CSCI471/971 Advanced Computer Security
Spring 2013
Group 2
Designated Verifier Proxy Signatures

Shiwei Zhang, Christopher Tom Kochovski, Xiong Wen, and Neilston Pinto

School of Computer Science and Software Engineering
Faculty of Engineering and Information Sciences
University of Wollongong, Australia
Student Number: {3792444, 3860632, 4107755, 4404440}
Email: {sz653, ctk283, xw926, nlp872}@uowmail.edu.au

Abstract. The process in which the Designated Verifier Proxy Signature (DVPS) scheme works is that you have one party which is an original signer who has the ability to delegate signing capability to another party member, in this case this is the proxy signer. The proxy signer can then sign on behalf of the original signer. The validity or invalidity of a signature can only be verified by a designated verifier and no other verifying party by the use of a proof or challenge. In this paper, we will first outline the security preliminary we will use for our own Designated Verifier Proxy Signature scheme, and the signature schemes of the Proxy Signature (PS), Designated Verifier Signature (DVS) and Short Designated Verifier Proxy Signature (SDVPS). Finally, our own Designated Verifier Proxy Signature scheme will be presented with concrete and complete definitions, scheme, and properties which also includes security assumptions based on the Proxy, Designated Verifier Signature and Short Designated Verifier Proxy Signature scheme presented earlier in the paper. A discussion will finalise the paper covering the security of our own Designated Verifier Proxy Signature scheme.

Keywords: Proxy Signature, Designated Verifier Signature, Short Signature, Pairings, Designated Verifier Proxy Signature.

1 Introduction

With the fast development of computer technology, more and more traditional business processes are adapted to the digital world. Especially, signatures play an important role in e-business for authentication.

In 1976, Diffie and Hellman first introduced the notion of *digital signature* with the same property of traditional hand-writing signatures and opened a new direction in cryptography [3]. Besides hand-writing signatures, seals are also frequently used in tradition business to authenticate documents on behalf of legal persons by authorised individuals. In the digital word, the replacement of traditional seals is *proxy signature* which was first introduced by Mambo, Usuda and Okamoto [9]. The proxy signature allows a party to delegate signing capabilities to other participants so that they can sign on behalf of the entity within a given context. In addition, some documents may be personally or commercially sensitive that the property of public verifiability is improper. In order to solve that problem in reality, Chaum and Antwerpen [2] introduced the notion of *undeniable signature* so that the signature can only be verified by performing an interactive protocol with the signer, taking the advantage of *zero knowledge proof* (ZKP) [5]. However, undeniable signatures do not fully model the problem since the signature is still publicly verifiable by running the interactive ZKP concurrently. Hence, *Designated verifier signature* is introduced by Jakobsson, Sako and Impagliazzo [7] that only the designated verifier can verify the signature, using the technique of *non-interactive zero knowledge proof* (NIZK) [1].

Designated verifier signatures solve the designated verifier problem of hand-writing signed documents, whereas the same problem of sealed documents was not solved. For example, the chief financial officer in a company generates a report about the financial risks of the company and seals the report with the financial department seal. The chief financial officer sends the sealed report to the chief operating officer and requires that only he/she can verify the sealed report as it is commercially sensitive. To address this problem with seals, *designated verifier proxy signature* (DVPS) is a proper solution which combines the property of proxy signature and designated verifier signature [10]. In the previous scenario, the chief financial officer can be modelled as a proxy signer of the financial department (the original signer), and the chief operating officer can be modelled as the designated verifier. In 2005, Wang constructed a designated verifier proxy signature based on discrete logarithm problem, using delegation by warrant [10]. Later, Huang, Mu, Susilo and Zhang constructed *short designated verifier proxy signature* (SDVPS) based on elliptic curve and pairings [6]. The signature size of Huang et al.'s scheme is very short (around $1/20$ of Wang's scheme).

The rest of this paper is organised as following. First, we describe the preliminaries for readers in section 2.1. Then we explain proxy signature, designated verifier signature, short designated verifier proxy signature and one scheme for each signature type in sections 2.2 to 2.4. After that, we propose our designated verifier proxy signature in section 2.5. The security of the proposed scheme is discussed in section 3. Finally, we draw conclusions for this paper in section 4.

2 Technical contents

2.1 Preliminaries

Bilinear Pairing Let G_1 and G_2 be additive groups of prime order p with generator P_1 and P_2 respectively, and G_3 be multiplicative group of the same prime order p. There is an equation $\Phi(P_2) = P_1$ holds under assumption of isomorphism $G_2 \to G_1$. The e: $G_1 \times G_2 \to G_3$ is a bilinear pairing which has following properties:

- Bilinearity: $\forall P \in G_1, Q \in G_2, a, b \in Z_q$, $e(aP, bQ) = e(P, Q)^{ab}$.
- Non-degeneracy: $\exists P \in G_1, Q \in G_2$, such that $e(P, Q) \neq 1$.
- Computability: $\forall P \in G_1, Q \in G_2$, $\exists A$ which is an efficient algorithm to compute $e(P, Q)$.

Security assumptions

Definition 1. *Discrete Logarithm Problem (DLP): Let p and q be two large primes satisfying q is divisible by $p-1$, and a generator, g, of order q over $GF(p)$. Given (y, p, q, g), where $y = g^x \pmod p$ for some $x \in \mathbb{Z}_q$, derive x.*

Definition 2. *Bilinear Diffie-Hellman (BDH) Problem: Let $P \in \mathbb{G}_1$. Given aP, bP and cP, where $a, b, c \in_R \mathbb{Z}_p^*$, calculate $e(P, P)^{abc}$.*

Definition 3. *Decisional Bilinear Diffie-Hellman (DBDH) Problem: Let $P \in \mathbb{G}_1$. Given aP, bP and cP, where $a, b, c \in_R \mathbb{Z}_p^*$, and $h \in \mathbb{G}_3$, decide whether $h = e(P, P)^{abc}$.*

Definition 4. *Gap Bilinear Diffie-Hellman (GBDH) Problem: Let $P \in \mathbb{G}_1$. Given aP, bP and cP, where $a, b, c \in_R \mathbb{Z}_p^*$, calculate $e(P, P)^{abc}$ with the help of a DBDH Oracle.*

Non-interactive Zero Knowledge Proof (NIZK) The notion of Non-Interactive Zero-Knowledge (NIZK) is first introduced by Blum and Feldman [1] to prove a secret to someone else without revealing it.

Definition 5. *A non-interactive zero knowledge proof (NIZK) is a pair of probabilistic polynomial time algorithms (P, V) for a language $L \in NP$, if the following three properties are satisfied.*

- *Completeness*: For any $x \in L$ with witness w where $|x| = k$, we require

$$Pr[\sigma \leftarrow \{0,1\}^k; \pi \leftarrow P(\sigma, x, w) : V(\sigma, x, \pi) = 1] = 1$$

- *Soundness*: For any $x \notin L$, where $|x| = k$ and $negl()$ is a negligible function, we require

$$Pr[\sigma \leftarrow \{0,1\}^k; \pi \leftarrow P(\sigma, x) : V(\sigma, x, \pi) = 1] < negl(k)$$

- *Zero-knowledge*: For all $x \in L$ with any witness w, we require that there exists a probabilistic polynomial time simulator S such that the output proofs by P and S are computationally indistinguishable.

$$\{\sigma \leftarrow \{0,1\}^k; \pi \leftarrow P(\sigma, x, w) : (\sigma, x, \pi)\} = \{(\sigma, \pi) \leftarrow S(x) : (\sigma, x, \pi)\}$$

Remark that Designated Verifier Proof discussed in section 2.3 is Designated Verifier NIZK, which is a weak notion of NIZK proof.

2.2 Proxy Signature

Proxy signature is a digital signature scheme which allows the original signer to delegate the capability of signing to a third party to create digital signatures. In analogy, it is an electronic version corresponding to the transferable seal among the multiple seals. The invention of this signature scheme is driven by the demand for delegating signing operation effectively in an electronic context. Proxy signature has been used in many scenarios such as electronic commerce and distributed shared object systems.

There are two ways of classification for proxy signature. According to the application, it can be classified to full delegation, partial delegation and delegation by warrant schemes [6]. In full delegation, the proxy signer uses exactly the same secret s with the original signer to create signatures, which means the signatures created by proxy signer and original signer are indistinguishable. In this case, a proxy signer can sign a message that does not follow the original signer's intention without being detected. This venerability can be eliminated by partial delegation. In partial delegation, the proxy signer uses a different secret σ which is derived from original signer's secret s and sent from the original signer in a secure way. The signature created by proxy signer can be checked by using a certain verification equation which is different from the original equation. Therefore, the signature created by proxy signer can be distinguished from the one created by original signer.

The last type is delegation by warrant. In this type of delegation, warrant is used to certify the entrusted proxy signer along with the signature. There are two signature schemes based on public key signature scheme to implementing delegation by warrant depend on how warrant is composed. In the first signature scheme, warrant consists of a message and an original signer's signature for a public key of proxy signer P, or just a declaration about the designation of P. In this scheme, P signs a document using her own secret key to create a signature along with the warrant. As a result, a proxy signature, by this method, is composed of the original signer's signature and the warrant. It should be marked that there is no relation between the signature created by proxy signer P and the public key of original signer. In the second signature scheme, warrant composition is similar to the first scheme except the original signer's signature is for a newly generated public key. Then P uses the secret key that correspond to the new public key of original signer for signing along with the warrant. Unlike the first scheme, the public key of original signer is required in verification phase of verifier [9]. To sum up, among these three types of delegation, delegation by warrant is the most secure one as it has a property that it is able to restrict the documents to be signed. However, there is lack of valid period limitation of signing, which means a proxy signer can sign a signature forever [9].

According to knowledge of proxy private key, it can also be categorised as proxy-unprotected and proxy-protected [6]. Both proxy signer and original signer, who all know the proxy private key, can generate the proxy signature in proxy-unprotected scheme. But only the proxy signer can create proxy signature in proxy-protected scheme. Certain conditions should be satisfied when identifying whether a digital signature is a proxy signature. An example for partial delegation includes unforgeability, proxy signers deviation, secret-keys dependence, verifiability, distinguishability, identifiability and undeniability.

There are variants of proxy signature which added distinct features to the basic proxy signature, such as threshold proxy signature, one-time proxy signature, ID-based proxy signature and so on [6].

Mambo et al. [9] proposed a proxy signature scheme with partial delegated which is based on discrete logarithm problem. The detailed scheme is presented as following.

Definition 6. *A proxy signature scheme involves three parties, namely original signer S, proxy signer P and verifier V, and consists of following 3 algorithms and a protocol:*

- $OriginalKeyGen$: This key generation algorithm aims to generate the private-public key pair for the original signer. The algorithm calculates $y = g^x \pmod p$ where $x \in_R \mathbb{Z}_p \backslash \{0\}$ is the secret key of the original signer, p is a big prime number and g is the generator of $\mathbb{Z}_{p^*}$. Finally, the algorithm outputs (x, y) as the private-public key pair for the original signer.
  $OriginalKeyGen$:
  $$\begin{aligned} &x \in_R \mathbb{Z}_p \backslash \{0\} \\ &y \equiv g^x \pmod p \end{aligned}$$
  $Output$: (x, y).

- $Protocol$: This protocol is composed of 3 steps:
  Step 1 $ProxyKeyGen$:

To generate a proxy key pair for the proxy signer P of an original signer S, the original signer S randomly selects a number k from $\mathbb{Z}_p \setminus \{0\}$ and computes $K = g^k \pmod p$ and $z = x + kK \pmod{p-1}$.

Step 2 $ProxyDeliver$:
Then the original signer S sends (z, K) to the proxy signer P via a secure channel.

Step 3 $ProxyKeyVerify$:
After receiving the key pair, the proxy signer P verifies whether g^z and yK^K are congruent modulo p. If the verification is passed, the proxy signer P accepts the proxy private-public key pair. Otherwise, the proxy signer P requests restarting the protocol or aborts the protocol.

The protocol can be illustrated as below:

$$\begin{array}{lcl}
\text{Original Singer } S & & \text{Proxy Signer } P \\
k \in_R \mathbb{Z}_p \setminus \{0\} & & \\
K = g^k \pmod p & & \\
z = x + kK \pmod{p-1} & & \\
 & \xrightarrow{(z,K)} & \\
 & & g^z \overset{?}{\equiv} yK^K \pmod p
\end{array}$$

– $Sign$: To sign a message m_p, the proxy signer P uses z as the secret key and execute the signing operation in an ordinary signature scheme such as ElGamal. Then a created proxy signature is composed of message m_p, signature of the ordinary scheme and the public key of the proxy signer K.
$Sign$:

$$Sign(m_p, z)$$

$Output$: $(m_p, (\text{signature of the ordinary scheme}), K)$.

– $Verify$: To verify the signature created by proxy signer P, the verifier V follow the same verification operation with the original signature scheme except using a new public key y' which can be computed by the congruence $y' \equiv yK^K \pmod p$. If the check is passed, the verifier can confirm the signature is valid. Otherwise, the signature is invalid. The public key y of the original signer is used to generate the new public key for the verifier, so this can assure the involvement of the original signer in proxy signature.
$Verify$:

$$Verify(m, (\text{signature of the ordinary scheme}), K, y')$$

$Output$: $\top$ if the check passed. Otherwise, output $\bot$.

The proxy signature scheme can be applied to many ordinary signature schemes, one example for the application is proxy signature for ElGamal scheme, which will be demonstrated later. Given that the private-public key pair(z, K) of a proxy signer (P) which has been generated and sent by original signer (S) after verification by the P as described in proxy signature scheme above, in this example, there are two phases- sign and verify.

Proxy signature for ElGamal scheme:

- $Sign$: A proxy signer (P) signs using z as the secret key. P randomly chooses a number $w \in_R \mathbb{Z}^*_{p-1}$, computes $r = g^w \pmod p$ and $s' = w^{-1}(m_p - zr) \pmod{p-1}$, then sends $(m_p, (r, s', K))$ to the verifier V.
- $Verify$:The verifier V uses y' to verify the signature created by P, where y' can be computed by the congruence $y' \equiv yK^K \pmod p$(y: public key of original signer)

Then, the signing and verification of proxy signature for ElGamal scheme can be illustrated as following:

Proxy Singer P Verifier V

$$w \in_R \mathbb{Z}^*_{p-1}$$
$$r = g^w \pmod p$$
$$s' = w^{-1}(m_p - zr) \pmod{p-1}$$
$$\xrightarrow{(m_p,(r,s',K))}$$
$$y' \equiv yK^K \pmod p$$
$$g^{m_p} \overset{?}{\equiv} {y'}^r r^{s'} \pmod p$$

Another aspect of proxy signature has to be addressed is that it is possible for a dishonest proxy signer to misachieve the signature in some situation. So it is neccessary, in this case, to have a way to identify the deviating proxy signer and revoke the signing capability of the abused proxy signer.

In proxy signature scheme, there have two approches for proxy revocation. The one is to make a revocation list which can be seen publicly. The original signer can revoke the signing capability of an abused proxy signer by putting a mark in the revocation list according to certain convention with verifiers , then the verifier can know if the proxy signer is a revoked one by checking the revocation list. The other is to change the public key of the original signer, and accordingly update all proxies of honest proxy signers. It is noted that both methods require fast delivery of revocation information from original signer to all potential verifiers [9].

2.3 Designated Verifier Signature

Before the formal definition and scheme outline of the Designated Verifier Signature (DVS) scheme it is important to note and discuss briefly another digital signature scheme, which is the Undeniable Signature (US) scheme [2,4]. The US scheme has a unique property compared to the DVS scheme in that to check the validity or invalidity of a signature, interaction with the signer must occur by the verifier. The signer has the power to determine only when a signature can be verified but not who can verify that signature. So there is an interaction between the verifier and the signer, something that the DVS scheme does not have as will be described later in the paper [7,8].

The Designated Verifier Signature scheme proposed by Jakobsson et al. [7] tries to find a solution to this authentication and privacy issue of having the interaction between signer and verifier for the validity of a digital signature. The idea is to have a proof or challenge in which there is concrete evidence that is irrefutable by the verifying party and only that verifying party in determining the validity of the digital signature. This proof or challenge makes the Designated Verifier Signature scheme non-interactive as the signer does not need to provide further evidence to the designated verifier other than this proof or challenge accompanied by the signature. So this digital signature scheme, DVS, contains all the desired properties of an ordinary digital signature scheme, the Undeniable Signature scheme, and also non-interactive for verifying the validity of a signature [7].

Breaking down DVS for simplicity and clarification how DVS works is that you have two parties, the signer and the designated verifier; we will call the parties Alice and Bob respectively. When Alice wants to send some message to another party (in this case it is Bob) it sends along two pieces of information; the message signed by Alice and a challenge, θ, that only Bob will know what is true. The challenge essentially can be broken down into this statement; Instead of proving θ, Alice will prove the statement "*Either θ is true, or I am Bob*". After seeing this challenge Bob will trust that θ is true and thus ensuring that the signature is correct and most importantly from Alice if Bob has not signed this message before. However, if Bob passes the same message and challenge to another verifier (Cindy) then Cindy cannot determine what is true as Bob can prove that he is Bob. We get the desired non-interactive Designated Verifier Signature scheme [7].

Below is the Designated Verifier Signature definitions and scheme [7] that was described above and will be used later for our constructing our own digital signature scheme but detailed here for clarification and understanding:

Definition 7. *A designated verifier, V; is one where you have the signer, S, and a protocol (P_S, P_V) where S proves a statement π to V. V is considered a designated verifier if we have another verifier party, V_2 and two protocols, (P_S, P_V, P_{V_2}) and (P'_V, P_{V_2}) where V can produce identical transcript and V_2 cannot distinguish between the two; while V tries to prove the statement validity to V_2.*

Definition 8. *For a non-interactive Designated Verifier Signature scheme a Trap-door Commitment scheme needs to be used where the Trap-door Commitment scheme, c, is used while using the public key, pk_v, of the designated verifier.*

Definition 9. *A Designated Verifier Signature scheme involves two parties, namely the signer S, and the designated verifier V, and consists of the following five algorithms and is computationally secure based on the assumption of the Discrete Logarithm Problem (DLP):*

– $ParamGen$: The system Parameter Generation Algorithm (PGA) takes the system security parameter k as the input and outputs the system parameters appropriately. This PGA takes the system security parameter k as the input and outputs (p, q, g, H) where p and q are two large prime numbers such that $p = 2q + 1$, g is a generator of $\mathbb{G}_q \subset \mathbb{Z}_p$ and H is hash function that $H : \mathbb{Z}_p^3 \rightarrow \mathbb{Z}_q$.

$$param \leftarrow ParamGen(1^k)$$

$$param = (p, q, g, H) \leftarrow ParamGen(1^k)$$

– $KeyGen$: The Key Generation Algorithm (KGA) takes the system security parameter k as the input and generates private-public key pairs (sk_i, pk_i) for users U_i where U is a signer S or a designated verifier V. This KGA takes the system security parameter k as the input. The algorithm randomly selects a number x_i from $\mathbb{Z}_q$, and computes

$y_i = g^{x_i} \mod p$. Finally, the algorithm outputs (x_i, y_i) as the private-public key pair for a user U_i (signer S or a designated verifier V). $KeyGen(1^k)$:

$$(sk_i, pk_i), \leftarrow KeyGen(1^k)$$

$$\begin{aligned} &x_i \in_R \mathbb{Z}_q \\ &y_i = g^{x_i} \mod p \end{aligned}$$

Output: $(sk_i, pk_i) = (x_i, y_i)$.

- $Sign$: The signing algorithm takes a message m, the signer's private and public key pair (sk_s, pk_s), and the designated verifier's public key pk_v as the input, and generates a signature σ^* on m and a non-interactive zero knowledge proof π showing that σ^* is valid or the signer has the private key sk_v of the designated verifier V. The designated verifier signature σ is (σ^*, π). Signing a message m, the signer S calculates the basic signature $s = m^z \mod p$ using its private key z. The signer S also produces a non-interactive zero knowledge proof π and sets $\sigma = (s, \pi)$ as the DVS signature. $Sign(m, sk_s, pk_v)$:

$$\pi = PK\{(x_s, x_v) : (s = m^{x_s} \wedge y_s = g^{x_s}) \vee y_v = g^{x_v}\}$$

$$\sigma = (\sigma^*, \pi) \leftarrow Sign(m, sk_s, pk_v)$$

$$\begin{aligned} &(z, y_v, y_s) \leftarrow (sk_s, pk_v, pk_s) \\ &s = m^z \mod p \\ &w, r, t \in_R \mathbb{Z}_q \\ &c = g^w {y_v}^r \mod p \\ &G = g^t \mod p \\ &M = m^t \mod p \\ &h = H(c, G, M) \\ &d = t + z(h + w) \mod q \end{aligned}$$

Output: $\sigma = (\sigma^*, \pi) = (s, (w, r, G, M, d))$.

- $Verify$: The Proof Verification Algorithm (VPA) takes a message m, a designated verifier signature σ, the designated verifier's public key pk_v, and the signer's public key pk_p as the input, and outputs $\top$ (accept) or $\perp$ (reject). If the proof π in the signature σ is valid, the algorithm outputs $\top$. Otherwise, the algorithm outputs $\perp$. Remark that this Proof Verification Algorithm is equivalent to the Signature Verification Algorithm if and only if pk_v is the public key of the designated verifier who runs this algorithm and the corresponding private key is not leaked to any one. To verify the proof π in a signature σ on a message m, the designated verifier calculates and verifies the following equations. If all the checks are passed, the designated verifier accepts the proof. Otherwise, the proof is rejected. $Verify(m, \sigma, pk_v, pk_s)$:

$$b \leftarrow Verify(m, \sigma, pk_v, pk_s) \quad b \in \{\top, \perp\}$$

$$\begin{aligned}
&((\sigma^*, \pi), y_v, y_s) \leftarrow (\sigma, pk_v, pk_s)\\
&(s, (w, r, G, M, d)) \leftarrow (\sigma^*, \pi)\\
&c = g^w {y_v}^r \mod p\\
&h = H(c, G, M)\\
&G(y_s)^{h+w} \overset{?}{\equiv} g^d \pmod p\\
&Ms^{h+w} \overset{?}{\equiv} m^d \pmod p
\end{aligned}$$

Output: $\top$ if all checks passed. Otherwise, output $\perp$.

- $Simulate$: The Transcript Simulation Algorithm (TSA) takes a message m, a designated verifier's private-public key pair (sk_v, pk_v), and the signer's public key pk_s as the input, and output a designated verifier signature σ that is indistinguishable with the signature generated by $Sign(m, sk_s, pk_v)$. In the DVS scheme it is crucial that the designated verifier can simulate and generate an indistinguishable signature so the process in which the designated verifier V can simulate the transcript for arbitrary signature s is done using following calculations. $Simulate(m, sk_v, pk_v, pk_s)$:

$$\sigma \leftarrow Simulate(m, sk_v, pk_v, pk_s)$$

$$\begin{aligned}
&(x_v, y_v, y_s) \leftarrow (sk_v, pk_v, pk_s)\\
&s \in_R \mathbb{Z}_p\\
&d, \alpha, \beta \in_R \mathbb{Z}_q\\
&c = g^\alpha \mod p\\
&G = g^d (y_s)^{-\beta} \mod p\\
&M = m^d s^{-\beta} \mod p\\
&h = H(c, G, M)\\
&w = \beta - h \mod q\\
&r = (\alpha - w) x_v^{-1} \mod q
\end{aligned}$$

Output: $\sigma = (\sigma^*, \pi) = (s, (w, r, G, M, d))$.

Note 1. For the DVS scheme to be correct and ensure that the designated verifier V is able to simulate the signature the following must always be held true (see below); otherwise the above scheme would not be a DVS scheme under Jakobsson et al. [7] which is the basis for the DVS scheme above and the DVS scheme that we will be using to create our own signature scheme.

$$\begin{aligned}
\top &= Verify(m, Sign(m, sk_s, pk_v, pk_s), pk_v, pk_s)\\
\top &= Verify(m, Simulate(m, sk_v, pk_v, pk_s), pk_v, pk_s)
\end{aligned}$$

2.4 Short Designated Verifier Proxy Signature

Designated verifier proxy signature (DVPS) is a combination of proxy signature (PS) and designated verifier signature (DVS); the original signer can delegate its signing capability to the proxy signer who can then create a signature that only the designated verifier can verify it.

DVPS has extensive application in e-business. Here is an application to the scenario in section 1. Alice, the founder of a company, delegates her signing capability related to the financial department to the chief financial officer, Bob. After producing a financial risk analysis report in the company, Bob signs the report and produces a DVPS signature for the chief operating officer, Cindy. Due to the property of DVS in DVPS, the signature can only be verified by Cindy. If Cindy is malicious that she sells the report to David in other company for her own profit, David will not accept it since the report and the signature may be produced by Cindy and not Bob that the report may be not genuine and may be dangerous if used in David's company. Besides that, Alice also can delegate her signing capability to other employees in the company (CEO, CTO, CMO, etc.) in order to run the company well. Hence, authenticity and confidentiality of all documentation in the company is protected by DVPS.

As mentioned before, DVPS is actually a combination of PS and DVS. Thus, Wang extends a proxy signature scheme and a designated verifier signature scheme into a designated verifier proxy signature scheme [10]. However, the signature size is relatively large. Hence, Huang et al. constructed a DVPS scheme directly from elliptic curves and bilinear pairings, and the resulted signature size is very small [6]. They named their DVPS scheme as *short designated verifier proxy signature* (SDVPS).

Here is a comparison between Wang's DVPS scheme and SDVPS scheme. The signature of Wang's scheme is (r_p, K, D, s) where $r_p, K, D \in \mathbb{Z}_p$ and $s \in \mathbb{Z}_q$ and the signature of SDVPS is $\sigma \in \mathbb{Z}_q$. Let $|x|$ denote the bit length of x, $p = 1024$ bits and $q = 160$ bits. Then we have,

	Size (groups)	Size (bits)	
DVPS	$3\|\mathbb{Z}_p\| + \|\mathbb{Z}_q\|$	3232	$p = 1024$ bits
SDVPS	$\|\mathbb{Z}_q\|$	160	$q = 160$ bits

Table 1. Signature size comparison [6]

The dramatic signature size decrease makes it more practical in network transmission.

Both DVPS and SDVPS schemes work in delegation by warrant. Unlike DVPS, the SDVPS is based on GBDH problem (see section 2.1). The detailed SDVPS scheme by Huang et al. [6] involves the original signer Alice, the proxy signer Bob and the designated verifier Cindy, and consists of following protocol/algorithms.

- $ParamGen$: The system parameter generation algorithm takes l as the system security parameter and outputs the system public parameters $(\mathbb{G}_1, \mathbb{G}_M, q, e, P, H_0, H_1)$. In detail, $\mathbb{G}_1$ is a cyclic additive group of order $q \geq 2^l$. $\mathbb{G}_M$ is a multiplicative group of order q. e is a bilinear map where $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_M$. P is a generator of $\mathbb{G}_1$. H_0 and H_1 are two cryptographic hash functions where $H_0 : \{0,1\}^* \to \mathbb{G}_1$ and $H_1 : \{0,1\}^* \to \mathbb{Z}_q^*$.

$$(\mathbb{G}_1, \mathbb{G}_M, q, e, P, H_0, H_1) \leftarrow ParamGen(1^l)$$

- $KeyGen$: The key generation algorithm takes the security parameter l as the input and outputs a private-public key pair $(x_i, P_i = x_i P)$ for a user ($i = A$ for Alice, $i = B$ for

Bob and $i = C$ for Cindy) where x_i is randomly selected from $\mathbb{Z}_q^*$.

$$(x_i, P_i = x_iP) \leftarrow KeyGen(1^k) \quad x_i \in_R \mathbb{Z}_q^*, i \in \{A, B, C\}$$

– $ProxyKeyGen$: The proxy key generation protocol involves Alice and Bob and generates a proxy key for Bob in two steps.

Step 1 Alice first computes $Q_B = H_0(ID_B, P_B, m_w)$ using Bob's identity ID_B, Bob's public key P_B and the warrant m_w. Then Alice computes the proxy key $D_{AB} = x_AQ_B$ where x_A is Alice's private key and sends D_{AB} to Bob via an external security channel.

Step 2 Upon receiving the proxy key, Bob checks the equality of $e(D_{AB}, P) \stackrel{?}{=} e(Q_B, P_A)$. If the equation holds, Bob accepts the key and sets his full proxy key as (x_B, D_{AB}).

$$\begin{array}{lcl}
\text{Alice} & & \text{Bob} \\
Q_B = H_0(ID_B, P_B, m_w) & & \\
D_{AB} = x_AQ_B & & \\
 & \xrightarrow{D_{AB}} & \\
 & & e(D_{AB}, P) \stackrel{?}{=} e(Q_B, P_A) \\
 & & proxykey = (x_B, D_{AB})
\end{array}$$

– $Sign$: The deterministic signing algorithm takes the message m, Bob's $proxykey$, identity ID_B, public key P_B, warrant m_w and Cindy's public key P_C as input, and generate the SDVPS signature σ on message m. Remark that since $Q_B = H_0(ID_B, P_B, m_w)$ is always fixed, Bob can pre-compute the value and use this value directly for better performance.

$$\sigma \leftarrow Sign(m, x_b, D_{AB}, ID_B, P_B, m_w, P_C)$$

Since Q_B is always fixed, the algorithm can be simplified as

$$\sigma = H_1(m, e(D_{AB} + x_BQ_B, P_C)) \leftarrow Sign(m, proxykey, Q_B, P_C)$$

– $Verify$: The deterministic verification algorithm takes a message m, a signature σ, Cindy's private key x_C, Bob's identity ID_B, public key P_B, warrant m_w and Alice's public key P_A as input, and outputs $\top$ (accept) or $\bot$ (reject) by checking the equality of $\sigma \stackrel{?}{=} H_1(m, e(x_CQ_B, P_A + P_B))$. If the equation holds, then the signature σ is valid for message m and the algorithm outputs $\top$. Otherwise, the signature is rejected and the algorithm outputs $\bot$.
$Verify(m, \sigma, x_C, ID_B, P_B, m_w, P_A)$:

$$Q_B = H_0(ID_B, P_B, m_w)$$
$$\sigma \stackrel{?}{=} H_1(m, e(x_CQ_B, P_A + P_B))$$

Output: $\top$ if the equation holds. Otherwise, output $\bot$.

To ensure that the SDVPS has the property of *correctness*,

$$\top = Verify(m, Sign(m, x_b, D_{AB}, ID_B, P_B, m_w, P_C), x_C, ID_B, P_B, m_w, P_A)$$

we have

$$\begin{aligned}
&Q_B = H_0(ID_B, P_B, m_w)\\
&H_1(m, e(x_c Q_B, P_A + P_B)\\
&= H_1(m, e(x_c Q_B, x_A P + x_B P))\\
&= H_1(m, e((x_A + x_B) Q_B, x_c P))\\
&= H_1(m, e(D_{AB} + x_B Q_B, P_C))
\end{aligned}$$

In addition, the SDVPS has the property of *transcript simulation* derived from DVS since Cindy can always create signatures on arbitrary messages by a $Simulate$ algorithm which is actually a part of the $Verify$ algorithm. Instead of checking the equality of the equation, Cindy outputs $H_1(m, e(x_C Q_B, P_A + P_B))$ as the signature σ on message m.

– $Simulate(m, \sigma, x_C, ID_B, P_B, m_w, P_A)$:

$$\begin{aligned}
Q_B &= H_0(ID_B, P_B, m_w)\\
\sigma &= H_1(m, e(x_C Q_B, P_A + P_B))
\end{aligned}$$

Output: σ.

2.5 Proposed scheme

As described in section 2.2, proxy signatures can be classified into three classes based on the application: *full delegation*, *partial delegation* and *delegation by warrant*. In this subsection, we propose a new designated verifier proxy signature constructed from Mambo et al.'s proxy signature schemes [9], which is classified as *partial delegation*.

The key point of Mambo el al.'s proxy scheme is that it can be applied to any signatures based on the discrete logarithm problem. Since the designated verifier signature proposed by Jakobsson et al. [7] is based on the undeniable signature, which is also based on the discrete logarithm problem, we are able to convert the proxy signature to DVPS using Jakobsson et al.'s DVS scheme (JSI scheme). To be more specific, we use the proxy generation, proxy delivery and proxy verification parts of the proxy signature and the whole DVS scheme. In addition, there is a known attack by Lipmaa, Wang and Bao [8], showing that the JSI scheme is not disavowable. To counter this problem, we also take the advantage of Lipmaa et al.'s suggestion of adding the signature and the public keys to the hash function.

Outline Here is the outline of our proposed scheme and we follow the structure of [6].

Definition 10. *A designated verifier proxy signature scheme (DVPS), involving original signers S, proxy signers P and verifiers V, consists of following six algorithm or protocols:*

– $ParamGen$: The system parameter generation algorithm takes the system security parameter k as the input and outputs the system parameters.

$$param \leftarrow ParamGen(1^k)$$

– $KeyGen$: The key generation algorithm takes the system security parameter k as the input and generates private-public key pairs (sk_i, pk_i) for users U_i where U is a original signer S or a verifier V.

$$(sk_i, pk_i), \leftarrow KeyGen(1^k)$$

– $ProxyKeyGen_{(S_i,P^i_j)}$: In the proxy key generation protocol, the original signer S_i first uses his private key sk_{s_i} to create a proxy private-public key pair $(sk^{s_i}_{p_j}, pk^{s_i}_{p_j})$ for a proxy signer P^i_j. Then S_i sends $(sk^{s_i}_{p_j}, pk^{s_i}_{p_j})$ to P^i_j via a secure channel. Upon receiving the proxy key pair $(sk^{s_i}_{p_j}, pk^{s_i}_{p_j})$, the proxy signer P^i_j checks the validity of the proxy key pair using the original signer's public key pk_{s_i}. If the proxy key is invalid, the protocol is restarted or aborted.

$$(sk^{s_i}_{p_j}, pk^{s_i}_{p_j}) \leftarrow ProxyKeyGen_{(S_i,P^i_j)}(sk_{s_i}, pk_{s_i})$$

– $Sign$: The signing algorithm takes a message m, a proxy signer's private-public key pair (sk^s_p, pk^s_p), a verifier's public key pk_v, and a corresponding original signer's public key pk_s as the input, and generates a signature σ^* on m and a non-interactive zero knowledge proof π showing that σ^* is valid or the signer has the private key sk_v of the verifier V. The designated verifier proxy signature σ is (σ^*, π).

$$\sigma = (\sigma^*, \pi) \leftarrow Sign(m, sk^s_p, pk_v, pk^s_p, pk_s)$$

– $Verify$: The proof verification algorithm takes a message m, a designated verifier proxy signature σ, the designated verifier's public key pk_v, a proxy signer's public key pk^s_p and a corresponding original signer's public key pk_s as the input, and outputs $\top$ (accept) or $\perp$ (reject). If the proof π in the signature σ is valid, the algorithm outputs $\top$. Otherwise, the algorithm outputs $\perp$.

$$b \leftarrow Verify(m, \sigma, pk_v, pk^s_p, pk_s) \quad b \in \{\top, \perp\}$$

Remark that this proof verification algorithm is equivalent to the signature verification algorithm if and only if pk_v is the public key of the verifier who runs this algorithm and the corresponding private key is not leaked to any one.

– $Simulate$: The transcript simulation algorithm takes a message m, a verifier's private-public key pair (sk_v, pk_v), a proxy signer's public key pk^s_p and a corresponding original signer's public key pk_s as the input, and outputs a designated verifier proxy signature σ that is indistinguishable with the signature generated by $Sign(m, sk^s_p, pk_v)$.

$$\sigma \leftarrow Simulate(m, sk_v, pk_v, pk^s_p, pk_s)$$

To ensure the properties of *correctness* and *transcript simulation generation*, we require the following statements are always true, respectively.

$$\top = Verify(m, Sign(m, sk^s_p, pk_v, pk^s_p, pk_s), pk_v, pk^s_p, pk_s)$$
$$\top = Verify(m, Simulate(m, sk_v, pk_v, pk^s_p, pk_s), pk_v, pk^s_p, pk_s)$$

Scheme Based on the discrete logarithm problem (DLP) assumption, we construct our DVPS scheme as following.

– $ParamGen$: The system parameter generation algorithm takes the system security parameter k as the input and outputs (p, q, g, H) where p and q are two large prime numbers s.t. $p = 2q + 1$, g is a generator of $\mathbb{G}_q \subset \mathbb{Z}_p$ and H is a hash function that $H : \mathbb{Z}_p^7 \rightarrow \mathbb{Z}_q$.

$$param = (p, q, g, H) \leftarrow ParamGen(1^k)$$

– $KeyGen$: The key generation algorithm takes the system security parameter k as the input. The algorithm randomly selects a number x_i from $\mathbb{Z}_q$, and computes $y_i = g^{x_i} \mod p$. Finally, the algorithm outputs (x_i, p_i) as the private-public key pair for a user U_i (a original signer S or a verifier V).
$KeyGen(1^k)$:

$$\begin{aligned} &x_i \in_R \mathbb{Z}_q \\ &y_i = g^{x_i} \mod p \end{aligned}$$

Output: $(sk_i, pk_i) = (x_i, y_i)$.

– $ProxyKeyGen_{(S_i, P_j^i)}$: To generate a proxy key pair for the jth proxy signer of an original signer S_i, the original signer S_i randomly selects a number k from $\mathbb{Z}_q \setminus \{0\}$ and computes $K_j^i = g^k \mod p$ and $z_j^i = x_i + kK_j^i \mod q$. Then the original signer S_i sends (z_j^i, K_j^i) to the proxy signer P_j^i via a secure channel. After receiving the key pair, the proxy signer P_j^i verifies whether $g^{z_j^i}$ and $y_i {K_j^i}^{K_j^i}$ are congruent modulo p. If the verification is passed, the proxy signer P_j^i accepts the proxy private-public key pair. Otherwise, the proxy signer P_j^i requests restarting the protocol or aborts the protocol.
$ProxyKeyGen_{(S_i, P_j^i)}(sk_{s_i}, pk_{s_i})$:

Original Singer S_i Proxy Signer P_j^i

$$\begin{aligned} &x_i \leftarrow sk_{s_i} \\ &k \in_R \mathbb{Z}_q \setminus \{0\} \\ &K_j^i = g^k \mod p \\ &z_j^i = x_i + kK_j^i \mod q \end{aligned}$$

$$\xrightarrow{(z_j^i, K_j^i)}$$

$$\begin{aligned} &y_i \leftarrow pk_{s_i} \\ &g^{z_j^i} \overset{?}{\equiv} y_i {K_j^i}^{K_j^i} \pmod p \end{aligned}$$

Output: $(sk_{p_j}^{s_i}, pk_{p_j}^{s_i}) = (z_j^i, K_j^i)$.

– $Sign$: To sign a message m, the proxy signer P calculates the basic signature $s = m^z \mod p$ using its private key z. The proxy signer P also produces a non-interactive zero knowledge proof π and set $\sigma = (s, \pi)$ as the DVPS signature.

$$\pi = PK\{(x_s, x_v) : (s = m^{x_s} \wedge y_s = g^{x_s}) \vee y_v = g^{x_v}\}$$

$Sign(m, sk_p^s, pk_v, pk_p^s, pk_s)$:

$$\begin{aligned}
&(z, y_v, K, y_s) \leftarrow (sk_p^s, pk_v, pk_p^s, pk_s)\\
&s = m^z \mod p\\
&w, r, t \in_R \mathbb{Z}_q\\
&c = g^w {y_v}^r \mod p\\
&G = g^t \mod p\\
&M = m^t \mod p\\
&h = H(c, G, M, s, y_v, K, y_s)\\
&d = t + z(h + w) \mod q
\end{aligned}$$

Output: $\sigma = (\sigma^*, \pi) = (s, (w, r, G, M, d))$.

– $Verify$: To verify the proof π in a signature σ on a message m, the verifier calculates and verifies the following equations. If all checks are passed, the verifier accepts the proof. Otherwise, the proof is rejected.
  $Verify(m, \sigma, pk_v, pk_p^s, pk_s)$:

$$\begin{aligned}
&((\sigma^*, \pi), y_v, K, y_s) \leftarrow (\sigma, pk_v, pk_p^s, pk_s)\\
&(s, (w, r, G, M, d)) \leftarrow (\sigma^*, \pi)\\
&c = g^w {y_v}^r \mod p\\
&h = H(c, G, M, s, y_v, K, y_s)\\
&G(y_s K^K)^{h+w} \overset{?}{\equiv} g^d \pmod p\\
&M s^{h+w} \overset{?}{\equiv} m^d \pmod p
\end{aligned}$$

Output: $\top$ if also checks passed. Otherwise, output $\perp$.

– $Simulate$: The designated verifier V can simulate the transcript for an arbitrary signature s by following calculations:
  $Simulate(m, sk_v, pk_v, pk_p^s, pk_s)$:

$$\begin{aligned}
&(x_v, y_v, K, y_s) \leftarrow (sk_v, pk_v, pk_p^s, pk_s)\\
&s \in_R \mathbb{Z}_p\\
&d, \alpha, \beta \in_R \mathbb{Z}_q\\
&c = g^\alpha \mod p\\
&G = g^d (y_s K^K)^{-\beta} \mod p\\
&M = m^d s^{-\beta} \mod p\\
&h = H(c, G, M, s, y_v, K, y_s)\\
&w = \beta - h \mod q\\
&r = (\alpha - w) x_v^{-1} \mod q
\end{aligned}$$

Output: $\sigma = (\sigma^*, \pi) = (s, (w, r, G, M, d))$.

Properties Our scheme meets the properties of *correctness* and *transcript simulation generation*. Remark that all calculations below are done in $\mathbb{Z}_p$, that is, modulo p.

– We first show the correctness of the proxy key generation in $ProxyKenGen$.

$$g^z = yK^K \quad \text{where } K = g^k, z = x + kK, y = g^x$$

Proof.

$$\begin{aligned} yK^K &= g^x(g^k)^K \\ &= g^x g^{kK} \\ &= g^{x+kK} \\ &= g^z \end{aligned}$$

□

– *Correctness*:

$$\top = Verify(m, Sign(m, sk_p^s, pk_v, pk_p^s, pk_s), pk_v, pk_p^s, pk_s)$$

Proof.

$$\begin{aligned} G(y_s K^K)^{h+w} &= g^t(g^z)^{h+w} \\ &= g^{t+z(h+w)} \\ &= g^d \end{aligned}$$

$$\begin{aligned} Ms^{h+w} &= m^t(m^z)^{h+w} \\ &= m^{t+z(h+w)} \\ &= m^d \end{aligned}$$

□

– *Transcript simulation generation*:

$$\top = Verify(m, Simulate(m, sk_v, pk_v, pk_p^s, pk_s), pk_v, pk_p^s, pk_s)$$

Proof.

$$\begin{aligned} G(y_s K^K)^{h+w} &= g^d(y_s K^K)^{-\beta}(y_s K^K)^{h+\beta-h} \\ &= g^d(y_s K^K)^{-\beta+h+\beta-h} \\ &= g^d \end{aligned}$$

$$\begin{aligned} Ms^{h+w} &= m^d s^{-\beta} s^{h+\beta-h} \\ &= m^d s^{-\beta+h+\beta-h} \\ &= m^d \end{aligned}$$

□

3 Discussion

In this section, we discuss the security of our proposed scheme in section 2.5. Then we compare partial delegation with delegation by warrant.

3.1 Security

As Mambo et al.'s proxy signature [9] is proven secure for any underlying secure signatures under DLP assumption and the designated verifier signature is proven secure [7,8] under DLP assumption, by the construction, our proposed scheme is secure under DLP assumption.

For the following three types of adversaries:

Type I: This type of adversaries only knows the public keys of the original signer and the corresponding proxy signers and tries to forge a signature of the original signer and the corresponding proxy signers. Remark that a scheme secure against Type II and Type III adversaries implies that it is secure against Type I adversaries.

Type II: This type of adversaries knows the public keys of the original signer and private-public key pairs of the corresponding proxy signers. The adversary tries to forge a signature of the original signer.

Type III: This type of adversaries knows the private-public key pair of the original signer and public keys of the corresponding proxy signers. The adversary tries to forge a signature of the original signer.

Our proposed scheme is secure against all types of adversaries, since all attacks can be reduced to the discrete logarithm problem, which is assumed hard. However, our scheme is not secure against Type III adversaries if proxy private-public key pairs are generated after the adversary getting the private key of the original signer as a honest but curious original signer may record every random k involved in the algorithm $ProxyKeyGen$.

Another security concern is the distribution of the public keys of the proxy signers. In Mambo et al.'s proxy signature [9], the public key K of the proxy signer is sent to the verifier along with the basic signature σ as $\zeta = (\sigma, K)$. Here is the scenario: A malicious proxy signer P_1 may sign a message m with its proxy private key and declare the signature is signed by another proxy signer P_2 by declaring K_1 is P_2's proxy public key. Unless the verifier launches a standard *disavowal* protocol with P_2, the verifier cannot know whether the message is really signed by P_2 or not. However, the verifier may not care about this as all proxy signers function the same for the verifier. If a proxy signer tries to impersonate another proxy signer, we can check the real identity of the proxy signer by the *disavowal* protocol later. Besides the backward checking approach, we could also have forward checking approaches. Either a Public-Key Infrastructure (PKI) or the original signer can produce a certificate for the proxy public key. In this case, the scheme works as delegation by warrant not partial delegation and the size of the transmission package is increased. Hence, in our scheme, we use the backward checking approach.

3.2 Delegation

Since full delegation is not secure, we only discuss partial delegation and delegation by warrant in this subsection.

As mentioned by Mambo et al. [9], unlike delegation by warrant, the original signer cannot put a limit on partial delegation, including limiting the validity period of a proxy public key. Thus, key revocation is a hard problem. Mambo et al. proposed two solutions. One is to let the original signer maintain a revocation list of proxy public keys and then verifiers can verify it. Another is to change the public key of the original signer. Our scheme may use Mambo et al.'s solutions.

Someone may think that delegation by warrant is better than partial delegation. However, it depends on the demand. As mentioned in section 3.1, partial delegation does not require PKI for the proxy signer, whereas delegation by warrant does. In terms of size, partial delegation only increases the size of the original signature by one group element while delegation by warrant increases the size by a full certificate of the proxy signer's public key and the warrant itself. Since the certificate of the proxy signer's public key is fixed and is not necessary to be included in the signature as PKI handles it, only the warrant is required to be sent along with the signature. However, the warrant is larger than one group element in general. Hence, partial delegation is less secure but more efficient than delegation by warrant.

4 Conclusion

In this paper, we have reviewed the preliminary on Bilinear Pairing, Non-interactive Zero Knowledge Proof (NIZK) and security assumptions including Discrete Logarithm Problem (DLP), Bilinear Diffie-Hellman (BDH) Problem, Decisional Bilinear Diffie-Hellman (DBDH) Problem and Gap Bilinear Diffie-Hellman (GBDH) Problem in section 2.1. Then Proxy Signature, Designated Verifier Signature, as well as Short Designated Verifier Proxy Signature have been explained with Mambo et al.'s scheme, Jakobsson et al.s scheme (JSI scheme) and Huang et al.'s scheme respectively in sections 2.2 to 2.4. After that, we proposed a new designated verifier proxy signature with partial delegation based on Mambo et al.s proxy signature scheme and Jakobsson et al.s DVS scheme (JSI scheme) in section 2.5. Finally, security concerns of our DVPS scheme is discussed that our scheme maybe not be secure against TYPE III adversaries in section 3.

References

1. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 103–112. STOC '88, ACM, New York, NY, USA (1988)
2. Chaum, D., Antwerpen, H.: Undeniable signatures. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings, Lecture Notes in Computer Science, vol. 435, pp. 212–216. Springer New York (1990), http://dx.doi.org/10.1007/0-387-34805-0_20
3. Diffie, W., Hellman, M.: New directions in cryptography. Information Theory, IEEE Transactions on 22(6), 644–654 (1976)
4. Duan, S.: Certificateless undeniable signature scheme. Information Sciences 178(3), 742 – 755 (2008), ¡ce:title¿Including Special Issue "Ambient Intelligence"¡/ce:title¿
5. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the seventeenth annual ACM symposium on Theory of computing. pp. 291–304. STOC '85, ACM, New York, NY, USA (1985)
6. Huang, X., Mu, Y., Susilo, W., Zhang, F.: Short designated verifier proxy signature from pairings. In: Enokido, T., Yan, L., Xiao, B., Kim, D., Dai, Y., Yang, L. (eds.) Embedded and Ubiquitous Computing – EUC 2005 Workshops, Lecture Notes in Computer Science, vol. 3823, pp. 835–844. Springer Berlin Heidelberg (2005)

7. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: Maurer, U. (ed.) Advances in Cryptology — EUROCRYPT '96, Lecture Notes in Computer Science, vol. 1070, pp. 143–154. Springer Berlin Heidelberg (1996)
8. Lipmaa, H., Wang, G., Bao, F.: Designated verifier signature schemes: Attacks, new security notions and a new construction. In: Caires, L., Italiano, G., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 3580, pp. 459–471. Springer Berlin Heidelberg (2005)
9. Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures for delegating signing operation. In: Proceedings of the 3rd ACM conference on Computer and communications security. pp. 48–57. CCS '96, ACM, New York, NY, USA (1996)
10. Wang, G.: Designated-verifier proxy signature schemes. In: Sasaki, R., Qing, S., Okamoto, E., Yoshiura, H. (eds.) Security and Privacy in the Age of Ubiquitous Computing, IFIP Advances in Information and Communication Technology, vol. 181, pp. 409–423. Springer US (2005)