standard way of revoking certificates is using a *certificate revocation list* (CRL). The problem is that Joe has a template that allows him to generate exponentially many certificates, and so the CA cannot just list them all. The solution would be that each certificate would be concatenated with a *revocation tag*, and each party (for example Joe) would have a *revocation key*. Without knowledge of Joe's revocation key, any revocation tag on the certificates produced by joe would look like a random string. However, if Joe's revocation key is applied (in some specified way) to the revocation tag of any of Joe's certificates, the word "VOID" appears. Therefore, to revoke Joe's credit, the CA would simply post Joe's revocation key in the CRL, and whenever a party receives a certificate, it tries to apply all the revocation keys in the CRL to this certificate, if the word "VOID" appears then the party rejects the certificate. This feature can be obtained in the following way: when Joe requests a restricted key from the CA, the CA will generate a private-key K_{Joe} of a private-key cryptosystem[6]. The CA will then provide Joe with K_{Joe} and with the ability to sign only on messages of the form "The party Y is worthy of credit ; e" where e is an encryption of "VOID" using K_{Joe}. The ciphertext e is the revocation tag, and the key K_{Joe}. One point worth noting is that once Joe's credit is revoked, any party can link together *all* of Joe's transactions, including those made *before* the revocation took place.

Anonymity against the certificate authority One can require that even the CA will not be able to link Joe's transaction. This can be done if the delegating signature scheme satisfies indistinguishable signature w.r.t. the delegating user. Our construction can have this property if we assume existence of a public string that was generated once and for all by a trusted party.

4.3 Comparison with Other Solutions

There has been a lot of work about *pseudonym systems*, which are the same as described above, with a one difference. In these systems, while Joe's transaction *can* be linked to one another, no party *including the CA* can link a transaction back to joe. Pseudonym system were introduced by Chaum, that constructed them using *blind signatures* ([**?**]).

Group signatures, introduced by Chaum and van Heyst, can also be used to solve this problem ([**?**]) although perhaps not with all the extensions noted above. A group signature is similar to delegateable signatures with two differences: (a) the delegating user can only delegate the ability to sign on the *entire* message space $\{0,1\}^*$ (b) given any signature, the delegating user has a way to trace the party that created this signature. The set of parties that were delegated the ability to sign on messages are called the *group*. In fact our above approach can be seen as a way of constructing group signatures using delegateable signatures. This group signature scheme will have the following desirable properties:

1. The size of the group public verification key and the size of a signature are independent of the size of the group

2. Adding and removing (revoking) parties from the group does not require refreshing all the member's signing keys or refreshing the public verification key.

[6] Actually, we'll need to require that the crypto-system satisfies the additional feature that if K and K' are two different randomly selected keys then the encryption of "VOID" using K is indistinguishable from the encryption of "VOID" using K'. This property is achieved if the encryption scheme has *pseudorandom* ciphertexts, where private-key encryption schemes satisfying the latter property can be constructed assuming the existence of one-way functions.

Most known group signature schemes do not enjoy all these properties (see [**?**], [**?**]). However, known group signature schemes are more efficient than the one implied by our construction.

Acknowledgements

I wish to thank Oded Goldreich for helpful discussions and advice. In particular the idea of using CS proofs to make the signature size independent of the circuit size is due to Oded.

A Proof of Theorem 3.5

To prove that Construction 3.4 is indeed a delegateable signature scheme one needs to prove that this construction satisfies the three properties required in Definition 2.1. We shall skip the proof of the validity property, as it directly follows from the definition of the relation R and the perfect completeness of the NIZKPK.

A.1 Proof of security against existential forgery

Let A be an oracle algorithm. Suppose we run with A the experiment described in Property 2 of Definition 2.1, which is hereafter called Experiment 1. Suppose that the output of A is (m, α). To prove security against existential forgery we need to prove the following claim:

Claim A.1. *Let $s_{C_1}, \ldots, s_{C_k}$ be the restricted signature keys that A received from its oracle. Recall that A is* successful *if $\textsc{Verify}(v_{\mathsf{master}}, m, \alpha) = 1$ and $m \notin \mathcal{M}$ where $\mathcal{M} \stackrel{def}{=} L(C_1) \cup \cdots \cup L(C_k)$. The probability that A is successful is negligible.*

Proof. Suppose , toward the contradiction , that A is successful, with non-negligible probability ϵ. We'll build an adversary A' that manages to break the standard signature scheme ($\textsc{KeyGen}'$,$\textsc{Sign}'$,$\textsc{Verify}'$) with probability at least $\frac{\epsilon}{2}$. This will contradict the security of the standard signature scheme (and so we'll be done). Recall that to break the standard signature scheme, algorithm A' gets as input the verification key v and oracle access to the signing algorithm $\textsc{Sign}(s, \cdot)$. The goal of A' would be the output a pair of a message and a signature (m, τ) such that $\textsc{Verify}'(m, \tau) = 1$ but m was *not* queried by A' from its oracle. We now describe the algorithm A':

Algorithm A':

- Input: Verification key v (where $(s, v) \leftarrow \textsc{KeyGen}'(1^n)$.)
- Oracle access: oracle $S(\cdot)$ where $S(m) \stackrel{def}{=} \textsc{Sign}'(s, m)$.

A''s operation:

- Let $(\Sigma, aux_S, aux_E) \leftarrow \textsc{Gen}(1^n)$.
- We let $v_{\mathsf{master}} \stackrel{def}{=} v \circ \Sigma$, $s_{\mathsf{master}} \stackrel{def}{=} s \circ v \circ \Sigma$. (Note that A' only knows v_{master} and does not know all of s_{master}).
- Algorithm A' run algorithm A with input v_{master}. Note that the distribution of v_{master} is identical to the distribution of A's input in Experiment 1.
- Suppose that the i^{th} query A makes to its oracle is the circuit C_i. To answer it, A' first uses the oracle S to produce $\tau_i \leftarrow S(C_i) = \textsc{Sign}(s, C_i)$. Algorithm A' then replies to A with $s_{C_i} \stackrel{def}{=} C_i \circ \tau_i \circ v \circ \Sigma$. Note that this is again exactly the same answer that A would get in the Experiment 1, when it has an oracle to $\textsc{Delegate}(s_{\mathsf{master}}, \cdot)$.
- Let (m, α) be the final output of algorithm A, where $\alpha = \pi \circ 1^l$. (Note that if A is successful then we know that $\textsc{Verify}(v_{\mathsf{master}}, m, \alpha) = 1$ which means that $\textsc{VerProof}(\Sigma, \langle m, v, 1^l \rangle, \pi) = 1$)
- Using the extractor $\textsc{Ext}$, algorithm A' obtains $\langle C, \tau \rangle \leftarrow \textsc{Ext}(\Sigma, \langle m, v, 1^l \rangle, \pi, aux_E)$ and outputs (C, τ).

As both the input A gets and the answers it receives from its oracle are distributed exactly the same as in Experiment 1 we know that with probability ϵ , algorithm A will be successful. This means that A will output a pair (m, α) such that $m \notin \mathcal{M}$ and $\textsc{Verify}(v_{\mathsf{master}}, m, \alpha) = 1$. (Recall that if $C_1, \ldots, C_k$ are the queries that A made, then $\mathcal{M} \stackrel{def}{=} L(C_1) \cup \cdots \cup L(C_k)$.) Since $\textsc{VerProof}(\Sigma, \langle m, v, 1^l \rangle, \pi) = 1$, by the proof of knowledge property of the NIZKPK , the output of the extractor $\langle C, \tau \rangle$ satisfies in this case $(\langle m, v, 1^l \rangle, \langle C, \tau \rangle) \in R$ with probability at least $1 - \frac{\epsilon}{2}$. In this case, $C(m) = 1$ and $\textsc{Verify}'(v, C, \tau) = 1$. We claim that it can't be the case that $C = C_i$ for some i. This is because if this was the case then we would have that $C_i(m) = 1$ and so $m \in L(C_i) \subseteq \mathcal{M}$. Therefore we see that with overall probability $\frac{\epsilon}{2}$ we've actually succeeded in breaking the standard signature scheme ($\textsc{Gen}'$,$\textsc{Sign}'$,$\textsc{Verify}'$).

Note that in order to prove this claim we did not need to use the zero knowledge property of the NIZKPK, but rather only its proof of knowledge property. □

A.2 Proof of Indistinguishable Signatures

Let A be an oracle algorithm. Recall the experiment described in property 3 (Indistinguishable signatures) of Definition 2.1 (We copy it here for the reader's convenience):

Experiment 2:

1. Generate the keys: $(s_{\mathsf{master}}, v_{\mathsf{master}}) \leftarrow \textsc{KeyGen}(1^n)$
2. Run A with oracle to the delegating algorithm w.r.t. s_{master} (i.e. run A with oracle to $\textsc{Delegate}(s_{\mathsf{master}}, \cdot)$).
3. At this stage A outputs is (m, C_1, C_2) where $|C_1| = |C_2|$ and $C_1(m) = C_2(m) = 1$.
4. Choose $b \leftarrow_{\mathrm{R}} \{1, 2\}$.
5. Generate a restricted signing key s_{C_b}: Let $s_{C_b} \leftarrow \textsc{Delegate}(s_{\mathsf{master}}, C_b)$.
6. Provide A with $\textsc{Sign}(s_{C_b}, m)$.
7. Let $b' \in \{0, 1\}$ be the final output of A.

To prove this property we need to prove the following claim:

Claim A.2. *The probability that $b' = b$ is at most $\frac{1}{2} + negl(n)$.*

Proof. Suppose that A succeeds with probability q. Recall the definition of the signature algorithm $\textsc{Sign}$:

$$\textsc{Sign}(\underbrace{C \circ \tau \circ \Sigma}_{s_C}, m) \stackrel{def}{=} \underbrace{\textsc{Prove}(\Sigma, \langle m, v, 1^{|C|} \rangle, \langle C, \tau \rangle) \circ 1^{|C|}}_{\alpha}$$

Suppose now that we replace the proof with a *simulated proof* of the same statement. That is we let

$$\textsc{Sign}'_{aux_S}(\underbrace{C \circ \tau \circ \Sigma}_{s_C}, m) \stackrel{def}{=} \underbrace{\textsc{Sim}(\Sigma, \langle m, v, 1^{|C|} \rangle, aux_S) \circ 1^{|C|}}_{\alpha}$$

By the zero knowledge property of the NIZKPK, no efficient algorithm could distinguish between $\textsc{Sign}$ and $\textsc{Sign}'$. Let Experiment 2' be the same as Experiment 2 with the one difference that in step 6 we replace SIGN with SIGN$'$ and let p be the probability that $b = b'$ in this experiment. As noted above, the zero knowledge property of the NIZKPK implies that

$$|p - q| \leq negl(n)$$

.

But, as SIGN$'$ is *independent* of the circuit C it gets as input and only depends on C's size, in Experiment 2' the algorithm A *gets no information* about circuit C_n that was chosen. Therefore we get that $p \leq \frac{1}{2}$. It follows that $q \leq \frac{1}{2} + negl(n)$. □

B Extensions

One can extend the basic definition of delegateable signatures to achieve some extra properties, all of which can be obtained through simple modifications to our construction. We first describe here the possible extensions. After each extension we provide a rough sketch of the modifications that need to be made to our construction to achieve it. Some of these extensions are useful for the application presented in Section 4.

Removing the dependence on the circuit size. One troubling property of our construction is that the size of the restricted key s_C and the size of a signature made using s_C have a polynomial dependence on the size of C. It is possible to change the size of a signature made using s_C to have only a polylogarithmic dependence on the size of C (along with a polynomial dependence on the security parameter). However, we only know how to do this if make our signature scheme an *interactive* one. That is, the signing process involves an interaction with the verifier. By doing this we lose the property of signature *transferability*, as the verifier can not proved that the transcript of its interaction with the signer was generated properly. Independently, we can reduce the dependence of $|s_C|$ on $|C|$ if the circuit C has a shorter representation. For instance it may happen that a circuit C of size n^3 is actually the result of performing the standard reduction on a Turing machine of size n and running time n^3.

> **Sketch of modifications needed to the construction:** This can be done using computationally sound (CS) proofs, defined and constructed by Kilian ([**?**]) and Micali ([**?**]). CS proofs enable a prover to prove an **NP** statement with communication complexity that is polylogarithmic in the theorem and witness size and polynomial in the security parameter. As CS proofs systems are a constant-round Arthur-Merlin protocol, one can apply the Fiat-Shamir heuristic ([**?**]) on this protocol and obtain a *non-interactive* version. However, as this is only a heuristic, in this case there is no guarantee of security.

Sets of non-deterministic complexity. The construction allows to delegate the ability to sign on any set L of messages such that $L \in \mathbf{P}/Poly$ (a set $L \subseteq \{0,1\}^*$ is in $\mathbf{P}/Poly$ if there exists a polynomial sized circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that for any $x \in \{0,1\}^*$, $x \in L \iff C_{|x|}(x) = 1$). However, one may sometimes want to delegate signature ability of sets of higher complexity. In particular, it may be useful to be able to delegate the ability to sign on sets L that are in $\mathbf{NP}/Poly$, where $L \in \mathbf{NP}/Poly$ if there exists a polynomial sized two-input circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that for any $x \in \{0,1\}^*$, $x \in L$ iff $\exists w \in \{0,1\}^{p(|x|)}$ s.t. $C_{|x|}(x, w) = 1$. If for $x \in \{0,1\}^*$ such a string w exists, then w is called a *witness* for x. In view of this, we'll define a *non-deterministic* circuit to be a circuit C that takes two strings as inputs. If C is a non-deterministic circuit then we define $L(C) \stackrel{def}{=} \{x | \exists w \in \{0,1\}^* \text{ s.t. } C(x, w) = 1\}$.

A delegateable signature scheme can be extended to allowing delegating of signatures for messages in $L(C)$, where C is a *non-deterministic* circuit. In this case we'll modify the signing algorithm and it will take an additional parameter w which would be the *witness* that proves that indeed the message m that is to be signed is in $L(C)$ (i.e. the signing algorithm will be invoked in the following way: $\textsc{Sign}(s_C, m, w)$ where $C(m, w) = 1$).

> **Sketch of modifications needed to the construction:** In the current construction, when signing on a message m with a restricted key $s_C = C \circ \tau \circ v \circ \Sigma$, we prove knowledge of C, τ such that $C(m) = 1$ and $\textsc{Verify}'(v, C, \tau) = 1$ (recall that ($\textsc{KeyGen}'$,$\textsc{Sign}'$,$\textsc{Verify}'$) is the standard (non-delegateable) signature scheme we use as a component). In the modified construction we'll prove knowledge of C, τ, w such that $C(m, w) = 1$ and $\textsc{Verify}'(v, C, \tau) = 1$.

Restricting further delegation In some applications, one may want to discourage the party that received a restricted signing key s_C from sharing this key with others. One way to achieve this is the following. Suppose that each party p has in a directory some public information PK_p such that only p knows the corresponding secret information SK_p (one may think of PK_p as $f(SK_p)$ where f is a one-way permutation). It is possible to change the delegation mechanism so the delegation algorithm $\textsc{Delegate}$ will get as additional input the public information PK_p. It will then generate a restricted signing key s_{C,PK_p}. The additional property of s_{C,PK_p} would be that only parties that know the corresponding secret SK_p will be able to use it to sign messages. We stress that the delegation algorithm *need not* get SK_p as input. If this change was made then a party p that received a restricted key s_{C,PK_p} will not be able to share it with another party without disclosing its own private information SK_p. We stress that we still want the indistinguishable signatures property to hold. In particular this means that a signature on a message m using s_{C,PK_p} *should not* reveal the identity of p and so it should not reveal the information PK_p.

> **Sketch of modifications needed to the construction:** When creating a restricted key $s_{C,PK}$, instead of just signing on C we'll sign on the pair $\langle C, PK\rangle$. When signing on a message m we'll prove knowledge of C, PK, SK, τ such that $C(m) = 1$, $\textsc{Verify}'(v, \langle C, PK\rangle, \tau) = 1$ and $PK = f(SK)$.

Indistinguishable signatures with respect to the delegating user. In some scenarios, one may want the indistinguishable signatures property to hold even against the party that generated the master signature and verification keys. We'll want to require that this holds even if this party deviates from the prescribed generator algorithm when generating the master keys. The current definition does not imply this property, and indeed in the current construction, if the generator of the master keys retains the auxiliary information aux_S, he or she can use this information to extract from any signature using s_C the circuit C. Our construction can be modified to achieve this property of indistinguishable signatures w.r.t. the delegating user, but in order to do that we'll need to add a public reference string R. This string R will be generated once and for all by a trusted party. After that, any party can generate master keys w.r.t. the string R. It will hold that even the party that generated the the master keys is unable to distinguish between two signatures.

> **Sketch of modifications needed to the construction:** The public reference string would be the reference string of the NIZKPK, Σ. Note that the same reference string can be used for many signatures. The indistinguishable signatures property holds against any party that does not know the auxiliary information aux_S, aux_E.

More levels of delegation. One may want to allow a party that has received a restricted signing key s_C to delegate further to another party the ability to sign on messages that are in some subset of $L(C)$.

Sketch of modifications needed to the construction: Recall that in our construction $v_{\mathsf{master}} = v \circ \Sigma$. Denote $v_0 \stackrel{def}{=} v$. When creating a restricted signature key s_{C_1} for some circuit C_1, the delegation algorithm will generate new signature keys $(s_1, v_1) \leftarrow \textsc{KeyGen}'(1^n)$. The restricted key s_{C_1} will contain both keys s_1, v_1 along with a signature on the pair $\langle C_1, v_1\rangle$ (again, it will also contain $v_0 \circ \Sigma$). At the first level of delegation to sign a message m one will prove knowledge of C_1, v_1, τ_0, τ_1 such that

1. $C_1(m) = 1$
2. $\textsc{Verify}'(v_0, \langle C_1, v_1\rangle, \tau_0) = 1$
3. $\textsc{Verify}'(v_1, m, \tau_1) = 1$

Let C_2 be another circuit. If the holder of $s_{C_1} = s_1 \circ v_1 \circ \tau \circ v_0 \circ \Sigma$ wishes to delegate the ability to sign on messages in $L(C_1) \cap L(C_2)$ then it will generate new keys $(s_2, v_2) \leftarrow \textsc{KeyGen}'(1^n)$. It will then provide a signature on $\langle v_2, C_2\rangle$ using s_1.

In general, in order to sign a message m, one will prove knowledge of $C_1, \ldots, C_k$,$v_1, \ldots, v_k$, $\tau_0, \ldots, \tau_k$ such that

1. $C_1(m) = C_2(m) = \cdots = C_k(m) = 1$
2. For all $1 \le i \le k$, $\textsc{Verify}'(v_{i-1}, \langle C_i, v_i\rangle, \tau_{i-1}) = 1$
3. $\textsc{Verify}'(v_k, m, \tau_k) = 1$

A technical requirement would be that the signature will include the total sizes of all circuits involved (in unary).

C Non-interactive Zero-Knowledge Proof of Knowledge

Following is the definition of (adaptive multi-theorem) non-interactive zero knowledge proof systems.

Definition C.1. Let R be an **NP** relation.

A quadruple of algorithms ($\textsc{Gen}$,$\textsc{Prove}$, $\textsc{VerProof}$, $\textsc{Sim}$) is called a *adaptive multi-theorem non-interactive zero knowledge proof system* (NIZK) for $L = L(R)$ if it satisfies the following properties:

1. (Perfect Completeness) For any reference string Σ generated by the generator ,and any $(x, w) \in R$, the output of $\textsc{Prove}(\Sigma, x, w)$ must be a valid proof that $x \in L$. That is, we require that for any $(x, w) \in R$,
$$\Pr_{(\Sigma, aux_S)\leftarrow \textsc{Gen}(1^n)}[\textsc{VerProof}(\Sigma, x, \textsc{Prove}(\Sigma, x, w))] = 1 = 1$$
2. (Adaptive Computational Soundness) For any *polynomial* non-uniform adversary A we require that with only negligible probability, given Σ, A can find a false theorem x (i.e. $x \notin L$) and a proof π for x that passes verification with respect to Σ. Formally, we require that
$$\Pr_{(\Sigma, aux_S)\leftarrow \textsc{Gen}(1^n)}[(x, \pi) \leftarrow A(\Sigma); \textsc{VerProof}(\Sigma, x, \pi) = 1 \wedge x \notin L] = negl(n)$$

3. (Adaptive multi-theorem zero knowledge) No polynomial non-uniform adversary A, can distinguish between oracle access to the prover and oracle access to the simulator. Formally we require that the results of the following two experiments are computationally indistinguishable:

   Experiment 1: Running A with oracle to prover:

   (a) Generate reference string : Let $(\Sigma, aux_S) \leftarrow \textsc{Gen}(1^n)$.

   (b) Run the adversary A with oracle to the prover (i.e. with oracle to $\textsc{Prove}(\Sigma, \cdot, \cdot)$.)

   (c) The result of the experiment is the output of A.

   Experiment 2: Running A with oracle to simulator:

   (a) Generate reference string: Let $(\Sigma, aux_S) \leftarrow \textsc{Gen}(1^n)$.

   (b) Run the adversary A with oracle to the simulator. That is, we run A with oracle to the following algorithm $S_{\Sigma,aux_S}(\cdot, \cdot)$: $S_{\Sigma,aux_S}(x, w) \stackrel{def}{=} \textsc{Sim}(\Sigma, x, aux_S)$.

   (c) The result of the experiment is the output of A.

   In both experiments we can assume without loss of generality that A's queries to the oracle are limited to (x, w) such that $(x, w) \in R$ (otherwise the oracle can answer with $\perp$).

   > **Comment:** As remarked above, we have chosen one of several possible variants for defining NIZK. In particular, we have chosen to require *computational soundness*, and to require the adversary that tries to distinguish between the simulator and the prover to supply a witness with each query it makes. We've also chosen to require that the same generator algorithm Gen is used in the actual proof and the simulated proof (with the only difference being that in the actual proof we discard the auxiliary information aux_S). These choices were made for simplifying the presentation and because they provide sufficient security for our application.

Definition C.2. Let R be an **NP** relation. A tuple of algorithms (Gen, Prove , VerProof , Sim , Ext) is called an *(adaptive multi-theorem) non-interactive zero knowledge proof of knowledge* (NIZKPK) for R if the following holds:

1. Gen is an algorithm that on input 1^n outputs three strings (Σ, aux_S, aux_E).

2. Let $\textsc{Gen}'$ be an algorithm that on input 1^n runs $\textsc{Gen}(1^n)$ and discards Gen's third output (i.e., $\textsc{Gen}'(1^n) \stackrel{def}{=} (\Sigma, aux_S)$ where $(\Sigma, aux_S, aux_E) \leftarrow \textsc{Gen}(1^n)$), then (Gen,Prove,VerProof,Sim) is an (adaptive multi-theorem) non-interactive zero-knowledge proof (NIZK) for $L(R)$.

3. For any polynomial non-uniform adversary A we consider the following experiment:

   (a) Generate reference string: Let (Σ, aux_S, aux_E).

   (b) Let $(x, \pi) \leftarrow A(\Sigma)$.

   (c) Let $w \leftarrow \textsc{Ext}(\Sigma, x, \pi, aux_E)$.

   We say that Ext is *successful* if either $\textsc{VerProof}(\Sigma, x, \pi) = 0$ or $(x, w) \in R$. We require that the probability that Ext is successful is at least $1 - negl(n)$.

C.1 Construction of Non-interactive Zero Knowledge Proofs of Knowledge

Let R be an **NP** relation. We can assume w.l.o.g that there exists a polynomial $p(\cdot)$ such that $(x, w) \in R$ implies that $|w| = p(|x|)$ (otherwise we build a proof for a 'padded' relation R). Our construction of a NIZKPK for R uses two components: a standard NIZK ($\textsc{Gen}'$,$\textsc{Prove}'$,$\textsc{VerProof}'$,$\textsc{Sim}'$) for an **NP** language L that we shall specify later and a public key encryption scheme ($\textsc{KeyGen}$,$\textsc{Enc}$,$\textsc{Dec}$). Recall that a NIZKPK is composed of 5 components: ($\textsc{Gen}$,$\textsc{Prove}$,$\textsc{VerProof}$,$\textsc{Sim}$,$\textsc{Ext}$).

Construction C.3. *A NIZKPK for R*

- The output of the generator $\textsc{Gen}$ on input 1^n will be $(\Sigma' \circ e, aux'_S \circ e, d)$ where $(\Sigma', aux'_S) \leftarrow \textsc{Gen}'(1^n)$ and $(d, e) \leftarrow \textsc{KeyGen}(1^n)$. The notation will be

$$(\underbrace{\Sigma' \circ e}_{\Sigma}, \underbrace{aux'_S \circ e}_{aux_S}, \underbrace{d}_{aux_E}) \leftarrow \textsc{Gen}(1^n)$$

- Recall that for $x \in \{0,1\}^*$ the prover needs to prove knowledge of a string w such that $(x, w) \in R$. To do this the prover will encrypt w using e and so obtain $y \leftarrow \textsc{Enc}(e, w; r)$ (where r is the randomness used for the encryption). The proof will be $y \circ \pi'$, where π' would be a proof in the standard NIZK system that there exist w and r such that $(x, y) \in R$ and $y = \textsc{Enc}(e, w; r)$. Formally, we define L to be the following **NP** language:

$$L \stackrel{def}{=} \{\langle e, x, y\rangle | \exists w, r \in \{0,1\}^* \text{ s.t. } y = \textsc{Enc}(e, w; r) \wedge (x, w) \in R\}$$

  We assume that the standard NIZK system ($\textsc{Gen}'$,$\textsc{Prove}'$,$\textsc{VerProof}'$,$\textsc{Sim}'$) is a NIZK for the language L. On input $(x, w) \in R$ the prover $\textsc{Prove}$ will choose a uniform string r for the encryption, let $y \stackrel{def}{=} \textsc{Enc}(e, w; r)$ and $\pi' \leftarrow \textsc{Prove}'(\Sigma', \langle e, x, y\rangle, \langle w, r\rangle)$. The output of $\textsc{Prove}$ will be $y \circ \pi'$.

- The verifier $\textsc{VerProof}$, on input $\pi = y \circ \pi'$ and with reference string $\Sigma = \Sigma' \circ e$ will output 1 if and only if $\textsc{VerProof}'(\Sigma, \langle e, x, y\rangle, \pi') = 1$.

- The simulator $\textsc{Sim}$ on input a theorem x and with auxiliary information $aux'_S \circ e$ will generate $y \leftarrow \textsc{Enc}(e, 0^{p(|x|)})$ and let $\pi' \leftarrow \textsc{Sim}'(aux'_S, \langle e, x, y\rangle)$. The simulator's output will be $y \circ \pi'$.

- The extractor $\textsc{Ext}$, on input a proof $y \circ \pi'$ and with auxiliary input $aux_E = d$ will output $\textsc{Dec}(d, y)$.

D Boolean Circuits

We review the standard definition of boolean circuits. A *boolean circuit* C with n inputs and k outputs is a directed acyclic graph with each node labeled with one of the following labels $\wedge, \vee, \neg$ $, in_1, \ldots, in_n$, $out_1, \ldots, out_k$. We require that for any of the labels $in_1, \ldots, in_n$, $out_1, \ldots, out_k$ there will be exactly one node labeled with this label. We call a node labeled in_i for some $1 \leq i \leq n$ an *input node*. We call a node labeled out_j for $1 \leq j \leq k$ an *output node*. We also require that nodes labeled in_i for $1 \leq i \leq n$ will have in-degree 0, and that these will be the only nodes with in-degree 0. We require that nodes labeled out_j for $1 \leq j \leq k$ will have out-degree 0 and in degree 1, nodes labeled $\wedge$ or $\vee$ will have in-degree 2 and nodes labeled $\neg$ will have in-degree 1. A circuit C can be *applied* to an input $x = x_1 \ldots x_n$ where $x_i \in \{0,1\}$ for $1 \leq i \leq n$ in the following way: we give a boolean value $h(u)$ to each node u. At first we give the value x_i to the node labeled in_i for $1 \leq i \leq n$. Then we perform the following process iteratively while there exists a node u without a value but all its predecessors in the graph have a value do:

1. If u is an output node, then it has only one predecessor u', and we let $h(u) \leftarrow h(u')$.
2. If u is labeled $\wedge$ then it has two predecessors u_1, u_2 and we let $h(u) = 1$ if both $h(u_1)$ and $h(u_2)$ are equal to 1. Otherwise we let $h(u) = 0$.
3. If u is labeled $\vee$ then it has two predecessors u_1, u_2 and we let $h(u) = 0$ if both $h(u_1)$ and $h(u_2)$ are equal to 0. Otherwise we let $h(u) = 1$.
4. If u is labeled with $\neg$ then it has one predecessor u' we let $h(u) = 1 - h(u')$.

It can be shown that when this process terminates, all output nodes have a value. The *output* of a circuit C with n inputs and k outputs on input $x \in \{0,1\}^n$ is the string $y = y_1 \dots y_k$ (where $y_j \in \{0,1\}$) such that y_j is the value of the node labeled with out_j after performing this process. This string y is denoted by $C(x)$.

E Standard Digital Signatures

We review the standard definition of security against existential forgery for signature schemes. Our definition is taken from [**?**], Section 6.1.

Definition E.1. A *signature scheme* is a triple of algorithms (KeyGen,Sign,Verify) that satisfies the following properties:

1. On input 1^n, the algorithm Gen outputs a pair of strings (s, v).
2. For every pair (s, v) in the range of $G(1^n)$ and for every $m \in \{0,1\}^*$, algorithms Sign and Verify satisfy Verify$(v, m, \text{Sign}(s, m)) = 1$.
3. For every polynomial non-uniform algorithm A we consider the following experiment:

   (a) generate the keys $(s, v) \leftarrow$ KeyGen(1^n).

   (b) Run A with oracle to the signing algorithm w.r.t. the signing key s. That is run $A^{\text{Sign}(s,\cdot)}$.

   Let $m_1, \dots, m_k$ be the queries that A asked from its oracle and let (m, α) be A's output. We say that A is *successful* if $m \neq m_i$ for all $1 \leq i \leq k$ and Verify$(v, m, \alpha) = 1$. The probability that A is successful is negligible.

F Delegateable Signatures are Secure Against a Stronger Attack

Lemma F.1. *Let (*KeyGen,Delegate,Sign,Verify*) be a delegateable signature scheme.Let A be a two-oracle algorithm. If we perform the following experiment:*

1. *generate the keys $(s_{\mathsf{master}}, v_{\mathsf{master}}) \leftarrow$* KeyGen$(1^n)$.
2. *Run A with the following two oracles:*

   (a) The delegation algorithm w.r.t. the key s_{master}. That is, the oracle Delegate$(s_{\mathsf{master}}, \cdot)$.

(b) A signing oracle that given any message m and circuit C such that $m \in L(C)$, produces a signature on m using s_C. That is the following oracle S :

$$S(m, C) \stackrel{def}{=} \begin{cases} \textsc{Sign}(\textsc{Delegate}(s_{\mathsf{master}}, C), m) & C(m) = 1 \\ \perp & \textit{otherwise} \end{cases}$$

We define $\mathcal{M} \stackrel{def}{=} L(C_1) \cup \ldots L(C_k) \cup \{m_1, \ldots, m_j\}$ where $s_{C_1}, \ldots, s_{C_k}$ are the restricted keys that A got from its first oracle and $\{m_1, \ldots, m_j\}$ are the messages A asked for signatures on from its second oracle. We let (m, τ) be the output of A. We say that A is successful *if* $\textsc{Verify}(v_{\mathsf{master}}, m, \tau) = 1$ *and $m \notin \mathcal{M}$.*

Then,

$$\Pr[A \textit{ is successful}] = \textit{negl}(n)$$

Proof Sketch: Suppose that we perform the experiment stated in the Lemma with one modification. We'll change the definition of the oracle S in the following way: on input a message m and a circuit C such that $C(m) = 1$ the oracle S will construct a circuit C' such that $|C'| = |C|$ and $L(C') = \{m\}$ and use $s_{C'}$ to sign on m. That is, we define

$$S(m, C) \stackrel{def}{=} \begin{cases} \textsc{Sign}(\textsc{Delegate}(s_{\mathsf{master}}, C'), m) & C(m) = 1 \\ \perp & \text{otherwise} \end{cases}$$

where C' is a circuit that satisfies $|C'| = |C|$ and $L(C') = \{m\}$.

We claim that the probability that A succeeds in the modified experiment is equal (up to a negligible factor) to its probability of success in the original experiment. Indeed, any non-negligible difference could be used to construct an adversary that would contradict Property 3 (Indistinguishable Signatures) of the delegateable signature scheme.

However, the probability of success in the modified experiment must be negligible because otherwise we can use A to contradict Property 2 (Security against existential forgery). Indeed we can construct an adversary A' that will simulate A using the oracle to $\textsc{Delegate}(s_{\mathsf{master}}, \cdot)$ alone. Let $s_{C_1}, \ldots, s_{C_k}$ be the restricted keys that A gets from its first oracle, and let $m_1, \ldots, m_j$ be the messages that A gets signatures on from its second oracle. The restricted keys that A' will get from its oracle will be $s_{C_1}, \ldots, s_{C_k}, s_{C'_1}, \ldots, s_{C'_j}$ where $L(C'_i) = \{m_i\}$ for any $1 \le i \le j$. Therefore, if A manages to output a valid signature on a message $m \notin L(C_1) \cup \cdots \cup L(C_k) \cup \{m_1, \ldots, m_j\}$, then $m \notin L(C_1) \cup \cdots \cup L(C_k) \cup L(C'_1) \cup L(C'_j)$. Therefore, any success of A is a success of A', and so A' has the same probability of success as A.

$\square$