# Proxy Signatures Secure Against Proxy Key Exposure

Jacob C. N. Schuldt[1], Kanta Matsuura[1], and Kenneth G. Paterson[2] *

[1] Institute of Industrial Science, University of Tokyo,
4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan.
{schuldt, kanta}@iis.u-tokyo.ac.jp
[2] Information Security Group,
Royal Holloway, University of London,
Egham, Surrey, TW20 0EX, UK
kenny.paterson@rhul.ac.uk

**Abstract.** We provide an enhanced security model for proxy signatures that captures a more realistic set of attacks than previous models of Boldyreva *et al.* and of Malkin *et al.*. Our model is motivated by concrete attacks on existing schemes in scenarios in which proxy signatures are likely to be used. We provide a generic construction for proxy signatures secure in our enhanced model using sequential aggregate signatures; our construction provides a benchmark by which future specific constructions may be judged. Finally, we consider the extension of our model and constructions to the identity-based setting.

**Keywords:** proxy signatures, provable security.

## 1 Introduction

A proxy signature scheme allows an entity, the *delegator*, to delegate his signing capabilities to another entity, the *proxy*, which can then construct signatures on behalf of the delegator. A signature constructed by the proxy, called a *proxy signature*, will not only convince a verifier that the signature was indeed constructed by the proxy, but also that the proxy was delegated the signing rights of the delegator. In a *multi level* scheme, the proxy has the option of re-delegating the signing rights obtained from the delegator, to another proxy.

The concept of proxy signatures was first proposed by Mambo, Usuda and Okamoto in [24]. Among the ideas presented in [24], the concept of *delegation by warrant*, in which a signed warrant is used to describe the delegation, has received the most attention. Kim, Park and Won [16] expanded on this idea and suggested that a proxy key could be generated from the warrant. One of the main advantages of the use of warrants is that it is possible to include any type of security policy in the warrant to describe the restrictions under which the

* This author's research was supported by the European Commission under contract IST-2002- 507932 (ECRYPT).

delegation is valid. Most proxy signature schemes uses a variant of this approach and it is often expected that new proxy signature schemes will implement the functionality of warrants.

Since their introduction, many proxy signature schemes have been proposed (e.g. see [24, 27, 18, 19, 2, 35, 26]) and many extensions (e.g. see [32, 36, 29, 17, 30]) have been considered. However, the initial security notion introduced by Mambo, Usuda and Okamoto (slightly expanded by Lee, Kim and Kim [18]), was based on a list of security aims, and no security model in which schemes could be analysed was given. The lack of formal security definitions had a huge impact on the security of the initially proposed schemes. Many constructions were shown to be insecure, then fixed, only to be shown insecure again (e.g. see [24, 18, 19, 31]). This not only illustrates the need for well defined security models and a rigorous security analysis, but also indicates that the security of proxy signatures is more subtle than was initially assumed.

*Security models for proxy signatures.* Boldyreva, Palacio and Warinschi [2] were the first to introduce a proper security model for proxy signatures and to propose a provably secure proxy signature scheme. These results provided a significant improvement over previous treatments of proxy signatures in terms of security analysis and also highlighted security concerns with the trivial scheme in which the delegator signs the public key of the proxy and proxy signatures are constructed with the private key of the proxy. Malkin, Obana and Yung [22] later proposed an extended security model, allowing multi-level proxy signatures, and showed that proxy signatures are equivalent to key-insulated signatures [8]. However, if we consider the typical environments in which proxy signatures will be used, then these models do not capture all desired properties of proxy signatures. We expand on this next.

The use of warrants demands special attention in both the definition and security model of proxy signatures. If warrants are not explicitly modeled, it might be possible for an adversary to alter the warrant under which a proxy has made a signature on the delegator's behalf, even though the scheme has been proved secure. This is clearly an undesirable property, since users of proxy signatures should be able to rely on warrants not being mutable once a proxy signature has been created. Even though the schemes presented in [2] use warrants, these are not a part of the presented security model. However, the model presented in [22] rectifies this and explicitly models the warrants.

The security models of [2, 22] are both in the registered key model, meaning that the adversary is required to submit both the public and the private key of all users used in the security game, except the challenge user. Although this might be convenient when constructing proofs of security, it does not capture attacks where the adversary derives and registers a public key for which he cannot compute the corresponding private key. These types of attacks are also known from multi- and aggregate signatures (e.g. see [1, 5]), and are relevant in practice since users may not be required to prove knowledge of their private key when registering a public key (for example, due to efficiency concerns). Furthermore, the attacks seem to pose a real threat to some proxy signature schemes. As

an example of this, consider the construction proposed by Zhang, Safavi-Naini and Lin [35]. This construction efficiently combines the Boneh-Lynn-Shacham signature scheme [6] with the identity-based signature scheme of Hess [15] to create a proxy signature scheme. But the construction is insecure in a security model which does not use the registered key model. In this case, an adversary will be able to produce proxy signatures on behalf of a user, without that user having delegated his signing rights (details of this are given in Appendix A). This illustrates the need for a security model which can guarantee security of a proxy signature scheme when used in the more practical setting where the registered key model is not required.

Proxy signatures are often proposed for use in applications where signing is done in a potentially hostile environment. In this setting, it is assumed that secure storage is available for a long term key pair (e.g. key storage in a TPM within a laptop), but that it is not possible to perform all signature computations within the fully trusted device due to the number of signature requests or the amount of data that needs to be signed. Hence, these computations are performed on a less trusted device (e.g. by the operating system on a laptop which might become infected with malware). To limit the potential damage resulting from compromise of the less trusted device, a limited set of signing rights for the long term key pair can be delegated to this device, which can then act as a signing proxy. Thereby, only the limited proxy key is exposed in a compromise. However, this raises the concern that compromised proxy keys might somehow leak information about the long term key. This is relevant not only in the case where delegation is performed to protect a long term key, but also in the general case of delegation of signing rights from one entity to another. The security model of [2] does not model this possibility, since an adversary is not allowed to gain access to any proxy keys. The model of [22] has only limited support for proxy key exposure, since an adversary is only allowed access to proxy keys which a user has obtained by *self-delegation*, i.e. by delegating his signing rights to himself. However, this is not sufficient to guarantee security in an environment where any proxy key can potentially be exposed, and the assumption that only self-delegated proxy keys are at risk of being exposed seems unnatural and restrictive. Indeed, systems that rely on proxy key material (of any type) not revealing information about long term keys are already in use today (e.g. in applications such as the Grid Security Infrastructure [9]). So it is important to extend the adversarial capabilities to allow a richer set of proxy key exposures in order to correctly model the threats against these systems. However, if the adversary gains access to arbitrary proxy keys, many of the existing proxy signature schemes become insecure. In particular, the scheme proposed by Malkin *et al.* [22, 23] will be insecure, since private keys double as proxy keys in an ordinary delegation (i.e. a non-self-delegation). Schemes where this is not the case might also be vulnerable. For example, the triple Schnorr scheme whose security is analyzed by Boldyreva *et al.* [2] has the weakness that an adversary can compute the (long term) private key of a user upon exposure of a proxy key for a delegation procedure for which the adversary has a transcript (details of this attack

are given in Appendix B). Lastly, we emphasize that to model the compromise of a proxy correctly, the adversary should be given access to *all* information held by the proxy. For example, a closer look at [22, 23] reveals that the adversary there is given access to an oracle that returns a single self-delegated proxy key, but that the proxy information in the concrete scheme can potentially contain many keys (the scheme generates all keys used as self-delegated proxy keys in the initial key generation phase, and therefore should include any keys needed for further delegation in the proxy information). We argue that the approach taken in [22] is not sufficient to model the threat posed by a proxy compromise.

*Our contributions.* First and foremost, we define a refined security model for proxy signatures along with the security notion *Proxy Signature Unforgeability Under an Adaptive Chosen Message Attack with Proxy Key Exposure* (`ps-uf-pke`). In addition to more accurately capturing the threats against proxy signatures, we claim that our model and security notion are more direct and clear when compared to the model given in [22]. Hence they more easily allow proposed schemes to be proven secure. Our model is strictly stronger than the models of [2] and [22] in that our model allows an adversary to gain access to any proxy key and does not require the registered key model. Hence, a scheme secure in our model will also be secure in the models of [2] and [22], whereas the converse does not necessarily hold (in fact, as mentioned above, the schemes proposed in [2] and [22] will be insecure in our model).

We then present a simple generic construction for proxy signatures using sequential aggregate signatures. This is closely related to the delegation-by-certificate and aggregate-based constructions of [2], but our security proof is in our enhanced security model. We discuss how the construction can be instantiated (in the random oracle model) to give efficient proxy signature schemes with security relying on either the bilinear Diffie-Hellman assumption or the assumption that RSA is a claw-free permutation. We also discuss how a scheme secure in the standard model can be obtained.

Lastly, we sketch how to extend our security model to the identity-based setting and give a fairly simple generic construction that is secure in the extended model. We also discuss the possibilities for instantiating this construction.

Since our constructions are relatively simple and easy to prove secure, they provide a performance benchmark, both in terms of security and efficiency, for any new proxy signature schemes.

## 2 Preliminaries

*Notation.* Let $PK = (pk_1, \ldots, pk_n)$ be a list of public keys (or any other strings). We use the notation $PK_{i\ldots j}$ with $i \leq j$ to indicate the sublist of keys from the $i$-th key to the $j$-th key in $PK$, e.g. $PK_{2\ldots 4} = (pk_2, pk_3, pk_4)$. By $PK.(pk_{n+1})$ we mean that the key $pk_{n+1}$ is appended to the end of $PK$. Lastly we will use the notation $m_1||m_2$ to mean the concatenation of the strings $m_1$ and $m_2$. When elements that are not strings appear in a concatenation, we will assume that they will be encoded as a string before the actual concatenation takes place.

*Signature schemes.* We briefly recall the definitions of an ordinary signature scheme and a sequential aggregate signature scheme.

A signature scheme, $\mathcal{S}$, is given by the following algorithms:

- `Setup` which on input a security parameter $1^k$ generates a set of global system parameters `params`. We assume that `params` are made publicly available and will not write `params` as an explicit argument to the functions defined below.
- `KeyGen` which generates a public/private key pair $(pk, sk)$.
- `Sign` which on input $(sk, m)$, where $m$ is a message to be signed, generates a signature $\sigma$ on $m$.
- `Verify` which on input $(pk, m, \sigma)$, outputs either `accept` or `reject`.

A signature scheme is said to be *sound* if for all $(pk, sk) \leftarrow$ `KeyGen` and all messages $m$, we have that

$$\Pr[\texttt{Verify}(pk, m, \texttt{Sign}(sk, m)) = \texttt{accept}] = 1$$

where the probability is taken over all random coin tosses made in the `KeyGen`, `Sign` and `Verify` algorithms. A signature $\sigma$ is said to be *valid* on $m$ under public key $pk$ if $\texttt{Verify}(pk, m, \sigma) = \texttt{accept}$.

The standard notion of security for signature schemes is *Existential Unforgeability under a Chosen Message Attack* (euf-cma) [11].

A sequential aggregate signature scheme, $\mathcal{AS}$, is given by the following algorithms:

- `Setup` and `KeyGen` which are similar to the corresponding algorithms of a ordinary signature scheme.
- `AggSign` which takes as input $(sk, m, \sigma_{agg})$, where $sk$ is a private key, $m$ is a message to be signed and $\sigma_{agg}$ is a sequential aggregate signature on messages $(m_1, \ldots, m_n)$ under public keys $(pk_1, \ldots, pk_n)$, constructed by previous calls to `AggSign`. The output of `AggSign` is a sequential aggregate signature $\sigma'_{agg}$ on messages $(m_1, \ldots, m_n, m)$ under public keys $(pk_1, \ldots, pk_n, pk)$ where $pk$ is the public key corresponding to $sk$. Note that we can construct an ordinary signature scheme by using an "empty" sequential aggregate signature as part of the input to `AggSign`.
- `AggVerify` takes as input $((pk_1, \ldots, pk_n), (m_1, \ldots, m_n), \sigma_{agg})$ and outputs `accept` or `reject`.

A sequential aggregate signature scheme is said to be *sound* if for all $n \geq 1$, all $(pk_i, sk_i) \leftarrow$ `KeyGen` $\;\; i \in \{1, \ldots, n\}$, all messages $(m_1, \ldots, m_n)$ and all sequential aggregated signatures constructed as $\sigma_i \leftarrow \texttt{AggSign}(sk_i, m_i, \sigma_{i-1}), i \in \{1, \ldots, n\}$ with $\sigma_0 = \emptyset$, we have that

$$\Pr[\texttt{AggVerify}((pk_1, \ldots, pk_n), (m_1, \ldots, m_n), \sigma_n) = \texttt{accept}] = 1$$

where the probability is taken over all random coin tosses used in the `KeyGen`, `AggSign` and `AggVerify` algorithms. Validity of sequential aggregate signatures is defined as one would expect.

There exist two different security notions for sequential aggregate signatures, introduced by Lysyanskaya *et al.* [21] and Lu *et al.* [20], respectively. The difference between the two notions is that the latter requires the registered key model whereas the former does not. In this paper we will insist that the registered key model is not required and use the notion defined in [21], referred to as *Existential Unforgeability in the Sequential Aggregate Chosen-Key Model*.

## 3 Proxy Signatures

Before formally defining a proxy signature scheme, we will briefly discuss a few basic assumptions and the format of a proxy signature.

We will assume that users can be uniquely identified by their public keys. So a delegation chain consisting of an original delegator and a number of proxies will be uniquely identified by an ordered list of their public keys. This requirement can be met in practice by requiring the certification authority not to issue certificates for two different users on the same public key. This simple expedient is much simpler to realise than relying on proofs of knowledge (that are implicit in the registered key model).

A proxy signature scheme is required to implement a *proxy identification* algorithm, which, when given a valid proxy signature, outputs the identities (i.e. public keys) of the proxies in the delegation chain. Since we require this function to be publicly available (i.e. no secret information is required to run the algorithm), we have chosen to explicitly include a list $PK$ of the public keys in the proxy signature itself. This does not represent a restriction, since the requirement of a public identification algorithm forces the keys to be part of a proxy signature anyway. For simplicity, we will also require the original delegator to add his public key to $PK$, making a proxy signature "self-verifiable", i.e. only the signature and a message is required for verification.

It will also be required that a proxy signature contains a list of warrants $W$ for the delegation chain. It is common not to specify the format of warrants since a concrete security policy might depend on the particular usage of the proxy signatures. However, it is also common to assume that some information about the delegation is a part of the warrant to prevent trivial attacks against the scheme. We consider the combination of these two assumptions to be bad practice and suggest that the definition of a proxy scheme should explicitly include all elements which are required for the scheme to be secure. This will help prevent implementation flaws from the use of non-standard or perhaps empty warrants.

A multi-level proxy signature scheme is an extension of an ordinary signature scheme $\mathcal{S} = \{\texttt{Setup}, \texttt{Keygen}, \texttt{Sign}, \texttt{Verify}\}$ with the following additional algorithms:

- (`Delegate`, `ProxyKeyGen`) which is a pair of randomized interactive algorithms for delegation of signing rights.
  - `Delegate` is run by the delegator with input $(PK, W, pk_d, pk_p, sk, w)$, where $PK$ and $W$ are lists of (public keys of) previous delegators and

previous warrants in the delegation chain, $pk_d$ and $pk_p$ are the public keys of the delegator and the proxy, $sk$ is the private key for which signing rights are delegated, and $w$ is the warrant for the current delegation. If the delegator is delegating his own signing rights (i.e. the lists $PK$ and $W$ are empty), we will set $sk = sk_d$ where $sk_d$ is the private key of the delegator itself. However, if the delegator is delegating signing rights for a proxy key $psk$ he has obtained playing the role of a proxy in a previous delegation, we will set $sk = psk$. `Delegate` will interact with `ProxyKeyGen` to perform the delegation, but will have no local output.

- `ProxyKeyGen` is run by the proxy and takes as input $(pk_d, pk_p, sk_p)$ where $pk_d$ is the public key of the delegator and $(pk_p, sk_p)$ is the public/private key pair of the proxy. Upon completion of the interaction with `Delegate`, `ProxyKeyGen` returns the local output $(PK', W', psk)$, where $PK'$ and $W'$ are lists of public keys of the delegators and warrants in the delegation chain, extended with the public key of the proxy and the warrant of the current delegation, and $psk$ is a private proxy key which can be used to create proxy signatures on behalf of the delegator.

– `ProxySign` is run by the proxy and takes as input $(PK, W, psk, m)$ where $PK$ and $W$ are the delegators and warrants in the delegation chain, $psk$ is a proxy key and $m$ is a message to be signed. The output of `ProxySign` is a proxy signature $(PK, W, p\sigma)$ where $p\sigma$ is a signature on the message $m$ created with the proxy key $psk$. We say that the proxy signature is generated by the proxy on behalf of the delegator.

– `ProxyVerify` is run by the verifier and takes as input $(m, (PK, W, p\sigma))$ where $m$ is a message and $(PK, W, p\sigma)$ is a proxy signature as generated by the `ProxySign` algorithm. The output of `ProxyVerify` is either `accept` or `reject`. Note that `ProxyVerify` does not take any public keys as input since these are assumed to be part of $PK$ in the proxy signature itself.

Note that a properly generated proxy signature will have one more element in $PK$ than in $W$ since no warrant will be added by the signing proxy. From the explicit inclusion of both $PK$ and $W$ in the proxy signature, it is clear that the public keys of the delegators and the warrants in the delegation chain can be extracted from a proxy signature. Hence, there is no need to define functions which provides this functionality.

The above definition can be seen as a multi-level extension of the definition given in [2], but with explicit modeling of warrants. Compared to the definition given in [22], there are only minor differences which do not impact the functionality of the scheme.

*Notation for delegation.* To make the notation more clear, we will write

$$(PK', W', psk) \leftarrow \begin{bmatrix} \texttt{Delegate}(PK, W, pk_d, pk_p, sk, w); \\ \texttt{ProxyKeyGen}(pk_d, pk_p, sk_p); \end{bmatrix}$$

for the interaction between the algorithms `Delegate` and `ProxyKeyGen` with the inputs $(PK, W, pk_d, pk_p, sk, w)$ and $(pk_d, pk_p, sk_p)$ respectively, and let $psk$ be the proxy key output by `ProxyKeyGen`.

*Soundness.* We say that a proxy signature scheme is sound if, firstly, the basic signature scheme $\mathcal{S}$ is sound, and secondly, for all $n \geq 1$, for all possible delegation chains of users with public/private key pairs and proxy keys generated as

$$(pk_i, sk_i) \leftarrow \texttt{KeyGen} \quad \text{for } i \in \{1, \ldots, n\}, \qquad psk_1 \leftarrow sk_1 \quad \text{and}$$
$$(PK_i,\, W_i, psk_i) \leftarrow \begin{bmatrix} \texttt{Delegate}(PK_{i-1},\, W_{i-1}, pk_{i-1}, pk_i, psk_{i-1}, w_{i-1}); \\ \texttt{ProxyKeyGen}(pk_{i-1}, pk_i, sk_i); \end{bmatrix}$$
$$\text{for } i \in \{2, \ldots\, n\},$$

and all messages $m$ satisfying the warrants $W_n = (w_1, \ldots, w_{n-1})$, we have that

$$\Pr\left[\texttt{ProxyVerify}(m, \texttt{ProxySign}(PK_n,\, W_n, psk_n, m)) = \texttt{accept}\right] = 1,$$

where the above probability is taken over all random coins used by the `KeyGen`, `Delegate`, `ProxyKeyGen` and `ProxySign` algorithms.

## 4 Security Model

We define the security notion *Existential Unforgeability under an Adaptive Chosen Message Attack with Proxy Key Exposure* (`ps-uf-pke`) for multi-level proxy signature schemes. The security notion is based on the security game defined below, played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We first introduce some notation and features of the security model, and then give formal definitions.

In the game, $\mathcal{A}$ will control all but a single user, $u^*$, whose public/private key pair $(pk^*, sk^*)$ will be generated by the challenger, and only $pk^*$ will be made available to $\mathcal{A}$. The public/private key pairs of all the other users will be generated by $\mathcal{A}$, and it will not be required of $\mathcal{A}$ to register generated keys or prove knowledge of the private keys corresponding to the public keys used in the game. This means that $\mathcal{A}$ is allowed to generate and use public keys for which he cannot compute the private key.

The goal of the adversary in the game is to produce a forgery. In this case, a forgery is one of the following: $(i)$ an ordinary signature which verifies under $u^*$'s public key, $(ii)$ a proxy signature that appears to be constructed by $u^*$ on behalf of one of the users controlled by the adversary, or $(iii)$ a proxy signature on behalf of $u^*$ that is computed by one of the users controlled by the adversary which has not been delegated the signing rights of $u^*$. We will of course have some requirements on the forgeries to exclude trivial cases, e.g. it is required for a type $(i)$ or type $(ii)$ forgery that the signature was not obtained in a query to the challenger. However, when considering a message/proxy signature pair $(m, (PK, W, p\sigma))$ produced by the adversary as a type $(ii)$ forgery, we will treat any query on a different $m$ or with a different $PK$ or $W$ list, as being unrelated. By this we mean that a forgery will be considered to be valid even if the adversary, for example, has received a proxy signature on the same message $m$ from the same delegation chain $PK$, but with a different set of warrants $W'$.

Lastly, in a type (*iii*) forgery we will allow the adversary to place $u^*$ anywhere in the delegation chain except as the last proxy, which would make the forgery a type (*ii*) forgery (i.e. we will not restrict $u^*$ to be the original delegator).

For convenience, during the game the challenger will maintain two sets of lists: $pskList(*, *)$ and $delList(*, *, *)$. Each list $pskList(PK, W)$ holds all proxy keys generated by $u^*$ in delegations from the delegation chain with the public keys in the list $PK$ and with the warrants in the list $W$. This list will be used by the challenger to respond to the various queries made by the adversary during the game. Each list $delList(PK, W, w)$ holds the public keys of users to whom $u^*$ has re-delegated the signing rights of one of the keys in $pskList(PK, W)$ with the warrant $w$. This list is only used to define valid type (*iii*) forgeries. If $u^*$ delegates the signing rights of his own private key under the warrant $w$, the public key of the proxy will be stored in $delList(\{\}, \{\}, w)$ using empty lists, $\{\}$, as the previous public key and warrant lists.

The security game is formally defined as follows:

**Setup** The challenger $\mathcal{C}$ runs `Setup` with input $1^k$ and generates the public/private key pair of $u^*$ by running $(pk^*, sk^*) \leftarrow$ `KeyGen`. $\mathcal{C}$ then passes $pk^*$ to the adversary $\mathcal{A}$ and stores $sk^*$.

**Queries** While $\mathcal{A}$ is running, it can adaptively make any of the following queries which are answered by $\mathcal{C}$:

1. *Ordinary signature.* On input $m$ from $\mathcal{A}$, $\mathcal{C}$ runs $\sigma \leftarrow \texttt{Sign}(sk^*, m)$ and returns $\sigma$ to $\mathcal{A}$.
2. *Delegation to $u^*$.* On input $pk_d$ from $\mathcal{A}$, $\mathcal{C}$ interacts with $\mathcal{A}$ through the delegation protocol by running $\texttt{ProxyKeyGen}(pk_d, pk^*, sk^*)$. Upon completion, $\mathcal{C}$ will obtain the proxy information $(PK', W', psk)$. If no $pskList(PK', W')$ list exists, $\mathcal{C}$ creates one and adds $psk$ to it. Otherwise, $\mathcal{C}$ just adds $psk$ to the existing $pskList(PK', W')$ list.
3. *Delegation from $u^*$.* For clarity, we will define an oracle for each of the three different types of delegation the adversary can request $u^*$ to perform:
   (a) *Delegation of $sk^*$.* On input $(pk_p, w)$ from $\mathcal{A}$, $\mathcal{C}$ interacts with $\mathcal{A}$ by running $\texttt{Delegate}(\{\}, \{\}, pk^*, pk_p, sk^*, w)$. Upon completion of the delegation protocol, $\mathcal{C}$ adds $pk_p$ to the list $delList(\{\}, \{\}, w)$.
   (b) *Re-delegation of psk.* On input $(PK, W, j, pk_p, w)$ where $j \in \mathbb{N}$, $\mathcal{C}$ looks up the $j$-th proxy key, $psk_j$, in $pskList(PK, W)$. If no such key exists, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ interacts with $\mathcal{A}$ by running $\texttt{Delegate}(PK, W, pk^*, pk_p, psk_j)$. When the delegation is complete, $\mathcal{C}$ adds $pk_p$ to $delList(PK, W, w)$.
   (c) *Self-delegation.* On input $(PK, W, j, w)$, $\mathcal{C}$ sets $sk = sk^*$ if $PK = W = \{\}$ and $j = 1$. Otherwise, $\mathcal{C}$ sets $sk$ to be the $j$-th proxy key in $pskList(PK, W)$ (if this proxy key does not exist, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$). Then $\mathcal{C}$ interacts with itself by running

$$(PK', W', psk) \leftarrow \begin{bmatrix} \texttt{Delegate}(PK, W, pk^*, pk^*, sk, w); \\ \texttt{ProxyKeyGen}(pk^*, pk^*, sk^*); \end{bmatrix}$$

When the delegation is complete, $\mathcal{C}$ adds $psk$ to $pskList(PK', W')$ and send the transcript of the delegation to $\mathcal{A}$.

4. *Proxy signature.* On input $(PK, W, j, m)$, $\mathcal{C}$ looks up the $j$-th proxy key, $psk_j$, in $pskList(PK, W)$ and returns $\perp$ to $\mathcal{A}$ if no such key exists. Otherwise, $\mathcal{C}$ computes $(PK', W, p\sigma) \leftarrow \texttt{ProxySign}(PK, W, psk_j, m)$ and sends $(PK', W, p\sigma)$ to $\mathcal{A}$.
5. *Proxy key exposure.* On input $(PK, W, j)$, $\mathcal{C}$ returns the $j$-th proxy key in $pskList(PK, W)$ if such a key exists. Otherwise, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

**Forgery** The adversary outputs a forgery and halts. The forgery can be of one of the following forms:

(i) *Ordinary signature of $u^*$.* The adversary outputs $(m, \sigma)$. This forgery is said to be *valid* if $\texttt{Verify}(pk^*, m, \sigma) = \texttt{accept}$ and $m$ has not been submitted in an ordinary signature query.

(ii) *Proxy signature of $u^*$.* The adversary outputs a message/signature tuple, $(m, (PK, W, p\sigma))$, where the last key in $PK$ is $pk^*$. This forgery is said to be *valid* if $\texttt{ProxyVerify}(m, (PK, W, p\sigma)) = \texttt{accept}$, $(PK, W, *, m)$ has not been submitted in a proxy signature query and $(PK, W, *)$ has not been submitted in a proxy key exposure query.

(iii) *Proxy signature on behalf of $u^*$.* The adversary outputs a message/signature tuple, $(m, (PK, W, p\sigma))$, as a forgery, where the last key in $PK$ is different from $pk^*$. Let $PK = (pk_1, \ldots, pk_n)$. The forgery is said to be *valid* if $\texttt{ProxyVerify}(m, (PK, W, p\sigma)) = \texttt{accept}$ and there exists an $1 \leq i^* \leq n-1$ such that $pk_{i^*} = pk^*$, $pk_{i^*+1} \notin delList(PK_{1\ldots i^*}, W_{1\ldots i^*-1}, w_{i^*})$ and $(PK_{1\ldots i^*}, W_{1\ldots i^*-1}, *)$ has not been submitted in a proxy key exposure query.

If the forgery output by the adversary is valid, return 1 as a result of the game. Otherwise, return 0.

Note that a type $(ii)$ forgery $(m, (PK, W, p\sigma))$ is not considered to be valid in our model if the adversary has exposed *any* of the proxy keys generated by $u^*$ in a delegation from the users $PK$ with the warrants $W$, or requested a signature on $m$ with one of these keys. Multiple keys can exists if the delegation is randomized and the adversary makes identical delegation requests multiple times. However, since all signatures created with these proxy keys will verify under the same conditions, a compromise of just one of them should be considered as a complete compromise of the delegation from the users $PK$ under warrants $W$.

Let $\mathbf{Gm}^{\texttt{ps-uf-pke}}_{\mathcal{PS},\mathcal{A}}(k)$ be the outcome of running the above security game with proxy signature scheme $\mathcal{PS}$, adversary $\mathcal{A}$ and security parameter $k$. We then define the advantage of the adversary in the security game as

$$\mathbf{Adv}^{\texttt{ps-uf-pke}}_{\mathcal{PS},\mathcal{A}}(k) = \Pr[\mathbf{Gm}^{\texttt{ps-uf-pke}}_{\mathcal{PS},\mathcal{A}}(k) = 1]$$

where the probability is taken over all random coins tosses made by the adversary and the challenger.

**Definition 1** *An adversary $\mathcal{A}$ is said to be a $(\epsilon, t, q_d, q_s)$-forger of a proxy signature scheme if $\mathcal{A}$ has advantage at least $\epsilon$ in the above game, runs in time at most*

*t and makes at most $q_d$ and $q_s$ delegation and signing queries to the challenger. A proxy signature scheme is said to be $(\epsilon, t, q_d, q_s)$-secure if no $(\epsilon, t, q_d, q_s)$-forger exists.*

## 5 Proxy Schemes Based on Sequential Aggregation

We will now present a generic proxy signature construction that satisfy the security definition given in Section 4, using a sequential aggregate signature scheme that is existentially unforgeable in the sequential aggregate chosen-key model. To guarantee that no information about a user's long term secret key is leaked if proxy keys are exposed, we will let a proxy generate a fresh independent key pair $(pk, sk)$ in a delegation, create a certificate for $pk$ and keep $sk$ as the proxy key. The generated public keys will be stored in a separate list $FK$. To avoid trivial attacks against the scheme, we will use the idea of Boldyreva *et al.* [2], and introduce symbols `dlg`, `sgn` and `prx`, which will be attached to the content being signed in, respectively, a delegation, an ordinary signature and a proxy signature.

*Construction 1* Let $\mathcal{AS} = \{\texttt{Setup}', \texttt{KeyGen}', \texttt{AggSign}, \texttt{AggVerify}\}$ be a sequential aggregate signature scheme and let the symbols `dlg`, `sgn` and `prx` be defined as different strings. Then a multi-level proxy signature scheme can be constructed as follows:

- `Setup`, `KeyGen`. Same as the corresponding algorithms from the sequential aggregate signature scheme.
- $\texttt{Sign}(sk, m)$ Compute $\sigma \leftarrow \texttt{AggSign}(sk, \texttt{sgn}||m, \emptyset)$, where $\emptyset$ indicates an "empty" sequential aggregate signature, and return $\sigma$ as a signature.
- $\texttt{Verify}(pk, m, \sigma)$ Return the output of $\texttt{AggVerify}(pk, \texttt{sgn}||m, \sigma)$.
- $\texttt{Delegate}(PK, W, pk_d, pk_p, sk, w)$ Depending on $(PK, W)$, take one of the following actions:
  - If $PK$ and $W$ are empty lists (i.e. $sk$ is an ordinary private key), construct the lists $PK' = (pk_d, pk_p)$, $FK = ()$ and $W' = (w)$. Compute $\sigma_{del} \leftarrow \texttt{AggSign}(sk, \texttt{dlg}||PK'||FK||W', \emptyset)$ and send the delegation message $(PK', FK, W', \sigma_{del})$ to the proxy.
  - If $PK$ and $W$ are not empty (i.e. $sk$ is a proxy key), construct $PK' = PK.(pk_p)$ and $W' = W.(w)$. Parse $sk$ as $(FK, \sigma_{del}, sk_{prx})$, compute

    $$\sigma'_{del} \leftarrow \texttt{AggSign}(sk_{prx}, \texttt{dlg}||PK'||FK||W', \sigma_{del})$$

    and send the delegation message $(PK', FK, W', \sigma'_{del})$ to the proxy.
- $\texttt{ProxyKeyGen}(pk_d, pk_p, sk_p)$ When $(PK', FK, W', \sigma_{del})$ is received from the delegator, generate a fresh proxy key pair $(pk'_p, sk'_p) \leftarrow \texttt{KeyGen}$ and construct $FK' = FK.(pk'_p)$. Compute $\sigma''_{del} \leftarrow \texttt{AggSign}(sk_p, \texttt{dlg}||PK'||FK'||W', \sigma'_{del})$, set $psk = (FK', \sigma''_{del}, sk'_p)$ and output $(PK', W', psk)$.
- $\texttt{ProxySign}(PK, W, psk, m)$ Parse the proxy key $psk$ as $(FK, \sigma_{del}, sk_p)$ and compute $p\sigma \leftarrow \texttt{AggSign}(sk_p, \texttt{prx}||PK||FK||W||m, \sigma_{del})$. Return the tuple $(PK, W, (FK, p\sigma))$ as a proxy signature.

– `ProxyVerify`$(m, (PK, W, (FK, p\sigma)))$ Assume that $PK$ contains $n+1$ elements. Construct

$$m_i = \texttt{dlg}||PK_{1\ldots i+1}||FK_{1\ldots i}||\,W_{1\ldots i} \quad \text{for } i \in \{1,\ldots,n\} \quad \text{and}$$
$$\overline{m} = (m_1,\ldots,m_n,\texttt{prx}||PK||FK||\,W||m).$$

Return the output of `AggVerify`$(PK, \overline{m}, p\sigma)$.

**Theorem 2** *Let $\mathcal{AS}$ be a $(t, q_s, \epsilon)$-unforgeable sequential aggregate signature scheme. Then Construction 1 yields a $(t', q'_s, q'_d, \epsilon')$-unforgeable proxy signature scheme where $\epsilon = \epsilon'/2q_d$, $t = t'$ and $q_s = q'_s + q'_d$.*

The proof of this theorem is given in Appendix C.

The above construction can be instantiated with a number of different sequential aggregate signature schemes to give proxy signature schemes with various security properties. For example, if the (fully aggregate) scheme of Boneh *et al.* [5] is used, we obtain a proxy signature scheme which is secure in the random oracle model under the Computational co-Diffie-Hellman assumption, a natural generalization of the CDH assumption suited to bilinear groups. Notice, however, that since a proxy signature will potentially include many public keys, but only one aggregate signature, the most efficient scheme (in terms of proxy signature size) is achieved by minimizing the size of the public keys and not the size of the aggregate signature. The scheme of [5] easily allows this modification, and using this we obtain a very efficient scheme, even if only single-level delegations are considered. Instantiating the scheme with the MNT elliptic curves [25], we can achieve a public key size of 168 bits and an aggregate signature size of 1008 bits, giving a proxy signature size of 1512 bits, all for a security level of approximately 80 bits. Hence, the scheme provides proxy signatures which are less than half the size of the triple Schnorr signatures as they are presented in [2], while satisfying a stronger definition of security and providing self-verifiability and multi-level capabilities. Note, however, that the triple Schnorr scheme allows faster verification.

To achieve an RSA-based proxy signature scheme, we can use the sequential aggregate signature technique proposed by Lysyanskaya *et al.* [21], which is secure in the random oracle model given that a claw-free permutation family is used in the construction. Note that the RSA-based instantiation proposed in [21] has the disadvantage that the aggregate signature will grow with one bit for each signer. To avoid this expansion, the slightly more computationally expensive RSA-family of trap-door permutations with common domain proposed by Hayashi *et al.* [13] can be used. It should be mentioned that to avoid the need for key certification, a few extra properties are needed to guarantee that each public key does define a permutation over the common domain; details are in [21]. However, with these minor changes, we obtain a scheme at the 80-bit security level having an aggregate signature size of 1024 bits and, assuming that all users use the same encryption exponent, a public key size of 1024 bits, giving a proxy signature size of 4096 bits for a single-level delegation.

Lastly, it is also possible to construct a scheme which is secure in the standard model. However, the sequential aggregate scheme proposed by Lu *et al.* [20] cannot be used for this purpose since this scheme is dependent on the registered key model for security. In fact, to our knowledge, it is still an open problem to construct an efficient sequential aggregate signature scheme which is secure in the standard model and which does not require the registered key model. This leaves only the "trivial" construction from an ordinary signature scheme (in which the aggregate signature is simply a concatenation of ordinary signatures). Using this together with, for example, the signature scheme of Boneh and Boyen [3], gives a scheme which is secure, albeit somewhat inefficient, in the standard model under the q-Strong Diffie-Hellman assumption. Instantiating the scheme with an elliptic curve similar to the one suggested above, it is possible to achieve a public key size of 336 bits (assuming all users use the same group generator and that redundant parts of the public key are left out), a signature size of 1176 bits, and a proxy signature size of 4536 bits for a single-level delegation, all at a security level of 80 bits. We note that if one is willing to downgrade the security requirements and use the registered key model, the scheme of Lu *et al.* [20] can be used, but a direct application will not be efficient due to the large size of the public keys.

## 6 Identity-based Constructions

Identity-based cryptography was originally proposed by Shamir [28] more than two decades ago, but identity-based encryption was first efficiently instantiated recently by Boneh and Franklin [4]. The construction methods presented in [4] inspired the extension of many existing cryptographic primitives to the identity-based setting along with efficient constructions. Among these, specific identity-based proxy signatures were also constructed (see, for example, [34, 33, 12]).

Both our definition of a proxy signature scheme given in Section 3 and the security model presented in Section 4, can easily be extended to the identity-based setting. However, due to space restrictions, we will not give the full definitions here, but only briefly discuss the changes needed to obtain the identity-based formulations.

*Identity-based proxy signatures.* First of all, in an identity-based setting, the presence of a *master entity* is assumed. The role of the master entity is to initially generate a set of public system parameters and a *master key*, which the master entity will use to generate private keys corresponding to identities in the scheme. An identity-based signature scheme is given by the algorithms $\mathcal{IBS} = \{\texttt{Setup}, \texttt{Extract}, \texttt{Sign}, \texttt{Verify}\}$, where `Setup` generates the system parameters and the master key, `Extract` generates a private key for an identity and `Sign` and `Verify` implement similar functionality to the corresponding algorithms of an ordinary signature scheme, with the exception that public keys are replaced by identities. An identity-based proxy signature scheme extends an identity-based signature scheme with the algorithms {`Delegate`, `ProxyKeyGen`,

`ProxySign`, `ProxyVerify`}, which implement similar functionality to the corresponding algorithms for an ordinary proxy signature scheme, with the exception that all public keys are replaced by identities.

*Security model.* The security notion *Identity-Based Proxy Signature Unforgeability Under an Adaptive Chosen Message Attack with Proxy Key Exposure* (`id-ps-uf-pke`) can be defined by introducing the following changes to the security game in Section 4:

**Setup** The adversary is no longer given a public key pair, but only the system parameters.

**Queries** The adversary is allowed to make similar queries to those in the ordinary security game, using identities instead of public keys. The adversary will furthermore be allowed to adaptively request the private keys of identities.

**Forgery** The adversary is allowed to choose an identity $ID^*$ for which he will produce a forgery (in the ordinary game the adversary was forced to produce a forgery for $pk^*$ chosen by the challenger), but it is required that he has not requested the private key of $ID^*$ during the game. Besides this, the restrictions on the forgery from the ordinary game apply.

With the above changes to the security game, the advantage of the adversary can be defined exactly as in the ordinary security game and $(\epsilon, t, q_d, q_s)$-security for an identity-based proxy signature scheme can be formulated exactly as in Definition 1.

*Construction* Having defined the identity-based security model, it remains to be seen if Construction 1 will yield a secure identity-based proxy signature scheme, using an identity-based sequential aggregate signature scheme [14]. Looking at the definition of `ProxyKeyGen` reveals one problem though: a proxy is required to generate a fresh key pair in a delegation. This represents a limitation in the identity-based setting, since only the master entity can generate a private key corresponding to a given identity[3]. However, note that it is not necessary for the key pair generated in `ProxyKeyGen` to be identity-based (i.e. consist of an identity and a private key) for the overall scheme to maintain its identity-based properties. In fact, a key pair from an ordinary sequential aggregate signature scheme will suffice. Since signatures from an ordinary and an identity-based sequential aggregate signature scheme cannot generally be aggregated in the same signature, $\sigma_{del}$ and $p\sigma$ in Construction 1 will have to be split into two parts – one part containing aggregated identity-based signatures and the other containing aggregated ordinary signatures. However, with these small changes, a secure identity-based proxy signature scheme can be obtained.

**Theorem 3** *Let a $(t', q'_s, \epsilon')$-unforgeable sequential aggregate signature scheme and a $(t'', q''_s, \epsilon'')$-unforgeable identity-based sequential aggregate signature scheme*

[3] We note that a user can generate private keys for new identities if a hierarchical identity-based signature scheme is used, but due to space limitations, we will not discuss this alternative approach here.

*be given. Then the above modifications to Construction 1 yields a* $(t, q_s, q_d, \epsilon)$*-unforgeable identity-based proxy signature scheme where* $\epsilon = q_d\epsilon' + \epsilon''$*,* $t = \min(t', t'')$ *and* $q_s + q_d = \min(q'_s, q''_s)$*.*

The proof of this theorem is very similar to that of the proof of Theorem 2 and will not be given here. The main difference from the proof of Theorem 2 is that a successful forgery against the identity-based proxy signature scheme will potentially lead to either a forgery of the identity-based or the ordinary signature scheme, depending on the type of the proxy signature forgery. However, these different types of proxy signature forgeries are already considered in the proof of Theorem 2 although only a forgery for the single underlying scheme is produced.

When instantiating the above construction, all of the options for an ordinary sequential aggregate signature scheme discussed in Section 5 can be used. However, the choice of an identity-based sequential aggregate signature scheme is less obvious. One would imagine that the scheme by Gentry and Ramzan [10] would be an ideal candidate, but this scheme is based on all users agreeing on a random string $w$, which is used in the signing process, before signatures can be aggregated, and the scheme will become insecure if the same $w$ is used for different aggregate signatures[4]. The latter property means that the Gentry-Ramzan scheme does not have the full flexibility of a sequential aggregate signature scheme, since an existing aggregate signature cannot be aggregated with two different signatures to yield two new aggregate signatures. In our construction, this would mean that a proxy could only delegate the signing rights of a proxy key once. To our knowledge, no other identity-based sequential aggregate signature scheme (which provides full aggregation) has been proposed, and it remains an open problem to construct such a scheme. However, schemes that provide partial aggregation (i.e. the size of the aggregate signature is not independent of the number of signers) have been proposed and can be used to instantiate our construction. For example, the scheme proposed by Herranz [14], which is secure in the random-oracle model under the Computational co-Diffie-Hellman assumption, can be used to achieve a fairly efficient scheme.

## 7 Conclusion

In this paper, we have motivated the introduction of a new security model for proxy signatures that enhances the existing models of [2, 22]. The new model incorporates warrants, allows unregistered public keys, and lets the attacker recover proxy private keys. These extensions were motivated by practical considerations as well as attacks on existing schemes. We showed how our new security definition could be achieved through a generic construction involving sequential aggregate signatures, and considered concrete and efficient instantiations of the construction. Finally, we sketched how our models and constructions can be extended to the identity-based setting.

[4] This is not just a property of the security proof given in [10], but will enable an adversary to construct selective forgeries.

In the full version, we complete the routine investigation of the security and performance trade-offs of our schemes and provide the full details of the identity-based setting. We also consider how hierarchical identity-based signatures can be used to construct efficient identity-based proxy signatures.

## References


1. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM Conference on Computer and Communications Security*, pages 390–399. ACM, 2006.
2. Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. Cryptology ePrint Archive, Report 2003/096, 2003. `http://eprint.iacr.org/`.
3. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Cachin and Camenisch [7], pages 56–73.
4. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.
5. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.
6. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, 2004.
7. Christian Cachin and Jan Camenisch, editors. *Advances in Cryptology - EUROCRYPT 2004, Proceedings*, volume 3027 of *LNCS*. Springer, 2004.
8. Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Strong key-insulated signature schemes. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *LNCS*, pages 130–144. Springer, 2003.
9. Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.
10. Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2006.
11. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
12. Chunxiang Gu and Yuefei Zhu. An efficient id-based proxy signature scheme from pairings. Cryptology ePrint Archive, Report 2006/158, 2006. `http://eprint.iacr.org/`.
13. Ryotaro Hayashi, Tatsuaki Okamoto, and Keisuke Tanaka. An RSA family of trap-door permutations with a common domain and its applications. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Public Key Cryptography*, volume 2947 of *LNCS*, pages 291–304. Springer, 2004.
14. Javier Herranz. Deterministic identity-based signatures for partial aggregation. Cryptology ePrint Archive, Report 2005/313, 2005. `http://eprint.iacr.org/`.
15. Florian Hess. Efficient identity based signature schemes based on pairings. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *LNCS*, pages 310–324. Springer, 2002.

16. Seungjoo Kim, Sangjoon Park, and Dongho Won. Proxy signatures, revisited. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *ICICS*, volume 1334 of *LNCS*, pages 223–232. Springer, 1997.
17. S. Lal and A. K. Awasthi. Proxy blind signature scheme. Cryptology ePrint Archive, Report 2003/072, 2003. `http://eprint.iacr.org/`.
18. Byoungcheon Lee, Heesun Kim, and Kwangjo Kim. Strong proxy signature and its application. In *SCIS*, pages 603–608, 2001.
19. Jung-Yeun Lee, Jung Hee Cheon, and Seungjoo Kim. An analysis of proxy signatures: Is a secure channel necessary? In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2003.
20. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. Cryptology ePrint Archive, Report 2006/096, 2006. `http://eprint.iacr.org/`.
21. Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Cachin and Camenisch [7], pages 74–90.
22. Tal Malkin, Satoshi Obana, and Moti Yung. The hierarchy of key evolving signatures and a characterization of proxy signatures. In Cachin and Camenisch [7], pages 306–322.
23. Tal Malkin, Satoshi Obana, and Moti Yung. The hierarchy of key evolving signatures and a characterization of proxy signatures. Cryptology ePrint Archive, Report 2004/052, 2004. `http://eprint.iacr.org/`.
24. M. Mambo, K Usuda, and E. Okamoto. Proxy signatures for delegating signing operation. In *Proceedings of the 3st ACM conference on Computer and Communications Security*, pages 48–57, 1996.
25. A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions, Fundamentals.*, E84-A, No. 5:1234–1243, 2001.
26. Takeshi Okamoto, Atsuo Inomata, and Eiji Okamoto. A proposal of short proxy signature using pairing. In *ITCC (1)*, pages 631–635. IEEE Computer Society, 2005.
27. Takeshi Okamoto, Mitsuru Tada, and Eiji Okamoto. Extended proxy signatures for smart cards. In Masahiro Mambo and Yuliang Zheng, editors, *ISW*, volume 1729 of *LNCS*, pages 247–258. Springer, 1999.
28. Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO 84*, pages 47–53, 1984.
29. Hung-Min Sun. An efficient nonrepudiable threshold proxy signature scheme with known signers. *Computer Communications*, 22(8):717–722, 1999.
30. Guilin Wang. Designated-verifier proxy signature schemes. In Ryôichi Sasaki, Sihan Qing, Eiji Okamoto, and Hiroshi Yoshiura, editors, *SEC*, pages 409–424. Springer, 2005.
31. Guilin Wang, Feng Bao, Jianying Zhou, and Robert H. Deng. Security analysis of some proxy signatures. In Jong In Lim and Dong Hoon Lee, editors, *ICISC*, volume 2971 of *LNCS*, pages 305–319. Springer, 2003.
32. Huaxiong Wang and Josef Pieprzyk. Efficient one-time proxy signatures. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *LNCS*, pages 507–522. Springer, 2003.
33. Jing Xu, Zhenfeng Zhang, and Dengguo Feng. ID-based proxy signature using bilinear pairings. Cryptology ePrint Archive, Report 2004/206, 2004. `http://eprint.iacr.org/`.

34. Fangguo Zhang and Kwangjo Kim. Efficient ID-based blind signature and proxy signature from bilinear pairings. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *ACISP*, volume 2727 of *LNCS*, pages 312–323. Springer, 2003.
35. Fangguo Zhang, Reihaneh Safavi-Naini, and Chih-Yin Lin. New proxy signature, proxy blind signature and proxy ring signature schemes from bilinear pairing. Cryptology ePrint Archive, Report 2003/104, 2003. `http://eprint.iacr.org/`.
36. K. Zhang. Threshold proxy signature schemes. In *Proc. 1st International Information Security Workshop*, pages 282–290, 1997.

## A Key Registration Attack on ZSL-scheme

We briefly illustrate how an adversary can mount an attack on the proxy signature scheme proposed by Zhang, Safavi-Naini and Lin [35], if key registration is not required as a part of the security model.

In the construction presented in [35], delegation is done by letting the delegator construct a Boneh-Lynn-Shacham signature [6] on the warrant $w$, i.e. by computing $\sigma_d = s_d H(w)$ (where $s_d \in \mathbb{Z}_q$ is the private key of the delegator and $H$ is a hash function onto a bilinear group $\mathbb{G}$ of prime order $q$), and then sending $\sigma_d$ to the proxy. Upon receiving $\sigma_d$, the proxy generates a private proxy key $psk$ by computing his own signature on the warrant, $\sigma_p = s_p H(w)$, where $s_p \in \mathbb{Z}_q$ is the private key of the proxy, and setting $psk = \sigma_d + \sigma_p = (s_d + s_p)H(w)$.

A proxy signature is then created by using the identity-based signature scheme of Hess [15], letting $w$ act as the signing identity and $psk$ as the private key for this identity. A verifier can construct a master public key for the Hess signature scheme in which $psk$ is the private key of $w$, by summing the public keys for the delegator and the proxy, i.e. by setting $pk_d + pk_p = (s_d + s_p)P$ where $P$ generates $\mathbb{G}$, and then verify the proxy signature as a signature by the identity $w$.

However, this construction is insecure if the registered key model is not used. This can easily be seen as follows: let $pk^* = s^*P$ be the public key of the challenge user and let the adversary choose the public key $pk = s_a P - pk^*$ for a malicious proxy (note that the adversary cannot compute the private key corresponding to this public key). Then, for any warrant $w$, the adversary can compute $s_a H(w) = (s_a - s^* + s^*)H(w) = \sigma^* + \sigma_a$ and thereby construct the private key needed for creating proxy signatures on behalf of the challenge user, without the challenge user having delegated his signing rights.

## B Proxy Key Exposure Attack on BPW-scheme

We briefly illustrate how an adversary can recover the private key of the user in the triple Schnorr proxy signature scheme analyzed by Boldyreva, Palacio and Warinschi [2], if a proxy key is exposed.

The key observation is that the value $t = G(0||pk_d||pk_p||w, Y) \cdot sk_p + s \mod q$ is a part of the proxy key, where $G$ is a hash function, $pk_d$ and $pk_p$ are the public keys of the delegator and the proxy, $w$ is a warrant, $(Y, s)$ are values sent by the

delegator to the proxy in a delegation, and $sk_p$ is the private key of the proxy. Since it is not assumed that there is a secure channel between the delegator and the proxy, $(Y, s)$ can be observed by the adversary, and if a proxy key is exposed, the adversary can recover the private key $sk_p$ of the proxy, simply by computing $sk_p = (t - s) \cdot G(0||pk_d||pk_p||w, Y)^{-1} \mod q$.

## C Proof of Theorem 2

*Proof.* The proof is by contradiction: we will assume that an adversary $\mathcal{A}$ that $(t', q'_s, q'_d, \epsilon')$-breaks Construction 1 exists, and from this, construct an adversary $\mathcal{B}$ that $(t, q_s, \epsilon)$-breaks the underlying sequential aggregate signature scheme.

Initially, $\mathcal{B}$ will be given a challenge public key $pk'$ and access to a sequential aggregate signing oracle $\mathcal{O}_{sig}(m, \sigma_{agg})$ for the secret key $sk'$ corresponding to $pk'$. Firstly, $\mathcal{B}$ flips a fair coin $c$. If $c = 0$, $\mathcal{B}$ sets $pk^* = pk'$ and $sk^* = \emptyset$. Otherwise, $\mathcal{B}$ generates a fresh key pair $(pk^*, sk^*) \leftarrow \texttt{KeyGen}$, and chooses $i^* \in \{1, \ldots, q'_d\}$ ($\mathcal{B}$ will later use $pk'$ instead of a fresh key in the $i^*$-th delegation query by $\mathcal{A}$). For ease of notation, we define the following function for signature generation by $\mathcal{B}$:

$$\texttt{Sign}_{\mathcal{B}}(sk, m, \sigma_{agg}) = \begin{cases} \mathcal{O}_{sig}(m, \sigma_{agg}) & \text{if } sk = \emptyset \\ \texttt{AggSign}(sk, m, \sigma_{agg}) & \text{otherwise} \end{cases}$$

$\mathcal{B}$ runs $\mathcal{A}$ with input $pk^*$. As the challenger in the security game, $\mathcal{B}$ will maintain a set of lists $pskList(*, *)$ while $\mathcal{A}$ is running. Each list $pskList(PK, W)$ will hold all proxy keys generated by $\mathcal{B}$ for the delegation chain with the public keys $PK$ and the warrants $W$. While running, $\mathcal{A}$ can make various queries which $\mathcal{B}$ will answer as follows (note that, to answer the queries, $\mathcal{B}$ simply implements the challenger by using his access to the signing oracle and taking into account the value of $c$):

- *Ordinary signature.* On input $m$ from $\mathcal{A}$, $\mathcal{B}$ returns $\texttt{Sign}_{\mathcal{B}}(sk^*, \texttt{sgn}||m, \emptyset)$.
- *Delegation to $u^*$.* $\mathcal{A}$ submits the delegation message $(PK, FK, W, \sigma_{del})$. If $c = 0$, or $c = 1$ and this is *not* the $i^*$-th delegation query, $\mathcal{B}$ generates a fresh key pair $(pk, sk) \leftarrow KeyGen$, constructs $FK' = FK.(pk)$ and sets $sk_{prx} = sk$. If $c = 1$ and this *is* the $i^*$-th delegation query, $\mathcal{B}$ constructs $FK' = FK.(pk^*)$ and sets $sk_{prx} = \emptyset$. Then $\mathcal{B}$ computes $\sigma'_{del} \leftarrow \texttt{Sign}_{\mathcal{B}}(sk_{prx}, \texttt{dlg}||PK||FK'||W, \sigma_{del})$ and stores $psk = (FK', \sigma'_{del}, sk_{prx})$ in $pskList(PK, W)$.
- *Delegation from $u^*$.* There are three different types of queries $\mathcal{A}$ can make:
  1. *Delegation of $sk^*$* On input $(pk_p, w)$ from $\mathcal{A}$, $\mathcal{B}$ constructs the lists $PK' = (pk^*, pk_p)$, $FK = ()$ and $W' = (w)$. Then $\mathcal{B}$ computes the signature $\sigma_{del} \leftarrow \texttt{Sign}_{\mathcal{B}}(sk^*, \texttt{dlg}||PK'||FK'||W', \emptyset)$ and sends the delegation message $(PK', FK, W', \sigma_{del})$ to $\mathcal{A}$.
  2. *Re-delegation of psk.* On input $(PK, W, j, pk_p, w)$ from $\mathcal{A}$, where $j \in \mathbb{N}$, $\mathcal{B}$ looks up the $j$-th proxy key in $pskList(PK, W)$ and parses it as $(FK, \sigma_{del}, sk_{prx})$. Then $\mathcal{B}$ constructs $PK' = PK.(pk_p)$ and $W' = W.(w)$, computes $\sigma'_{del} \leftarrow \texttt{Sign}_{\mathcal{B}}(sk_{prx}, \texttt{dlg}||PK'||FK||W', \sigma_{del})$, and sends the delegation message $(PK', FK, W', \sigma'_{del})$ to $\mathcal{A}$.

3. *self-delegation.* Depending on the input $(PK, W, j, w)$ submitted by $\mathcal{A}$, $\mathcal{B}$ will do one of the following:
   - If $PK$ and $W$ are empty (self-delegation of $sk^*$), $\mathcal{B}$ constructs the lists $PK' = (pk^*, pk^*)$, $FK = ()$ and $W' = (w)$, and sets $sk_{self} = sk^*$ and $\sigma_{self} = \emptyset$.
   - If $PK$ and $W$ are not empty (delegation of $psk$), $\mathcal{B}$ looks up the $j$-th proxy key in $pskList(PK, W)$ and parses it as $(FK, \sigma_{del}, sk_{prx})$. Then $\mathcal{B}$ constructs $PK' = PK.(pk^*)$ and $W' = W.(w)$, and sets $sk_{self} = sk_{prx}$ and $\sigma_{self} = \sigma_{del}$

   Then $\mathcal{B}$ computes $\sigma'_{del} \leftarrow \mathtt{Sign}_{\mathcal{B}}(sk_{self}, \mathtt{dlg}||PK'||FK||W', \sigma_{self})$. Now, if $c = 0$, or $c = 1$ and this is *not* the $i^*$-th delegation query, $\mathcal{B}$ generates $(pk, sk) \leftarrow \mathtt{KeyGen}$ and constructs $FK' = FK.(pk)$. Otherwise, $\mathcal{B}$ just constructs $FK' = FK.(pk^*)$ and sets $sk = \emptyset$. Finally, $\mathcal{B}$ computes $\sigma''_{del} \leftarrow \mathtt{Sign}_{\mathcal{B}}(sk_{self}, \mathtt{dlg}||PK'||FK'||W', \sigma'_{del})$, stores the proxy key $psk = (FK', \sigma''_{del}, sk)$ in $pskList(PK', W')$ and sends the transcript $(PK', FK, W', \sigma'_{del})$ to $\mathcal{A}$.

– *Proxy signature.* On input $(PK, W, j, m)$ from $\mathcal{A}$, $\mathcal{B}$ looks up the $j$-th proxy key, in $pskList(PK, W)$ and parses it as $(FK, \sigma_{del}, sk_{prx})$. Then $\mathcal{B}$ computes the signature $p\sigma \leftarrow \mathtt{Sign}_{\mathcal{B}}(sk_{prx}, \mathtt{prx}||PK||FK||W||m, \sigma_{del})$ and returns $(PK, W, (FK, p\sigma))$ to $\mathcal{A}$.
– *Proxy key exposure.* On input $(PK, W, j)$, $\mathcal{B}$ looks up the $j$-th proxy key in $pskList(PK, W)$ and parses it as $(FK, \sigma_{del}, sk_{prx})$. If $sk_{prx} = \emptyset$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ returns $(FK, \sigma_{del}, sk_{prx})$ to $\mathcal{A}$.

Note that $pk^*$ will be drawn from the same distribution as public keys generated by `KeyGen` and that $\mathcal{B}$'s choice of $c$ will be completely hidden from $\mathcal{A}$, unless an abort occurs.

If $\mathcal{B}$ is not forced to abort, $\mathcal{A}$ will eventually output a forgery. We will classify forgeries into two different categories:

**Category A** forgeries are either a valid type $(i)$ forgery $(m, \sigma)$, a valid type $(ii)$ forgery $(m, (PK, W, (FK, p\sigma)))$ where the last key in $FK$ was not generated by $\mathcal{B}$, or a valid type $(iii)$ forgery $(m, (PK, W, (FK, p\sigma)))$ where the $(i^* - 1)$-th key in $FK$ was not generated by $\mathcal{B}$.

**Category B** forgeries are all valid forgeries that are not in Category A, i.e. a type $(ii)$ or type $(iii)$ forgery where $\mathcal{B}$ has generated the public key in $FK$ which corresponds to $u^*$'s position in the delegation chain of the forgery.

Informally, Category A forgeries correspond to forgeries where $\mathcal{A}$ has forged a signature under $u^*$'s long term key, and Category B forgeries correspond to forgeries where $\mathcal{A}$ has forged a signature under one of the keys generated by $u^*$ in a delegation, but for which $\mathcal{A}$ has not received the corresponding private key.

Consider the case where $c = 0$. In this case, $\mathcal{B}$ sets $pk^* = pk'$. If $\mathcal{A}$ constructs a valid Category A forgery, then

– if the forgery is of type $(i)$ i.e. $(m, \sigma)$, then $\mathcal{A}$ will not have requested a signature on $m$ (since the forgery is valid), and $\mathcal{B}$ will therefore not have

submitted $(\texttt{sgn}||m, \emptyset)$ to his own signing oracle. Hence, $\sigma$ is a valid forgery of a sequential aggregate signature of length 1 on the message $\texttt{sgn}||m$ under the the public key $pk^* = pk'$.

- if the forgery is of type $(ii)$ i.e. $(m, (PK, W, (FK, p\sigma)))$, where the last key $pk_n \in PK$ is equal to $pk^* = pk'$, then $\mathcal{B}$ will not have submitted $(\texttt{dlg}||PK||FK||W, \sigma_{del})$ for any $\sigma_{del}$ to his own signing signing oracle (since this is a Category A forgery). Hence, $p\sigma$ will be a valid forgery of a sequential aggregate signature containing a signature on the message $\texttt{dlg}||PK||FK||W$ under $pk^* = pk'$.
- if the forgery is of type $(iii)$ i.e. $(m, (PK, W, (FK, p\sigma)))$, $p\sigma$ will be a valid forgery for the same reasons as in a type $(ii)$ forgery, just having $pk^*$ appearing at a different position in $PK$.

If $\mathcal{A}$, on the other hand, constructs a Category B forgery, $\mathcal{B}$ will abort.

Now consider the case where $c = 1$. In this case $\mathcal{B}$ inserts $pk'$ as a fresh key in a delegation query. If $\mathcal{A}$ outputs a Category A forgery, $\mathcal{B}$ will abort. However, if $\mathcal{A}$ outputs a category B forgery $(m, (PK, W, (FK, p\sigma)))$, which will be of either type $(ii)$ or type $(iii)$, $p\sigma$ will be a sequential aggregate signature containing a signature under a key $pk$ generated by $\mathcal{B}$ in a delegation query (i.e. $pk$ will appear as the last key in $FK'$ for a proxy key $(FK', \sigma_{del}, sk_{prx}) \in pskList(PK_{1\ldots i}, W_{1\ldots i-1})$ for some $i$), and for which $\mathcal{A}$ has not asked for the proxy key containing the corresponding private key. With probability $1/q_d$, $\mathcal{B}$ will have chosen $pk = pk'$. In this case, $\mathcal{B}$ outputs $p\sigma$ as a valid forgery for the underlying sequential aggregate signature scheme. Otherwise, $\mathcal{B}$ will abort.

Note that if $c = 0$, $\mathcal{B}$ provides a perfect simulation for $\mathcal{A}$ and does not need to abort before $\mathcal{A}$ outputs a forgery. Also note that if $c = 1$, $\mathcal{A}$ is constructing a Category B forgery and $\mathcal{B}$ has guessed the correct value of $i^*$ (i.e. guessed the key $pk_{i^*}$ which $\mathcal{A}$ will use in a forgery and inserted $pk_{i^*} = pk'$), $\mathcal{B}$ will not have to abort either since $\mathcal{A}$ will not compromise the key $pk_{i^*}$ in order to produce a valid forgery.

Let $E_1$ be the event that $\mathcal{A}$ produces a Category A forgery, $E_2$ be the event that $\mathcal{A}$ produces a Category B forgery, and $E_3$ be the event that $\mathcal{B}$ guesses the correct value of $i^*$ in a Category B forgery. The success probability $\epsilon'$ of $\mathcal{A}$ can be expressed as $\epsilon' = \Pr[E_1] + \Pr[E_2]$. The success probability of $\mathcal{B}$ can be expressed as

$$\begin{aligned}
\epsilon &= \Pr[c = 0 \wedge E_1] + \Pr[c = 1 \wedge E_2 \wedge E_3] \\
&= 1/2 \Pr[E_1] + \Pr[E_3 | c = 1 \wedge E_2] \Pr[c = 1 | E_2] \Pr[E_2] \\
&= 1/2 \Pr[E_1] + 1/q_d \cdot 1/2 \cdot \Pr[E_2] \\
&\geq \epsilon'/2q_d
\end{aligned}$$

Hence, the theorem follows.