(Strong) multi-designated verifiers signatures secure against rogue key attack

Yunmei Zhang
*University of Wollongong, yz528@uow.edu.au*

Man Ho Allen Au
*University of Wollongong, aau@uow.edu.au*

Guomin Yang
*University of Wollongong, gyang@uow.edu.au*

Willy Susilo
*University of Wollongong, wsusilo@uow.edu.au*

Follow this and additional works at: [https://ro.uow.edu.au/eispapers](https://ro.uow.edu.au/eispapers)

Part of the **Engineering Commons, and the Science and Technology Studies Commons**
(Strong) multi-designated verifiers signatures secure against rogue key attack

Abstract
Designated verifier signatures (DVS) allow a signer to create a signature whose validity can only be verified by a specific entity chosen by the signer. In addition, the chosen entity, known as the designated verifier, cannot convince any body that the signature is created by the signer. Multi-designated verifiers signatures (MDVS) are a natural extension of DVS in which the signer can choose multiple designated verifiers. DVS and MDVS are useful primitives in electronic voting and contract signing. In this paper, we investigate various aspects of MDVS and make two contributions. Firstly, we revisit the notion of unforgeability under rogue key attack on MDVS. In this attack scenario, a malicious designated verifier tries to forge a signature that passes through the verification of another honest designated verifier. A common counter-measure involves making the knowledge of secret key assumption (KOSK) in which an adversary is required to produce a proof-of-knowledge of the secret key. We strengthened the existing security model to capture this attack and propose a new construction that does not rely on the KOSK assumption. Secondly, we propose a generic construction of strong MDVS.

Keywords
signatures, verifiers, designated, attack, multi, key, strong, rogue, against, secure

Disciplines
Engineering | Science and Technology Studies

Publication Details

This journal article is available at Research Online: https://ro.uow.edu.au/eispapers/186
Designated verifier signatures (DVS) allow a signer to create a signature whose validity can only be verified by a specific entity chosen by the signer. In addition, the chosen entity, known as the designated verifier, cannot convince any body that the signature is created by the signer. Multi-designated verifiers signatures (MDVS) are a natural extension of DVS in which the signer can choose multiple designated verifiers. DVS and MDVS are useful primitives in electronic voting and contract signing. In this paper, we investigate various aspects of MDVS and make two contributions. Firstly, we revisit the notion of unforgeability under rogue key attack on MDVS. In this attack scenario, a malicious designated verifier tries to forge a signature that passes through the verification of another honest designated verifier. A common counter-measure involves making the knowledge of secret key assumption (KOSK) in which an adversary is required to produce a proof-of-knowledge of the secret key. We strengthened the existing security model to capture this attack and propose a new construction that does not rely on the KOSK assumption. Secondly, we propose a generic construction of strong MDVS.

1 Introduction

Designated verifier signatures/proofs (DVS/DVP) were introduced by Jakobsson, Sako and Impagliazzo [12], and independently by Chaum [6] in 1996. A DVS scheme allows a signer Alice to convince a designated verifier Bob that Alice has endorsed the message while Bob cannot transfer this conviction to anyone else. The underlying principle of DVS is that a signature is a non-interactive proof that asserts the validity of the statement “Alice has endorsed a message” or “the signer has Bob’s secret key”. While Bob is convinced that Alice has endorsed the message, he cannot convince Carol as the proof could have been produced by Bob himself. In the same paper, Jakobsson et al. introduce the concept of strong DVS (SDVS) in which the private key of Bob is required to verify the signature. Recall that DVS itself discloses the information that the signature is produced by Alice or Bob. If an external party, Carol, is confident that Bob has not created the signature, she knows Alice has endorsed the message. An example is that the signature is captured by Carol before it reaches Bob. This
requirement is formalized as privacy of signer’s identity in [19]. It is required that without Bob’s private key, Carol cannot tell if a signature is created by Alice or another signer.

In [12], the concept of multiple verifiers has been discussed and in the rump session of Crypto’03, Desmedt [10] proposed the notion of multi-designated verifiers signatures (MDVS) as a generalization of DVS. It was later formalized in [14]. Since then, a number of MDVS constructions [12, 14, 17, 20, 7, 15, 16, 8, 24, 5, 22, 23] with different features in different settings have been proposed. Interested readers may refer to [23] for a survey.

The problem of rogue key attack in DVS was first discussed in [12]. In the discussion, the goal of a malicious verifier Bob is to convince an external party Carol that the signer Alice has endorsed the message. For example, Bob can create his public key as the output of a hash function using a random number as input. Later, when Bob reveals the value of the random number, everyone will be convinced that the signatures must have been created by Alice. One of the counter-measures suggested is to require Bob to prove the knowledge of his secret key. Another type of rogue key attack specifically targeting MDVS was discussed in [21]. In this attack, a malicious verifier Carol creates her public key as a function of other honest verifiers’ public keys so that she could create a signature that passes the verification of other honest verifiers. Again, the suggested counter-measure is to require the verifier to prove the knowledge of her secret key. Note that no formal model has been proposed to capture the attack.

We remark that the two types of rogue key attacks are different in nature. The former is against non-transferability while the latter is against unforgeability. In this paper, our focus is on the latter.

As discussed, a counter-measure against rogue key attack in MDVS is to require the adversary to produce a proof-of-knowledge of the secret key. In practice, this implies all users would have to produce a proof-of-knowledge of the secret key to the certification authority (CA) before the CA certifies the corresponding public key. This solution requires a change in the current PKI and is regarded as costly [2]. Thus, it is desirable to design MDVS secure against rogue key attack in the plain model. In respond to this, we provide a partial solution by proposing the first MDVS scheme that is formally proven unforgeable under rogue key attack.

It is known that if we encrypt the DVS under the designated verifier’s public key, the resulting scheme would be a strong DVS. Nonetheless, a subtle issue discussed in [14] prevent such generic transformation to be applicable to the case of MDVS. Specifically, the challenge is to ensure correctness of the resulting scheme since it is entirely possible for a signer to encrypt different values under different designated verifier’s public key so that a signature could be regarded as valid by some of the designated verifiers only. We tackle this issue with an hybrid encryption using a simple one-way secure encryption and a symmetric encryption and show that the unforgeability under rogue key attack is preserved in our generic transformation. Specifically, we make the following contributions.
1.1 Contribution.

1. We present a formal definition for MDVS that captures existential forgery under rogue key attack.
2. We propose a construction that is provably secure against rogue key attack in our model.
3. We present a generic construction of strong MDVS secure against rogue key attack.

Organization. The rest of the paper is organized as follows. In Section 2, we review the syntax of a MDVS scheme and its security definitions. We discuss the rogue key attack on MDVS, its formal definition and our proposed solution in Section 3. In Section 4, we present a generic construction of strong MDVS. We conclude our paper in Section 5.

2 Preliminary

If \( n \) is a positive integer, we use \([n]\) to denote the set \( \{1, \ldots, n\} \). We review the following well-known computational assumptions.

**Definition 1 (DL Assumption).** Let \( G = \langle g \rangle \) be a cyclic group of prime order \( p \). The discrete logarithm assumption states that given a tuple \((g, Z) \in (G, G)\), it is computationally infeasible to compute the value \( z \in \mathbb{Z}_p \) such that \( Z = g^z \).

**Definition 2 (CDH Assumption).** Let \( G = \langle g \rangle \) be a cyclic group of prime order \( p \). The computational Diffie-Hellman assumption states that given a tuple \((g, g^a, g^b) \in (G, G, G)\), it is computationally infeasible to compute the value \( g^{ab} \).

2.1 Syntax

We adapt the definitions and security models of MDVS from various literatures [14, 15]. A MDVS scheme consists of four algorithms, namely, Setup, Gen, Sign, Verify, whose functions are enumerated below.

\[
\text{param} \leftarrow \text{Setup}(1^\lambda): \text{On input a security parameter } \lambda, \text{this algorithm outputs the public parameter param for the system. Note that this algorithm is optional if all users could generate their key pairs without any coordination. Nonetheless, to the best of our knowledge, all existing schemes requires the users to create their keys based on some commonly known system parameters. We assume param is an implicit input to all algorithms listed below.}
\]

\[
(pk, sk) \leftarrow \text{Gen}(): \text{This algorithm outputs a key pair } (pk, sk) \text{ for a user (who can take the role of a signer or a designated verifier). If } (pk, sk) \text{ is an output of the algorithm Gen()}, \text{we say } pk \text{ is the corresponding public key of sk (and vice versa).}
\]
\((\sigma, V) \leftarrow \text{Sign}(sk_S, V, m)\) : On input a message \(m\), a secret key of a signer \(sk_S\) (whose public key is \(pk_S\)) and a set of designated verifiers’ public keys \(V\), this algorithm outputs a signature \(\sigma\), which is a designated verifier signature of \(m\) with respect to the public key \(pk_S\).

valid/invalid \(\leftarrow \text{Verify}(pk_S, \sigma, V, m, sk_V)\) : On input a public key \(pk_S\), a message \(m\), a signature \(\sigma\) with a set of designated verifiers’ public keys \(V\) and a private key \(sk_V\) such that the corresponding public key \(pk_V \in V\), this algorithm verifies the signature and outputs valid/invalid.

A MDVS scheme must possess Correctness, Unforgeability and Source-Hiding, to be reviewed below.

Correctness. For any security parameter \(\lambda\) and \(\text{param} \leftarrow \text{Setup}(1^\lambda)\), \((pk_S, sk_S) \leftarrow \text{Gen}()\) and \(V = \{pk_{V_1}, \ldots, pk_{V_n}\}\) such that \((pk_i, sk_i) \leftarrow \text{Gen}()\) for \(i \in [n]\). For any message \(m\), if \((\sigma, V) \leftarrow \text{Sign}(sk_S, V, m)\), then valid \(\leftarrow \text{Verify}(pk_S, \sigma, V, m, sk_V)\) for all \(i \in [n]\). Furthermore, for any values \(\sigma, V, m, pk_S\), if there exists a private key \(sk_V\) such that its corresponding public key \(pk_V \in V\) and that valid \(\leftarrow \text{Verify}(pk_S, \sigma, V, m, sk_V)\), then for any private key \(sk_V\), it holds that valid \(\leftarrow \text{Verify}(pk_S, \sigma, V, m, sk_V)\) if the corresponding public key \(pk_V \in V\).

Unforgeability. The following game between a challenger \(C\) and an adversary \(A\) formally captures the requirement of Unforgeability.

Setup \(C\) invokes \(\text{Setup}(1^\lambda)\) and subsequently \(\text{Gen}()\) to obtain \(\text{param} = (pk_S, sk_S)\), \(\{(pk_i, sk_i)\}_{i \in [n]}\). Denote the set \(\{pk_i\}_{i \in [n]}\) by \(V\). \((\text{param}, pk_S, V)\) is given to \(A\).

Query \(A\) is allowed to make the following queries:
- Corruption Query. \(A\) submits a public key \(pk_{V_i} \in V\) and receives \(sk_{V_i}\).
- Signature Query. \(A\) submits a message \(m\) and receives \((\sigma, V) \leftarrow \text{Sign}(sk_S, V, m)\).

Output \(A\) submits \((\sigma^*, m^*)\) and wins if and only if
1. There exists a public key \(pk_{V_i} \in V\) such that valid \(\leftarrow \text{Verify}(pk_S, \sigma^*, V, m^*, sk_{V_i})\).
2. \(A\) has not submitted a Signature Query with input \(m^*\).
3. There exists a public key \(pk_{V_i} \in V\) such that \(A\) has not submitted a Corruption Query as input.

Definition 3 (Unforgeability). A MDVS scheme is unforgeable if no PPT adversary wins the above game with non-negligible probability.

As stated in [14], the adversary is not given an oracle for signature verification as he can verify any signatures by corrupting some of the verifiers.

Source hiding. It means that given a message \(m\) and a signature \((\sigma, V)\), it is infeasible to determine who from the original signer or the designated verifiers all together created the signature, even if all the secret keys are known. The formal definition is adapted from Definition 3 of [11] for normal DVS into that for MDVS.
**Definition 4 (Source Hiding).** A MDVS scheme is source hiding if there exists a PPT simulation algorithm Sim that on input a public key $pk_s$, a set of key pairs $(pk_{V_i}, sk_{V_i})_{i \in [n]}$ and a message $m$, outputs a tuple $(\sigma, V)$ (such that $V = \{pk_{V_i} \}_{i \in [n]}$) that is indistinguishable to $(\sigma, V) \leftarrow \text{Sign}(sk_s, V, m)$ (where $sk_s$ is the corresponding private key of $pk_s$). In other words, for all PPT algorithm $D$, for any security parameter $\lambda$, $\text{param} \leftarrow \text{Setup}(\lambda)$, $(pk_s, sk_s) \leftarrow \text{Gen}()$, $\{(pk_{V_i}, sk_{V_i}) \leftarrow \text{Gen()} \}_{i \in [n]}$ and any message $m$, it holds that:

$$\Pr \left[ (\sigma_0, V) \leftarrow \text{Sign}(sk_s, V, m) \text{ s.t. } b' \leftarrow D(\sigma_0, pk_s, sk_s, \{(pk_{V_i}, sk_{V_i}) \}_{i \in [n]}, m) \text{ s.t. } b = b' \right] - 1/2 = \text{negl}(\lambda)$$

where $\text{negl}(\lambda)$ represents a negligible function in $\lambda$. A function $\text{negl}(\lambda)$ is said to be negligible in $\lambda$ if for all polynomial $q(\lambda)$, there exists a value $k_0$ such that for every $\lambda > k_0$, $\text{negl}(\lambda) < 1/q(\lambda)$.

### 2.2 Strong Multi-Designated Verifiers Signatures

**Strong Multi-Designated Verifiers Signatures.** It is desirable in many scenarios that, besides the signer and the verifier, a third party cannot tell if a signature for the verifier is created by that particular signer or by someone else. This concept appeared in [12] and is formally defined as privacy of signer’s identity (PSI) in [13]. This applies to the case of multiple designated verifiers and the property PSI for MDVS is defined in [14].

**Privacy of signer’s identity.** The following game between a challenger $C$ and an adversary $A$ formally captures the requirement of PSI.

- **Setup** $C$ invokes $\text{Setup}(\lambda)$ and subsequently $\text{Gen}()$ to obtain $(\text{param}, (pk_{S_1}, sk_{S_1}), (pk_{S_2}, sk_{S_2}), \{(pk_{V_i}, sk_{V_i}) \}_{i \in [n]})$. Denote the set $\{pk_{V_i} \}_{i \in [n]}$ by $V$.

- **Query** $A$ is allowed to make the following queries:
  - Verification Query. $A$ submits $(m, \sigma, V, pk_{S_1} : c \in \{0, 1\}, V \in V)$ and receives $\text{valid}/\text{invalid} \leftarrow \text{Verify}(pk_{S_1}, \sigma, V \cup pk_{V_i}, m, sk_{V_i})$.
  - Signature Query. $A$ submits a message $m$, a bit $b$ and receives $(\sigma, V) \leftarrow \text{Sign}(sk_{S_1}, V, m)$.

- **Challenge** At some point $A$ submits a message $m^*$. $C$ flips a fair coin $b$ and returns $(\sigma^*, V) \leftarrow \text{Sign}(sk_{S_2}, V, m)$.

- **Query** $A$ continues to make verification and signature queries.

- **Output** $A$ submits a bit $b'$ and wins if and only if $b' = b$.

$A$’s advantage in the game PSI is defined as the probability that $A$ wins the game minus $1/2$.

**Definition 5 (Privacy of signer’s identity).** A MDVS scheme is said to possess privacy of signer’s identity if no PPT adversary has non-negligible advantage in game PSI.

A strong MDVS scheme is a MDVS scheme that possesses privacy of signer’s identity.
3 Rouge Key Attack in MDVS and its Solution

We first review the generic construction of MDVS from discrete logarithm-based ring signatures [14]. In the next subsection, we describe how a malicious designated verifier could launch a rogue key attack to make an honest verifier into accepting a forged signature. We stress that this attack is outside the original security model and does not imply the scheme is insecure. Rather, we would like to show that a signature that passes the verification of a particular honest designated verifier could have been created by a real signer or some other malicious verifiers. Finally, we propose a fix.

3.1 Generic Construction of MDVS [14]

The generic construction utilizes ring signatures as building blocks and requires that all the keys are discrete logarithm-based. Readers are referred to [18] for the formal definition of a ring signature scheme. Roughly speaking, a ring signature is a signature created from one of the possible signers in a set of signers (often called a ring of signers). The ring of signers are created in an ad-hoc manner by the actual signer. The formation is spontaneous in that the members can be completely unaware of being conscripted into the ring. In the generic construction of MDVS, ring signatures supporting a ring size of 2 is required.

- Setup. This is equivalent to the parameter generation of the ring signature scheme (if any).
- Gen. This is equivalent to the key generation of the ring signature scheme. The generic construction requires the key of the ring signature to be of the form \((g^x, x)\) where \(g^x\) is included in the parameter, \(x\) is the signing key and \(g^x\) is the corresponding public key.
- Sign. Let the signer’s key pair be \((g^x, x_S)\) and the set of designated verifiers’ key pairs be \(\{(g^{x_V_i}, x_{V_i})\}_{i=1}^n\). The signer computes \(g^{X_V} = \prod_{i \in [n]} g^{x_{V_i}}\). Next, the signer creates a ring signature on message \(m\) on the ring \(\{g^{x_S}, g^{x_V}\}\) using the secret key \(x_S\). Denote the output as \(\sigma\). This value, together with the set \(\{g^{x_{V_i}}\}_{i \in [n]}\), is outputted as the multi-designated verifier signature.
- Verify. To verify the signature \((\sigma, \{g^{x_{V_i}}\}_{i \in [n]})\) on message \(m\), a verifier computes \(g^{X_V} = \prod_{i \in [n]} g^{x_{V_i}}\). Then it employs the verification algorithm of the ring signature scheme on the ring \(\{g^{x_S}, g^{X_V}\}\). The unforgeability property comes from the fact that to create a ring signature on the ring \(\{g^{x_S}, g^{x_V}\}\), one needs to know \(x_S\) or \(x_V\). Since the adversary does not know \(x_S\) or \(x_V\), forging a signature implies breaking the unforgeability of the underlying ring signature scheme. On the other hand, the source hiding property comes from the fact that if all secret keys of the verifiers are known, one can construct a PPT Sim which computes \(x_V = \sum_{i \in [n]} x_{V_i}\) and uses it to

\[1\] Since the adversary cannot corrupt all the verifiers, it does not know the value \(x_V\), which is equal to \(\sum_{i \in [n]} x_{V_i}\).
create a ring signature on behalf of the ring \( \{ g^{x_S}, g^{x_V} \} \). Due to the anonymity of ring signature, no PPT algorithm can distinguish a signature created by the real signer using \( x_S \) or by \( \text{Sim} \) using \( x_V \).

### 3.2 Rouge Key Attack and Its Defence

**Existential Forgery under Rouge Key Attack.** Rouge key attack against a concrete scheme in [14] has been discussed in [21]. Here we extend the attack to the generic construction of [14]. Suppose an adversary’s goal is to convince an honest designated verifier into accepting a forged signature. Let \( g^{x_S}, g^{x_{V'}} \) be the public keys of the targeted signer and designated verifier respectively. To cheat the verifier, the adversary randomly generates a value \( x_A \) and crafts a mal-formed public key \( K = g^{x_A}/g^{x_{V'}} \). Next, the adversary computes \( g^{x_V} = Kg^{x_{V'}} = g^{x_A} \).

Since the adversary is in possession of \( x_A \), he can create a ring signature on the ring \( \{ g^{x_S}, g^{x_{V'}} \} \). He outputs the signature, together with the set of designated verifiers as \( \{ g^{x_{V'}}, K \} \). Consequently, the designated verifier would accept a forged signature created by the adversary instead of the signer. We denote attack of this kind as forgery against rogue key attack (RKA).

**A Proposed Fix.** The problem comes from the extra power given to the adversary to create malformed public key. The fix suggest in [21] is to require the certification authority to check the validity of the public key before issuing a digital certificate. In terms of modelling, this implies the stronger certified key model in which the users are required to conduct a proof-of-knowledge of his secret key to the CA. As argue in [2], this requires modification of the client and CA functioning software. We propose another way that could withstand this attack in the plain model based on a technique used in multisignatures [2] and batch verification of digital signatures [1]. In a nutshell, \( g^{x_{V'}} \) is defined to be \( \prod_{i \in [n]} (g^{x_{V_i}})^{h_i} \), where \( h_i = H(g^{x_S}, g^{x_{V_1}}, \ldots, g^{x_{V_n}}, m, i) \) for a hash function \( H \) which shall be modelled as a random oracle. Observe that with this modification, the value \( x_{V'} \) can still be computed if all the values \( x_{V_i} \) are known. On the other hand, if one of the secret keys, say \( x_{V_i} \), is unknown, the value \( x_{V'} \) cannot be computed since the probability of “canceling” \( g^{x_{V_i}} \) in the computation of \( g^{x_{V'}} \) is negligible assuming the values \( h_i \) are randomly distributed and are only known after the value of the public keys are chosen.

### 3.3 Formal Security Definition for Unforgeability Under Rogue Key Attack

To formally assert the security of our proposed solution, we define a security model which intends to capture attack of this kind.\(^2\) We believe a verification query with the target verifier may be of use to the adversary since the adversary might try to submit mal-formed signatures to learn information about the target verifier’s verification procedure.

\(^2\) While rogue key attack on MDVS is discussed in [21], no formal security model has been proposed to capture such an attack.
Unforgeability Against Rogue Key Attack. The following game between a challenger \(C\) and an adversary \(A\) formally captures the requirement of UF-RKA.

**Setup** \(C\) invokes Setup(1\(^\lambda\)) and subsequently Gen() to obtain (param, (pk\(_S\), sk\(_S\)), (pk\(_V\), sk\(_V\)). (param, pk\(_S\), pk\(_V\)) is given to \(A\).

**Query** \(A\) is allowed to make the following queries:
- Verification Query. \(A\) submits a set of public keys \(\mathcal{V}\), a signature \((\sigma, \mathcal{V} \cup pk\(_V\))\), a message \(m\) and receives valid/invalid \(\leftarrow Verify(pk\(_S\), \sigma, \mathcal{V} \cup pk\(_V\), m, sk\(_V\))\).
- Signature Query. \(A\) submits a message \(m\), a set of public keys \(\mathcal{V}\) and receives \((\sigma, \mathcal{V}, m)\). Note that \(A\) can submit an arbitrary set of verifiers of his choice (even a set without \(pk\(_V\))

**Output** \(A\) submits \((\sigma^*, m^*)\) and a set of public keys \(\mathcal{V}^*\) and wins if and only if
1. valid \(\leftarrow Verify(pk\(_S\), \sigma^*, \mathcal{V}^* \cup pk\(_V\), m^*, sk\(_V\))\).
2. \(A\) has not submitted a Signature Query with input \((m^*, \mathcal{V}^* \cup pk\(_V\))\).

**Definition 6 (UF-RKA).** A MDVS scheme is unforgeable under rogue key attack if no PPT adversary wins the above game with non-negligible probability.

We believe UF-RKA for MDVS is a stronger notion compared with the notion Unforgeability.

### 3.4 A Concrete Construction

We present a concrete MDVS scheme from a commonly used two-party ring signature following the generic construction together with our proposed fix.

- **Setup.** Let \(G = \langle g \rangle\) be a cyclic group of prime order \(p\). Output param as \((G, p, g)\).
- **Gen.** Choose a hash function \(H : \{0, 1\}^* \rightarrow \mathbb{Z}_p\) which will be modelled as a random oracle\(^3\). Randomly generate \(x \in \mathbb{Z}_p\), compute \(g^x\). Output pk as \((g^x, H)\) and sk as \(x\).
- **Sign.** On input the signer’s key pair \((pk\(_S\), sk\(_S\))\), a set of designated verifier’s public keys \(\mathcal{V} = \{pk\(_{V_1}\), \ldots, pk\(_{V_n}\)\} and a message \(m\), parse \(pk\(_S\)\) as \((Y\(_S\), H\(_S\))\), \(sk\(_S\)\) as \(x\), \(pk\(_V\)\) as \((Y\(_{V_i}\), H\(_{V_i}\))\). Compute \(Y = \prod_{i \in \mathcal{V}} Y\(_{V_i}\)^{h\(_{V_i}\)}\) where \(h\(_{V_i}\) = H\(_{V_i}\)(pk\(_S\), pk\(_{V_1}\), \ldots, pk\(_{V_n}\), m)\).
  1. Randomly generate \(r, c_2, z_2 \in \mathbb{Z}_p\), compute \(T_1 = g^r\), \(T_2 = Y^x g^{z_2}\).
  2. Compute \(c = H_S(T_1, T_2, pk\(_S\), pk\(_{V_1}\), h_1, \ldots, pk\(_{V_n}\), h_n, Y, m)\) and \(c_1 = c - c_2\).
  3. Compute \(z_1 = r - c_1 x\).
Output the signature as \((c_1, c_2, z_1, z_2, Y)\). Note that \((c_1, c_2, z_1, z_2)\) is a ring signature on message \(m\) with respective to ring \(\{Y\(_S\), Y\}\).

\(^3\) We abuse the notation and assume a full domain hash. In the following when we write \(c = H(X, Y)\) where \(X\) and \(Y\) may be elements from different domains, we assume a suitable encoding scheme is employed to convert \(X, Y\) into a bit-string.
We remark that the above steps constitute a standard signature proof-of-knowledge of 1-out-of-2 discrete logarithms (which can be viewed as a two-party ring signature). This can be presented as follows using the Camenisch and Stadler notation [4].

\[
\text{SPK } \{(\alpha) : Y_S = g^\alpha \vee Y = g^{\alpha'} \} \{m\}
\]

- **Verify.** To verify the signature \((c_1, c_2, z_1, z_2, \{pk_{V_i}\}_{i \in [n]})\) on message \(m\), a verifier parses \(pk_{V_i}\) as \((Y_{V_i}, H_{V_i})\) and computes \(Y = \prod_{i \in [n]} Y_{V_i}^{H_{V_i}(pk_S, pk_{V_1}, \ldots, pk_{V_n}, m)}\). Output valid if and only if

\[
c_1 + c_2 = H_S(Y_{c_2}^{z_1}, Y_{c_2}^{z_2}, Y_{c_1}, Y_{c_2}^{z_1}, pk_{V_1}, h_1, \ldots, pk_{V_n}, h_n, Y, m)
\]

and invalid otherwise.

Regarding the security of our concrete construction, we have the following theorem, whose proof shall appear in the full version of the paper due to page limitation.

**Theorem 1.** Our concrete construction is secure under the discrete logarithm assumption in the random oracle model. Specifically, it satisfies

- definition 3 under the discrete logarithm assumption in the random oracle model;
- definition 4 unconditionally;
- definition 6 under the discrete logarithm assumption in the random oracle model.

## 4 Generic Strong MDVS

Strong DVS can be constructed from DVS via encrypting the signature under the designated verifier’s public key. However, the intuitive solution of encrypting the signature under each designated verifier’s public key in the case of multiple designated verifiers is not satisfactory. As discussed in [14], this intuitive solution creates a subtle issue in correctness. Specifically, if some of the encryptions are not executed properly, the signer could create an “invalid” signature that would be regarded as valid by some verifiers.

### 4.1 Overview of Our Generic Construction

To tackle this challenge, we observe that it is straightforward to use a verifiable encryption [3] which allows the signer to create a proof that all ciphertext decrypts to the same value. By verifying the proof, all verifiers are assured that all the verifiers obtains the same value for signature verification. This solution is, however, expensive. Looking at an abstract level, the goal of this encryption is to ensure all verifiers obtains the same value via decryption. This can be achieved,
perhaps somewhat interestingly, using a very weak one-way encryption with an explicit “IND-CPA” attack. Denote such an encryption scheme as $\mathcal{WE}$. That is, given a message $k$, anyone can check if the ciphertext $C$ decrypts to it. It is easy for a verifier to check locally if all the encryptions of the designated verifier signature are properly done.

This creates another problem. Since $\mathcal{WE}$ is only one-way secure, the ciphertext might leak information about the signature being encrypted and thus privacy of signer’s identity is not guaranteed. Thus, we employ a hybrid approach. $\mathcal{WE}$ is used to encrypt a symmetric key $k$ under all the designated verifiers public keys into ciphertexts $C_1, \ldots, C_n$. The ordinary MDVS is encrypted with a symmetric key encryption $\mathcal{SE}$ with key $k$. Also among the key $k$ cannot be recovered from the ciphertext $C_i$’s, no information about the MDVS can be learnt as long as the symmetric encryption $\mathcal{SE}$ is secure. Looking ahead, we assume $\mathcal{SE}$ to be an idealized cipher for the ease of security analysis. This means that our generic construction is secure in the ideal cipher model, which is equivalent to the random oracle model due to the result of [9].

4.2 Building Block of Our Generic Construction

While conceptually simple, two properties regarding $\mathcal{WE}$ are needed. The first one is an efficient and explicit “IND-CPA” attack. The second one is an efficient and explicit malleability attack which allows anyone to transform a ciphertext $C$ under public key $Y$ into another ciphertext $C'$ under public key $Y'$ so that they are encrypting the same message. The malleability attack on $\mathcal{WE}$ is needed in the proof of security for multiple designated verifiers.

Below we define the requirement of the weakly secure encryption $\mathcal{WE}$ as follows.

- $\text{param}_\mathcal{WE} \leftarrow \mathcal{WE}.\text{Setup}(1^\lambda)$: On input a security parameter $\lambda$, this algorithm outputs the public parameter $\text{param}_\mathcal{WE}$ for the system. We assume $\text{param}_\mathcal{WE}$ is an implicit input to all algorithms listed below.
- $(\mathcal{WE}.\text{pk}, \mathcal{WE}.\text{sk}) \leftarrow \mathcal{WE}.\text{Gen}()$: This algorithm outputs a key pair $(\mathcal{WE}.\text{pk}, \mathcal{WE}.\text{sk})$.
- $C_{\mathcal{WE}} \leftarrow \mathcal{WE}.\text{Enc}(\mathcal{WE}.\text{pk}, m)$: On input a message $m$ and a public key of the receiver $\mathcal{WE}.\text{pk}$, this algorithm outputs the ciphertext $C_{\mathcal{WE}}$.
- $m \leftarrow \mathcal{WE}.\text{Dec}(\mathcal{WE}.\text{sk}, C_{\mathcal{WE}})$: On input a secret key $\mathcal{WE}.\text{sk}$, a ciphertext $C_{\mathcal{WE}}$, this algorithm outputs the plaintext $m$.
- $0/1 \leftarrow \mathcal{WE}.\text{iAtk}((\mathcal{WE}.\text{pk}, \mathcal{WE}.\text{sk}, m))$: This is an attack on indistinguishability of ciphertext. On input a public key $\mathcal{WE}.\text{pk}$, a ciphertext $C_{\mathcal{WE}}$ and a plaintext $m$, output 1 if and only if $m = \mathcal{WE}.\text{Dec}(\mathcal{WE}.\text{sk}, C_{\mathcal{WE}})$, where $\mathcal{WE}.\text{sk}$ is the corresponding private key of $\mathcal{WE}.\text{pk}$ and 0 otherwise. Note that $\mathcal{WE}.\text{sk}$ is not an input to this algorithm.
- $(C'_{\mathcal{WE}}, \mathcal{WE}.\text{sk}') \leftarrow \mathcal{WE}.\text{mAtk}((\mathcal{WE}.\text{pk}, C_{\mathcal{WE}}))$: This is an attack on malleability of ciphertext. On input a public key $\mathcal{WE}.\text{pk}$, a ciphertext $C_{\mathcal{WE}}$, output $C'_{\mathcal{WE}}, \mathcal{WE}.\text{pk}'$ such that the distribution of $C'_{\mathcal{WE}}$ is indistinguishable to that of $\mathcal{WE}.\text{Enc}(\mathcal{WE}.\text{pk}', \mathcal{WE}.\text{Dec}(\mathcal{WE}.\text{sk}, C_{\mathcal{WE}}))$. Note that the algorithm does not output the corresponding secret key for $\mathcal{WE}.\text{pk}'$. 

We require the one-way security of $\mathcal{WE}$, which is formally defined as the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

**Setup** $\mathcal{C}$ invokes $\text{Setup}(1^\lambda)$ and subsequently $\text{Gen}()$ to obtain $(\text{param}_{\mathcal{WE}}, \mathcal{WE}.pk, \mathcal{WE}.sk)$.

**Challenge** $\mathcal{C}$ picks a random message $m$, compute $C_{\mathcal{WE}} \leftarrow \mathcal{WE}.\text{Enc}(\mathcal{WE}.pk, m)$.

$(\text{param}_{\mathcal{WE}}, \mathcal{WE}.pk, C_{\mathcal{WE}})$ is given to $\mathcal{A}$.

**Output** $\mathcal{A}$ outputs $m'$ and win if and only if $m = m'$.

$\mathcal{WE}$ is one-way secure if no PPT adversary $\mathcal{A}$ wins the above game with non-negligible probability.

We propose a construction of $\mathcal{WE}$ based on the Elgamal encryption in a cyclic group equipped with a bilinear map.

- $\mathcal{WE}.\text{Setup}(1^\lambda)$: Generate a pair of groups $\mathbb{G}, \mathbb{G}_T$ of the same prime order $p$ of $\lambda$-bit and a bilinear map $\hat{e}: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Let $g$ be a generator of $\mathbb{G}$. Set $\text{param}_{\mathcal{WE}} = (\mathbb{G}, \mathbb{G}_T, p, g, \hat{e})$.
- $\mathcal{WE}.\text{Gen}()$: Randomly pick $u \in \mathbb{Z}_p$, compute $U = g^u$. Set $(\mathcal{WE}.pk, \mathcal{WE}.sk) = (U, u)$.
- $\mathcal{WE}.\text{Enc}(U, m)$: On input a message $m \in \mathbb{G}$, randomly generate $r \in \mathbb{Z}_p$, output $C_{\mathcal{WE}} = (C, D)$ as $(mU^r, g^r)$.
- $\mathcal{WE}.\text{Dec}(u, (C, D))$: Output $C/D^u$.
- $\mathcal{WE}.\text{iAtk}(U, (C, D), m)$: Output 1 if and only if
  \[
  \hat{e}(C/m, g) = (D, U)
  \]
  and 0 otherwise.
- $\mathcal{WE}.\text{mAtk}(U, (C, D))$: Randomly pick $e, f \in \mathbb{Z}_p$, compute $U' = Ug^e$. Compute $C' = CD^e, C' = CU'^f, D' = Dg^f$. Output $((C', D'), U')$.

Note that if $U = g^u, C = mU^r, D = g^r$, it is easy to see that $C' = m(Ug^e)^{-r/f}, D' = g^{r+f}$ and $U' = Ug^e$. Thus, $(C', D')$ is encrypting the message $m$ under the public key $U'$ with the correct distribution.

Next, we show that our construction of $\mathcal{WE}$ is one-way secure under the computational Diffie-Hellman assumption.

**Proof.** Suppose there exists an adversary $\mathcal{A}$ that can win the game one-way security, we show how to construction an algorithm $\mathcal{S}$ that solves the CDH problem in a group equipped with a bilinear map. $\mathcal{S}$ is given $(\mathbb{G}, \mathbb{G}_T, \hat{e}, p, g, g^a, g^b)$ and its goal is to output $g^{ab}$.

$\mathcal{S}$ randomly picks a value $C$, gives $\text{param}_{\mathcal{WE}} = (\mathbb{G}, \mathbb{G}_T, p, g, \hat{e}), U = g^a, (C, D) = (C, g^b)$ to $\mathcal{A}$. Note that this implicit set the message being encrypted as $m = C/g^{ab}$. $\mathcal{A}$ returns with a value $m'$. $\mathcal{S}$ computes $C/m'$ and outputs it as the solution to the CDH problem. \(\Box\)
4.3 Our Generic Construction of Strong MDVS

We present our generic construction of Strong MDVS. Let $\mathcal{MS} = (\mathcal{MS}$.Setup, $\mathcal{MS}$.Gen, $\mathcal{MS}$.Sign, $\mathcal{MS}$.Verify) be a secure MDVS scheme. Let $\mathcal{WE} = (\mathcal{WE}$.Setup, $\mathcal{WE}$.Gen, $\mathcal{WE}$.Enc, $\mathcal{WE}$.Dec, $\mathcal{WE}$.iAtk, $\mathcal{WE}$.mAtk) be a one-way secure encryption. Let $H$ be a hash function and $\mathcal{SE}$ be a symmetric key encryption. We use $\mathcal{SE}$.Enc and $\mathcal{SE}$.Dec to denote encryption and decryption operation of $\mathcal{SE}$ using key $k$. $H$, $\mathcal{SE}$ will be modelled as a random oracle and an ideal cipher respectively. We show how to construct a strong MDVS scheme $(\text{Setup}, \text{Gen}, \text{Sign}, \text{Verify})$ as follows.

- Setup. On input security parameter $1^\lambda$, invoke $\text{param}_{\mathcal{MS}} \leftarrow \mathcal{MS}$.Setup$(1^\lambda)$ and $\text{param}_{\mathcal{WE}} \leftarrow \mathcal{WE}$.Setup$(1^\lambda)$, specify a weak encryption $\mathcal{WE}$, a hash function $H$ and a symmetric cipher $\mathcal{SE}$. Set $\text{param} = (\text{param}_{\mathcal{MS}}, \text{param}_{\mathcal{WE}}, H, \mathcal{SE})$.

- Gen. Invoke $(\mathcal{MS}.pk, \mathcal{MS}.sk) \leftarrow \mathcal{MS}.Gen(1^\lambda)$. Output $\mathcal{pk} = (\mathcal{MS}.pk, \mathcal{WE}.pk)$ and $\mathcal{sk} = (\mathcal{MS}.sk, \mathcal{WE}.sk)$.

- Sign. Let $\mathcal{pk}_S = (\mathcal{MS}.pk_S, \mathcal{WE}.pk_S)$ and $\mathcal{sk}_S = (\mathcal{MS}.sk_S, \mathcal{WE}.sk_S)$ be the key pair of the signer. Let $m$ be the message to be signed. Parse the set of verifiers to be $V = \{pk_{V_1}, \ldots, pk_{V_n}\}$ such that $pk_{V_i} = (\mathcal{MS}.pk_{V_i}, \mathcal{WE}.pk_{V_i})$.

Denote by $V_{\mathcal{MS}}$ the set $\{\mathcal{MS}.pk_{V_1}, \ldots, \mathcal{MS}.pk_{V_n}\}$. The signer randomly picks $k \in_R \{0,1\}^\lambda$. For $i = 1$ to $n$, compute

$$C_i = \mathcal{WE}.\text{Enc}(\mathcal{WE}.pk_{V_i}, k)$$

Next, compute $\tau = H(\mathcal{MS}.sk_S, \mathcal{pk}_{V_1}, \ldots, \mathcal{pk}_{V_n}, C_n, m)$. Invoke $\langle \sigma_{\mathcal{MS}}, V_{\mathcal{MS}} \rangle \leftarrow \text{Sign}(\mathcal{MS}.sk_S, \mathcal{MS}$.sk, $m || \tau$). Invoke $E = \mathcal{SE}$.Enc$(\sigma_{\mathcal{MS}} || \tau || pk_S)$.

Output the signature as $(E, V, \{C_i\}_{i \in [n]}$).

- Verify. To verify a signature $(E, V, \{C_i\}_{i \in [n]}$ on message $m$, a verifier $V$ parses $pk_{V_i}$ as $(\mathcal{MS}.pk_{V_i}, \mathcal{WE}.pk_{V_i})$ for all $pk_{V_i} \in V$ and uses his secret key $(\mathcal{MS}.sk_{V_i}, \mathcal{WE}.sk_{V_i})$ as follows.

  - Locate the index $i$ such that $pk_{V_i} = \mathcal{pk}_{V_i}$. Use his secret key to compute $k = \mathcal{WE}$.Dec$(C_i, \mathcal{WE}.sk_{V_i})$.
  - For all $j \in [n] \setminus \{i\}$, check if $1 = \mathcal{WE}$.iAtk$(\mathcal{WE}.pk_i, C_i, k)$. Output invalid if any of the check outputs 0.
  - Compute $\sigma_{\mathcal{MS}}, \tau, pk_S$ by $\mathcal{SE}$.Dec$(E)$.
  - Output invalid if $\tau \neq H(pk_S, pk_{V_1}, \ldots, pk_{V_n}, C_n, m)$.
  - Parse $pk_S$ as $(\mathcal{MS}.pk_S, \mathcal{WE}.pk_S)$.
  - Invoke valid/invalid $\leftarrow \mathcal{MS}$.Verify$(\mathcal{MS}.pk_S, \sigma_{\mathcal{MS}}, \{\mathcal{MS}.pk_{V_1}, \ldots, \mathcal{MS}.pk_{V_n}\}, m || \tau, \mathcal{MS}$.sk$_{V_i}$).

Regarding the security of our generic construction, we have the following theorem, whose proof shall appear in the full version of the paper due to page limitation.

**Theorem 2.** Our generic construction satisfies definition $d$ if the underlying MDVS scheme $\mathcal{MS}$ satisfies definitions $d$ for $d \in \{3, 4, 6\}$. Furthermore, our generic construction satisfies definition 5 if $\mathcal{WE}$ is one-way secure in the random oracle model.
5 Conclusion

In this paper, we formalized the security notion unforgeability under rogue key attack for MDVS. We proposed an efficient construction that is provably secure in the proposed model. In addition, we present a generic transformation that converts any secure MDVS scheme into a strong MDVS scheme. We leave the construction of constant size strong MDVS scheme secure under our definitions as an open problem.

References


