Multi-Designated Detector Watermarking for Language Models

Zhengan Huang, Gongxian Zeng, Xin Mu, Yu Wang, and Yue Yu

Pengcheng Laboratory, Shenzhen, China
zhahuang.sjtu@gmail.com, gxzeng@cs.hku.hk, mux@pcl.ac.cn,
wangy12@pcl.ac.cn, yuy@pcl.ac.cn

Abstract. In this paper, we initiate the study of *multi-designated detector watermarking (MDDW)* for large language models (LLMs). This technique allows model providers to generate watermarked outputs from LLMs with two key properties: (i) only specific, possibly multiple, designated detectors can identify the watermarks, and (ii) there is no perceptible degradation in the output quality for ordinary users. We formalize the security definitions for MDDW and present a framework for constructing MDDW for any LLM using multi-designated verifier signatures (MDVS). Recognizing the significant economic value of LLM outputs, we introduce claimability as an optional security feature for MDDW, enabling model providers to assert ownership of LLM outputs within designated-detector settings. To support claimable MDDW, we propose a generic transformation converting any MDVS to a claimable MDVS. Our implementation of the MDDW scheme highlights its advanced functionalities and flexibility over existing methods, with satisfactory performance metrics.

Keywords: Watermarking; Claimability; Off-the-record; Multi-designated verifier signature; Language model

1 Introduction

Generative artificial intelligence (AI) technique, e.g., large language models (LLMs), has been widely adopted in the field of language generation and has achieved excellent performance in a variety of downstream tasks. These tasks span from machine translation [HAS+23], dialogue system [HD23] to code generation [NIR+23] and medicine [TTE+23].

However, the abuse of LLMs may lead to several potential harms, including the generation of fake news [HS24] and instances of academic dishonesty, such as cheating on writing and coding assignments [KGW+23]. Another potential risk is that the proliferation of data fabricated by LLMs complicates the acquisition of superior models, as this data is not sourced from the real world and has to be excluded before training [RKX+23].

Thus, a crucial challenge lies in *distinguishing between texts generated by LLMs and those written by humans.*

Currently, the main approach to address the above issue is to training another AI model for detection, such as GPTZero[⋆]. This approach hinges on the critical assumption that texts generated by LLMs exhibit unique characteristics that can be identified by AI. However, a significant flaw in this assumption lies in the fact that LLMs are deliberately engineered to produce content that is indistinguishable from human-created works. As a result, any "black-box" detection method is prone to *high false positive and/or false negative rates* as LLMs become more realistic. Existing detectors (e.g., GPTZero, Detectgpt [MLK+23]) do not offer guarantees of accuracy. In fact, there have already been instances where students were falsely accused, making headlines in the news [Fow23,Jim23].

Recently, some schemes with formal guarantees of negligible error bounds have been proposed.

For example, [KGW+23] introduces watermarking for LLMs to achieve such formal guarantees. They demonstrate that a watermark can be embedded in the outputs of LLMs with large enough entropy. However, their watermarking scheme significantly alters the distribution of the generated texts and the watermark detection relies on this alteration. Thus, the methodology of the watermark detection in [KGW+23] has led to *degradation in the quality of the watermarked texts*.

To achieve the perfect quality (i.e., there is no degradation in the quality of watermarked outputs), [CGZ24] proposes a method for embedding a watermark if the outputs of LLMs are sufficiently random. The main idea is that it generates the watermark with a pseudorandom function (PRF) and utilizes the watermark (i.e., a pseudorandom number) to sample the outputs of LLMs. The perfect quality of the output is ensured by the *undetectability*, the notion of which is formalized in [CGZ24] and requires that without the knowledge of the secret key of the underlying PRF, the watermarked output is computationally indistinguishable from that output by the original LLM. However, the solution proposed in [CGZ24] has a notable limitation. Both the watermark generation and detection require a secret key, rendering the scheme *a privately detectable watermarking or a symmetric watermarking*. If the detection is desired to be outsourced, the secret key has to be shared to others, which could compromise its unforgeability.

To address these challenges, [FGJ+23] introduces an asymmetric solution called *publicly detectable watermarking (PDW)*, where the generation of a watermark requires a secret key of the signature, and public detectability is achieved through the public verification of the signature. In [FGJ+23], a security notion called *distortion-freeness* is proposed for PDW, serving as the asymmetric version of undetectability from [CGZ24], to ensure that the watermarking scheme maintains the quality of the LLM output. More specifically, distortion-freeness in [FGJ+23] is defined as follows: "without the secret watermarking key, no PPT machine can distinguish plain LLM output from watermarked LLM output". This definition aligns with the way undetectability is defined in [CGZ24]. However, in the asymmetric setting, the public key is usually known to the public, and

[⋆] GPTZero, https://gptzero.me/

a distinguisher with the public key can easily find out whether the LLM output is watermarked or not, since the watermarking solution in [FGJ+23] is publicly detectable. In other words, in the public-key setting, the distortion-freeness defined in [FGJ+23] and the completeness are contradictory. Furthermore, for PDW, any third party can detect watermarks, which may compromise privacy and other interests (e.g., economic interests). This universal detectability is often undesirable, particularly when restricted detectability is critical.

Contributions. In this paper, we initiate the study of *multi-designated detector watermarking (MDDW)* for LLMs. Generally speaking, for the watermarked LLM outputs generated via MDDW, multi-designated detectors are allowed to detect the watermarks, and all the other parties cannot distinguish the watermarked outputs from the original LLM outputs. The contributions of this paper is summarized as follows.

- We introduce a new primitive called multi-designated detector watermarking (MDDW), and formalize its security notions.
- We offer a framework for constructing MDDW from any LLM and multi-designated verifier signature (MDVS), and show that it achieves the required security properties.
- We provide a general method to transfer any MDVS to a claimable MDVS. Then, applying the above framework, we obtain a claimable MDDW, which allows the model provider to provably claim that some candidate texts are indeed generated by the LLMs.
- When considering only a single designated verifier, we present a more efficient concrete designated detector watermarking (DDW) in Appendix H, compared with the above solution from MDVS. We also provide a detailed experimental evaluation in Sec. 5 to show that our schemes are practical.

MDDW primitive. In MDDW, there are three kinds of roles involved: *model providers*, *designated detectors* and *users*. *Model providers* are the ones who deliver the LLM service to users and execute the watermarking scheme during the text generation phase. *Designated detectors* are the ones responsible for detecting whether some text was output by the model (by extracting and validating the watermark). *Users* are the ones who use the model.

An MDDW scheme consists of five algorithms: a setup algorithm Setup, a key generation algorithm WatKG for model providers, a key generation algorithm DetKG for designated detectors, a watermarking algorithm WatMar and a detection algorithm Detect.

Upon receiving the public parameter pp output by the setup algorithm Setup, the model provider can invoke WatKG to generate its key pair $(\mathsf{wpk}, \mathsf{wsk})$ for watermarking, and a designated detector can call DetKG to generate its key pair $(\mathsf{dpk}, \mathsf{dsk})$.

The watermarking algorithm WatMar, invoked by a model provider, takes the public parameter pp, the model provider's secret watermarking key wsk_i, the public keys of all the designated detectors $(\mathsf{dpk}_j)_{j\in S}$ (where S is the index

set of the designated detectors, and the same below), and a prompt $\boldsymbol{p}$ as input, and outputs a text $\boldsymbol{t}$ embedded with a watermark.

The detection algorithm Detect, invoked by a designated detector, takes the public parameter pp, a public watermarking key wpk_i, the designated detector's secret key $\mathsf{dsk}_{j'}$, the public keys of all the designated detectors $(\mathsf{dpk}_j)_{j\in S}$ (where $j' \in S$), and a candidate watermarked text $\boldsymbol{t}$ as input, and outputs a bit b, indicating whether $\boldsymbol{t}$ was watermarked.

Similar to [CGZ24,FGJ+23], the watermarking scheme just uses the LLM as a black box. Thus, it works without adopting a specific LLM or using specific configuration (e.g., the specific decoder algorithm, model parameters, etc). In addition, the detection also works without access to the LLMs or corresponding configurations.

MDDW security. It is important to note that, compared with the security notions of symmetric watermarking and publicly detectable watermarking, the security requirements in the multi-designated detector setting are much more complex.

The first challenge is ensuring consistency. If a designated detector successfully detects a watermark in a given text, how can they be assured that other designated detectors will also be able to detect it successfully? Another challenge is preventing designated detectors from provably convincing ordinary users that a text output by the LLMs is watermarked. If they can do so, the multi-designated detector functionality would be rendered useless. Moreover, in certain scenarios, even if designated detectors are unable to convince ordinary users of the watermark, it is crucial that the model provider can assert the truth. This mechanism is very helpful in MDDW, as it allows model providers to claim copyrights on content such as novels, pictures, or videos created by LLMs, which can yield considerable economic gains.

In this paper, we require that MDDW should satisfy the following properties.

- ***Completeness.*** Completeness requires for any designated detector set S and any candidate text $\boldsymbol{t}$ generated for the designated detectors in S with MDDW, as long as $\boldsymbol{t}$ is long enough (to embed a watermark), each detector in S should be able to extract and validate the watermark from $\boldsymbol{t}$ successfully using their individual secret detection keys.
- ***Consistency.*** Consistency aims to ensure that for any text $\boldsymbol{t}$, including those created maliciously, two designated detectors with uncompromised secret keys should not yield different detection results. Furthermore, if one designated detector accepts that the text $\boldsymbol{t}$ is generated by the LLM, all other designated detectors in S should obtain the same outcome.
- ***Soundness.*** Soundness is formalized to ensure that no PPT adversary can forge a watermarked text in the name of some model provider, such that some designated detector, whose secret key is not compromised, would accept the text is output by the corresponding model owned by this model provider.
- ***Distortion-freeness.*** As discussed before, the purpose of distortion-freeness is to guarantee that the watermarking scheme does not degrade the quality of the LMM output. Specifically, distortion-freeness for MDDW formalized

to require that for any PPT distinguisher without the secret detection keys of the designated detectors in S, a watermarked text generated for the designated detectors in S by MDDW should be indistinguishable from a text output by the original LLM.

- ***Robustness.*** It is likely that the text obtained from MDDW is modified before publication. The watermark detector should be robust enough to detect a watermark even if the text has been artificially altered, provided the text semantics are preserved. However, in extreme cases where a significant portion of the text is modified, the watermark should normally become undetectable, as the text effectively becomes the adversary's creation. Thus, we use a "soft" definition of robustness. Informally, if a continuous segment of text with length δ remains unaltered, then it is highly unlikely that any designated detector will fail to detect the watermark. We term this property δ-robust.
- ***Off-the-record property for designated set.*** This property is formalized to ensure that a text $\boldsymbol{t}$ can be simulated by all the designated detectors in S, such that it is indistinguishable from a watermarked text generated for them by MDDW, from the perspective of any third party, even if they possess all the secret detection keys of the designated detectors in S. Intuitively, this prevents the designated detectors from convincing a third party that the text $\boldsymbol{t}$ is watermarked by the model provider, as it can be simulated by the designated detectors themselves. Moreover, this property allows users to deny using the LLMs when confronted with third-party suspicions, even if some designated detectors have exposed their secret keys.

In addition, we introduce two *optional* security properties for MDDW: *off-the-record property for any subset* and *claimability*.

- ***Off-the-record property for any subset.*** This property is similar to the off-the-record property for designated set, with key differences: the plausible-looking text $\boldsymbol{t}$ can be simulated by *any subset* of the designated detectors in S, and it is indistinguishable from a watermarked text generated for the designated detectors in S by MDDW, from the perspective of any third party, even if they possess the secret detection keys of *the subset that produced* $\boldsymbol{t}$. It is crucial to note that the requirement of "any subset" enhances the meaningfulness of the off-the-record property. For example, if two designated detectors in S are unable to communicate, the off-the-record property for designated set loses its significance. Thus, off-the-record for any subset is a stronger security notion.
- ***Claimability.*** Claimability is designed to give the model provider a means to convincingly demonstrate the public that specific watermarked texts were generated by the provider's LLM. This requires two algorithms: Claim and ClmVer. The model provider runs Claim with their secret watermarking key to create proof π for some watermarked text $\boldsymbol{t}$. The public then uses ClmVer to verify this proof π. Claimability requires that (i) proofs created with Claim must be correctly validated by ClmVer, (ii) only the model provider

can create proofs to claim the LLM's output, and (iii) no one can falsely accuse another provider of generating the watermarked text $\boldsymbol{t}$.

MDDW construction. The following outlines a technical overview of our MDDW construction.

Framework. We propose a framework for building MDDW, which applies to any LLM, based on a multi-designated verifier signature (MDVS) scheme. Our method is inspired by [CGZ24,FGJ+23].

Before providing the high-level description of our framework, we firstly abstract the LLMs as in [CGZ24,FGJ+23,CHS24], disregarding the specifics of their implementations. LLMs have a "vocabulary" $\mathcal{T}$ consisting of words or word fragments called "tokens". These models employ neural networks to process and generate tokens, trained on varied datasets to learn and predict language patterns. During text generation, an LLM takes a prompt $\boldsymbol{p}$ as input and produces a sequence of tokens $\boldsymbol{t}$, with each subsequent token predicted based on the preceding context. For notation, let Model denote an auto-regressive language model, which accepts a prompt $\boldsymbol{p} \in \mathcal{T}$ and previously generated tokens $\boldsymbol{t}$ as input, outputting subsequent tokens over $\mathcal{T}$. For any polynomial n, $\mathsf{GenModel}_n$ iteratively invokes Model to generate n tokens.

Next, we briefly recall another building block of our framework, MDVS. An MDVS scheme consists of five algorithms: a setup algorithm Setup, a signing key generation algorithm SignKG for signers, a verification key generation algorithm VerKG for verifiers, a signing algorithm Sign and a verification algorithm Verify. In a nutshell, given a MDVS signature σ generated for the designated verifiers in a set S by Sign, only the designated verifiers in S can check the validity of the signature with Verify.

Now, we show a high-level description of our framework as follows. For a visual depiction, please see Fig. 1.

The setup algorithm of MDDW is the same is the setup algorithm of MDVS. The public/secret watermarking (resp., detection) key pairs are generated by invoking the signing (resp., verification) key generation algorithm of the MDVS.

Given the watermarking secret key wsk (which belongs to a model provider), a designated detector set S (whose public keys are $\{\mathsf{dpk}_j\}_{j\in S}$), and a prompt $\boldsymbol{p}$, the watermarking algorithm WatMar, run by the model provider, proceeds as follows.

(i) *Prompt Processing.* The model provider begins by using $\mathsf{GenModel}_\ell$ (given a prompt $\boldsymbol{p}$ and the previously generated tokens $\boldsymbol{t}$) to generate text in blocks of ℓ tokens.

(ii) *Generating MDVS signatures.* Take the last ℓ tokens of the current token sequence $\boldsymbol{t}$ as message m. The model provider generates a signature $\hat{\sigma}$ on m for the designated detectors in S, with the signing algorithm of the underlying MDVS. It is crucial to note that the model provider's secret key wsk is the signing secret key, and $\{\mathsf{dpk}_j\}_{j\in S}$ are actually the corresponding verification public keys of the designated detectors in S.

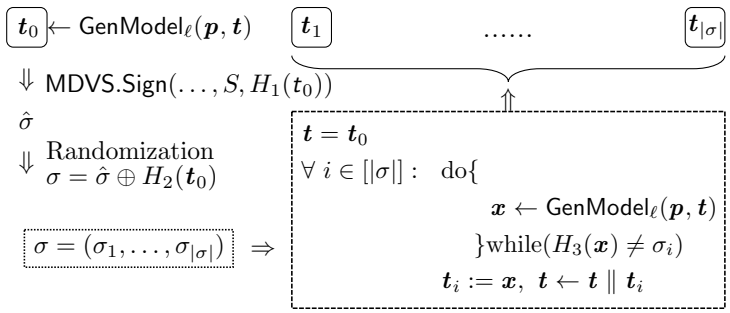


Fig. 1 The MDDW framework based on MDVS

(iii) *Randomizing MDVS signatures*[**]. This process involves performing the XOR operation on the MDVS signature $\hat{\sigma}$ and the hash value of the last ℓ tokens of the current token sequence $\boldsymbol{t}$. For simplicity, we denote the XOR result as σ.

(iv) *Sampling tokens with rejection.* Let σ_i represent the i^{th} bit of σ, for $i \in [|\sigma|]$. Firstly, generate a candidate ℓ-bit token sequence $\boldsymbol{x}$ using $\mathsf{GenModel}_\ell$ with given a prompt $\boldsymbol{p}$ and the current token sequence $\boldsymbol{t}$. Compute the hash value of $\boldsymbol{x}$. If the hash value matches σ_1, then accept $\boldsymbol{x}$ and append it to the end of $\boldsymbol{t}$. Otherwise, reject $\boldsymbol{x}$ and generate a new candidate sequence. For σ_2, σ_3, and continuing through $\sigma_{|\sigma|}$, repeat the above steps by calculating the hash value of the last ℓ bits of the current $\boldsymbol{t}$ and comparing it to σ_i ($i \in \{2, \cdots, |\sigma|\}$), appending the sequence only if they match.

That is the main idea for generating the MDDW watermarked texts.

The detection process is similar to the watermarking process, so we will not delve into many details here. Given the watermarking public key wpk (which belongs to a model provider), a designated detector set S (whose public keys are $\{\mathsf{dpk}_j\}_{j \in S}$), a secret detection key $\mathsf{dsk}_{j'}$ belonging to a detector $j' \in S$, and a token sequence $\boldsymbol{t}$, the detection algorithm Detect, run by the detector j', proceeds as follows. First, divide $\boldsymbol{t}$ into contiguous blocks of ℓ tokens each. Then, using the predetermined method (which involves hashing and comparing operations) and the verification algorithm of the underlying MDVS, extract a watermark or signature from the token block (if one is embedded) with the help of the verification secret key of the designated detector (i.e., $\mathsf{dsk}_{j'}$).

Security analysis. Our generic MDDW construction is based on an MDVS scheme, ensuring that many of its security properties are directly derived from those of the MDVS. For example, the consistency of our MDDW is inherited directly from the consistency of the underlying MDVS, the soundness of our MDDW is

[**] Very recently, [FGJ+23] also independently proposes a similar randomization step in their watermarking framework in their latest version, in order to eliminate the need for pseudorandom signatures. However, as discussed in the aforementioned section, it still fails to achieve meaningful distortion-freeness in the public-key setting.

also guaranteed by the existential unforgeability of the underlying MDVS, and so on. Moreover, following [CGZ24,FGJ+23], we also assume that any contiguous block of ℓ tokens contains at least α bits of min-entropy. With this assumption and the properties of random oracle, we can show that our MDDW achieves distortion-freeness.

Regarding the optional security requirements, the off-the-record property for any subset of our MDDW is implied by the off-the-record property for any subset of the underlying MDVS, and some MDVS schemes provide this security [DHM+20,MPR22]. However, to the best of our knowledge, it seems that currently no known security property of MDVS can ensure the claimability of our MDDW.

To ensure claimability in our generic MDDW, we introduce the security notion of claimability for MDVS, an extension of the claimability notions for designated verifier signatures [YHW+23] and ring signatures [PS19].

Similar to the claimability notion for MDDW, claimability for MDVS requires two algorithms: $\mathsf{Claim}_{\mathrm{mdvs}}$ and $\mathsf{ClmVer}_{\mathrm{mdvs}}$. A genuine signer uses $\mathsf{Claim}_{\mathrm{mdvs}}$ with their secret signing key to generate proof π for a signature σ they created. The public then uses $\mathsf{ClmVer}_{\mathrm{mdvs}}$ to verify this proof π. This security also requires that (i) proofs created with $\mathsf{Claim}_{\mathrm{mdvs}}$ must be correctly validated by $\mathsf{ClmVer}_{\mathrm{mdvs}}$, (ii) no one can successfully claim a signature that they did not generate, and (iii) no one can falsely accuse others of being the signer.

By applying this claimability to the underlying MDVS, we can show that our generic MDDW achieves claimability.

Construction of MDVS with claimability. We provide a generic method to transform any MDVS into an MDVS with claimability, abbreviated as CMDVS. The main idea, with some details omitted, is as follows. To generate a CMDVS signature, first produce an MDVS signature σ_{mdvs}. Then, use a standard signature scheme to sign the signer's public key and the MDVS signature σ_{mdvs}, producing a standard signature. Next, commit the signer's public key, the designated verifiers' public keys and the standard signature with a commitment scheme to produce a commitment com. The CMDVS signature comprises $(\sigma_{\mathrm{mdvs}}, \mathsf{com})$. To make a claim, the signer reveals the opening of the commitment com, including the standard signature and the randomness used for the commitment. The claim verification process involves checking the correctness of the opening and the validity of the standard signature.

Due to the correctness of the standard signature and the commitment scheme, the generated claim will be verified successfully. Regarding the second requirement of claimability, note that the CMDVS signature contains com, produced by committing the signer's public key, the designated verifiers' public keys and the standard signature. If another party tries to claim that they generated the CMDVS signature, they would need to provide an opening of com with their own public key, distinct from the actual signer's public key. This would imply two different openings for the same commitment, contradicting the binding property of the commitment scheme. As for the third requirement of claimability, it primarily relies on the existential unforgeability of the standard signature. This is

because the generated claim contains a standard signature, which is inherently difficult to forge without access to the signer's secret key.

Discussions. Here, we provide some discussions, which contains some further contributions and future works.

More efficient concrete DDW. Our generic framework of MDDW implies a generic designated detector watermarking (DDW), when there is only one designated detector (i.e., there is only one designated verifier in the corresponding MDVS scheme). However, when plugging the concrete MDVS (e.g., [AYSZ14]) with one designated verifier into our framework, the resulting DDW is not very efficient. In Appendix H, we consider a more efficient concrete DDW. The concrete DDW is based on the designated verifier signature (DVS) [LV05,SBWP03]. Compared with the DDW scheme from the MDVS [AYSZ14], the DDW from DVS [LV05,SBWP03] has shorter length of watermark (appropriately only 1/4 of the length of the DDW from [AYSZ14]). Thus, given a fixed length of the text output by the LLM, we can embed more watermarks, which also implies that the DDW from DVS [LV05,SBWP03] is more robust. The experimental results in Sec. 5 also show that DDW in Appendix H is practical.

Bit Length of watermarks for any subset. When considering the off-the-record property for any subset, we have the following theorem for the bit length of watermarks, with a proof inspired by [DHM+20, Theorem 1].

Theorem 1. *If an MDDW scheme supports the off-the-record property for any subset and soundness, then the size of the generated watermarks must be $\Omega(n)$, where n is the number of the designated detectors (i.e., $|S| = n$).*

Proof. Suppose that there exists a distinguisher D who knows all detectors' secret keys. Formally, the off-the-record property for any subset requires the existence of a PPT algorithm FgeAS (please refer to Def. 13), that takes the public parameter pp, the public key of the signer wpk_i, the public keys of the designated detectors $(\mathsf{dpk}_j)_{j\in S}$, the secret keys of the designated detectors in the corrupted set $(\mathsf{dsk}_j)_{j\in S_{\text{cor}}}$ and a prompt $\boldsymbol{p}$ as input, and outputs a text $\boldsymbol{t}$, which is indistinguishable from a real one output by WatMar.

Given a text generated by FgeAS with a corrupted detector set $S_{\text{cor}} \subseteq S$, D can distinguish whether some designated detector is in S_{cor} or not, by verifying the validity of the watermark (i.e., the MDVS signature). Note that the off-the-record property for any subset and soundness guarantee that only the detectors in S_{cor} would accept the watermark. Thus, D can determine the set S_{cor} when given a text generated by FgeAS. Therefore, the watermark must contain enough bits to indicate S_{cor}. Since $S_{\text{cor}} \subseteq S$ and it can be an arbitrary subset, there are $2^{|S|} = 2^n$ kinds of different subsets. Thus, the bit length of watermark is at least $\log_2 2^{|S|} = \log_2 2^n = n$. Considering that the text output by WatMar is indistinguishable from that output by FgeAS, the bit length of watermark embedded in the text output by WatMar is also $\Omega(n)$. □

Strengthening the security models. It is important to note that in this paper, claimability is defined as *optional* for MDDW. If it is considered a mandatory

security requirement, then an MDDW scheme must include the algorithms Claim and ClmVer. In this case, several security models (e.g., the game defining soundness) can be strengthened by granting the adversary access to the claiming oracle. Similar enhancements can be applied to the security models of MDVS. We will not elaborate on the details and leave this for future work.

Error-correcting code. We note that several works [FGJ+23,QYH+24,CG24] have considered the use of error-correction codes to enhance watermarking schemes. It improves robustness and relaxes the entropy assumption by allowing the watermarking scheme to tolerate errors caused by adversarial modifications or low entropy token distribution in rejection sampling. We emphasize that error-correction codes can also be applied to our work to achieve the same effects. Since error-correction codes are essentially specialized coding schemes that alter the form of signatures, they would not affect our MDDW framework.

Empirical attacks. We stress that, similar to existing works like [KGW+23,CGZ24,FGJ+23], our scheme is also vulnerable to certain empirical attacks, like the emoji attack and translation attack. For a more detailed introduction to these attacks, please refer to [KGW+23,CGZ24]. Actually, [CGZ24] concludes that no undetectable watermarking schemes can be completely unremovable.

Other watermarking schemes. Recently, there is a line of works using cryptographic tools to construct watermarking schemes for LLMs, besides the aforementioned works [KGW+23,CGZ24,FGJ+23]. Here, we summarized some works as follows.

Piet et al. [PSF+23] systematically analyze watermarking schemes in the secret key setting. Their study focuses on assessing generation quality and robustness to minor edits for practical protocol parameters. They conclude that distortion-freeness is too strong a property for practice. However, as pointed out in [FGJ+23], this conclusion was drawn from quality assessment performed by the chat version of Llama 2 7B [TMS+23], and they remark that Llama 2 7B's quality assessment likely does not generalize, since higher fidelity models may reveal weaknesses in distortion-inducing watermarking schemes.

Zhang et al. [ZEF+23] formally prove that "strong" robustness is impossible in watermarking schemes. They further demonstrate that their attack works in practice against a range of secret-key watermarking schemes (including the [KGW+23] scheme). On the other hand, [ZEF+23] also points out that "weak watermarking schemes" do exist and can be useful. To be noticed, our security notion of robustness is a kind of weak notions.

Qu et al. [QYH+24] present a watermarking scheme for LLMs that uses error correction codes to gain robustness. Similarly, Christ and Gunn [CG24] construct pseudorandom error-correction codes and applies it to show an undetectable watermarking scheme.

Roadmap. We recall some preliminaries in Sec. 2. Then, we introduce the primitive of MDDW and formalize its security notions in Sec. 3. In Sec. 4, we provide a framework of constructing MDDW for any LLM from MDVS, and show that

it achieves all required security properties. Finally, we present the experimental results in Sec. 5.

2 Preliminaries

Notations. We assume that the security parameter λ is an (implicit) input to all algorithms. For any $n \in \mathbb{N}$, let $[n] := \{1, 2, \cdots, n\}$. For a finite set S, let $|S|$ denote the number of elements in S, and $x \leftarrow S$ denote the process of sampling x from S uniformly at random. For a distribution Dist, we use $x \leftarrow \mathsf{Dist}$ to denote the process of sampling x from Dist. A function f of λ is *negligible* if $f \in O(\frac{1}{\mathsf{poly}(\lambda)})$ for any polynomial $\mathsf{poly}(\cdot)$. For simplicity, we write $f(\lambda) \leq \mathsf{negl}(\lambda)$ to mean that f is negligible. Throughout this paper, let ϵ denote the empty list or empty string.

For a random variable X, the *min-entropy* $\mathbf{H}_\infty(X)$ is $-\log(\max_x \Pr[X = x])$.

Let $a||b$ denote the tail-to-head concatenation of a and b. In a bit string a, a_i represents the i^{th} bit of a, unless indicated otherwise.

Throughout this paper, bold lower-case letters denote vectors or sequence of tokens, e.g., $\boldsymbol{t} = (t_1, \ldots, t_{|\boldsymbol{t}|})$ is a $|\boldsymbol{t}|$-dimension vector, and usually the number of dimensions can be inferred from the context. For a vector or sequence of tokens $\boldsymbol{t}$, let $\boldsymbol{t}[i]$ denote the i^{th} element of $(t_1, \cdots, t_{|\boldsymbol{t}|})$, and $\boldsymbol{t}[-i]$ denote the i^{th} last element of $(t_1, \cdots, t_{|\boldsymbol{t}|})$. For $i \leq j$, $\boldsymbol{t}[i:j]$ denotes $(t_i, t_{i+1}, \cdots, t_j)$, and $\boldsymbol{t}[-i:]$ denotes the last i elements of $(t_1, \cdots, t_{|\boldsymbol{t}|})$. We sometimes adapt these notations to bit strings, such as $a[1]$ and $a[-i]$.

2.1 Language models

We follow [KGW+23,CGZ24,FGJ+23] in our definition of a language model, and refer to language models simply as *models* in this paper.

Definition 1 (Auto-regressive model). *An* auto-regressive model Model *over token vocabulary* $\mathcal{T}$ *is a deterministic algorithm that takes as input a prompt* $\boldsymbol{p} \in \mathcal{T}$ *and tokens previously output by the model* $\boldsymbol{t} \in \mathcal{T}$ *and outputs a probability distribution* $p = \mathsf{Model}(\boldsymbol{p}, \boldsymbol{t})$ *over* $\mathcal{T}$*.*

For all $\ell \in \mathbb{N}$, $\mathsf{GenModel}_\ell$ wraps around Model to implement a generative model. $\mathsf{GenModel}_\ell$ iteratively generates ℓ tokens. Decode is the specific decoding method, e.g., argmax.

Algorithm 1 Generative model $\mathsf{GenModel}_\ell(\boldsymbol{p} \in \mathcal{T}, \boldsymbol{t} \in \mathcal{T})$

1: **for** $i = 1, \cdots, \ell$ **do**
2: $\quad \boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{Decode}(\mathsf{Model}(\boldsymbol{p}, \boldsymbol{t}))$
3: **return** $\boldsymbol{t}$

Following [FGJ+23], we assume that any contiguous block of $\ell \in \mathbb{N}$ tokens contains at least α bits of min-entropy, i.e., no particular sample is more than $2^{-\alpha}$ likely to happen.

Assumption 1 *For any prompt* $\boldsymbol{p}$ *and tokens* $\boldsymbol{t}$*, the new tokens* $\boldsymbol{t'} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t}) \in \mathcal{T}^\ell$ *were sampled from distributions with min-entropy at least* α*.*

Entities. In the scenarios discussed in this paper, three types of entities are involved.

- *Model provider.* The model provider delivers the large language model (LLM) service, generating LLM outputs for specified configurations based on received prompts. An honest model provider will execute the watermarking scheme using their watermarking secret key during the text generation phase.
- *Designated detector.* The designated detector, using their own detection secret key, checks if a text is watermarked with respect to a specific model provider, without access to the model weights or the model provider's secret watermarking key.
- *User.* Users create prompts that are sent to the LLM to receive the model output in return.

2.2 Multi-designated verifier signature

We recall the definition of multi-designated verifier signature (MDVS) from [DHM+20,MPR22], with some adjustments.

An MDVS scheme, associated with message space $\mathcal{M}$, is a tuple of algorithms $\mathsf{MDVS} = (\mathsf{Setup}, \mathsf{SignKG}, \mathsf{VerKG}, \mathsf{Sign}, \mathsf{Verify})$, where

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter λ, the setup algorithm outputs a public parameter pp.
- $\mathsf{SignKG}(\mathsf{pp}) \to (\mathsf{spk}, \mathsf{ssk})$: On input pp, the signing key generation algorithm outputs a public key spk and a secret key ssk for a signer.
- $\mathsf{VerKG}(\mathsf{pp}) \to (\mathsf{vpk}, \mathsf{vsk})$: On input pp, the verification key generation algorithm outputs a public key vpk and a secret key vsk for a verifier.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, m) \to \sigma$: On input pp, a signing secret key ssk_i, public keys of the designated verifiers $\{\mathsf{vpk}_j\}_{j\in S}$, and a message m, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_{j'}, \{\mathsf{vpk}_j\}_{j\in S}, m, \sigma) \to b$: On input pp, a signing public key spk_i, a secret key $\mathsf{vsk}_{j'}$ of a verifier such that $j' \in S$, public keys of the designated verifiers $\{\mathsf{vpk}_j\}_{j\in S}$, a message m, and a signature σ, the verification algorithm outputs a bit $b \in \{0, 1\}$.

In this paper, we require that an MDVS scheme should satisfy correctness, consistency, existential unforgeability, and off-the-record property for designated set.

Definition 2 (Correctness). *We say that* MDVS *is* correct*, if for any signer* i*, any message* $m \in \mathcal{M}$*, any verifier identity set* S *and any* $j' \in S$*, it holds that*

$$\Pr\left[\begin{array}{l|l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) & \\ (\mathsf{spk}_i, \mathsf{ssk}_i) \leftarrow \mathsf{SignKG}(\mathsf{pp}) & \mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_{j'}, \\ \{(\mathsf{vpk}_j, \mathsf{vsk}_j) \leftarrow \mathsf{VerKG}(\mathsf{pp})\}_{j\in S} & \quad \{\mathsf{vpk}_j\}_{j\in S}, m, \sigma) \neq 1 \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, m) & \end{array}\right] = 0.$$

$\mathbf{G}^{\text{cons}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}_S,\mathcal{O}_V}(\mathsf{pp})$

If $\exists\ j_0^*, j_1^* \in S^*$, such that

$\mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_{i^*}, \mathsf{vsk}_{j_0^*}, \{\mathsf{vpk}_j\}_{j\in S^*}, m^*, \sigma^*) \neq \mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_{i^*}, \mathsf{vsk}_{j_1^*}, \{\mathsf{vpk}_j\}_{j\in S^*}, m^*, \sigma^*)$,

where all keys are the honestly generated outputs of the key generation oracles, and $\mathcal{O}_{VK}$ is never queried on j_0^* or j_1^*

then return 1

Return 0

$\mathbf{G}^{\text{unforg}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}_S,\mathcal{O}_V}(\mathsf{pp})$

where $\mathcal{O}_{SK}$ is never queried on i^*, and $\mathcal{O}_S$ is never queried on (i^*, S^*, m^*)

If $\exists\ j^* \in S^*$, such that

$\mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_{i^*}, \mathsf{vsk}_{j^*}, \{\mathsf{vpk}_j\}_{j\in S^*}, m^*, \sigma^*) = 1$, where all keys are the honestly generated outputs of the key generation oracles, and $\mathcal{O}_{VK}$ is never queried on j^*

then return 1

Return 0

Fig. 2 Games $\mathbf{G}^{\text{cons}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ and $\mathbf{G}^{\text{unforg}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ for MDVS, and the oracles are given in Fig. 3

Signer Key Generation Oracle $\mathcal{O}_{SK}(i)$:
1. On the first call to $\mathcal{O}_{SK}$ on i, compute $(\mathsf{spk}_i, \mathsf{ssk}_i) \leftarrow \mathsf{SignKG}(\mathsf{pp})$, output and store $(\mathsf{spk}_i, \mathsf{ssk}_i)$.
2. On subsequent calls, simply output $(\mathsf{spk}_i, \mathsf{ssk}_i)$.

Verifier Key Generation Oracle $\mathcal{O}_{VK}(j)$:
1. On the first call to $\mathcal{O}_{VK}$ on j, compute $(\mathsf{vpk}_j, \mathsf{vsk}_j) \leftarrow \mathsf{VerKG}(\mathsf{pp})$, output and store $(\mathsf{vpk}_j, \mathsf{vsk}_j)$.
2. On subsequent calls, simply output $(\mathsf{vpk}_j, \mathsf{vsk}_j)$.

Public Signer Key Generation Oracle $\mathcal{O}_{SPK}(i)$:
1. $(\mathsf{spk}_i, \mathsf{ssk}_i) \leftarrow \mathcal{O}_{SK}(i)$; output spk_i.

Public Verifier Key Generation Oracle $\mathcal{O}_{VPK}(j)$:
1. $(\mathsf{vpk}_j, \mathsf{vsk}_j) \leftarrow \mathcal{O}_{VK}(j)$; output vpk_j.

Signing Oracle $\mathcal{O}_S(i, S, m)$:
1. $(\mathsf{spk}_i, \mathsf{ssk}_i) \leftarrow \mathcal{O}_{SK}(i)$, $\{\mathsf{vpk}_j \leftarrow \mathcal{O}_{VPK}(j)\}_{j\in S}$.
2. Output $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, m)$.

Verification Oracle $\mathcal{O}_V(i, j' \in S, S, m, \sigma)$:
1. $\mathsf{spk}_i \leftarrow \mathcal{O}_{SPK}(i)$, $(\mathsf{vpk}_{j'}, \mathsf{vsk}_{j'}) \leftarrow \mathcal{O}_{VK}(j')$
2. $\{\mathsf{vpk}_j \leftarrow \mathcal{O}_{VPK}(j)\}_{j\in S}$.
3. Output $b \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_{j'}, \{\mathsf{vpk}_j\}_{j\in S}, m, \sigma)$.

Fig. 3 The oracles for the games defining security notions for MDVS

Definition 3 (Consistency). *We say that* MDVS *is* consistent*, if for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\text{cons}}_{\mathsf{MDVS},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\text{cons}}_{\mathsf{MDVS},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{cons}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ *is shown in Fig. 2.*

$\underline{\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeDS}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $b \leftarrow \{0,1\}$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}^{(b)}_{otr\text{-}chl},\mathcal{O}_V}(\mathsf{pp})$

where $\mathcal{O}^{(0)}_{otr\text{-}chl}(i,S,m)$ outputs $\mathcal{O}_S(i,S,m)$, $\mathcal{O}^{(1)}_{otr\text{-}chl}(i,S,m)$ outputs $\sigma \leftarrow \mathsf{FgeDS}(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vsk}_j\}_{j\in S}, m)$ (where all keys are the honestly generated outputs of the key generation oracles),

let Q denote the set of query-response of $\mathcal{O}^{(b)}_{otr\text{-}chl}$,

all queries (i,S,m) to $\mathcal{O}^{(b)}_{otr\text{-}chl}$ should satisfy that $\mathcal{O}_{SK}$ is never queried on i,

and all queries (i',j',S',m',σ') to $\mathcal{O}_V$ should satisfy "$\nexists\ (*,*,*,\sigma) \in Q$ s.t. $\sigma = \sigma'$"

If $b' = b$, then return 1

Return 0

$\underline{\mathbf{G}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $b \leftarrow \{0,1\}$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}^{(b)}_{otr\text{-}chl},\mathcal{O}_V}(\mathsf{pp})$

where $\mathcal{O}^{(0)}_{otr\text{-}chl}(i,S,S_{\text{cor}},m)$ outputs $\mathcal{O}_S(i,S,m)$, $\mathcal{O}^{(1)}_{otr\text{-}chl}(i,S,S_{\text{cor}},m)$ outputs $\sigma \leftarrow \mathsf{FgeAS}(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \{\mathsf{vsk}_j\}_{j\in S_{\text{cor}}}, m)$ (where all keys are the honestly generated outputs of the key generation oracles),

let Q denote the set of query-response of $\mathcal{O}^{(b)}_{otr\text{-}chl}$,

all queries (i,S,S_{cor},m) to $\mathcal{O}^{(b)}_{otr\text{-}chl}$ should satisfy that (1) $S_{\text{cor}} \subset S$, (2) $\mathcal{O}_{SK}$ is never queried on i, and (3) for all $j \in S \setminus S_{\text{cor}}$, $\mathcal{O}_{VK}$ is never queried on j,

all queries j to $\mathcal{O}_{VK}$ should satisfy "$\nexists\ (*,S,S_{\text{cor}},*,*) \in Q$ s.t. $j \in S \setminus S_{\text{cor}}$",

and all queries (i',j',S',m',σ') to $\mathcal{O}_V$ should satisfy "$\nexists\ (*,*,*,*,\sigma) \in Q$ s.t. $\sigma = \sigma'$"

If $b' = b$, then return 1

Return 0

Fig. 4 Games $\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeDS}}(\lambda)$ and $\mathbf{G}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)$ for MDVS, and the oracles are given in Fig. 3

Definition 4 (Unforgeability). *We say that* MDVS *is* unforgeable, *if for any PPT adversary* $\mathcal{A}$,

$$\mathbf{Adv}^{\text{unforg}}_{\mathsf{MDVS},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\text{unforg}}_{\mathsf{MDVS},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{unforg}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ *is shown in Fig. 2.*

Definition 5 (Off-the-record for designated set). *We say that* MDVS *is* off-the-record for designated set, *if there is a PPT algorithm* FgeDS, *taking* $(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vsk}_j\}_{j\in S}, m)$ *as input and outputting* σ, *such that for any PPT adversary* $\mathcal{A}$,

$$\mathbf{Adv}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeDS}}(\lambda) = |\Pr[\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeDS}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeDS}}(\lambda)$ *is shown in Fig. 4.*

Now, we recall the off-the-record property for any subset for MDVS [DHM+20]. Note that unlike [DHM+20], in this paper, the off-the-record property is defined as *optional* for MDVS. Note that off-the-record property for any subset implies off-the-record property for designated set.

Definition 6 (Off-the-record for any subset). *We say that* MDVS *is* off-the-record for any subset*, if there is a PPT algorithm* FgeAS*, taking* $(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \{\mathsf{vsk}_j\}_{j\in S_{\mathrm{cor}}}, m)$ *(where* $S_{\mathrm{cor}} \subset S$*) as input and outputting* σ*, such that for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda) = |\Pr[\mathbf{G}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)$ *is shown in Fig. 4.*

3 Multi-designated detector watermarking

In this section, we introduce a primitive called *multi-designated detector watermarking (MDDW)*, and formalize its security notions.

An MDDW scheme for an auto-regressive model Model over token vocabulary $\mathcal{T}$ is a tuple of algorithms $\mathsf{MDDW} = (\mathsf{Setup}, \mathsf{WatKG}, \mathsf{DetKG}, \mathsf{WatMar}, \mathsf{Detect})$, where

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter λ, the setup algorithm outputs a public parameter pp.
- $\mathsf{WatKG}(\mathsf{pp}) \to (\mathsf{wpk}, \mathsf{wsk})$: On input pp, the watermarking key generation algorithm outputs a public/secret key pair $(\mathsf{wpk}, \mathsf{wsk})$ for watermarking.
- $\mathsf{DetKG}(\mathsf{pp}) \to (\mathsf{dpk}, \mathsf{dsk})$: On input pp, the detection key generation algorithm outputs a public/secret key pair $(\mathsf{dpk}, \mathsf{dsk})$ for a detector.
- $\mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{p}) \to \boldsymbol{t}$: On input pp, a watermarking secret key wsk_i, public keys of the designated detectors $\{\mathsf{dpk}_j\}_{j\in S}$, and a prompt $\boldsymbol{p} \in \mathcal{T}$, the watermarking algorithm outputs $\boldsymbol{t} \in \mathcal{T}$.
- $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_i, \mathsf{dsk}_{j'}, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t}) \to b$: On input pp, a watermarking public key wpk_i, a secret key $\mathsf{dsk}_{j'}$ of a detector such that $j' \in S$, public keys of the designated detectors $\{\mathsf{dpk}_j\}_{j\in S}$, and a candidate watermarked text $\boldsymbol{t} \in \mathcal{T}$, the detection algorithm outputs a bit $b \in \{0, 1\}$.

We provide further explanations here. The watermarking key generation algorithm WatKG (run by the model providers) and the detection key generation algorithm DetKG (run by the detectors) generate their respective public/secret key pairs. To generate watermarked texts that can only be detected by designated detectors in a set S, the model provider (holding watermarking secret key wsk_i) runs $\mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{p})$ to produce $\boldsymbol{t}$. A detector $j' \in S$ can then execute $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_i, \mathsf{dsk}_{j'}, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t})$ using their own detection secret key $\mathsf{dsk}_{j'}$ to check whether $\boldsymbol{t}$ is watermarked .

Regarding security, we require that an MDDW scheme should satisfy *completeness*, *consistency*, *soundness*, *distortion-freeness*, *robustness*, and *off-the-record property for designated set*. The formal definitions of these security requirements are as follows.

$\underline{\mathbf{G}^{\text{cons}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, \boldsymbol{t}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}_W,\mathcal{O}_D}(\mathsf{pp})$

If $\exists\ j_0, j_1 \in S^*$, such that $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_{i^*}, \mathsf{dsk}_{j_\beta}, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*) = \beta$ for all $\beta \in \{0,1\}$,
where all keys are the honestly generated outputs of the key generation oracles, and
$\mathcal{O}_{DK}$ is never queried on j_0 or j_1
then return 1

Return 0

$\underline{\mathbf{G}^{\text{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, \boldsymbol{t}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}_W,\mathcal{O}_D}(\mathsf{pp})$
where $\mathcal{O}_{WK}$ is never queried on i^*, and all $\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots$ (denoting the watermarked texts that $\mathcal{A}$ receives when querying $\mathcal{O}_W$ on $(i^*, S^*, *)$) satisfy $\mathsf{NOLap}_k(\boldsymbol{t}^*, \boldsymbol{t}_1, \boldsymbol{t}_2, \cdots) = 1$

If $\exists\ j' \in S^*$, such that $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_{i^*}, \mathsf{dsk}_{j'}, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*) = 1$
where all keys are the honestly generated outputs of the key generation oracles, and
$\mathcal{O}_{DK}$ is never queried on j',
then return 1

Return 0

$\underline{\mathbf{G}^{\text{dist-fr}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $b \leftarrow \{0,1\}$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}_{\mathsf{Model}},\mathcal{O}^{(b)}_{M\text{-}chl},\mathcal{O}_D}(\mathsf{pp})$
where $\mathcal{O}_{\mathsf{Model}}(\boldsymbol{p})$ outputs $\mathsf{Model}(\boldsymbol{p})$, $\mathcal{O}^{(0)}_{M\text{-}chl}(i, S, \boldsymbol{p})$ outputs $\mathcal{O}_W(i, S, \boldsymbol{p})$, $\mathcal{O}^{(1)}_{M\text{-}chl}(i, S, \boldsymbol{p})$ outputs $\mathsf{GenModel}(\boldsymbol{p})$, let Q denote the set of query-response of $\mathcal{O}^{(b)}_{M\text{-}chl}$,
all queries $(i, S, \boldsymbol{p})$ to $\mathcal{O}^{(b)}_{M\text{-}chl}$ satisfy that "i has never been queried to $\mathcal{O}_{SK}$, and $\forall\ j \in S$, j has never been queried to $\mathcal{O}_{DK}$",
all queries j to $\mathcal{O}_{DK}$ satisfy that "$\nexists\ (i, S, *, \boldsymbol{t}) \in Q$ s.t. $j \in S$",
and all queries $(i', j', \boldsymbol{t}')$ to $\mathcal{O}_D$ satisfy "$\forall\ (i', S, *, \boldsymbol{t}) \in Q$ s.t. $j' \in S$, $\mathsf{NOLap}_k(\boldsymbol{t}', \boldsymbol{t}) = 1$"

If $b' = b$, then return 1

Return 0

Fig. 5 Games $\mathbf{G}^{\text{cons}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$, $\mathbf{G}^{\text{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ and $\mathbf{G}^{\text{dist-fr}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ for MDDW, and the oracles are given in Fig. 6

Definition 7 (Completeness). *We say that* MDDW *is* δ-complete*, if for any i, any prompt $\boldsymbol{p} \in \mathcal{T}$, any detector identity set S and any $j' \in S$, it holds that*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda),\\ (\mathsf{wpk}_i, \mathsf{wsk}_i) \leftarrow \mathsf{WatKG}(\mathsf{pp}),\\ \{(\mathsf{dpk}_j, \mathsf{dsk}_j) \leftarrow \mathsf{DetKG}(\mathsf{pp})\}_{j\in S},\\ \boldsymbol{t} \leftarrow \mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{p})\\ \quad s.t.\ |\boldsymbol{t}| \geq \delta\end{array} : \begin{array}{r}\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_i, \mathsf{dsk}_{j'},\\ \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t}) \neq 1\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Watermarking Key Generation Oracle: $\mathcal{O}_{WK}(i)$
1. On the first call to $\mathcal{O}_{WK}$ on i, compute $(\mathsf{wpk}_i, \mathsf{wsk}_i) \leftarrow \mathsf{WatKG}(\mathsf{pp})$, output and store $(\mathsf{wpk}_i, \mathsf{wsk}_i)$.
2. On subsequent calls, simply output $(\mathsf{wpk}_i, \mathsf{wsk}_i)$.

Detecting Key Generation Oracle: $\mathcal{O}_{DK}(j)$
1. On the first call to $\mathcal{O}_{DK}$ on j, compute $(\mathsf{dpk}_j, \mathsf{dsk}_j) \leftarrow \mathsf{DetKG}(\mathsf{pp})$, output and store $(\mathsf{dpk}_j, \mathsf{dsk}_j)$.
2. On subsequent calls, simply output $(\mathsf{dpk}_j, \mathsf{dsk}_j)$.

Public Watermarking Key Generation Oracle: $\mathcal{O}_{WPK}(i)$
1. $(\mathsf{wpk}_i, \mathsf{wsk}_i) \leftarrow \mathcal{O}_{WK}(i)$; output wpk_i.

Public Detecting Key Generation Oracle: $\mathcal{O}_{DPK}(j)$
1. $(\mathsf{dpk}_j, \mathsf{dsk}_j) \leftarrow \mathcal{O}_{DK}(j)$; output dpk_j.

Watermarking Oracle: $\mathcal{O}_W(i, S, \boldsymbol{p})$
1. $(\mathsf{wpk}_i, \mathsf{wsk}_i) \leftarrow \mathcal{O}_{WK}(i)$, $\{\mathsf{dpk}_j \leftarrow \mathcal{O}_{DPK}(j)\}_{j \in S}$.
2. Output $\boldsymbol{t} \leftarrow \mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j \in S}, \boldsymbol{p})$.

Detecting Oracle: $\mathcal{O}_D(i, j' \in S, S, \boldsymbol{t})$
1. $\mathsf{wpk}_i \leftarrow \mathcal{O}_{WPK}(i)$, $(\mathsf{dpk}_{j'}, \mathsf{dsk}_{j'}) \leftarrow \mathcal{O}_{DK}(j')$.
2. $\{\mathsf{dpk}_j \leftarrow \mathcal{O}_{DPK}(j)\}_{j \in S}$.
3. Output $b \leftarrow \mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_i, \mathsf{dsk}_{j'}, \{\mathsf{dpk}_j\}_{j \in S}, \boldsymbol{t})$.

Fig. 6 The oracles for defining security notions of MDDW

Definition 8 (Consistency). *We say that* MDDW *is* consistent*, if for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\mathrm{cons}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\mathrm{cons}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\mathrm{cons}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ *is shown in Fig. 5.*

Definition 9 (Soundness). *We say that* MDDW *is* k-sound*, if for any PPT adversary* $\mathcal{A}$ *and any prompt* $\boldsymbol{p} \in \mathcal{T}$*,*

$$\mathbf{Adv}^{\mathrm{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\mathrm{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\mathrm{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ *is shown in Fig. 5, and the predicate* $\mathsf{NOLap}_k(\boldsymbol{t}^*, \boldsymbol{t}_1, \boldsymbol{t}_2, \cdots)$ *in Fig. 5 outputs 1 if* $\boldsymbol{t}^*$ *does not share a* k*-length window of tokens with any of the genuinely-watermarked texts* $\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots$*, and outputs 0 otherwise.*

Definition 10 (Distortion-freeness). *We say that* MDDW *is* k-distortion-free*, if for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\text{dist-fr}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = |\Pr[\mathbf{G}^{\text{dist-fr}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{dist-fr}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ *is shown in Fig. 5, and the predicate* $\mathsf{NOLap}_k(\boldsymbol{t}', \boldsymbol{t})$ *in Fig. 5 outputs 1 if* $\boldsymbol{t}'$ *does not share a* k*-length window of tokens with* $\boldsymbol{t}$*, and outputs 0 otherwise.*

Remark 1. Given that MDDW is established in the public-key framework, we adopt the term "distortion-freeness" as used in [FGJ+23]. We stress that our definition of distortion-freeness, like that in [FGJ+23], is formalized in the multi-query setting, allowing the distinguisher to make multiple oracle queries adaptively. This aligns with the undetectability definition in [CGZ24]. In other words, our distortion-freeness can also be viewed as the undetectability definition from [CGZ24] in the MDDW context.

Definition 11 (Robustness). *We say that* MDDW *is* k-robust*, if for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\mathrm{rob}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\mathrm{rob}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\mathrm{rob}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ *is shown in Fig. 7, and the predicate* $\mathsf{NOLap}_k(\boldsymbol{t}, \boldsymbol{t}^*)$ *in Fig. 7 outputs 1 if* $\boldsymbol{t}$ *does not share a* k*-length window of tokens with* $\boldsymbol{t}^*$*, and outputs 0 otherwise.*

Definition 12 (Off-the-record for designated set). *We say that* MDDW *is* off-the-record for designated set*, if there is a PPT algorithm* FgeDS*, taking* $(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dsk}_j\}_{j\in S}, \boldsymbol{p})$ *as input and outputting* $\boldsymbol{t}$*, such that for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\text{otr-ds}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeDS}}(\lambda) = |\Pr[\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeDS}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeDS}}(\lambda)$ *is shown in Fig. 7.*

Now, we introduce two *optional* security notions for MDDW: *off-the-record property for any subset* and *claimability*. It's important to note that adherence to these two securities is *not* mandatory for every MDDW scheme.

Definition 13 (Off-the-record for any subset). *We say that* MDDW *is* off-the-record for any subset*, if there is a PPT algorithm* FgeAS*, taking* $(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \{\mathsf{dsk}_j\}_{j\in S_{\mathrm{cor}}}, \boldsymbol{p})$ *(where* $S_{\mathrm{cor}} \subset S$*) as input and outputting* $\boldsymbol{t}$*, such that for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\text{otr-as}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeAS}}(\lambda) = |\Pr[\mathbf{G}^{\text{otr-as}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeAS}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{otr-as}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeAS}}(\lambda)$ *is shown in Fig. 7.*

Definition 14 (Claimability). *We say that* MDDW *is* k-claimable*, if there are two PPT algorithms* Claim *and* ClmVer *(where* Claim *takes* $(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t})$ *as input and outputs a claim* π*, and* ClmVer *takes* $(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t}, \pi)$ *as input and outputs a bit), such that*

$\underline{\mathbf{G}^{\text{rob}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, \boldsymbol{p}^*, st) \leftarrow \mathcal{A}_1^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}_W,\mathcal{O}_D}(\mathsf{pp})$

where $\mathcal{O}_{WK}$ is never queried on i^*

$\boldsymbol{t}^* \leftarrow \mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_{i^*}, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{p}^*)$

where all keys are the honestly generated outputs of the key generation oracles

$\boldsymbol{t} \leftarrow \mathcal{A}_2^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}_W,\mathcal{O}_D}(\boldsymbol{t}^*, st)$

where $\mathsf{NOLap}_k(\boldsymbol{t}, \boldsymbol{t}^*) = 0$, and all queries i to $\mathcal{O}_{WK}$ satisfy $i \neq i^*$

If $\exists\ j \in S^*$ s.t. $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_{i^*}, \mathsf{dsk}_j, \boldsymbol{t}) = 0$, then return 1

Return 0

$\underline{\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeDS}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $b \leftarrow \{0,1\}$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}^{(b)}_{otr\text{-}chl},\mathcal{O}_D}(\mathsf{pp})$

where $\mathcal{O}^{(0)}_{otr\text{-}chl}(i, S, \boldsymbol{p})$ outputs $\mathcal{O}_W(i, S, \boldsymbol{p})$, $\mathcal{O}^{(1)}_{otr\text{-}chl}(i, S, \boldsymbol{p})$ outputs $\boldsymbol{t} \leftarrow \mathsf{FgeDS}(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dsk}_j\}_{j\in S}, \boldsymbol{p})$ (where all keys are the honestly generated outputs of the key generation oracles),

let Q denote the set of query-response of $\mathcal{O}^{(b)}_{otr\text{-}chl}$,

all queries $(i, S, \boldsymbol{p})$ to $\mathcal{O}^{(b)}_{otr\text{-}chl}$ satisfy that $\mathcal{O}_{WK}$ is never queried on i,

and all queries $(i', j', S', \boldsymbol{t}')$ to $\mathcal{O}_D$ satisfy "$\nexists\ (*, *, *, \boldsymbol{t}) \in Q$ s.t. $\boldsymbol{t} = \boldsymbol{t}'$"

If $b' = b$, then return 1

Return 0

$\underline{\mathbf{G}^{\text{otr-as}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeAS}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $b \leftarrow \{0,1\}$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}^{(b)}_{otr\text{-}chl},\mathcal{O}_D}(\mathsf{pp})$

where $\mathcal{O}^{(0)}_{otr\text{-}chl}(i, S, S_{\text{cor}}, \boldsymbol{p})$ outputs $\mathcal{O}_W(i, S, \boldsymbol{p})$, $\mathcal{O}^{(1)}_{otr\text{-}chl}(i, S, S_{\text{cor}}, \boldsymbol{p})$ outputs $\boldsymbol{t} \leftarrow \mathsf{FgeAS}(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \{\mathsf{dsk}_j\}_{j\in S_{\text{cor}}}, \boldsymbol{p})$ (where all keys are the honestly generated outputs of the key generation oracles),

let Q denote the set of query-response of $\mathcal{O}^{(b)}_{otr\text{-}chl}$,

all queries $(i, S, S_{\text{cor}}, \boldsymbol{p})$ to $\mathcal{O}^{(b)}_{otr\text{-}chl}$ satisfy that (1) $S_{\text{cor}} \subset S$, (2) $\mathcal{O}_{WK}$ is never queried on i, and (3) for all $j \in S \setminus S_{\text{cor}}$, $\mathcal{O}_{DK}$ is never queried on j,

all queries j to $\mathcal{O}_{DK}$ satisfy "$\nexists\ (*, S, S_{\text{cor}}, *, *) \in Q$ s.t. $j \in S \setminus S_{\text{cor}}$",

and all queries $(i', j', S', \boldsymbol{t}')$ to $\mathcal{O}_D$ satisfy "$\nexists\ (*, *, *, *, \boldsymbol{t}) \in Q$ s.t. $\boldsymbol{t} = \boldsymbol{t}'$"

If $b' = b$, then return 1

Return 0

Fig. 7 Games $\mathbf{G}^{\text{rob}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$, $\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeDS}}(\lambda)$ and $\mathbf{G}^{\text{otr-as}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeAS}}(\lambda)$ for MDDW, and the oracles are given in Fig. 6

1. *for any i and any detector identity set S,*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{wpk}_i, \mathsf{wsk}_i) \leftarrow \mathsf{WatKG}(\mathsf{pp}) \\ ((\mathsf{dpk}_j, \mathsf{dsk}_j) \leftarrow \mathsf{DetKG}(\mathsf{pp}))_{j\in S} \\ \boldsymbol{t} \leftarrow \mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{p}) \\ \pi \leftarrow \mathsf{Claim}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t})\end{array} : \begin{array}{r}\mathsf{ClmVer}(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \\ \boldsymbol{t}, \pi) \neq 1\end{array}\right] = 0.$$

2. *for any PPT adversary $\mathcal{A}$,*

$$\mathbf{Adv}^{\text{clm-unf}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

   where $\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ is shown in Fig. 8, and the predicate $\mathsf{NOLap}_k(\boldsymbol{t}, \boldsymbol{t}^*)$ in Fig. 8 outputs 1 if $\boldsymbol{t}$ does not share a k-length window of tokens with $\boldsymbol{t}^*$, and outputs 0 otherwise.

3. *for any PPT adversary $\mathcal{A}$,*

$$\mathbf{Adv}^{\text{non-fram}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\text{non-fram}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

   where $\mathbf{G}^{\text{non-fram}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ is shown in Fig. 8.

Here, we provide further explanations. When a model provider intends to claim that certain watermarked texts are generated by their LLM (e.g., to establish copyright), they run Claim with their watermarking secret key to generate a claim π. Any one can use ClmVer to verify the claim.

The notion of claimability for MDDW is inspired by the claimability concepts in ring signature [PS19] and designated-verifier signature [YHW+23]. Generally speaking, the first requirement in the claimability definition is correctness. The second requirement is that even an adversarial model provider cannot successfully claim a watermarked text they did not produce. The third requirement is that no one can falsely accuse another provider of generating watermarked texts.

4 MDDW construction

In this section, we present a framework for building MDDW from MDVS and demonstrate its fulfillment of the required security properties. Additionally, we show that if the underlying MDVS scheme is off-the-record for any subset, the constructed MDDW also achieves the off-the-record property for any subset. Furthermore, by introducing the notion of claimability for MDVS, we demonstrate that our generic MDDW scheme achieves claimability when the underlying MDVS scheme possesses this property. Finally, we provide a generic method for constructing claimable MDVS by showing a transformation that converts any MDVS scheme into one that is claimable.

$\underline{\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, \boldsymbol{p}^*, st) \leftarrow \mathcal{A}_1^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}_W,\mathcal{O}_D,\mathcal{O}_{Clm}}(\mathsf{pp})$

where $\mathcal{O}_{WK}$ is never queried on i^*

$\boldsymbol{t}^* \leftarrow \mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_{i^*}, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{p}^*)$

where all keys are the honestly generated outputs of the key generation oracles

$(\pi^*, i') \leftarrow \mathcal{A}_2^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}_W,\mathcal{O}_D,\mathcal{O}_{Clm}}(\boldsymbol{t}^*, st)$

where all queries i to $\mathcal{O}_{WK}$ satisfy $i \neq i^*$, and all queries $(*, *, \boldsymbol{t})$ to $\mathcal{O}_{Clm}$ satisfy $\mathsf{NOLap}_k(\boldsymbol{t}, \boldsymbol{t}^*) = 1$

If $(\mathsf{ClmVer}(\mathsf{pp}, \mathsf{wpk}_{i'}, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*, \pi^*) = 1) \wedge (i' \neq i^*)$, then return 1

Return 0

$\underline{\mathbf{G}^{\text{non-fram}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, \boldsymbol{t}^*, \pi^*) \leftarrow \mathcal{A}^{\mathcal{O}_{WK},\mathcal{O}_{DK},\mathcal{O}_{WPK},\mathcal{O}_{DPK},\mathcal{O}_W,\mathcal{O}_D,\mathcal{O}_{Clm}}(\mathsf{pp})$

where $\mathcal{O}_{WK}$ is never queried on i^*, and $\mathcal{O}_{Clm}$ is never queried on $(i^*, S^*, \boldsymbol{t})$ satisfying $\mathsf{NOLap}_k(\boldsymbol{t}, \boldsymbol{t}^*) = 1$

If $\mathsf{ClmVer}(\mathsf{pp}, \mathsf{wpk}_{i^*}, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*, \pi^*) = 1$ and

$\exists\ \mu \in S^*$, $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_{i^*}, \mathsf{dsk}_\mu, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*)$=1, then return 1

Return 0

Claiming Oracle $\mathcal{O}_{Clm}(i, S, \boldsymbol{t})$:

(1) $(\mathsf{wpk}_i, \mathsf{wsk}_i) \leftarrow \mathcal{O}_{WK}(i)$, $\{\mathsf{dpk}_j \leftarrow \mathcal{O}_{DPK}(j)\}_{j\in S}$.

(2) Output $\pi \leftarrow \mathsf{Claim}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t})$.

Fig. 8 Games $\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ and $\mathbf{G}^{\text{non-fram}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ for MDDW, and some oracles are given in Fig. 6

4.1 Generic construction of MDDW

Let $\mathsf{MDVS} = (\mathsf{MDVS.Setup}, \mathsf{SignKG}, \mathsf{VerKG}, \mathsf{Sign}, \mathsf{Verify})$ be an MDVS scheme with signature length $\mathsf{len}_{\text{sig}}$. Let $\mathcal{M}$ denote the message space of MDVS, and $\mathcal{SG}$ denote the signature space of MDVS. Let $H_1 : \{0,1\}^* \to \mathcal{M}$, $H_2 : \{0,1\}^* \to \mathcal{SG}$ and $H_3 : \{0,1\}^* \to \{0,1\}$ be hash functions, which will all be modeled as random oracles in the security proof.

Our MDDW scheme $\mathsf{MDDW}_{n,\ell,\mathsf{len}_{\text{sig}}} = (\mathsf{Setup}, \mathsf{WatKG}, \mathsf{DetKG}, \mathsf{WatMar}, \mathsf{Detect})$, for some predefined parameters n and ℓ, is as follows.[***]

- $\mathsf{Setup}(1^\lambda)$: Compute $\mathsf{pp} \leftarrow \mathsf{MDVS.Setup}(1^\lambda)$, and return pp as the public parameter of MDDW.
- $\mathsf{WatKG}(\mathsf{pp})$: Generate $(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathsf{SignKG}(\mathsf{pp})$, and set $\mathsf{wpk} = \mathsf{spk}$ and $\mathsf{wsk} = \mathsf{ssk}$. Return $(\mathsf{wpk}, \mathsf{wsk})$.
- $\mathsf{DetKG}(\mathsf{pp})$: Generate $(\mathsf{vpk}, \mathsf{vsk}) \leftarrow \mathsf{VerKG}(\mathsf{pp})$, and set $\mathsf{dpk} = \mathsf{vpk}$ and $\mathsf{dsk} = \mathsf{vsk}$. Return $(\mathsf{dpk}, \mathsf{dsk})$.

[***] Note that according to Assumption 1, any contiguous block of ℓ tokens contains at least α bits of min-entropy.

- $\mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{p})$: Its description is shown in **Algorithm 2**.
- $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_i, \mathsf{dsk}_{j'}, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t})$: Its description is shown in **Algorithm 3**.

Algorithm 2 $\mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{p})$

$\mathsf{ssk}_i \leftarrow \mathsf{wsk}_i$, $\{\mathsf{vpk}_j\}_{j\in S} \leftarrow \{\mathsf{dpk}_j\}_{j\in S}$, $\boldsymbol{t} \leftarrow \epsilon$
while $|\boldsymbol{t}| + \ell + \ell \cdot \mathsf{len}_{\mathrm{sig}} < n$ **do**
  $\boldsymbol{t} \leftarrow \boldsymbol{t} || \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, H_1(\boldsymbol{t}[-\ell:]))$
  $\boldsymbol{\sigma} \leftarrow \hat{\boldsymbol{\sigma}} \oplus H_2(\boldsymbol{t}[-\ell:])$
  $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \epsilon$, $\boldsymbol{m} \leftarrow \epsilon$
  while $\boldsymbol{\sigma} \neq \epsilon$ **do**
    $\sigma_{\mathrm{bit}} \leftarrow \boldsymbol{\sigma}[1]$, $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[2:]$
    $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    while $H_3(\boldsymbol{m}||\boldsymbol{x}||\boldsymbol{\sigma}_{\mathrm{prev}}) \neq \sigma_{\mathrm{bit}}$ **do**
      $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{x}$, $\boldsymbol{t} \leftarrow \boldsymbol{t}||\boldsymbol{x}$, $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \boldsymbol{\sigma}_{\mathrm{prev}}||\sigma_{\mathrm{bit}}$
if $|\boldsymbol{t}| < n$ **then**
  $\boldsymbol{t} \leftarrow \boldsymbol{t} || \mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{p}, \boldsymbol{t})$
return $\boldsymbol{t}$

Algorithm 3 $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_i, \mathsf{dsk}_{j'}, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t})$

$\mathsf{spk}_i \leftarrow \mathsf{wpk}_i$, $\mathsf{vsk}_{j'} \leftarrow \mathsf{dsk}_{j'}$, $\{\mathsf{vpk}_j\}_{j\in S} \leftarrow \{\mathsf{dpk}_j\}_{j\in S}$
for $\mu = 1, 2, \cdots, n - (\ell + \ell \cdot \mathsf{len}_{\mathrm{sig}})$ **do**
  $\widetilde{m} \leftarrow H_1(\boldsymbol{t}[\mu : \mu + \ell - 1])$, $\boldsymbol{\sigma} \leftarrow \epsilon$, $\boldsymbol{m} \leftarrow \epsilon$
  for $\varphi = 1, 2, \cdots, \mathsf{len}_{\mathrm{sig}}$ **do**
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{t}[\mu + \varphi \cdot \ell : \mu + \varphi \cdot \ell + \ell - 1]$
    $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}||H_3(\boldsymbol{m}||\boldsymbol{\sigma})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \boldsymbol{\sigma} \oplus H_2(\boldsymbol{t}[\mu : \mu + \ell - 1])$
  if $\mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_{j'}, \{\mathsf{vpk}_j\}_{j\in S}, \widetilde{m}, \hat{\boldsymbol{\sigma}}) = 1$ **then**
    return 1
return 0

Now, we show that our proposed MDDW satisfies the required security properties defined in Sec. 3. Formally, we have the following theorem.

Theorem 2. *The constructed MDDW scheme* MDDW *possesses the following properties:*

- (Completeness). *If Assumption 1 holds and* MDVS *satisfies correctness, then* MDDW *is ℓ-complete.*
- (Consistency). *If the underlying* MDVS *is consistent, then* MDDW *is also consistent.*

– (Soundness). *If the underlying* MDVS *is unforgeable, then* MDDW *is* ℓ*-sound.*
– (Distortion-freeness). *If Assumption 1 holds, then* MDDW *is* ℓ*-distortion-free.*
– (Robustness). MDDW *is* $(2\mathsf{len}_{\mathrm{sig}}+2)\ell$*-robust.*
– (Off-the-record for designated set). *If the underlying* MDVS *is off-the-record for designated set, then* MDDW *is also off-the-record for designated set.*

Remark 2. If the underlying MDVS scheme MDVS meets strong unforgeability, MDDW might achieve $(\mathsf{len}_{\mathrm{sig}}+1)\ell$-soundness.

For the optional off-the-record property for any subset, we have the following theorem.

Theorem 3 (Off-the-record for any subset). *If* MDVS *is off-the-record for any subset, then* MDDW *is also off-the-record for any subset.*

The proofs of Theorem 2 and Theorem 3 are placed in Appendix D and Appendix E, respectively.

4.2 MDDW construction with claimability

To make the above generic MDDW scheme achieve claimability, the underlying MDVS scheme needs to meet some corresponding security property. However, to the best of our knowledge, no claimability notion for MDVS has been introduced before. Here, we firstly introduce the notion of claimability for MDVS, and then formally prove that if the underlying MDVS scheme meets claimability, the above generic MDDW scheme is claimable.

Claimability for MDVS. The notion of claimability for MDVS extends from the established claimability concepts for ring signature [PS19] and designated-verifier signature [YHW+23]. It is crucial to emphasize that in this paper, claimability is defined as an optional requirement for MDVS.

Definition 15 (Claimability for MDVS). *We say that* $\mathsf{MDVS}=(\mathsf{Setup},\mathsf{SignKG},\mathsf{VerKG},\mathsf{Sign},\mathsf{Verify})$ *is* claimable*, if there are two PPT algorithms* Claim *and* ClmVer *(where* Claim *takes* $(\mathsf{pp},\mathsf{ssk}_i,\{\mathsf{vpk}_j\}_{j\in S},\sigma)$ *as input and outputs a claim* π*, and* ClmVer *takes* $(\mathsf{pp},\mathsf{spk}_i,\{\mathsf{vpk}_j\}_{j\in S},\sigma,\pi)$ *as input and outputs a bit), such that*

1. *for any signer* i*, any message* $m\in\mathcal{M}$*, and any verifier identity set* S*,*

$$\Pr\left[\begin{array}{l|l}\mathsf{pp}\leftarrow\mathsf{Setup}(1^\lambda) & \\ (\mathsf{spk}_i,\mathsf{ssk}_i)\leftarrow\mathsf{SignKG}(\mathsf{pp}) & \\ \{(\mathsf{vpk}_j,\mathsf{vsk}_j)\leftarrow\mathsf{VerKG}(\mathsf{pp})\}_{j\in S} & :\mathsf{ClmVer}(\mathsf{pp},\mathsf{spk}_i,\{\mathsf{vpk}_j\}_{j\in S},\sigma,\pi)\neq 1\\ \sigma\leftarrow\mathsf{Sign}(\mathsf{pp},\mathsf{ssk}_i,\{\mathsf{vpk}_j\}_{j\in S},m) & \\ \pi\leftarrow\mathsf{Claim}(\mathsf{pp},\mathsf{ssk}_i,\{\mathsf{vpk}_j\}_{j\in S},\sigma) & \end{array}\right]=0.$$

$\underline{\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, m^*, st) \leftarrow \mathcal{A}_1^{\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}_S,\mathcal{O}_V,\mathcal{O}_{Clm}}(\mathsf{pp})$

where $\mathcal{O}_{SK}$ is never queried on i^*

$\sigma^* \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_{i^*}, \{\mathsf{vpk}_j\}_{j\in S^*}, m^*)$

where all keys are the honestly generated outputs of the key generation oracles

$(\pi^*, i') \leftarrow \mathcal{A}_2^{\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}_S,\mathcal{O}_V,\mathcal{O}_{Clm}}(\sigma^*, st)$

where all queries i to $\mathcal{O}_{SK}$ satisfy $i \neq i^*$, and all queries $(*, *, \sigma)$ to $\mathcal{O}_{Clm}$ satisfy $\sigma \neq \sigma^*$

If $(\mathsf{ClmVer}(\mathsf{pp}, \mathsf{spk}_{i'}, \{\mathsf{vpk}_j\}_{j\in S^*}, \sigma^*, \pi^*) = 1) \wedge (i' \neq i^*)$, then return 1

Return 0

$\underline{\mathbf{G}^{\text{non-fram}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$(i^*, S^*, m^*, \sigma^*, \pi^*) \leftarrow \mathcal{A}^{\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}_S,\mathcal{O}_V,\mathcal{O}_{Clm}}(\mathsf{pp})$

where $\mathcal{O}_{SK}$ is never queried on i^*, and $\mathcal{O}_{Clm}$ is never queried on (i^*, S^*, σ^*)

If $\mathsf{ClmVer}(\mathsf{pp}, \mathsf{spk}_{i^*}, \{\mathsf{vpk}_j\}_{j\in S^*}, \sigma^*, \pi^*) = 1$, then return 1

Return 0

Claiming Oracle $\mathcal{O}_{Clm}(i, S, \sigma)$:

(1) $(\mathsf{spk}_i, \mathsf{ssk}_i) \leftarrow \mathcal{O}_{SK}(i)$, $\{\mathsf{vpk}_j \leftarrow \mathcal{O}_{VPK}(j)\}_{j\in S}$.

(2) Output $\pi \leftarrow \mathsf{Claim}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \sigma)$.

Fig. 9 Games $\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ and $\mathbf{G}^{\text{non-fram}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ for MDVS

2. *for any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\mathbf{Adv}^{\text{clm-unf}}_{\mathsf{MDVS},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDVS},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ *is shown in Fig. 9.*

3. *for any PPT adversary* $\mathcal{A}$,

$$\mathbf{Adv}^{\text{non-fram}}_{\mathsf{MDVS},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\text{non-fram}}_{\mathsf{MDVS},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{non-fram}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ *is shown in Fig. 9.*

Remark 3. In the above definition, the adversary $\mathcal{A}$ succeeds in game $\mathbf{G}^{\text{non-fram}}_{\mathsf{MDVS},\mathcal{A}}(\lambda)$ if its output tuple $(i^*, S^*, m^*, \sigma^*, \pi^*)$ satisfies $\mathsf{ClmVer}(\mathsf{pp}, \mathsf{spk}_{i^*}, \{\mathsf{vpk}_j\}_{j\in S^*}, \sigma^*, \pi^*) = 1$. It's worth noting that unlike the prerequisites in the definition of claimability for ring signatures [PS19] and designated-verifier signatures [YHW+23], our third condition doesn't mandate that "σ^* is accepted by some designated verifier in S^*". Thus, the security requirement in our claimability for MDVS appears to be more stringent.

Generic MDDW scheme with claimability. For the claimability of our generic MDDW scheme, we have the following theorem.

Theorem 4 (Claimability). *If the underlying MDVS scheme* MDVS *is claimable, then the generic MDDW scheme* MDDW *in Sec. 4.1 is* $(\mathsf{len}_{\mathrm{sig}}+1)\ell$*-claimable.*

The proof of Theorem 4 is placed in Appendix F.

4.3 Instantiation of claimable MDVS

To instantiate claimable MDVS (CMDVS), we show a transformation that converts any MDVS into a CMDVS, with the help of a standard digital signature, a pseudorandom function and a commitment scheme. Our method is inspired by [PS19,YHW+23], where [PS19] shows how to construct claimable ring signature and [YHW+23] shows how to construct claimable DVS from ring signature.

The intuition of our method is as follows. To generate a CMDVS signature, the signer firstly generates an MDVS signature σ_{MDVS} with the signing algorithm of the underlying MDVS, signs σ_{MDVS} with the the standard signature scheme to obtain a standard signature σ_{Sig}, and then takes the commitment scheme to commit σ_{Sig}, obtaining a commitment com. The generated CMDVS signature consists of $(\sigma_{\mathsf{MDVS}}, \mathsf{com})$. When making a claim, the signer just opens the commitment com, outputting σ_{Sig} and the randomness used to generate com. To verify the claim, one firstly checks if the opening is correct, and then checks if the standard signature is valid. The unforgeability of the standard signature scheme guarantees that the claim is indeed generated by the signer.

The detailed construction of CMDVS is as follows.

Let $\mathsf{MDVS} = (\mathsf{Setup}, \mathsf{SignKG}, \mathsf{VerKG}, \mathsf{Sign}, \mathsf{Verify})$ be an MDVS scheme. Let $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme, $\mathsf{PRF} = (\mathsf{KG}, \mathsf{Eval})$ be a pseudorandom function, and $\mathsf{Commit} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Decom})$ be a commitment scheme. The definitions of signature, pseudorandom function and commitment are given in Appendix B for completeness.

Our CMDVS scheme $\mathsf{CMDVS} = (\mathsf{Setup}, \mathsf{SignKG}, \mathsf{VerKG}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Claim}, \mathsf{ClmVer})$ is shown in Fig. 10.

For security, we present the following theorem, the proof of which is given in Appendix G.

Theorem 5. *If* MDVS *satisfies correctness, consistency, unforgeability, and off-the-record property for designated set (resp., for any subset),* Sig *satisfies unforgeability, and* Commit *is hiding and binding, then* CMDVS *is an MDVS scheme achieving correctness, consistency, unforgeability, off-the-record property for designated set (resp., for any subset), and claimability.*

5 Evaluation

In this experiment section, we evaluate our proposed scheme from the perspective of computational overhead. Specifically, we analyze the time required for text generation and watermark detection using three different LLMs. The details are as follows:

Schemes. To highlight the performance of our scheme, we will implement the following schemes.

$\underline{\mathsf{Setup}(1^\lambda)}$:
$\mathsf{pp}_{\mathsf{MDVS}} \leftarrow \mathsf{MDVS.Setup}(1^\lambda)$, $\mathsf{pp}_{\mathsf{Sig}} \leftarrow \mathsf{Sig.Setup}(1^\lambda)$, $\mathsf{pp}_{\mathsf{Commit}} \leftarrow \mathsf{Commit.Setup}(1^\lambda)$
Return $\mathsf{pp} = (\mathsf{pp}_{\mathsf{MDVS}}, \mathsf{pp}_{\mathsf{Sig}}, \mathsf{pp}_{\mathsf{Commit}})$

$\underline{\mathsf{SignKG}(\mathsf{pp})}$:
$k \leftarrow \mathsf{PRF.KG}(1^\lambda)$, $(\mathsf{ssk}_{\mathsf{Sig}}, \mathsf{spk}_{\mathsf{Sig}}) \leftarrow \mathsf{Sig.KG}(\mathsf{pp}_{\mathsf{Sig}})$
$(\mathsf{ssk}_{\mathsf{MDVS}}, \mathsf{spk}_{\mathsf{MDVS}}) \leftarrow \mathsf{MDVS.SignKG}(\mathsf{pp}_{\mathsf{MDVS}})$
Return $\mathsf{spk} = (\mathsf{spk}_{\mathsf{Sig}}, \mathsf{spk}_{\mathsf{MDVS}})$, $\mathsf{ssk} = (k, \mathsf{ssk}_{\mathsf{Sig}}, \mathsf{ssk}_{\mathsf{MDVS}})$

$\underline{\mathsf{VerKG}(\mathsf{pp})}$:
Return $(\mathsf{vsk}_{\mathsf{MDVS}}, \mathsf{vpk}_{\mathsf{MDVS}}) \leftarrow \mathsf{MDVS.VerKG}(\mathsf{pp}_{\mathsf{MDVS}})$

$\underline{\mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, m)}$:
$\sigma_{\mathsf{MDVS}} \leftarrow \mathsf{MDVS.Sign}(\mathsf{pp}_{\mathsf{MDVS}}, \mathsf{ssk}_{\mathsf{MDVS},i}, \{\mathsf{vpk}_j\}_{j\in S}, m)$
$r_{\mathsf{Sig}} \leftarrow \mathsf{PRF.Eval}(k_i, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}, 0))$, $\sigma_{\mathsf{Sig}} \leftarrow \mathsf{Sig.Sign}(\mathsf{pp}_{\mathsf{Sig}}, \mathsf{ssk}_{\mathsf{Sig},i}, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}); r_{\mathsf{Sig}})$
$r_{\mathsf{Commit}} \leftarrow \mathsf{PRF.Eval}(k_i, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}, 1))$
$\mathsf{com} \leftarrow \mathsf{Commit.Com}(\mathsf{pp}_{\mathsf{Commit}}, (\mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \sigma_{\mathsf{Sig}}); r_{\mathsf{Commit}})$
Return $\sigma = (\sigma_{\mathsf{MDVS}}, \mathsf{com})$

$\underline{\mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_{j'}, \{\mathsf{vpk}_j\}_{j\in S}, \sigma, m)}$:
Return $b \leftarrow \mathsf{MDVS.Verify}(\mathsf{pp}_{\mathsf{MDVS}}, \mathsf{spk}_{\mathsf{MDVS},i}, \mathsf{vsk}_{\mathsf{MDVS},j'}, \{\mathsf{vpk}_{\mathsf{MDVS},j}\}_{j\in S}, m, \sigma_{\mathsf{MDVS}})$

$\underline{\mathsf{Claim}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \sigma)}$:
$r_{\mathsf{Sig}} \leftarrow \mathsf{PRF.Eval}(k_i, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}, 0))$, $r_{\mathsf{Commit}} \leftarrow \mathsf{PRF.Eval}(k_i, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}, 1))$
$\sigma_{\mathsf{Sig}} \leftarrow \mathsf{Sig.Sign}(\mathsf{pp}_{\mathsf{Sig}}, \mathsf{ssk}_{\mathsf{Sig},i}, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}); r_{\mathsf{Sig}})$
If $\mathsf{Commit.Decom}(\mathsf{pp}_{\mathsf{Commit}}, \mathsf{com}, r_{\mathsf{Commit}}, (\mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \sigma_{\mathsf{Sig}})) = 0$: Return $\perp$
Return $\pi = (r_{\mathsf{Commit}}, \sigma_{\mathsf{Sig}})$

$\underline{\mathsf{ClmVer}(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \sigma, \pi)}$:
If $\mathsf{com} \neq \mathsf{Commit.Com}(\mathsf{pp}_{\mathsf{Commit}}, (\mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \sigma_{\mathsf{Sig}}); r_{\mathsf{Commit}})$: Return 0
Return $b = \mathsf{Sig.Verify}(\mathsf{pp}_{\mathsf{Sig}}, \mathsf{spk}_{\mathsf{Sig},i}, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}), \sigma_{\mathsf{Sig}})$

Fig. 10 Construction of CMDVS

1. **Original LLMs.** The first scheme is the original LLM scheme without any watermarking solutions. We set the performance of this scheme as the baseline for our experiments.
2. PDW. This is the publicly detectable watermarking scheme [FGJ+23] based on the BLS signature [BLS04], which is recalled in Appendix C for completeness. Although this scheme [FGJ+23] does not actually achieve distortion-freeness (when the public keys are known, as discussed in the Introduction), it is more relevant to compare our scheme with an asymmetric solution in our experiments. For those interested in the performance of a symmetric solution (e.g., [CGZ24]), [FGJ+23] shows that its performance exceeds that of [CGZ24] in both text generation and watermark detection. The BLS signature is recalled in Appendix C for completeness.

3. **DDW.** This is the designated-detector watermarking scheme built within the MDDW framework (in Sec. 4) when $|S| = 1$, using a concrete designated-verifier signature [LV05,SBWP03]. The details of DDW are given in Appendix H.
4. **MDDW.** This is the MDDW scheme based on the framework in Sec. 4 with the underlying MDVS proposed in [AYSZ14], which is recalled in Appendix C.

Metrics. In this paper, we evaluate our scheme from the perspective of computational overhead. We measure the computational overhead by the *time of text generation* and the *time of watermark detection*. More exactly,

- Time of text generation. It measures the time required to generate the text of a pre-defined length (e.g., 50, 100, etc.).
- Time of watermark detection. It measures the time required to successfully detect a watermark in a given watermarked text.

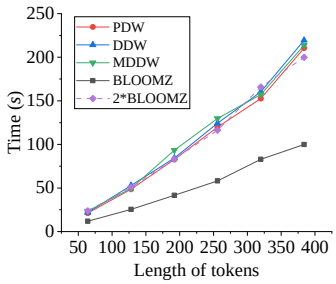
Implementation. We implement our experiments in Python language (version 3.8.10) based on elliptic curve groups with key size of 128 bits. The experiments are conducted on a PC, running Ubuntu 20.04 on one Intel® Core™ i9-10900K CPU@3.70 GHz and NVIDIA GeForce RTX 3080, and using 64 GB memory. We employ three LLMs to evaluate our different schemes, including **BLOOMZ** 3B model [MWS+22], **OPT** 1.3B model [ZRG+22] and **Gemma** 2B model [TMH+24].

Aligned with prior works, we embed our watermarks in *tokens*, (each contains 3 to 5 words), to ensure sufficient entropy. Aligned with prior works, we embed watermarks in *tokens* of 3 to 5 words for sufficient entropy. The parameters are set empirically (e.g., when a token contains 5 words in BLOOMZ, the outputs of the watermarking schemes contain no odd text), rather than from rigorous theoretical analysis.

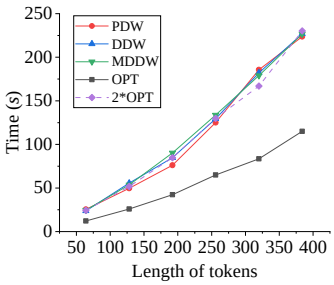
Experimental results. Here, we present the experimental results. These are illustrated in Fig. 11. Specifically, Fig. 11a shows the generation time when using the BLOOMZ model. Fig. 11b displays the generation time when using the OPT model. Fig. 11c illustrates the generation time when using the Gemma model. Fig. 11d depicts the time required for watermark detection when using the BLOOMZ model. Note that the computation of watermark detection is unrelated to the LLMs, hence we present only the results for the BLOOMZ model.

Time of text generation. From Fig. 11a to Fig. 11c, we observe that the text generation time for our schemes (DDW and MDDW) is nearly identical to that of PDW. However, it is important to note that PDW cannot support distortion-freeness, while ours not only satisfies distortion-freeness, but also satisfies other advanced security properties (e.g., off-the-record property).

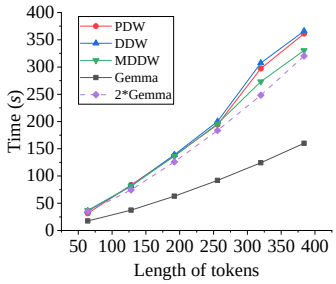
The dashed lines in Fig. 11a to Fig. 11c indicate twice the text generation times of the original LLMs. It is evident that the watermarking schemes, including PDW and our proposed methods, exhibit approximately the same time shown



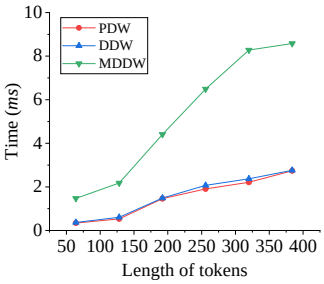
(a) Generation time for BLOOMZ



(b) Generation time for OPT



(c) Generation time for Gemma



(d) Watermark detection time

Fig. 11 Figures for time of text generation and watermark detection (the dashed lines are twice of the generation time of the corresponding LLMs)

by the dashed line. Since both the scheme in PDW and our schemes employ rejection sampling, there is only approximately 1/2 probability that a randomly sampled token meets the requirement. Theoretically, the expected time to sample a token is about 1.5 times that of the original LLM. The remaining time is dedicated to additional computations, such as generating signatures, hash computation, etc.

From Fig. 11a to Fig. 11c, it appears that DDW does not offer advantages over MDDW. In fact, these figures show the time of text generation when the output lengths are the same. Given the same output length, the number of watermarks in MDDW is only one-fourth of that in DDW in our experiments. Furthermore, from the perspective of generating a watermark (excluding the embedding time), the generation time for four watermarks in DDW is almost the same as that for one watermark in MDDW. Thus, the DDW in Appendix H is more efficient when considering only one designated verifier.

Time of watermark detection. Fig. 11d shows the time required for watermark detection. From the figure, it is evident that the detection time for DDW is comparable to that of PDW and significantly shorter than that of MDDW. A detailed theoretical analysis in Appendix H reveals that MDDW demands more exponential computations, while DDW relies on one bilinear map computation (so does PDW using the BLS signature with minor modifications in Appendix C). Although it is challenging to determine which type of computation is more

time-consuming purely from a theoretical standpoint, the experimental results clearly indicate that DDW is more efficient when considering only one designated verifier. Additionally, its performance is very close to that of PDW.

Overall, our scheme offers more advanced functionalities and greater flexibility than existing schemes (e.g., PDW), and the performance of our solutions is also acceptable.

Acknowledgements. We thank Sam Gunn and Miranda Christ for their valuable feedback on the initial version of this paper.

References

AYSZ14. Man Ho Au, Guomin Yang, Willy Susilo, and Yunmei Zhang. (strong) multidesignated verifiers signatures secure against rogue key attack. *Concurrency and Computation: Practice and Experience*, 26(8):1574–1592, 2014.

BLS04. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of cryptology*, 17:297–319, 2004.

CG24. Miranda Christ and Sam Gunn. Pseudorandom error-correcting codes. *arXiv preprint arXiv:2402.09370*, 2024.

CGZ24. Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. In *The Thirty Seventh Annual Conference on Learning Theory*, pages 1125–1139. PMLR, 2024.

CHS24. Aloni Cohen, Alexander Hoover, and Gabe Schoenbach. Enhancing watermarked language models to identify users. Cryptology ePrint Archive, Paper 2024/759, 2024. https://eprint.iacr.org/2024/759.

DHM+20. Ivan Damgård, Helene Haagh, Rebekah Mercer, Anca Nitulescu, Claudio Orlandi, and Sophia Yakoubov. Stronger security and constructions of multi-designated verifier signatures. In *TCC 2020*, pages 229–260. Springer, 2020.

DORS08. Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

FGJ+23. Jaiden Fairoze, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, and Mingyuan Wang. Publicly detectable watermarking for language models. Cryptology ePrint Archive, Paper 2023/1661, 2023. https://eprint.iacr.org/2023/1661.

Fow23. Geoffrey A Fowler. We tested a new chatgpt-detector for teachers. it flagged an innocent student. *Washington Post*, 2023.

HAS+23. Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. How good are gpt models at machine translation? a comprehensive evaluation. *arXiv preprint arXiv:2302.09210*, 2023.

HD23. Vojtěch Hudeček and Ondřej Dušek. Are llms all you need for task-oriented dialogue? *arXiv preprint arXiv:2304.06556*, 2023.

HS24. Yue Huang and Lichao Sun. Fakegpt: Fake news generation, explanation and detection of large language models, 2024.

Jim23. Kayla Jimenez. Professors are using chatgpt detector tools to accuse students of cheating. but what if the software is wrong. *USA Today*, 2023.

KGW+23. John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *ICML 2023*, volume 202. PMLR, 2023.

LV05. Fabien Laguillaumie and Damien Vergnaud. Designated verifier signatures: Anonymity and efficient construction from any bilinear map. In *SCN 2004*, pages 105–119. Springer, 2005.

MLK+23. Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature. In *International Conference on Machine Learning*, pages 24950–24962. PMLR, 2023.

MPR22. Ueli Maurer, Christopher Portmann, and Guilherme Rito. Multi-designated receiver signed public key encryption. In *EUROCRYPT 2022*, pages 644–673. Springer, 2022.

MWS+22. Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*, 2022.

NIR+23. Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. Lever: Learning to verify language-to-code generation with execution. In *ICML 2023*, pages 26106–26128. PMLR, 2023.

PS19. Sunoo Park and Adam Sealfon. It wasn't me! Repudiability and claimability of ring signatures. In *CRYPTO 2019*, pages 159–190. Springer, 2019.

PSF+23. Julien Piet, Chawin Sitawarin, Vivian Fang, Norman Mu, and David A. Wagner. Mark my words: Analyzing and evaluating language model watermarks. *CoRR*, abs/2312.00273, 2023.

QYH+24. Wenjie Qu, Dong Yin, Zixin He, Wei Zou, Tianyang Tao, Jinyuan Jia, and Jiaheng Zhang. Provably robust multi-bit watermarking for ai-generated text via error correction code. *arXiv preprint arXiv:2401.16820*, 2024.

RKX+23. Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *ICML 2023*, pages 28492–28518. PMLR, 2023.

SBWP03. Ron Steinfeld, Laurence Bull, Huaxiong Wang, and Josef Pieprzyk. Universal designated-verifier signatures. In *ASIACRYPT 2003*, pages 523–542. Springer, 2003.

Shi08. Kyung-Ah Shim. Rogue-key attacks on the multi-designated verifiers signature scheme. *Information processing letters*, 107(2):83–86, 2008.

TMH+24. Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

TMS+23. Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

TTE+23. Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. Large language models in medicine. *Nature medicine*, 29(8):1930–1940, 2023.

YHW+23. Kyosuke Yamashita, Keisuke Hara, Yohei Watanabe, Naoto Yanai, and Junji Shikata. Designated verifier signature with claimability. In *Pro-*

ceedings of the 10th ACM Asia Public-Key Cryptography Workshop, pages 21–32, 2023.

ZEF+23. Hanlin Zhang, Benjamin L Edelman, Danilo Francati, Daniele Venturi, Giuseppe Ateniese, and Boaz Barak. Watermarks in the sand: Impossibility of strong watermarking for generative models. *arXiv preprint arXiv:2311.04378*, 2023.

ZRG+22. Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

A Preliminaries: cryptographic assumptions and lemmas

Let $\mathsf{GenG}(1^\lambda)$ be a bilinear map setup algorithm, which takes the security parameter λ as input and outputs $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, p)$, where p is a prime of $\Theta(\lambda)$ bits, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order p, g is a generator of $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a nondegenerate and efficiently computable bilinear map.

Definition 16 (The DBDH assumption). *We say that the* decisional bilinear Diffie-Hellman (DBDH) assumption *holds with respect to* GenG*, if for any PPT adversary* $\mathcal{D}$*,*

$$\mathbf{Adv}^{\mathrm{dbdh}}_{\mathsf{GenG},\mathcal{D}}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{dbdh}}_{\mathsf{GenG},\mathcal{D}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda),$$

where $\mathbf{G}^{\mathrm{dbdh}}_{\mathsf{GenG},\mathcal{D}}(\lambda)$ *is in Fig. 12.*

Definition 17 (The CBDH assumption). *We say that the* computational bilinear Diffie-Hellman (CBDH) assumption *holds with respect to* GenG*, if for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\mathrm{cbdh}}_{\mathsf{GenG},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\mathrm{cbdh}}_{\mathsf{GenG},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathbf{G}^{\mathrm{cbdh}}_{\mathsf{GenG},\mathcal{A}}(\lambda)$ *is in Fig. 12.*

Definition 18 (The GBDH assumption). *We say that the* gap bilinear Diffie-Hellman (GBDH) assumption *holds with respect to* GenG*, if for any PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\mathrm{gbdh}}_{\mathsf{GenG},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\mathrm{gbdh}}_{\mathsf{GenG},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathbf{G}^{\mathrm{gbdh}}_{\mathsf{GenG},\mathcal{A}}(\lambda)$ *is in Fig. 12.*

For two random variables X_1 and X_2 over a set S, their statistical distance is denoted as $\mathbf{SD}(X_1, X_2) = \frac{1}{2}\sum_{x\in S} |\Pr[X_1 = x] - \Pr[X_2 = x]|$.

Lemma 1 ([DORS08]). *Assume that a family of hash functions* $\{H_x : \{0,1\}^n \to \{0,1\}^\ell\}_{x\in X}$ *is universal (i.e., for all* $a \neq b \in \{0,1\}^n$*,* $\Pr_{x\leftarrow X}[H_x(a) = H_x(b)] = 2^\ell$*). Then, for any random variable* W*,* $\mathbf{SD}((H_X(W), X), (U_\ell, X)) \leq \frac{1}{2}\sqrt{2^{-\mathbf{H}_\infty(W)}2^\ell}$*, where* U_ℓ *denotes the uniform distribution over* $\{0,1\}^\ell$*.*

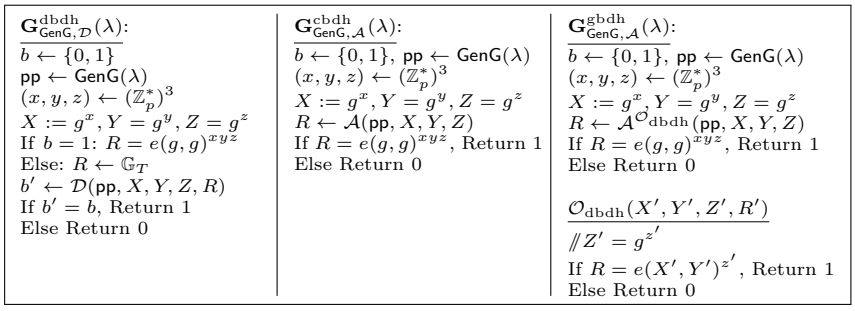


Fig. 12 Games $\mathbf{G}^{\mathrm{dbdh}}_{\mathsf{GenG},\mathcal{D}}(\lambda)$, $\mathbf{G}^{\mathrm{cbdh}}_{\mathsf{GenG},\mathcal{A}}(\lambda)$ and $\mathbf{G}^{\mathrm{gbdh}}_{\mathsf{GenG},\mathcal{A}}(\lambda)$

B Preliminaries: pseudorandom function, commitment, and signature

Definition 19 (Pseudorandom function). *A pseudorandom function (PRF)* PRF *is a pair of polynomial time algorithms* $(\mathsf{KG}, \mathsf{Eval})$ *that works as follows:*

- $\mathsf{KG}(1^\lambda) \to k$*: On inputting a security parameter* 1^λ*, it outputs a key* k*.*
- $\mathsf{Eval}(k, x) \to r$*: On inputting a key* k *and a string* $x \in \{0,1\}^*$*, it outputs a string* $r \in \{0,1\}^\lambda$*.*

A PRF should satisfy the following condition. For any sufficiently large security parameter 1^λ*, and* $k \leftarrow \mathsf{KG}(1^\lambda)$*, any truly random function* F *whose range is the same as* $\mathsf{Eval}(k, \cdot)$*, and any PPT distinguisher* $\mathcal{D}$*, it holds that*

$$|\Pr[\mathcal{D}^{\mathsf{Eval}(k,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{D}^{F(\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda).$$

Definition 20 (Commitment). *A* commitment *scheme with message space* $\mathcal{M}$ *contains three PPT algorithms* $\mathsf{Commit} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Decom})$*:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$*: The setup algorithm takes the security parameter* 1^λ *as input and output a public parameter* pp*.*
- $\mathsf{Com}(\mathsf{pp}, m; r_{\mathsf{com}}) \to \mathsf{com}$*: The commitment algorithm takes as input the public parameter* pp *and* $m \in \mathcal{M}$*, with an inner randomized input* r_{com}*, and outputs a commitment* com*.*
- $\mathsf{Decom}(\mathsf{pp}, \mathsf{com}, r_{\mathsf{com}}, m) \to b$*: The decommitment algorithm takes as input the public parameter pp, a commitment* com*, a decommitment* r_{com} *and a message* $m \in \mathcal{M}$*, and outputs a bit* $b \in \{0,1\}$ *depending on whether* m *is the committed message of* com*. One simple method is to re-generate the commitment via* Com *algorithm with input* $(\mathsf{pp}, m, r_{\mathsf{com}})$ *and check if the output equals to* com*.*

A commitment scheme enjoys the following properties:

– **Correctness.** *For all $m \in \mathcal{M}$ and all $r_{\mathsf{com}} \in \mathcal{RS}_{\mathsf{Com}}$, it holds that*

$$\Pr[\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{com} \leftarrow \mathsf{Com}(\mathsf{pp}, m; r_{\mathsf{com}}) : \mathsf{Decom}(\mathsf{pp}, \mathsf{com}, r_{\mathsf{com}}, m) = 1] = 1.$$

– **Binding.** *For any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{com}, m, r_{\mathsf{com}}, m', r'_{\mathsf{com}}) \leftarrow \mathcal{A}(\mathsf{pp})\end{array} : \begin{array}{c} m \neq m' \\ \wedge\ \mathsf{Decom}(\mathsf{pp}, \mathsf{com}, r_{\mathsf{com}}, m) = 1 \\ \wedge\ \mathsf{Decom}(\mathsf{pp}, \mathsf{com}, r'_{\mathsf{com}}, m') = 1\end{array}\right] \leq \mathsf{negl}(\lambda).$$

– **Hiding.** *For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*

$$\left|\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), b \leftarrow \{0,1\} \\ (m_0, m_1, st) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ \mathsf{com} \leftarrow \mathsf{Com}(\mathsf{pp}, m_b) \\ b' \leftarrow \mathcal{A}_2(\mathsf{com}, st)\end{array} : b' = b\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

If $\mathcal{A}$ is unbounded and $\mathsf{negl}(\lambda)$ is fixed to be 0, we say that the scheme Commit *is* perfect hiding.

Definition 21 (Signature). *A signature scheme for a message space $\mathcal{M}$ consists of four algorithms* $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Sign}, \mathsf{Verify})$.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$*: On input the security parameter 1^λ, the setup algorithm outputs a public parameter* pp.
- $\mathsf{KG}(\mathsf{pp}) \to (pk, sk)$*: On input* pp*, the key generation algorithm outputs a key pair (pk, sk).*
- $\mathsf{Sign}(\mathsf{pp}, sk, m) \to \sigma$*: On input* pp*, sk and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ.*
- $\mathsf{Verify}(\mathsf{pp}, pk, m, \sigma) \to b$*: On input* pp*, pk, m and a signature σ, the verification algorithm outputs a bit b.*

Correctness requires that for all $m \in \mathcal{M}$, it holds that:

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp})\end{array} : \mathsf{Verify}(\mathsf{pp}, pk, m, \mathsf{Sign}(\mathsf{pp}, sk, m)) = 1\right] = 1.$$

The signature should satisfy *existential unforgeability*. In other words, we say Sig is existentially unforgeable, if the following probability is negligible,

$$\mathbf{Adv}^{\mathrm{unforg}}_{\mathsf{Sig},\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{l}Q \leftarrow \emptyset,\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_S}(\mathsf{pp}, pk)\end{array} : \begin{array}{c}(m^*, *) \notin Q \\ \mathsf{Verify}(\mathsf{pp}, pk, m^*, \sigma^*) = 1\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the oracle $\mathcal{O}_S$ is defined as follows:

– $\mathcal{O}_S(m')$: On receiving m', find $(m', *)$ in Q. If some entry $(m', \sigma') \in Q$ is found, then output the corresponding σ'. Otherwise, compute $\sigma' \leftarrow \mathsf{Sign}(\mathsf{pp}, sk, m')$, set $Q \leftarrow Q \cup \{(m', \sigma')\}$, and output σ'.

C Preliminaries: A MDVS scheme in [AYSZ14] and BLS signature [BLS04]

MDVS. Here, we recall a MDVS scheme [AYSZ14] in Fig. 13.

It is based on discrete logarithm setting. Essentially, the signature is a ring signature with only two party. One party is the signer and the other one is all the designated verifiers. Since it is based on discrete logarithm and the key pairs of the verifiers are in the form of $(sk = x, pk = g^x)$, we can compute a public key for a "virtual" role representing all the designated verifiers in this way: $pk = \prod_{i \in S} pk_i$. Finally, adopting ring signatures to generate a signature for the signer and the virtual role. There are some improvements against rogue key attack [Shi08,AYSZ14]. Here we omit the details.

The existential unforgeability of the ring signature guarantees the existential unforgeability of the scheme [AYSZ14]. Off-the-record property for designated set lies in that all designated verifiers can together generate such a ring signature, and the security of anonymity of ring signature guarantees the indistinguishability from a real signature output by the signer in [AYSZ14]. Although consistency is not defined in [AYSZ14], the scheme is consistent, since the verification of the ring signature is public.

One of the advantages of the scheme [AYSZ14] is that the size of the MDVS signature is constant, no matter how many designated verifiers there are.

BLS signature. BLS signature [BLS04] is recalled in Fig. 14. Here, we make a bit modification. More exactly, the original signature is $\sigma := h^{sk}$ and the verification is to check whether $e(\sigma, g) = e(h, pk_{\mathrm{s}})$ or not. Here the signature would be a hash value, i.e., $\sigma := \mathsf{H}'(e(h^{sk}, g))$, and then the verification is to check whether $\sigma = \mathsf{H}'(e(h, pk_{\mathrm{s}}))$.

D Proof of Theorem 2

We provide the proofs of completeness, consistency, soundness, distortion-freeness, robustness, and off-the-record property for designated set of MDDW here.

D.1 Proof of completeness (in Theorem 2)

Proof. For any i, any prompt $\boldsymbol{p} \in \mathcal{T}^*$, any detector set S, any $j' \in S$, any normally generated pp, $(\mathsf{wpk}_i, \mathsf{wsk}_i)$ and $\{\mathsf{dpk}_j, \mathsf{dsk}_j\}_{j \in S}$, and for any token $\widehat{\boldsymbol{t}} \leftarrow \mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j \in S}, \boldsymbol{p})$, if WatMar successfully embeds a message/signature pair in $\widehat{\boldsymbol{t}}$, obviously the correctness of the underlying MDVS scheme guarantees $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_i, \mathsf{dsk}_{j'}, \{\mathsf{dpk}_j\}_{j \in S}, \widehat{\boldsymbol{t}}) = 1$. The only possibility that WatMar fails in embedding is that the rejection sampling algorithm fails to find the next batch of tokens whose hash is consistent with the target bit (i.e., line 10-11 in Algorithm 2). Note that by Assumption 1, any ℓ consecutive tokens generated by $\mathsf{GenModel}_\ell$ contains α bits of entropy. So no one can predict the input to the random oracle in line 10 of Algorithm 2 except with probability $2^{-\alpha}$, which is negligible. In other words, with overwhelming probability, the random oracle in

$\underline{\mathsf{Setup}(1^\lambda)}$:
$(\mathbb{G}, g, p) \leftarrow \mathsf{GenG}(1^\lambda)$ // g is a generator of $\mathbb{G}$ with prime order p
Choose a hash function H: $\{0,1\}^* \to \mathbb{Z}_p^*$
Return $\mathsf{pp} = (\mathbb{G}, g, p, \mathsf{H})$

$\underline{\mathsf{KG}(\mathsf{pp})}$:
Return $(sk \leftarrow \mathbb{Z}_p^*,\ pk := g^{sk})$

$\underline{\mathsf{Sign}(\mathsf{pp}, m, sk_\mathrm{s}, \mathsf{S} = (pk_{\mathrm{v}_i})_{i\in[n]})}$:
$Y := 1$
For $i \in [n]$: $h_i := \mathsf{H}(pk_\mathrm{s}, (pk_{\mathrm{v}_i})_{i\in[n]}, m, i)$, $Y = Y \cdot pk_{\mathrm{v}_i}^{h_i}$
$(r, c_2, z_2) \leftarrow (\mathbb{Z}_p^*)^3$, $T_1 := g^r$, $T_2 := Y^{c_2} g^{z_2}$
$c := \mathsf{H}(T_1, T_2, pk_\mathrm{s}, (pk_{\mathrm{v}_i}, h_i)_{i\in[n]}, m, Y)$
$c_1 := c - c_2$, $z_1 := r - c_1 \cdot sk_\mathrm{s}$
Return $\sigma = (c_1, c_2, z_1, z_2)$

$\underline{\mathsf{Verify}(\mathsf{pp}, m, pk_\mathrm{s}, \sigma, \mathsf{S})}$:
$Y := 1$
For $i \in [n]$: $h_i := \mathsf{H}(pk_\mathrm{s}, (pk_{\mathrm{v}_i})_{i\in[n]}, m, i)$, $Y = Y \cdot pk_{\mathrm{v}_i}^{h_i}$
$T_1' := pk_\mathrm{s}^{c_1} g^{z_1}$, $T_2' := Y^{c_2} g^{z_2}$
$c' := \mathsf{H}(T_1', T_2', pk_\mathrm{s}, (pk_{\mathrm{v}_i}, h_i)_{i\in[n]}, m, Y)$
If $(c' = c_1 + c_2)$: Return $b = 1$
Else Return $b = 0$

Fig. 13 Algorithms for multi-designated verifiers signature MDVS in [AYSZ14]

$\underline{\mathsf{Setup}(1^\lambda)}$:
$(\mathbb{G}, \mathbb{G}_T, e, g, p) \leftarrow \mathsf{GenG}(1^\lambda)$
Choose a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$
Choose a hash function $\mathsf{H}' : \mathbb{G}_T \to \{0,1\}^l$
Return $\mathsf{pp} = (\mathbb{G}, \mathbb{G}_T, e, g, p, \mathsf{H}, \mathsf{H}')$

$\underline{\mathsf{KG}(\mathsf{pp})}$:
$sk \leftarrow \mathbb{Z}_p^*$, $pk := g^{sk}$
Return (sk, pk)

$\underline{\mathsf{Sign}(\mathsf{pp}, m, sk_\mathrm{s})}$:
$h \leftarrow \mathsf{H}(m)$
Return $\sigma := \mathsf{H}'(e(h^{sk}, g))$

$\underline{\mathsf{Verify}(\mathsf{pp}, m, pk_\mathrm{s}, \sigma)}$:
If $(\sigma = \mathsf{H}'(e(h, pk_\mathrm{s})))$: Return 1
Return 0

Fig. 14 BLS signature

line 10 of Algorithm 2 will return a uniformly and independently sampled bit. Hence, with overwhelming probability, we derive that for each sampling attempt of the next batch of tokens, it will succeed in finding a consistent hash value with probability $\frac{1}{2}$. After $\mathsf{poly}(\lambda)$ attempts, the rejection sampling will find the next batch of tokens with probability $1 - 2^{-\mathsf{poly}(\lambda)}$, i.e., the probability that WatMar fails is negligible. □

D.2 Proof of consistency (in Theorem 2)

Proof. For any PPT adversary $\mathcal{A}$ attacking the consistency of MDDW, we show a PPT adversary $\mathcal{B}$ attacking the consistency of the underlying MDVS as follows.

Upon receiving pp (note that the pp generated by Setup of MDDW is the same as that generated by the underlying MDVS.Setup), $\mathcal{B}$ sends pp to $\mathcal{A}$. $\mathcal{B}$ maintains three local arrays, $L_{\mathrm{ro},1}$, $L_{\mathrm{ro},2}$ and $L_{\mathrm{ro},3}$, to keep track of $\mathcal{A}$'s random oracle queries. Then, $\mathcal{B}$ answers $\mathcal{A}$'s oracle queries as below:

- $\mathcal{O}_{RO,\kappa}(str)$ $(\kappa \in \{1,2,3\})$: If there is some $(str, y) \in L_{\mathrm{ro},\kappa}$, $\mathcal{B}$ returns y; otherwise, $\mathcal{B}$ samples $y \leftarrow \mathcal{M}$, adds (str, y) to $L_{\mathrm{ro},\kappa}$, and returns y.
- $\mathcal{O}_{WK}(i)$: $\mathcal{B}$ queries its own oracle $\mathcal{O}_{SK}(i)$ to obtain $(\mathsf{spk}_i, \mathsf{ssk}_i)$, and returns $(\overline{\mathsf{wpk}_i, \mathsf{wsk}_i}) = (\mathsf{spk}_i, \mathsf{ssk}_i)$ to $\mathcal{A}$.
- $\mathcal{O}_{DK}(j)$: $\mathcal{B}$ queries its own oracle $\mathcal{O}_{VK}(j)$ to obtain $(\mathsf{vpk}_j, \mathsf{vsk}_j)$, and returns $(\overline{\mathsf{dpk}_j, \mathsf{dsk}_j}) = (\mathsf{vpk}_j, \mathsf{vsk}_j)$ to $\mathcal{A}$.
- $\mathcal{O}_{WPK}(i)$: $\mathcal{B}$ queries its own oracle $\mathcal{O}_{SPK}(i)$ to obtain spk_i, and returns $\overline{\mathsf{wpk}_i = \mathsf{spk}_i}$ to $\mathcal{A}$.
- $\mathcal{O}_{DPK}(j)$: $\mathcal{B}$ queries its own oracle $\mathcal{O}_{VPK}(j)$ to obtain vpk_j, and returns $\overline{\mathsf{dpk}_j = \mathsf{vpk}_j}$ to $\mathcal{A}$.
- $\mathcal{O}_W(i, S, \boldsymbol{p})$: $\mathcal{B}$ runs SimWatMar (with the help of its signing oracle $\mathcal{O}_S$) as shown in Algorithm 4. More specifically, Algorithm 4 is the same as Algorithm 2, except that the original hash functions H_1, H_2 and H_3 (in line 4, line 5 and line 10, respectively) in Algorithm 2 are replaced with random oracles, and the original Sign algorithm (in line 4) in Algorithm 2 is replaced with $\mathcal{B}$'s signing oracle $\mathcal{O}_S$. Finally, $\mathcal{B}$ returns the output $\boldsymbol{t}$ of SimWatMar to $\mathcal{A}$.
- $\mathcal{O}_D(i, j', S, \boldsymbol{t})$: $\mathcal{B}$ runs SimDetect (with the help of its verification oracle $\mathcal{O}_V$) as shown in Algorithm 5. More specifically, Algorithm 5 is the same as Algorithm 3, except that the original hash functions H_1, H_2 and H_3 (in line 3, line 7 and line 6, respectively) in Algorithm 3 are replaced with random oracles, and the original Verify algorithm (in line 8) in Algorithm 3 is replaced with $\mathcal{B}$'s verification oracle $\mathcal{O}_V$. Finally, $\mathcal{B}$ returns the output of SimDetect to $\mathcal{A}$.

Upon receiving $(i^*, S^*, \boldsymbol{t}^*)$ from $\mathcal{A}$, $\mathcal{B}$ checks whether there are $j_0^*, j_1^* \in S^*$ such that $\mathsf{SimDetect}^{\mathcal{O}_V, \{\mathcal{O}_{RO,\kappa}\}_{\kappa\in\{1,2,3\}}}(\mathsf{pp}, i^*, j_\beta^*, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*) = \beta$ for all $\beta \in \{0,1\}$.

- If not, $\mathcal{B}$ returns a random tuple $(i^*_{\mathsf{rand}}, S^*_{\mathsf{rand}}, m^*_{\mathsf{rand}}, \sigma^*_{\mathsf{rand}})$ as its final output.
- Otherwise, $\mathcal{B}$ runs the algorithm SimDetect to find the tuple $(\widetilde{m}, \hat{\sigma})$ such that

$$\mathsf{SimDetect}^{\mathcal{O}_V, \{\mathcal{O}_{RO,\kappa}\}_{\kappa\in\{1,2,3\}}}(\mathsf{pp}, i^*, j_1^*, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*) = 1,$$

  (i.e., $\mathcal{O}_V(i^*, j_1^*, S^*, \widetilde{m}, \hat{\boldsymbol{\sigma}}) = 1$ in line 8 of Algorithm 5). Then, $\mathcal{B}$ returns a random tuple $(i^*, S^*, \widetilde{m}, \hat{\boldsymbol{\sigma}}$ as its final output.

That is the construction of adversary $\mathcal{B}$.

It is easy to see that $\mathcal{B}$ perfectly simulates game $\mathbf{G}^{\mathrm{cons}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ for $\mathcal{A}$, and if $\mathcal{A}$ wins $\mathbf{G}^{\mathrm{cons}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$, then $\mathcal{B}$ also wins $\mathbf{G}^{\mathrm{cons}}_{\mathsf{MDVS},\mathcal{B}}(\lambda)$. So we obtain

$$\mathbf{Adv}^{\mathrm{cons}}_{\mathsf{MDVS},\mathcal{B}}(\lambda) \geq \mathbf{Adv}^{\mathrm{cons}}_{\mathsf{MDDW},\mathcal{A}}(\lambda).$$

□

Algorithm 4 $\mathsf{SimWatMar}^{\mathcal{O}_S,\{\mathcal{O}_{RO,\kappa}\}_{\kappa\in\{1,2,3\}}}(\mathsf{pp}, i, S, \boldsymbol{p})$

$\boldsymbol{t} \leftarrow \epsilon$
while $|\boldsymbol{t}| + \ell + \ell \cdot \mathsf{len}_{\mathrm{sig}} < n$ **do**
  $\boldsymbol{t} \leftarrow \boldsymbol{t} || \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \mathcal{O}_S(i, S, \underline{\mathcal{O}_{RO,1}}(\boldsymbol{t}[-\ell:]))$
  $\boldsymbol{\sigma} \leftarrow \hat{\boldsymbol{\sigma}} \oplus \underline{\mathcal{O}_{RO,2}}(\boldsymbol{t}[-\ell:])$
  $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \epsilon,\ \boldsymbol{m} \leftarrow \epsilon$
  while $\boldsymbol{\sigma} \neq \epsilon$ **do**
    $\sigma_{\mathrm{bit}} \leftarrow \boldsymbol{\sigma}[1],\ \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[2:]$
    $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    while $\underline{\mathcal{O}_{RO,3}}(\boldsymbol{m}||\boldsymbol{x}||\boldsymbol{\sigma}_{\mathrm{prev}}) \neq \sigma_{\mathrm{bit}}$ **do**
      $\boldsymbol{x} \leftarrow \underline{\mathsf{GenModel}_\ell}(\boldsymbol{p}, \boldsymbol{t})$
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{x}$
    $\boldsymbol{t} \leftarrow \boldsymbol{t}||\boldsymbol{x}$
    $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \boldsymbol{\sigma}_{\mathrm{prev}}||\sigma_{\mathrm{bit}}$
if $|\boldsymbol{t}| < n$ **then**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{p}, \boldsymbol{t})$
return $\boldsymbol{t}$

Algorithm 5 $\mathsf{SimDetect}^{\mathcal{O}_V,\{\mathcal{O}_{RO,\kappa}\}_{\kappa\in\{1,2,3\}}}(\mathsf{pp}, i, j', S, \boldsymbol{t})$

for $\mu = 1, 2, \cdots, n - (\ell + \ell \cdot \mathsf{len}_{\mathrm{sig}})$ **do**
  $\widetilde{m} \leftarrow \underline{\mathcal{O}_{RO,1}}(\boldsymbol{t}[\mu : \mu + \ell - 1])$
  $\boldsymbol{\sigma} \leftarrow \epsilon,\ \boldsymbol{m} \leftarrow \epsilon$
  for $\varphi \in [\mathsf{len}_{\mathrm{sig}}]$ **do**
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{t}[\mu + \varphi \cdot \ell : \mu + \varphi \cdot \ell + \ell - 1]$
    $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}||\underline{\mathcal{O}_{RO,3}}(\boldsymbol{m}||\boldsymbol{\sigma})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \boldsymbol{\sigma} \oplus \underline{\mathcal{O}_{RO,2}}(\boldsymbol{t}[\mu : \mu + \ell - 1])$
  if $\underline{\mathcal{O}_V(i, j', S, \widetilde{m}, \hat{\boldsymbol{\sigma}})} = 1$ **then**
    return 1
return 0

D.3 Proof of soundness (in Theorem 2)

Proof. For any PPT adversary $\mathcal{A}$ attacking the soundness of MDDW, we show a PPT adversary $\mathcal{B}$ attacking the existential unforgeability of the underlying MDVS as follows.

Upon receiving pp (note that the pp generated by Setup of MDDW is the same as that generated by the underlying MDVS.Setup), $\mathcal{B}$ sends pp to $\mathcal{A}$. $\mathcal{B}$ maintains three local arrays, $L_{\mathrm{ro},1}$, $L_{\mathrm{ro},2}$ and $L_{\mathrm{ro},3}$, to keep track of $\mathcal{A}$'s random oracle queries. Then, $\mathcal{B}$ answers $\mathcal{A}$'s oracle queries as below:

- $\underline{\mathcal{O}_{RO,\kappa}(str)}$ $(\kappa \in \{1,2,3\})$: If there is some $(str, y) \in L_{\mathrm{ro},\kappa}$, $\mathcal{B}$ returns y; otherwise, $\mathcal{B}$ samples $y \leftarrow \mathcal{M}$, adds (str, y) to $L_{\mathrm{ro},\kappa}$, and returns y.
- $\mathcal{O}_{WK}(i)$: $\mathcal{B}$ queries its own oracle $\mathcal{O}_{SK}(i)$ to obtain $(\mathsf{spk}_i, \mathsf{ssk}_i)$, and returns $(\underline{\mathsf{wpk}_i, \mathsf{wsk}_i}) = (\mathsf{spk}_i, \mathsf{ssk}_i)$ to $\mathcal{A}$.

- $\underline{\mathcal{O}_{DK}(j)}$: $\mathcal{B}$ queries its own oracle $\mathcal{O}_{VK}(j)$ to obtain $(\mathsf{vpk}_j, \mathsf{vsk}_j)$, and returns $(\overline{\mathsf{dpk}_j, \mathsf{dsk}_j}) = (\mathsf{vpk}_j, \mathsf{vsk}_j)$ to $\mathcal{A}$.
- $\underline{\mathcal{O}_{WPK}(i)}$: $\mathcal{B}$ queries its own oracle $\mathcal{O}_{SPK}(i)$ to obtain spk_i, and returns $\overline{\mathsf{wpk}_i = \mathsf{spk}_i}$ to $\mathcal{A}$.
- $\underline{\mathcal{O}_{DPK}(j)}$: $\mathcal{B}$ queries its own oracle $\mathcal{O}_{VPK}(j)$ to obtain vpk_j, and returns $\overline{\mathsf{dpk}_j = \mathsf{vpk}_j}$ to $\mathcal{A}$.
- $\underline{\mathcal{O}_W(i, S, \boldsymbol{p})}$: $\mathcal{B}$ runs $\mathsf{SimWatMar}$ (with the help of its signing oracle $\mathcal{O}_S$) as shown in Algorithm 4. More specifically, Algorithm 4 is the same as Algorithm 2, except that the original hash functions H_1, H_2 and H_3 (in line 4, line 5 and line 10, respectively) in Algorithm 2 are replaced with random oracles, and the original Sign algorithm (in line 4) in Algorithm 2 is replaced with $\mathcal{B}$'s signing oracle $\mathcal{O}_S$. Finally, $\mathcal{B}$ returns the output $\boldsymbol{t}$ of $\mathsf{SimWatMar}$ to $\mathcal{A}$.
- $\underline{\mathcal{O}_D(i, j', S, \boldsymbol{t})}$: $\mathcal{B}$ runs $\mathsf{SimDetect}$ (with the help of its verification oracle $\mathcal{O}_V$) as shown in Algorithm 5. More specifically, Algorithm 5 is the same as Algorithm 3, except that the original hash functions H_1, H_2 and H_3 (in line 3, line 7 and line 6, respectively) in Algorithm 3 are replaced with random oracles, and the original Verify algorithm (in line 8) in Algorithm 3 is replaced with $\mathcal{B}$'s verification oracle $\mathcal{O}_V$. Finally, $\mathcal{B}$ returns the output of $\mathsf{SimDetect}$ to $\mathcal{A}$.

Receiving $(i^*, S^*, \boldsymbol{t}^*)$ from $\mathcal{A}$, $\mathcal{B}$ firstly sets $\widehat{i^*} = i^*$ and $\widehat{S^*} = S^*$. Then, $\mathcal{B}$ checks whether there is some $j' \in S^*$ such that (i) $\mathsf{SimDetect}^{\mathcal{O}_V, \{\mathcal{O}_{RO,\kappa}\}_{\kappa\in\{1,2,3\}}}(\mathsf{pp}, i^*, j', (\mathsf{dpk}_j)_{i\in S^*}, \boldsymbol{t}^*) = 1$, and (ii) j' has never been queried to $\mathcal{O}_{DK}$ by $\mathcal{A}$.

- If so, $\mathcal{B}$ extracts $(\widetilde{m}, \hat{\boldsymbol{\sigma}})$ according to the steps in Algorithm 5 (from line 1 to line 7), and sets $(\widehat{m^*}, \widehat{\sigma^*}) = (\widetilde{m}, \hat{\boldsymbol{\sigma}})$.
- Otherwise, $\mathcal{B}$ uniformly samples $(\widehat{m^*}, \widehat{\sigma^*})$ from the message space and the signature space.

Finally, $\mathcal{B}$ returns $(\widehat{i^*}, \widehat{S^*}, \widehat{m^*}, \widehat{\sigma^*})$ as its final output.

That is the construction of adversary $\mathcal{B}$. Now we analyze its advantage.

It is easy to see that $\mathcal{B}$ perfectly simulates game $\mathbf{G}^{\text{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ for $\mathcal{A}$. Note that when $\mathcal{A}$ wins $\mathbf{G}^{\text{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$, $\mathcal{A}$'s final output $(i^*, S^*, \boldsymbol{t}^*)$ satisfies that there is some $j' \in S^*$, such that (i) $\mathcal{A}$ has never queried the oracle $\mathcal{O}_{WK}$ (resp., $\mathcal{O}_{DK}$) on i^* (resp., j'); (ii) $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_{i^*}, \mathsf{dsk}_{j'}, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*) = 1$; (iii) $\mathsf{NOLap}_\ell(\boldsymbol{t}^*, \boldsymbol{t}_1, \boldsymbol{t}_2, \cdots) = 1$, where $\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots$ are the watermarked texts that $\mathcal{A}$ receives via querying $\mathcal{O}_W$. The fact (ii) implies that there must be some $\mu' \in \{0, 1, \cdots, n - (\ell + \mathsf{len}_{\text{sig}})\}$, such that for $\widetilde{m} = \mathcal{O}_{RO,1}(\boldsymbol{t}^*[\mu' : \mu' + \ell - 1])$ and for $\hat{\boldsymbol{\sigma}}$ generated according to line 2-7 in Algorithm 3 (when $\mu = \mu'$), $\mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_{i^*}, \mathsf{vsk}_{j'}, \{\mathsf{vpk}_j\}_{j\in S^*}, \widetilde{m}, \hat{\boldsymbol{\sigma}}) = 1$. Moreover, the fact (iii) guarantees that during the process of answering $\mathcal{A}$'s $\mathcal{O}_W$-oracle queries, the random oracle $\mathcal{O}_{RO,1}$ has never been programmed on $\boldsymbol{t}^*[\mu' : \mu' + \ell - 1]$. So the property of random oracle guarantees that $\widetilde{m} = \mathcal{O}_{RO,1}(\boldsymbol{t}^*[\mu' : \mu' + \ell - 1])$ has never been submitted as a signing query by $\mathcal{B}$.

Hence, $\mathcal{B}$ wins $\mathbf{G}^{\text{unforg}}_{\mathsf{MDVS},\mathcal{B}}(\lambda)$ if $\mathcal{A}$ wins $\mathbf{G}^{\text{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$, i.e.,

$$\mathbf{Adv}^{\text{unforg}}_{\mathsf{MDVS},\mathcal{B}}(\lambda) \geq \mathbf{Adv}^{\text{sound}}_{\mathsf{MDDW},\mathcal{A}}(\lambda).$$

□

D.4 Proof of distortion-freeness (in Theorem 2)

Proof. We show the proof with a sequence of games.

Game $\mathbf{G}_0$: This game is the original $\mathbf{G}^{\text{dist-fr}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ when b is fixed to be 0. Note that in this game, when $\mathcal{A}$ queries the oracle $\mathcal{O}^{(0)}_{M\text{-}chl}$ on $(i, S, \boldsymbol{p})$, the challenger runs Algorithm 2 with input $(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{p})$ and returns the output to $\mathcal{A}$.

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that when $\mathcal{A}$ queries the oracle $\mathcal{O}^{(0)}_{M\text{-}chl}$ on $(i, S, \boldsymbol{p})$, the challenger runs Algorithm 2 with the following modifications: for any $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$ in line 3 of Algorithm 2, if $\boldsymbol{t}[-\ell:]$ has been queried to the random oracle $\mathcal{O}_{RO,2}$ by $\mathcal{A}$, then the challenger aborts the game and returns 0 as the final output.

Game $\mathbf{G}_2$: This game is the original $\mathbf{G}^{\text{dist-fr}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ when b is fixed to be 1.

We present the following two lemmas with postponed proofs.

Lemma 2. $|\Pr[\mathbf{G}_1 = 1] - \Pr[\mathbf{G}_0 = 1]| \leq \mathsf{negl}(\lambda)$.

Lemma 3. $|\Pr[\mathbf{G}_2 = 1] - \Pr[\mathbf{G}_1 = 1]| \leq \mathsf{negl}(\lambda)$.

Combining these two lemmas, we derive that for any PPT adversary $\mathcal{A}$,

$$\mathbf{Adv}^{\text{dist-fr}}_{\mathsf{MDDW},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Hence, what remains is to prove the above two lemmas.

Proof (of Lemma 2). Note that $\mathsf{GenModel}_\ell$ generates tokens with min-entropy at least α. So for each $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$ in line 3 of Algorithm 2, the probability that $\mathcal{A}$ has queried the random oracle $\mathcal{O}_{RO,2}$ on $\boldsymbol{t}[-\ell:]$ is at most $2^{-\alpha}$. Within Algorithm 2, the loop spanning from line 2 to line 12 will iterate at most $\frac{n}{\ell}$ times. Hence,

$$|\Pr[\mathbf{G}_2 = 1] - \Pr[\mathbf{G}_1 = 1]| \leq \frac{n}{\ell} \cdot 2^{-\alpha} \leq \mathsf{negl}(\lambda).$$

□

Proof (of Lemma 3). Compared with the oracle $\mathcal{O}^{(0)}_{M\text{-}chl}$ in $\mathbf{G}_1$, the oracle $\mathcal{O}^{(1)}_{M\text{-}chl}$ in $\mathbf{G}_2$ can be seen as that ignoring the process of checking whether $H_3(\boldsymbol{m}||\boldsymbol{x}||\boldsymbol{\sigma}_{\text{prev}}) = \sigma_{\text{bit}}$ (i.e., line 10 in Algorithm 2) in the oracle $\mathcal{O}^{(0)}_{M\text{-}chl}$ in $\mathbf{G}_1$, where H_3 is modeled as the random oracle $\mathcal{O}_{RO,3}$.

Note that in $\mathbf{G}_1$, to answer $\mathcal{A}$'s query $(i, S, \boldsymbol{p})$ (to $\mathcal{O}^{(0)}_{M\text{-}chl}$), while $|\boldsymbol{t}|+\ell+\mathsf{len}_{\text{sig}}$ (i.e., lines 2-12 in Algorithm 2), the challenger firstly generates ℓ tokens with $\mathsf{GenModel}_\ell$ (i.e., line 3); then, it samples the next $\mathsf{len}_{\text{sig}} \cdot \ell$ tokens satisfying that for each $i \in [\mathsf{len}_{\text{sig}}]$, the next i-th tuple of tokens (with length ℓ) are generated with $\mathsf{GenModel}_\ell$ (i.e., line 9) conditioned on the hash to be the i-th bit of $\boldsymbol{\sigma}$. Since each bit of $\boldsymbol{\sigma}$ is uniformly and independently distributed over $\{0, 1\}$, and $\mathsf{GenModel}_\ell$ generate tokens with min-entropy at least α, according to Lemma 1, for each of $\mathcal{A}$'s query $(i, S, \boldsymbol{p})$, the statistical distance of the answer from $\mathcal{O}^{(0)}_{M\text{-}chl}$ in $\mathbf{G}_1$ and that from $\mathcal{O}^{(1)}_{M\text{-}chl}$ in $\mathbf{G}_2$ is $\mathsf{len}_{\text{sig}} \cdot 2^{-\frac{\alpha+1}{2}}$, which is negligible.

The total number of $\mathcal{A}$'s oracle queries is polynomial. So we actually derive $\mathbf{SD}(\mathbf{G}_2, \mathbf{G}_1) \leq \mathsf{negl}(\lambda)$. □

□

D.5 Proof of robustness (in Theorem 2)

Proof. According to the descriptions of Algorithm 2 and Algorithm 3, it is easy to see that for every $(2\mathsf{len}_{\mathrm{sig}}+2)\ell$ consecutive tokens $\boldsymbol{t}$ generated from Algorithm 2, at least a pair of message and signature is embedded in $\boldsymbol{t}$. Hence, Algorithm 3 will extract the message/signature pair from $\boldsymbol{t}$, leading to a successful detection output. □

D.6 Proof of off-the-record property for designated set (in Theorem 2)

Proof. Since MDVS is off-the-record for designated set, there is a PPT algorithm $\mathsf{FgeDS}_{\mathsf{MDVS}}$, taking $(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vsk}_j\}_{j\in S}, m)$ as input and outputting σ, such that for all PPT adversary $\mathcal{A}'$,

$$\mathbf{Adv}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A}',\mathsf{FgeDS}_{\mathsf{MDVS}}}(\lambda) = |\Pr[\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A}',\mathsf{FgeDS}_{\mathsf{MDVS}}}(\lambda)=1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda).$$

Now we construct a PPT algorithm FgeDS for MDDW as shown in Fig. 6.

Algorithm 6 $\mathsf{FgeDS}(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dsk}_j\}_{j\in S}, \boldsymbol{p})$

$\mathsf{spk}_i \leftarrow \mathsf{wpk}_i$
$\{\mathsf{vsk}_j\}_{j\in S} \leftarrow \{\mathsf{dsk}_j\}_{j\in S}$
$\boldsymbol{t} \leftarrow \epsilon$
while $|\boldsymbol{t}| + \ell + \ell \cdot \mathsf{len}_{\mathrm{sig}} < n$ **do**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \mathsf{FgeDS}_{\mathsf{MDVS}}(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vsk}_j\}_{j\in S}, H_1(\boldsymbol{t}[-\ell:]))$
  $\boldsymbol{\sigma} \leftarrow \hat{\boldsymbol{\sigma}} \oplus H_2(\boldsymbol{t}[-\ell:])$
  $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \epsilon,\ \boldsymbol{m} \leftarrow \epsilon$
  while $\boldsymbol{\sigma} \neq \epsilon$ **do**
    $\sigma_{\mathrm{bit}} \leftarrow \boldsymbol{\sigma}[1],\ \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[2:]$
    $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    while $H_3(\boldsymbol{m}||\boldsymbol{x}||\boldsymbol{\sigma}_{\mathrm{prev}}) \neq \sigma_{\mathrm{bit}}$ **do**
      $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{x},\ \boldsymbol{t} \leftarrow \boldsymbol{t}||\boldsymbol{x}$
    $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \boldsymbol{\sigma}_{\mathrm{prev}}||\sigma_{\mathrm{bit}}$
if $|\boldsymbol{t}| < n$ **then**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{p}, \boldsymbol{t})$
return $\boldsymbol{t}$

For any PPT adversary $\mathcal{A}$ attacking the off-the-record property (for designated set) of MDDW, we show a PPT adversary $\mathcal{A}'$ attacking the off-the-record property (for designated set) of MDVS as follows.

Receiving pp (note that the pp generated by Setup of MDDW is the same as that generated by the underlying $\mathsf{MDVS.Setup}$), $\mathcal{A}'$ sends pp to $\mathcal{A}$. $\mathcal{A}'$ maintains three local arrays, $L_{\mathrm{ro},1}$, $L_{\mathrm{ro},2}$ and $L_{\mathrm{ro},3}$, to keep track of $\mathcal{A}$'s random oracle queries. Then, $\mathcal{A}'$ answers $\mathcal{A}$'s oracle queries as below:

- $\underline{\mathcal{O}_{RO,\kappa}(str)\ (\kappa \in \{1,2,3\})}$: If there is some $(str, y) \in L_{\mathrm{ro},\kappa}$, $\mathcal{A}'$ returns y; otherwise, $\mathcal{A}'$ samples $y \leftarrow \mathcal{M}$, adds (str, y) to $L_{\mathrm{ro},\kappa}$, and returns y.
- $\underline{\mathcal{O}_{WK}(i)}$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{SK}(i)$ to obtain $(\mathsf{spk}_i, \mathsf{ssk}_i)$, and returns $(\overline{\mathsf{wpk}_i, \mathsf{wsk}_i}) = (\mathsf{spk}_i, \mathsf{ssk}_i)$ to $\mathcal{A}$.
- $\underline{\mathcal{O}_{DK}(j)}$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{VK}(j)$ to obtain $(\mathsf{vpk}_j, \mathsf{vsk}_j)$, and returns $(\overline{\mathsf{dpk}_j, \mathsf{dsk}_j}) = (\mathsf{vpk}_j, \mathsf{vsk}_j)$ to $\mathcal{A}$.
- $\underline{\mathcal{O}_{WPK}(i)}$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{SPK}(i)$ to obtain spk_i, and returns $\overline{\mathsf{wpk}_i = \mathsf{spk}_i}$ to $\mathcal{A}$.
- $\underline{\mathcal{O}_{DPK}(j)}$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{VPK}(j)$ to obtain vpk_j, and returns $\overline{\mathsf{dpk}_j = \mathsf{vpk}_j}$ to $\mathcal{A}$.
- $\underline{\mathcal{O}^{(b)}_{otr\text{-}chl}(i, S, \boldsymbol{p})}$: $\mathcal{A}'$ runs $\mathsf{SimOtr\text{-}Chl}_{\mathrm{DS}}$ (with the help of its challenge oracle $\mathcal{O}^{(b)}_{otr\text{-}chl}$ for MDVS, denoted as $\mathcal{O}^{(b),\mathsf{MDVS}}_{otr\text{-}chl}$) as shown in Algorithm 7. In particular, Algorithm 7 is the same as Algorithm 4, except that the signing oracle $\mathcal{O}_S$ (in line 4) in Algorithm 4 is replaced with $\mathcal{A}'$'s challenge oracle $\mathcal{O}^{(b),\mathsf{MDVS}}_{otr\text{-}chl}$. Finally, $\mathcal{A}'$ returns the output $\boldsymbol{t}$ of $\mathsf{SimOtr\text{-}Chl}_{\mathrm{DS}}$ to $\mathcal{A}$.
- $\underline{\mathcal{O}_D(i, j', S, \boldsymbol{t})}$: $\mathcal{A}'$ runs $\mathsf{SimDetect}$ (with the help of its verification oracle $\mathcal{O}_V$) as shown in Algorithm 5. More specifically, Algorithm 5 is the same as Algorithm 3, except that the original hash functions H_1, H_2 and H_3 (in line 3, line 7 and line 6, respectively) in Algorithm 3 are replaced with random oracles, and the original Verify algorithm (in line 8) in Algorithm 3 is replaced with $\mathcal{A}'$'s verification oracle $\mathcal{O}_V$. Finally, $\mathcal{A}'$ returns the output of $\mathsf{SimDetect}$ to $\mathcal{A}$.

Finally, $\mathcal{A}'$ returns $\mathcal{A}$'s output b' as its own final output.

That is the construction of $\mathcal{A}'$.

It is obvious that $\mathcal{A}'$ perfectly simulates game $\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeDS}}(\lambda)$ for $\mathcal{A}$, and if $\mathcal{A}$ wins $\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeDS}}(\lambda)$, then $\mathcal{A}'$ also wins $\mathbf{G}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A}',\mathsf{FgeDS}_{\mathsf{MDVS}}}(\lambda)$. So we derive

$$\mathbf{Adv}^{\text{otr-ds}}_{\mathsf{MDVS},\mathcal{A}',\mathsf{FgeDS}_{\mathsf{MDVS}}}(\lambda) \geq \mathbf{Adv}^{\text{otr-ds}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeDS}}(\lambda).$$

□

E Proof of Theorem 3

Proof. Since MDVS is off-the-record for any subset, there is a PPT algorithm $\mathsf{FgeAS}_{\mathsf{MDVS}}$, taking $(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \{\mathsf{vsk}_j\}_{j\in S_{\mathrm{cor}}}, m)$ (where $S_{\mathrm{cor}} \subset S$) as input and outputting σ, such that for all PPT adversary $\mathcal{A}'$,

$$\mathbf{Adv}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A}',\mathsf{FgeAS}_{\mathsf{MDVS}}}(\lambda) = |\Pr[\mathbf{G}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A}',\mathsf{FgeAS}_{\mathsf{MDVS}}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda).$$

Now we construct a PPT algorithm FgeAS for MDDW as shown in Fig. 8.

Algorithm 7 $\mathsf{SimOtr\text{-}Chl}_{\mathrm{DS}}^{\mathcal{O}_{otr\text{-}chl}^{(b),\mathsf{MDVS}},\{\mathcal{O}_{RO,\kappa}\}_{\kappa\in\{1,2,3\}}}(\mathsf{pp}, i, S, \boldsymbol{p})$

$\boldsymbol{t} \leftarrow \epsilon$
while $|\boldsymbol{t}| + \ell + \ell \cdot \mathsf{len}_{\mathrm{sig}} < n$ **do**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \mathcal{O}_{otr\text{-}chl}^{(b),\mathsf{MDVS}}(i, S, \mathcal{O}_{RO,1}(\boldsymbol{t}[-\ell:]))$
  $\boldsymbol{\sigma} \leftarrow \hat{\boldsymbol{\sigma}} \oplus \mathcal{O}_{RO,2}(\boldsymbol{t}[-\ell:])$
  $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \epsilon,\ \boldsymbol{m} \leftarrow \epsilon$
  while $\boldsymbol{\sigma} \neq \epsilon$ **do**
    $\sigma_{\mathrm{bit}} \leftarrow \boldsymbol{\sigma}[1],\ \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[2:]$
    $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    while $\mathcal{O}_{RO,3}(\boldsymbol{m}||\boldsymbol{x}||\boldsymbol{\sigma}_{\mathrm{prev}}) \neq \sigma_{\mathrm{bit}}$ **do**
      $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{x}$
    $\boldsymbol{t} \leftarrow \boldsymbol{t}||\boldsymbol{x}$
    $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \boldsymbol{\sigma}_{\mathrm{prev}}||\sigma_{\mathrm{bit}}$
if $|\boldsymbol{t}| < n$ **then**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{p}, \boldsymbol{t})$
return $\boldsymbol{t}$

Algorithm 8 $\mathsf{FgeAS}(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \{\mathsf{dsk}_j\}_{j\in S_{\mathrm{cor}}}, \boldsymbol{p})$

$\mathsf{spk}_i \leftarrow \mathsf{wpk}_i$
$\{\mathsf{vpk}_j\}_{j\in S} \leftarrow \{\mathsf{dpk}_j\}_{j\in S}$
$\{\mathsf{vsk}_j\}_{j\in S_{\mathrm{cor}}} \leftarrow \{\mathsf{dsk}_j\}_{j\in S_{\mathrm{cor}}}$
$\boldsymbol{t} \leftarrow \epsilon$
while $|\boldsymbol{t}| + \ell + \ell \cdot \mathsf{len}_{\mathrm{sig}} < n$ **do**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \mathsf{FgeAS}_{\mathsf{MDVS}}(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \{\mathsf{vsk}_j\}_{j\in S_{\mathrm{cor}}}, H_1(\boldsymbol{t}[-\ell:]))$
  $\boldsymbol{\sigma} \leftarrow \hat{\boldsymbol{\sigma}} \oplus H_2(\boldsymbol{t}[-\ell:])$
  $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \epsilon,\ \boldsymbol{m} \leftarrow \epsilon$
  while $\boldsymbol{\sigma} \neq \epsilon$ **do**
    $\sigma_{\mathrm{bit}} \leftarrow \boldsymbol{\sigma}[1],\ \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[2:]$
    $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    while $H_3(\boldsymbol{m}||\boldsymbol{x}||\boldsymbol{\sigma}_{\mathrm{prev}}) \neq \sigma_{\mathrm{bit}}$ **do**
      $\boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{x},\ \boldsymbol{t} \leftarrow \boldsymbol{t}||\boldsymbol{x}$
    $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \boldsymbol{\sigma}_{\mathrm{prev}}||\sigma_{\mathrm{bit}}$
if $|\boldsymbol{t}| < n$ **then**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{p}, \boldsymbol{t})$
return $\boldsymbol{t}$

For any PPT adversary $\mathcal{A}$ attacking the off-the-record property (for any subset) of MDDW, we show a PPT adversary $\mathcal{A}'$ attacking the off-the-record property (for any subset) of MDVS as follows.

Receiving pp (note that the pp generated by Setup of MDDW is the same as that generated by the underlying $\mathsf{MDVS.Setup}$), $\mathcal{A}'$ sends pp to $\mathcal{A}$. $\mathcal{A}'$ maintains three local arrays, $L_{\mathrm{ro},1}$, $L_{\mathrm{ro},2}$ and $L_{\mathrm{ro},3}$, to keep track of $\mathcal{A}$'s random oracle queries. Then, $\mathcal{A}'$ answers $\mathcal{A}$'s oracle queries as below:

- $\mathcal{O}_{RO,\kappa}(str)$ $(\kappa \in \{1,2,3\})$: If there is some $(str, y) \in L_{\mathrm{ro},\kappa}$, $\mathcal{A}'$ returns y; otherwise, $\mathcal{A}'$ samples $y \leftarrow \mathcal{M}$, adds (str, y) to $L_{\mathrm{ro},\kappa}$, and returns y.
- $\mathcal{O}_{WK}(i)$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{SK}(i)$ to obtain $(\mathsf{spk}_i, \mathsf{ssk}_i)$, and returns $(\mathsf{wpk}_i, \mathsf{wsk}_i) = (\mathsf{spk}_i, \mathsf{ssk}_i)$ to $\mathcal{A}$.
- $\mathcal{O}_{DK}(j)$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{VK}(j)$ to obtain $(\mathsf{vpk}_j, \mathsf{vsk}_j)$, and returns $(\mathsf{dpk}_j, \mathsf{dsk}_j) = (\mathsf{vpk}_j, \mathsf{vsk}_j)$ to $\mathcal{A}$.
- $\mathcal{O}_{WPK}(i)$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{SPK}(i)$ to obtain spk_i, and returns $\mathsf{wpk}_i = \mathsf{spk}_i$ to $\mathcal{A}$.
- $\mathcal{O}_{DPK}(j)$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{VPK}(j)$ to obtain vpk_j, and returns $\mathsf{dpk}_j = \mathsf{vpk}_j$ to $\mathcal{A}$.
- $\mathcal{O}^{(b)}_{otr\text{-}chl}(i, S, S_{\mathrm{cor}}, \boldsymbol{p})$: $\mathcal{A}'$ runs $\mathsf{SimOtr\text{-}Chl}_{\mathrm{AS}}$ (with the help of its challenge oracle $\mathcal{O}^{(b)}_{otr\text{-}chl}$ for MDVS, denoted as $\mathcal{O}^{(b),\mathsf{MDVS}}_{otr\text{-}chl}$) as shown in Algorithm 9. In particular, Algorithm 9 is the same as Algorithm 4, except that the signing oracle $\mathcal{O}_S$ (in line 4) in Algorithm 4 is replaced with $\mathcal{A}'$'s challenge oracle $\mathcal{O}^{(b),\mathsf{MDVS}}_{otr\text{-}chl}$. Finally, $\mathcal{A}'$ returns the output $\boldsymbol{t}$ of $\mathsf{SimOtr\text{-}Chl}_{\mathrm{AS}}$ to $\mathcal{A}$.
- $\mathcal{O}_D(i, j', S, \boldsymbol{t})$: $\mathcal{A}'$ runs $\mathsf{SimDetect}$ (with the help of its verification oracle $\mathcal{O}_V$) as shown in Algorithm 5. More specifically, Algorithm 5 is the same as Algorithm 3, except that the original hash functions H_1, H_2 and H_3 (in line 3, line 7 and line 6, respectively) in Algorithm 3 are replaced with random oracles, and the original Verify algorithm (in line 8) in Algorithm 3 is replaced with $\mathcal{A}'$'s verification oracle $\mathcal{O}_V$. Finally, $\mathcal{A}'$ returns the output of $\mathsf{SimDetect}$ to $\mathcal{A}$.

Finally, $\mathcal{A}'$ returns $\mathcal{A}$'s output b' as its own final output.

That is the construction of $\mathcal{A}'$.

It is obvious that $\mathcal{A}'$ perfectly simulates game $\mathbf{G}^{\text{otr-as}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeAS}}(\lambda)$ for $\mathcal{A}$, and if $\mathcal{A}$ wins $\mathbf{G}^{\text{otr-as}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeAS}}(\lambda)$, then $\mathcal{A}'$ also wins $\mathbf{G}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A}',\mathsf{FgeAS}_{\mathsf{MDVS}}}(\lambda)$. So we derive

$$\mathbf{Adv}^{\text{otr-as}}_{\mathsf{MDVS},\mathcal{A}',\mathsf{FgeAS}_{\mathsf{MDVS}}}(\lambda) \geq \mathbf{Adv}^{\text{otr-as}}_{\mathsf{MDDW},\mathcal{A},\mathsf{FgeAS}}(\lambda).$$

□

F Proof of Theorem 4

Proof. Since MDVS is claimable, there are two PPT algorithms $\mathsf{Claim}_{\mathsf{MDVS}}$ and $\mathsf{ClmVer}_{\mathsf{MDVS}}$.

We construct two PPT algorithms Claim and ClmVer for MDDW as shown in Algorithm 10 and Algorithm 11, respectively.

Now, we show that MDDW with the two algorithms, Claim and ClmVer, satisfies each of the three conditions of Definition 14.

Algorithm 9 $\mathsf{SimOtr\text{-}Chl}_{\mathrm{AS}}^{\mathcal{O}_{otr\text{-}chl}^{(b),\mathsf{MDVS}},\{\mathcal{O}_{RO,\kappa}\}_{\kappa\in\{1,2,3\}}}(\mathsf{pp}, i, S, S_{\mathrm{cor}}, \boldsymbol{p})$

$\boldsymbol{t} \leftarrow \epsilon$
while $|\boldsymbol{t}| + \ell + \ell \cdot \mathsf{len}_{\mathrm{sig}} < n$ **do**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_{\ell}(\boldsymbol{p}, \boldsymbol{t})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \mathcal{O}_{otr\text{-}chl}^{(b),\mathsf{MDVS}}(i, S, S_{\mathrm{cor}}, \mathcal{O}_{RO,1}(\boldsymbol{t}[-\ell:]))$
  $\boldsymbol{\sigma} \leftarrow \hat{\boldsymbol{\sigma}} \oplus \mathcal{O}_{RO,2}(\boldsymbol{t}[-\ell:])$
  $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \epsilon,\ \boldsymbol{m} \leftarrow \epsilon$
  while $\boldsymbol{\sigma} \neq \epsilon$ **do**
    $\sigma_{\mathrm{bit}} \leftarrow \boldsymbol{\sigma}[1],\ \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[2:]$
    $\boldsymbol{x} \leftarrow \mathsf{GenModel}_{\ell}(\boldsymbol{p}, \boldsymbol{t})$
    while $\mathcal{O}_{RO,3}(\boldsymbol{m}||\boldsymbol{x}||\boldsymbol{\sigma}_{\mathrm{prev}}) \neq \sigma_{\mathrm{bit}}$ **do**
      $\boldsymbol{x} \leftarrow \mathsf{GenModel}_{\ell}(\boldsymbol{p}, \boldsymbol{t})$
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{x}$
    $\boldsymbol{t} \leftarrow \boldsymbol{t}||\boldsymbol{x}$
    $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \boldsymbol{\sigma}_{\mathrm{prev}}||\sigma_{\mathrm{bit}}$
if $|\boldsymbol{t}| < n$ **then**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{p}, \boldsymbol{t})$
return $\boldsymbol{t}$

Algorithm 10 $\mathsf{Claim}(\mathsf{pp}, \mathsf{wsk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t})$

$\mathsf{ssk}_i \leftarrow \mathsf{wsk}_i$
$\{\mathsf{vpk}_j\}_{j\in S} \leftarrow \{\mathsf{dpk}_j\}_{j\in S}$
$\mu \leftarrow 0$
$\mathsf{Count} \leftarrow 0$
while $\mu < n - (\ell + \ell \cdot \mathsf{len}_{\mathrm{sig}})$ **do**
  $\widetilde{m} \leftarrow H_1(\boldsymbol{t}[\mu : \mu + \ell - 1]),\ \boldsymbol{\sigma} \leftarrow \epsilon,\ \boldsymbol{m} \leftarrow \epsilon$
  for $\varphi \in [\mathsf{len}_{\mathrm{sig}}]$ **do**
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{t}[\mu + \varphi \cdot \ell : \mu + \varphi \cdot \ell + \ell - 1]$
    $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}||H_3(\boldsymbol{m}||\boldsymbol{\sigma})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \boldsymbol{\sigma} \oplus H_2(\boldsymbol{t}[\mu : \mu + \ell - 1])$
  $\mathsf{Count} \leftarrow \mathsf{Count} + 1$
  $\pi_{\mathsf{Count}} \leftarrow \mathsf{Claim}_{\mathsf{MDVS}}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \hat{\boldsymbol{\sigma}})$
  $\mu \leftarrow \mu + (\ell + \mathsf{len}_{\mathrm{sig}} \cdot \ell) \cdot \mathsf{Count}$
$\pi \leftarrow (\pi_1, \cdots, \pi_{\lfloor \frac{n - \mathsf{len}_{\mathrm{sig}} - \ell}{(\mathsf{len}_{\mathrm{sig}}+1)\cdot\ell} \rfloor})$
return π

Algorithm 11 $\mathsf{ClmVer}(\mathsf{pp}, \mathsf{wpk}_i, \{\mathsf{dpk}_j\}_{j\in S}, \boldsymbol{t}, \pi)$

$\mathsf{spk}_i \leftarrow \mathsf{wpk}_i$
$\{\mathsf{vpk}_j\}_{j\in S} \leftarrow \{\mathsf{dpk}_j\}_{j\in S}$
$(\pi_1, \cdots, \pi_{\lfloor \frac{n-\ell\cdot \mathsf{len}_{\mathrm{sig}}-\ell}{(\mathsf{len}_{\mathrm{sig}}+1)\cdot\ell} \rfloor}) \leftarrow \pi$
$\mu \leftarrow 0$
$\mathsf{Count} \leftarrow 0$
while $\mu < n - (\ell + \mathsf{len}_{\mathrm{sig}})$ **do**
  $\widetilde{m} \leftarrow H_1(\boldsymbol{t}[\mu : \mu + \ell - 1])$, $\boldsymbol{\sigma} \leftarrow \epsilon$, $\boldsymbol{m} \leftarrow \epsilon$
  for $\varphi \in [\mathsf{len}_{\mathrm{sig}}]$ **do**
    $\boldsymbol{m} \leftarrow \boldsymbol{m} || \boldsymbol{t}[\mu + \varphi \cdot \ell : \mu + \varphi \cdot \ell + \ell - 1]$
    $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} || H_3(\boldsymbol{m} || \boldsymbol{\sigma})$
  $\hat{\boldsymbol{\sigma}} \leftarrow \boldsymbol{\sigma} \oplus H_2(\boldsymbol{t}[\mu : \mu + \ell - 1])$
  $\mathsf{Count} \leftarrow \mathsf{Count} + 1$
  if $\mathsf{ClmVer}_{\mathsf{MDVS}}(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \hat{\boldsymbol{\sigma}}, \pi_{\mathsf{Count}}) = 1$ **then**
    return 1
  $\mu \leftarrow \mu + (\ell + \mathsf{len}_{\mathrm{sig}} \cdot \ell) \cdot \mathsf{Count}$
return 0

(1) For the first condition of Definition 14

The first condition of Definition 14 for MDDW is immediately fulfilled by the first condition of Definition 15 for the underlying MDVS (encompassing $\mathsf{Claim}_{\mathsf{MDVS}}$ and $\mathsf{ClmVer}_{\mathsf{MDVS}}$).

(2) For the second condition of Definition 14

We turn to the second condition of Definition 14 for MDDW.

For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ attacking the second condition of Definition 14 for MDDW, we show a PPT adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ attacking the second condition of Definition 15 for MDVS as follows.

Upon receiving pp (note that the pp generated by Setup of MDDW is the same as that generated by the underlying MDVS.Setup), $\mathcal{B}_1$ sends pp to $\mathcal{A}_1$. $\mathcal{B}$ maintains three local arrays, $L_{\mathrm{ro},1}$, $L_{\mathrm{ro},2}$ and $L_{\mathrm{ro},3}$, to keep track of $\mathcal{A}$'s random oracle queries. Then, $\mathcal{B}_1$ answers $\mathcal{A}_1$'s oracle queries as below:

- $\mathcal{O}_{RO,\kappa}(str)$ $(\kappa \in \{1,2,3\})$: If there is some $(str, y) \in L_{\mathrm{ro},\kappa}$, $\mathcal{B}_1$ returns y; otherwise, $\mathcal{B}_1$ samples $y \leftarrow \mathcal{M}$, adds (str, y) to $L_{\mathrm{ro},\kappa}$, and returns y.
- $\mathcal{O}_{WK}(i)$: $\mathcal{B}_1$ queries its own oracle $\mathcal{O}_{SK}(i)$ to obtain $(\mathsf{spk}_i, \mathsf{ssk}_i)$, and returns $(\overline{\mathsf{wpk}_i, \mathsf{wsk}_i}) = (\mathsf{spk}_i, \mathsf{ssk}_i)$ to $\mathcal{A}_1$.
- $\mathcal{O}_{DK}(j)$: $\mathcal{B}_1$ queries its own oracle $\mathcal{O}_{VK}(j)$ to obtain $(\mathsf{vpk}_j, \mathsf{vsk}_j)$, and returns $(\overline{\mathsf{dpk}_j, \mathsf{dsk}_j}) = (\mathsf{vpk}_j, \mathsf{vsk}_j)$ to $\mathcal{A}_1$.
- $\mathcal{O}_{WPK}(i)$: $\mathcal{B}_1$ queries its own oracle $\mathcal{O}_{SPK}(i)$ to obtain spk_i, and returns $\overline{\mathsf{wpk}_i = \mathsf{spk}_i}$ to $\mathcal{A}_1$.
- $\mathcal{O}_{DPK}(j)$: $\mathcal{B}_1$ queries its own oracle $\mathcal{O}_{VPK}(j)$ to obtain vpk_j, and returns $\overline{\mathsf{dpk}_j = \mathsf{vpk}_j}$ to $\mathcal{A}_1$.
- $\mathcal{O}_W(i, S, \boldsymbol{p})$: $\mathcal{B}_1$ runs SimWatMar (with the help of its signing oracle $\mathcal{O}_S$) as shown in Algorithm 4. More specifically, Algorithm 4 is the same as Algorithm 2, except that the original hash functions H_1, H_2 and H_3 (in line 4, line 5 and

line 10, respectively) in Algorithm 2 are replaced with random oracles, and the original Sign algorithm (in line 4) in Algorithm 2 is replaced with $\mathcal{B}_1$'s signing oracle $\mathcal{O}_S$. Finally, $\mathcal{B}_1$ returns the output $\boldsymbol{t}$ of SimWatMar to $\mathcal{A}_1$.

- $\underline{\mathcal{O}_D(i, j', S, \boldsymbol{t})}$: $\mathcal{B}_1$ runs SimDetect (with the help of its verification oracle $\mathcal{O}_V$) as shown in Algorithm 5. More specifically, Algorithm 5 is the same as Algorithm 3, except that the original hash functions H_1, H_2 and H_3 (in line 3, line 7 and line 6, respectively) in Algorithm 3 are replaced with random oracles, and the original Verify algorithm (in line 8) in Algorithm 3 is replaced with $\mathcal{B}_1$'s verification oracle $\mathcal{O}_V$. Finally, $\mathcal{B}_1$ returns the output of SimDetect to $\mathcal{A}_1$.
- $\underline{\mathcal{O}_{Clm}(i, S, \boldsymbol{t})}$: $\mathcal{B}_1$ runs SimClaim (with the help of its own claiming oracle $\mathcal{O}_{Clm}^{\mathsf{MDVS}}$, which denotes the claiming oralce for $\mathcal{B}$ in game $\mathbf{G}_{\mathsf{MDVS},\mathcal{B}}^{\text{clm-unf}}(\lambda)$) as shown in Algorithm 12. More specifically, Algorithm 12 is the same as Algorithm 10, except that the original hash functions H_1, H_2 and H_3 (in line 4, line 8 and line 7, respectively) in Algorithm 10 are replaced with random oracles, and the original Claim algorithm (in line 10) in Algorithm 10 is replaced with $\mathcal{B}_1$'s claiming oracle $\mathcal{O}_{Clm}^{\mathsf{MDVS}}$. Finally, $\mathcal{B}_1$ returns the output of SimClaim to $\mathcal{A}_1$.

Algorithm 12 SimClaim$(\mathsf{pp}, i, S, \boldsymbol{t})$

1: $\mu \leftarrow 0$
2: $\mathsf{Count} \leftarrow 0$
3: **while** $\mu < n - (\ell + \ell \cdot \mathsf{len}_{\mathrm{sig}})$ **do**
4: $\quad \widetilde{m} \leftarrow \mathcal{O}_{RO,1}(\boldsymbol{t}[\mu : \mu + \ell - 1])$, $\boldsymbol{\sigma} \leftarrow \epsilon$, $\boldsymbol{m} \leftarrow \epsilon$
5: $\quad$ **for** $\varphi \in [\mathsf{len}_{\mathrm{sig}}]$ **do**
6: $\quad\quad \boldsymbol{m} \leftarrow \boldsymbol{m} || \boldsymbol{t}[\mu + \varphi \cdot \ell : \mu + \varphi \cdot \ell + \ell - 1]$
7: $\quad\quad \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} || \mathcal{O}_{RO,3}(\boldsymbol{m} || \boldsymbol{\sigma})$
8: $\quad \hat{\boldsymbol{\sigma}} \leftarrow \boldsymbol{\sigma} \oplus \mathcal{O}_{RO,2}(\boldsymbol{t}[\mu : \mu + \ell - 1])$
9: $\quad \mathsf{Count} \leftarrow \mathsf{Count} + 1$
10: $\quad \pi_{\mathsf{Count}} \leftarrow \mathcal{O}_{Clm}^{\mathsf{MDVS}}(i, S, \hat{\boldsymbol{\sigma}})$
11: $\quad \mu \leftarrow \mu + (\ell + \mathsf{len}_{\mathrm{sig}} \cdot \ell) \cdot \mathsf{Count}$
12: $\pi \leftarrow (\pi_1, \cdots, \pi_{\lfloor \frac{n - \mathsf{len}_{\mathrm{sig}} - \ell}{(\mathsf{len}_{\mathrm{sig}} + 1) \cdot \ell} \rfloor})$
13: **return** π

Receiving $(i^*, S^*, \boldsymbol{p}^*)$ from $\mathcal{A}_1$, $\mathcal{B}_1$ uniformly samples $\theta \leftarrow \{1, \cdots, \lfloor \frac{n - \ell \cdot \mathsf{len}_{\mathrm{sig}} - \ell}{(\mathsf{len}_{\mathrm{sig}} + 1) \cdot \ell} \rfloor\}$, and generates $\boldsymbol{t}^*$ via running $\mathsf{SimWatMar}_\theta(\mathsf{pp}, i^*, S^*, \boldsymbol{p}^*)$ (as shown in Algorithm 13). In particular, Algorithm 13 is the same as Algorithm 4 except that when the while loop (spanning from line 2 to line 14 in Algorithm 4) enters its θ-th cycle, B_1 outputs $(i^*, S^*, m^* = \mathcal{O}_{RO,1}(\boldsymbol{t}[-\ell :]))$, and after receiving σ^* from the challenger, B_2 sets $\hat{\boldsymbol{\sigma}} = \sigma^*$ and then continues the procedure of Algorithm 4.

$\mathcal{B}_2$ sends $\boldsymbol{t}^*$ to $\mathcal{A}_2$, and then answers $\mathcal{A}_2$'s oracle queries in the same manner as $\mathcal{B}_1$ does.

Algorithm 13 $\mathsf{SimWatMar}_{\theta}^{\mathcal{O}_S,\{\mathcal{O}_{RO,\kappa}\}_{\kappa\in\{1,2,3\}}}(\mathsf{pp}, i, S, \boldsymbol{p})$

$\boldsymbol{t} \leftarrow \epsilon$
$\mathsf{Count} \leftarrow 0$
while $|\boldsymbol{t}| + \ell + \ell \cdot \mathsf{len}_{\mathrm{sig}} < n$ **do**
  $\mathsf{Count} \leftarrow \mathsf{Count} + 1$
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_{\ell}(\boldsymbol{p}, \boldsymbol{t})$
  if $\mathsf{Count} \neq \theta$ **then**
    $\hat{\boldsymbol{\sigma}} \leftarrow \mathcal{O}_S(i, S, \mathcal{O}_{RO,1}(\boldsymbol{t}[-\ell:]))$
  else
    $\mathcal{B}_1$ outputs $(i^*, S^*, m^* = \mathcal{O}_{RO,1}(\boldsymbol{t}[-\ell:]))$, and then $\mathcal{B}_2$ receives σ^* as input.
    $\hat{\boldsymbol{\sigma}} \leftarrow \sigma^*$
  $\boldsymbol{\sigma} \leftarrow \hat{\boldsymbol{\sigma}} \oplus \mathcal{O}_{RO,2}(\boldsymbol{t}[-\ell:])$
  $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \epsilon,\ \ \boldsymbol{m} \leftarrow \epsilon$
  while $\boldsymbol{\sigma} \neq \epsilon$ **do**
    $\sigma_{\mathrm{bit}} \leftarrow \boldsymbol{\sigma}[1],\ \ \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[2:]$
    $\boldsymbol{x} \leftarrow \mathsf{GenModel}_{\ell}(\boldsymbol{p}, \boldsymbol{t})$
    while $\mathcal{O}_{RO,3}(\boldsymbol{m}||\boldsymbol{x}||\boldsymbol{\sigma}_{\mathrm{prev}}) \neq \sigma_{\mathrm{bit}}$ **do**
      $\boldsymbol{x} \leftarrow \mathsf{GenModel}_{\ell}(\boldsymbol{p}, \boldsymbol{t})$
    $\boldsymbol{m} \leftarrow \boldsymbol{m}||\boldsymbol{x}$
    $\boldsymbol{t} \leftarrow \boldsymbol{t}||\boldsymbol{x}$
    $\boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \boldsymbol{\sigma}_{\mathrm{prev}}||\sigma_{\mathrm{bit}}$
if $|\boldsymbol{t}| < n$ **then**
  $\boldsymbol{t} \leftarrow \boldsymbol{t}||\mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{p}, \boldsymbol{t})$
return $\boldsymbol{t}$

Finally, receiving $\mathcal{A}_2$'s final output $(\widehat{\pi}, \widehat{i'})$, $\mathcal{B}_2$ parses $\widehat{\pi} = (\pi_1, \cdots, \pi_{\lfloor \frac{n-\ell\cdot\mathsf{len}_{\mathrm{sig}}-\ell}{(\mathsf{len}_{\mathrm{sig}}+1)\cdot\ell} \rfloor})$, and then returns $(\pi^*, i') = (\pi_\theta, \widehat{i'})$ as its final output.

That is the construction of $\mathcal{B}$. Now we turn to analyze $\mathcal{B}$'s advantage.

It is easy to see that $\mathcal{B}$ perfectly simulates game $\mathbf{G}^{\text{clm-unf}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ for $\mathcal{A}$. Since θ is uniformly sampled from $\{1, \cdots, \lfloor \frac{n-\ell\cdot\mathsf{len}_{\mathrm{sig}}-\ell}{(\mathsf{len}_{\mathrm{sig}}+1)\cdot\ell} \rfloor\}$, according to the description of ClmVer (in Algorithm 11), we derive

$$\mathbf{Adv}^{\text{clm-unf}}_{\mathsf{MDVS},\mathcal{B}}(\lambda) \geq \frac{1}{\lfloor \frac{n-\ell\cdot\mathsf{len}_{\mathrm{sig}}-\ell}{(\mathsf{len}_{\mathrm{sig}}+1)\cdot\ell} \rfloor} \mathbf{Adv}^{\text{clm-unf}}_{\mathsf{MDDW},\mathcal{A}}(\lambda).$$

(3) For the third condition of Definition 14

For any PPT adversary $\mathcal{A}$ attacking the third condition of Definition 14 for MDDW, we show a PPT adversary $\mathcal{A}'$ attacking the third condition of Definition 15 for MDVS as follows.

Receiving pp (note that the pp generated by Setup of MDDW is the same as that generated by the underlying MDVS.Setup), $\mathcal{A}'$ sends pp to $\mathcal{A}$. $\mathcal{A}'$ maintains three local arrays, $L_{\mathrm{ro},1}$, $L_{\mathrm{ro},2}$ and $L_{\mathrm{ro},3}$, to keep track of $\mathcal{A}$'s random oracle queries. Then, $\mathcal{A}'$ answers $\mathcal{A}$'s oracle queries as below:

- $\mathcal{O}_{RO,\kappa}(str)$ $(\kappa \in \{1,2,3\})$: If there is some $(str, y) \in L_{\mathrm{ro},\kappa}$, $\mathcal{A}'$ returns y; otherwise, $\mathcal{A}'$ samples $y \leftarrow \mathcal{M}$, adds (str, y) to $L_{\mathrm{ro},\kappa}$, and returns y.
- $\mathcal{O}_{WK}(i)$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{SK}(i)$ to obtain $(\mathsf{spk}_i, \mathsf{ssk}_i)$, and returns $(\overline{\mathsf{wpk}_i, \mathsf{wsk}_i}) = (\mathsf{spk}_i, \mathsf{ssk}_i)$ to $\mathcal{A}$.
- $\mathcal{O}_{DK}(j)$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{VK}(j)$ to obtain $(\mathsf{vpk}_j, \mathsf{vsk}_j)$, and returns $(\overline{\mathsf{dpk}_j, \mathsf{dsk}_j}) = (\mathsf{vpk}_j, \mathsf{vsk}_j)$ to $\mathcal{A}$.
- $\mathcal{O}_{WPK}(i)$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{SPK}(i)$ to obtain spk_i, and returns $\overline{\mathsf{wpk}_i = \mathsf{spk}_i}$ to $\mathcal{A}$.
- $\mathcal{O}_{DPK}(j)$: $\mathcal{A}'$ queries its own oracle $\mathcal{O}_{VPK}(j)$ to obtain vpk_j, and returns $\overline{\mathsf{dpk}_j = \mathsf{vpk}_j}$ to $\mathcal{A}$.
- $\mathcal{O}_W(i, S, \boldsymbol{p})$: $\mathcal{A}'$ runs $\mathsf{SimWatMar}$ (with the help of its signing oracle $\mathcal{O}_S$) as shown in Algorithm 4. More specifically, Algorithm 4 is the same as Algorithm 2, except that the original hash functions H_1, H_2 and H_3 (in line 4, line 5 and line 10, respectively) in Algorithm 2 are replaced with random oracles, and the original Sign algorithm (in line 4) in Algorithm 2 is replaced with $\mathcal{A}'$'s signing oracle $\mathcal{O}_S$. Finally, $\mathcal{A}'$ returns the output $\boldsymbol{t}$ of $\mathsf{SimWatMar}$ to $\mathcal{A}$.
- $\mathcal{O}_D(i, j', S, \boldsymbol{t})$: $\mathcal{A}'$ runs $\mathsf{SimDetect}$ (with the help of its verification oracle $\mathcal{O}_V$) as shown in Algorithm 5. More specifically, Algorithm 5 is the same as Algorithm 3, except that the original hash functions H_1, H_2 and H_3 (in line 3, line 7 and line 6, respectively) in Algorithm 3 are replaced with random oracles, and the original Verify algorithm (in line 8) in Algorithm 3 is replaced with $\mathcal{A}'$'s verification oracle $\mathcal{O}_V$. Finally, $\mathcal{A}'$ returns the output of $\mathsf{SimDetect}$ to $\mathcal{A}$.
- $\mathcal{O}_{Clm}(i, S, \boldsymbol{t})$: $\mathcal{A}'$ runs $\mathsf{SimClaim}$ (with the help of its own claiming oracle $\mathcal{O}_{Clm}^{\mathsf{MDVS}}$, which denotes the claiming oralce for $\mathcal{A}'$ in game $\mathbf{G}^{\text{non-fram}}_{\mathsf{MDVS},\mathcal{A}'}(\lambda)$) as shown in Algorithm 12. More specifically, Algorithm 12 is the same as Algorithm 10, except that the original hash functions H_1, H_2 and H_3 (in line 4, line 8 and line 7, respectively) in Algorithm 10 are replaced with random oracles, and the original Claim algorithm (in line 10) in Algorithm 10 is replaced with $\mathcal{A}'$'s claiming oracle $\mathcal{O}_{Clm}^{\mathsf{MDVS}}$. Finally, $\mathcal{A}'$ returns the output of $\mathsf{SimClaim}$ to $\mathcal{A}$.

Receiving $\mathcal{A}$'s final output $(i^*, S^*, \boldsymbol{t}^*, \pi^*)$, $\mathcal{A}'$ firstlys checks whether $\mathsf{ClmVer}(\mathsf{pp}, \mathsf{wpk}_{i^*}, \{\mathsf{dpk}_j\}_{j\in S^*}, \boldsymbol{t}^*, \pi^*) = 1$. If not, $\mathcal{A}'$ returns $(i^*, S^*, m^*_{\text{rand}}, \sigma^*_{\text{rand}}, \pi^*_{\text{rand}})$ as its own final output, where $(m^*_{\text{rand}}, \sigma^*_{\text{rand}}, \pi^*_{\text{rand}})$ are uniformly sampled. Otherwise, there must be some $\mathsf{Count} \in [\lfloor \frac{n-\ell\cdot\mathsf{len}_{\mathrm{sig}}-\ell}{(\mathsf{len}_{\mathrm{sig}}+1)\cdot\ell} \rfloor]$ satisfying $\mathsf{ClmVer}_{\mathsf{MDVS}}(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \hat{\boldsymbol{\sigma}}, \pi_{\mathsf{Count}}) = 1$, where $\hat{\boldsymbol{\sigma}}$ is generated according to lines 4-11 in Algorithm 11. In this case, $\mathcal{A}'$ extracts $\hat{\boldsymbol{\sigma}}$, π_{Count} and the corresponding $\widetilde{m}$ (for $\hat{\boldsymbol{\sigma}}$). Then, $\mathcal{A}'$ returns $(i^*, S^*, \widetilde{m}, \hat{\boldsymbol{\sigma}}, \pi_{\mathsf{Count}}, \pi^*)$ as its final output.

That is the construction of $\mathcal{A}'$. Now we turn to analyze $\mathcal{A}'$'s advantage.

It is easy to see that $\mathcal{A}'$ perfectly simulates game $\mathbf{G}^{\text{non-fram}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$ for $\mathcal{A}$, and $\mathcal{A}'$ wins $\mathbf{G}^{\text{clm-fram}}_{\mathsf{MDVS},\mathcal{A}'}(\lambda)$ if $\mathcal{A}$ wins $\mathbf{G}^{\text{non-fram}}_{\mathsf{MDDW},\mathcal{A}}(\lambda)$. So we derive

$$\mathbf{Adv}^{\text{non-fram}}_{\mathsf{MDVS},\mathcal{A}'}(\lambda) \geq \mathbf{Adv}^{\text{non-fram}}_{\mathsf{MDDW},\mathcal{A}}(\lambda).$$

□

G Proof of Theorem 5

Proof. Since the proofs that CMDVS satisfies correctness, consistency and existential unforgeability are straightforward, we will only focus on proving off-the-record property and claimability.

Off-the-record. Now, we prove that CMDVS is off-the-record for any subset if the underlying MDVS is off-the-record for any subset. The proof for CMDVS being off-the-record for designated set, assuming the underlying MDVS is off-the-record for designated set, is nearly identical and is thus omitted here.

Since MDVS is off-the-record for any subset, there is a PPT algorithm MDVS.FgeAS. We provide a forging algorithm FgeAS, based on MDVS.FgeAS, for CMDVS as shown in Fig. 15, where $\mathcal{COM}$ is the commitment space of Commit.

$\underline{\mathsf{FgeAS}(\mathsf{pp}, \mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \{\mathsf{spk}_j\}_{j\in S_{\mathrm{cor}}}, m):}$
$\sigma_{\mathsf{MDVS}} \leftarrow \mathsf{MDVS.FgeAS}(\mathsf{pp}_{\mathsf{MDVS}}, \mathsf{spk}_{\mathsf{MDVS},i}, \{\mathsf{vpk}_j\}_{j\in S}, \{\mathsf{vsk}_j\}_{j\in S_{\mathrm{cor}}}, m)$
$\mathsf{com} \leftarrow \mathcal{COM}$
Return $\sigma = (\sigma_{\mathsf{MDVS}}, \mathsf{com})$

Fig. 15 Forging algorithm FgeAS for CMDVS

For any PPT adversary $\mathcal{A}$, we consider the following sequence of games.

- Game_0: Game_0 is $\mathbf{G}^{\text{otr-as}}_{\mathsf{CMDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)$ when $b = 0$. Thus, we have
$$\Pr[\mathrm{Game}_0 = 1] = \Pr[\mathbf{G}^{\text{otr-as}}_{\mathsf{CMDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda) = 1 | b = 0].$$
- Game_1: Game_1 is the same as Game_0, except that when the adversary $\mathcal{A}$ queries to the signing oracle $\mathcal{O}^{(b)}_{otr\text{-}chl}$ on $(i, S, S_{\mathrm{cor}}, m)$, the challenger generates the MDVS signature σ_{MDVS} via the forging algorithm MDVS.FgeAS of the underlying MDVS. By the off-the-record property for any subset of MDVS, we derive
$$|\Pr[\mathrm{Game}_1 = 1] - \Pr[\mathrm{Game}_0 = 1]| \leq \mathsf{negl}(\lambda).$$
- Game_2: Game_2 is the same as Game_1, except that when the adversary $\mathcal{A}$ queries to the signing oracle $\mathcal{O}^{(b)}_{otr\text{-}chl}$ on $(i, S, S_{\mathrm{cor}}, m)$, the challenger chooses a random commitment com over the commitment space $\mathcal{COM}$. By the hiding property of Commit, we have
$$|\Pr[\mathrm{Game}_2 = 1] - \Pr[\mathrm{Game}_1 = 1]| \leq \mathsf{negl}(\lambda).$$

Note that Game_2 is equivalent to $\mathbf{G}^{\text{otr-as}}_{\mathsf{CMDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)$ when $b = 1$. Thus,

$$\Pr[\mathrm{Game}_2 = 1] = \Pr[\mathbf{G}^{\text{otr-as}}_{\mathsf{CMDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda) = 1 | b = 1].$$

Therefore, it holds that

$$\mathbf{Adv}^{\text{otr-as}}_{\mathsf{CMDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)$$

$$
\begin{aligned}
&= |\Pr[\mathbf{G}^{\text{otr-as}}_{\mathsf{CMDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)=1] - \frac{1}{2}| \\
&= \frac{1}{2}|\Pr[\mathbf{G}^{\text{otr-as}}_{\mathsf{CMDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)=1|b=0] - \Pr[\mathbf{G}^{\text{otr-as}}_{\mathsf{CMDVS},\mathcal{A},\mathsf{FgeAS}}(\lambda)=1|b=1]| \\
&= \frac{1}{2}|\Pr[\text{Game}_0=1] - \Pr[\text{Game}_2=1]| \leq \mathsf{negl}(\lambda).
\end{aligned}
$$

Claimability. Now, we show that CMDVS satisfies each of the three conditions of Definition 15.

(1) For the first condition of Definition 15

For any signer i, any message $m \in \mathcal{M}$ and any verifier identity set S, given $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{spk}_i, \mathsf{ssk}_i) \leftarrow \mathsf{SignKG}(\mathsf{pp})$, $\{(\mathsf{vpk}_j, \mathsf{vsk}_j) \leftarrow \mathsf{VerKG}(\mathsf{pp})\}_{j\in S}$, $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, m)$ and $\pi \leftarrow \mathsf{Claim}(\mathsf{pp}, \mathsf{ssk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \sigma)$, we have $\mathsf{ssk}_i = (k_i, \mathsf{ssk}_{\mathsf{Sig},i}, \mathsf{ssk}_{\mathsf{MDVS},i})$, $\sigma = (\sigma_{\mathsf{MDVS}}, \mathsf{com})$ and $\pi = (r_{\mathsf{Commit}}, \sigma_{\mathsf{Sig}})$. According to the descriptions of Sign and Claim in Fig. 10, we derive $r_{\mathsf{Sig}} = \mathsf{PRF.Eval}(k_i, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}, 0))$, $\sigma_{\mathsf{Sig}} = \mathsf{Sig.Sign}(\mathsf{pp}_{\mathsf{Sig}}, \mathsf{ssk}_{\mathsf{Sig},i}, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}); r_{\mathsf{Sig}})$, and $r_{\mathsf{Commit}} = \mathsf{PRF.Eval}(k_i, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}, 1))$. The correctness of Commit implies $\mathsf{Commit.Com}(\mathsf{pp}_{\mathsf{Commit}}, (\mathsf{spk}_i, \{\mathsf{vpk}_j\}_{j\in S}, \sigma_{\mathsf{Sig}}); r_{\mathsf{Commit}}) = \mathsf{com}$, and the correctness of Sig implies that $\mathsf{Sig.Verify}(\mathsf{pp}_{\mathsf{Sig}}, \mathsf{spk}_{\mathsf{Sig},i}, (\mathsf{spk}_i, \sigma_{\mathsf{MDVS}}), \sigma_{\mathsf{Sig}}) = 1$. Hence, ClmVer would output 1, which implies that the scheme CMDVS is correct.

(2) For the second condition of Definition 15

Assume that there exits a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that successfully attacks the second condition of Definition 15 with non-negligible advantage . We construct a PPT adversary $\mathcal{B}$ to break the binding property of Commit with non-negligible advantage as follows.

On receiving the public parameter $\mathsf{pp}_{\mathsf{Commit}}$, the adversary $\mathcal{B}$ generates $\mathsf{pp}_{\mathsf{MDVS}}$ and $\mathsf{pp}_{\mathsf{Sig}}$ via invoking MDVS.Setup and Sig.Setup, respectively. Subsequently, $\mathcal{B}$ sends $\mathsf{pp} = (\mathsf{pp}_{\mathsf{MDVS}}, \mathsf{pp}_{\mathsf{Sig}}, \mathsf{pp}_{\mathsf{Commit}})$ to $\mathcal{A}_1$, and then answers $\mathcal{A}_1$'s oracle queries by itself.

After receiving (i^*, S^*, m^*) from $\mathcal{A}_1$, where i^* is never queried to the oracle $\mathcal{O}_{SK}$ by $\mathcal{A}_1$, $\mathcal{B}$ proceeds as follows:

(1) $(\mathsf{spk}_{i^*}, \mathsf{ssk}_{i^*}) \leftarrow \mathcal{O}_{SK}(i^*)$, and $\{\mathsf{vpk}_j \leftarrow \mathcal{O}_{VPK}(j)\}_{j\in S^*}$, where the oracles $\mathcal{O}_{SK}$ and $\mathcal{O}_{VPK}$ are described in Fig. 3.
(2) $\sigma^* \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_{i^*}, \{\mathsf{vpk}_j\}_{j\in S^*}, m^*)$.

$\mathcal{B}$ sends σ^* to $\mathcal{A}_2$, and then answers $\mathcal{A}_2$'s oracle queries, with the condition that queries of i^* to $\mathcal{O}_{SK}$ and $(*, *, \sigma^*)$ to $\mathcal{O}_{Clm}$ are not permitted. It is important to note that ssk_{i^*} can be parsed as $\mathsf{ssk}_{i^*} = (k_{i^*}, \mathsf{ssk}_{\mathsf{Sig},i^*}, \mathsf{ssk}_{\mathsf{MDVS},i^*})$, and σ^* can be parsed as $\sigma^* = (\sigma^*_{\mathsf{MDVS}}, \mathsf{com}^*)$.

Upon receiving (π^*, i') from $\mathcal{A}$, $\mathcal{B}$ firstly checks if $(\mathsf{ClmVer}(\mathsf{pp}, \mathsf{spk}_{i'}, \{\mathsf{vpk}_j\}_{j\in S^*}, \sigma^*, \pi^*) = 1) \wedge (i' \neq i^*)$. If not, $\mathcal{B}$ returns uniformly sampled $(\mathsf{com}_{\text{rand}}, m_{\text{rand}}, r_{\text{rand}}, m'_{\text{rand}}, r'_{\text{rand}})$ as its final output. Otherwise, $\mathcal{B}$ parses $\pi^* = (r'_{\mathsf{Commit}}, \sigma'_{\mathsf{Sig}})$, and returns

$$
(\mathsf{com}^*, m = (\mathsf{spk}_{i^*}, \{\mathsf{vpk}_j\}_{j\in S^*}, \sigma^*_{\mathsf{Sig}}), r^*_{\mathsf{Commit}}, m' = (\mathsf{spk}_{i'}, \sigma'_{\mathsf{Sig}}), r'_{\mathsf{Commit}})
$$

as its final output, where σ^*_{Sig} is generated via "$r_{\mathsf{Sig}} \leftarrow \mathsf{PRF.Eval}(k_{i^*}, (\mathsf{spk}_{i^*}, \sigma^*_{\mathsf{MDVS}}, 0))$, $\sigma^*_{\mathsf{Sig}} \leftarrow \mathsf{Sig.Sign}(\mathsf{pp}_{\mathsf{Sig}}, \mathsf{ssk}_{\mathsf{Sig},i^*}, (\mathsf{spk}_{i^*}, \sigma^*_{\mathsf{MDVS}}); r_{\mathsf{Sig}})$", and r^*_{Commit} is generated via "$r^*_{\mathsf{Commit}} \leftarrow \mathsf{PRF.Eval}(k_{i^*}, (\mathsf{spk}_{i^*}, \sigma^*_{\mathsf{MDVS}}, 1))$".

Given that $i' \neq i^*$, it follows that $\mathsf{spk}_{i'} \neq \mathsf{spk}_{i^*}$, which in turn means that $m \neq m'$. Moreover, the correctness of Commit guarantees that $\mathsf{Commit.Decom}(\mathsf{pp}_{\mathsf{Commit}}, \mathsf{com}^*, r^*_{\mathsf{Commit}}, m) = 1$. The fact that $\mathsf{ClmVer}(\mathsf{pp}, \mathsf{spk}_{i'}, \{\mathsf{vpk}_j\}_{j\in S^*}, \sigma^*, \pi^*) = 1$ implies that $\mathsf{Commit.Decom}(\mathsf{pp}_{\mathsf{Commit}}, \mathsf{com}^*, r'_{\mathsf{Commit}}, m') = 1$.

Hence, if $\mathcal{A}$ successfully attacks the second condition of Definition 15 with non-negligible advantage, $\mathcal{B}$ will also successfully breaks the binding property of the Commit with non-negligible advantage.

(3) For the third condition of Definition 15

For any PPT adversary $\mathcal{A}$ attacking the third condition of Definition 15 (of CMDVS), its advantage is

$$\mathbf{Adv}^{\text{non-fram}}_{\mathsf{CMDVS},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\text{non-fram}}_{\mathsf{CMDVS},\mathcal{A}}(\lambda) = 1]. \tag{1}$$

Assume that $\mathcal{A}$ makes at most $\mathsf{poly}(\lambda)$ oracle queries.

Now, we consider the following sequence of games.

$\underline{\text{Game}_0}$: Game_0 is the same as $\mathbf{G}^{\text{non-fram}}_{\mathsf{CMDVS},\mathcal{A}}(\lambda)$. Thus, we have

$$\Pr[\text{Game}_0 = 1] = \Pr[\mathbf{G}^{\text{non-fram}}_{\mathsf{CMDVS},\mathcal{A}}(\lambda) = 1]. \tag{2}$$

$\underline{\text{Game}_1}$: Game_1 is the same as Game_0, except that at the beginning of this game, the challenger uniformly samples $\widehat{i} \leftarrow [\mathsf{poly}(\lambda)]$, and then if $\mathcal{A}$'s final output $(i^*, *, *, *, *)$ satisfies $i^* \neq \widehat{i}$, the challenger returns 0 directly.

Clearly, we derive

$$\Pr[\text{Game}_1 = 1] = \frac{1}{\mathsf{poly}(\lambda)} \Pr[\text{Game}_0 = 1]. \tag{3}$$

$\underline{\text{Game}_2}$: Game_2 is the same as Game_1, except that when $\mathcal{A}$ makes a query (i', S', σ') (to $\mathcal{O}_{Clm}$) satisfying that

(i) $i' = \widehat{i}$, and
(ii) σ' is not output by $\mathcal{O}_S$ on any query $(i', S', *)$,

the challenger returns $\perp$ to $\mathcal{A}$ as a response directly.

We present the following lemma with a postponed proof.

Lemma 4. $|\Pr[\text{Game}_2 = 1] - \Pr[\text{Game}_1 = 1]| \leq \mathsf{negl}(\lambda)$.

$\underline{\text{Game}_3}$: Game_3 is identical to Game_2, except that when responding to $\mathcal{A}$'s query $(i', *, *)$ to either $\mathcal{O}_S$ or $\mathcal{O}_{Clm}$ where $i' = \widehat{i}$, the challenger samples r_{Sig} uniformly at random (instead of using $\mathsf{PRF.Eval}$ to compute r_{Sig}) during the $\mathcal{O}_S(\widehat{i}, *, *)$ or $\mathcal{O}_{Clm}(\widehat{i}, *, *)$ operations.

Due to the pseudorandomness of PRF, we derive

$$|\Pr[\text{Game}_3 = 1] - \Pr[\text{Game}_2 = 1]| \leq \mathsf{negl}(\lambda). \tag{4}$$

Once more, we introduce the following lemma, with the proof to be addressed subsequently.

Lemma 5. $\Pr[\text{Game}_3 = 1] \leq \mathsf{negl}(\lambda)$.

Combining Eqs. (1)-(4), Lemma 4 and Lemma 5, we obtain

$$\mathbf{Adv}^{\text{non-fram}}_{\mathsf{CMDVS},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Therefore, what remains is to prove the above two lemmas.

Proof (of Lemma 4). Let Evt be the event that $\mathcal{A}$ makes a query $(i' = \widehat{i}, S', \sigma')$ to $\mathcal{O}_{Clm}$, where σ' is not output by $\mathcal{O}_S$ on any query $(\widehat{i}, S', *)$, and $\mathcal{O}_{Clm}(\widehat{i}, S', \sigma')$ does not return $\perp$.

It is straightforward to see that

$$|\Pr[\text{Game}_2 = 1] - \Pr[\text{Game}_1 = 1]| \leq \Pr[\mathsf{Evt}].$$

Assume that $\Pr[\mathsf{Evt}]$ is non-negligible.

Note that σ' can be parsed as $\sigma' = (\sigma'_{\mathsf{MDVS}}, \mathsf{com}')$.

The fact that "$\mathcal{O}_{Clm}(\widehat{i}, S', \sigma')$ does not return $\perp$" means that $\mathsf{Claim}(\mathsf{pp}, \mathsf{ssk}_{\widehat{i}}, \{\mathsf{vpk}_j\}_{j\in S'}, \sigma')$ returns some $\pi' = (\sigma'_{\mathsf{Sig}}, r'_{\mathsf{Commit}})$, where $(\mathsf{ssk}_{\widehat{i}}, \{\mathsf{vpk}_j\}_{j\in S'})$ are the honestly generated keys. According to the description of the algorithm Claim as shown in Fig. 10, we obtain $\mathsf{Commit.Decom}(\mathsf{pp}_{\mathsf{Commit}}, \mathsf{com}', r'_{\mathsf{Commit}}, (\mathsf{spk}_{\widehat{i}}, \{\mathsf{vpk}_j\}_{j\in S'}, \sigma'_{\mathsf{Sig}})) = 1$.

If σ'_{MDVS} is found in some response, generated by the challenger for $\mathcal{A}$'s $\mathcal{O}_S$-oracle query $(i', S', *)$ satisfying $i' = \widehat{i}$, we claim that in this case $\Pr[\mathsf{Evt}] = 0$. The reason is as follows. Assume that there is such a response $\sigma'' = (\sigma'_{\mathsf{MDVS}}, \mathsf{com}'')$, returned by the challenger for $\mathcal{A}$'s $\mathcal{O}_S$-oracle query $(\widehat{i}, S', *)$. It is important to note that $\sigma' = (\sigma'_{\mathsf{MDVS}}, \mathsf{com}')$ is not output by $\mathcal{O}_S$ on any query $(\widehat{i}, S', *)$, so we derive $\mathsf{com}'' \neq \mathsf{com}'$. Since $\sigma'' = (\sigma'_{\mathsf{MDVS}}, \mathsf{com}'')$ is generated via the algorithm Sign, according to the description of Sign (as shown in Fig. 10), the correctness of Commit guarantees that $\mathsf{Commit.Decom}(\mathsf{pp}_{\mathsf{Commit}}, \mathsf{com}'', r'_{\mathsf{Commit}}, (\mathsf{spk}_{\widehat{i}}, \sigma'_{\mathsf{Sig}})) = 1$, where $\sigma'_{\mathsf{Sig}} = \mathsf{Sig.Sign}(\mathsf{pp}_{\mathsf{Sig}}, \mathsf{ssk}_{\mathsf{Sig},\widehat{i}}, (\mathsf{spk}_{\widehat{i}}, \sigma'_{\mathsf{MDVS}}); \mathsf{PRF.Eval}(k_{\widehat{i}}, (\mathsf{spk}_{\widehat{i}}, \sigma'_{\mathsf{MDVS}}, 0)))$ and $r'_{\mathsf{Commit}} = \mathsf{PRF.Eval}(k_{\widehat{i}}, (\mathsf{spk}_{\widehat{i}}, \sigma'_{\mathsf{MDVS}}, 1))$. Thus, the fact that $\mathsf{com}'' \neq \mathsf{com}'$ implies that $\mathsf{Commit.Decom}(\mathsf{pp}_{\mathsf{Commit}}, \mathsf{com}', r'_{\mathsf{Commit}}, (\mathsf{spk}_{\widehat{i}}, \{\mathsf{vpk}_j\}_{j\in S'}, \sigma'_{\mathsf{Sig}})) = 0$, which further implies that $\mathcal{O}_{Clm}$ will return $\perp$, i.e., Evt will not occur.

Hence, σ'_{MDVS} is not included in any response generated by the challenger for $\mathcal{A}$'s $\mathcal{O}_S$-oracle query $(i', S', *)$ satisfying $i' = \widehat{i}$. In other words, the challenger has never computed $\mathsf{PRF.Eval}(k_{\widehat{i}}, (\mathsf{spk}_{\widehat{i}}, \sigma'_{\mathsf{MDVS}}, *))$.

Note that $\Pr[\mathsf{Evt}]$ is non-negligible. When Evt occurs, during the computation of $\mathcal{O}_{Clm}(\widehat{i}, S', \sigma')$ (i.e., $\mathsf{Claim}(\mathsf{pp}, \mathsf{ssk}_{\widehat{i}}, \{\mathsf{vpk}_j\}_{j\in S'}, \sigma')$), if we uniformly sample r'_{Sig} and r'_{Commit} instead of computing them via $\mathsf{PRF.Eval}$, it still holds that $\mathsf{Commit.Decom}(\mathsf{pp}_{\mathsf{Commit}}, \mathsf{com}', r'_{\mathsf{Commit}}, (\mathsf{spk}_{\widehat{i}}, \{\mathsf{vpk}_j\}_{j\in S'}, \sigma'_{\mathsf{Sig}})) = 1$ with overwhelming probability, because of the pseudorandomness of PRF. Note that the truly random r'_{Sig} and the original one generated with $\mathsf{PRF.Eval}$ will lead to two different σ'_{Sig}'s, contradicting the binding property of Commit.

Therefore, the assumption that $\Pr[\mathsf{Evt}]$ is incorrect. We conclude that $\Pr[\mathsf{Evt}] \leq \mathsf{negl}(\lambda)$. □

Proof (of Lemma 5). Consider the following two events.

- Evt_1: Let Evt_1 denote the event that $\mathcal{A}$ wins Game_3, and σ^* (from $\mathcal{A}$'s final output) is not included in any response generated by the challenger for $\mathcal{A}$'s $\mathcal{O}_S$-oracle query $(i', *, *)$ satisfying $i' = \widehat{i}$.
- Evt_2: Let Evt_2 denote the event that $\mathcal{A}$ wins Game_3, and σ^* (from $\mathcal{A}$'s final output) is included in some response generated by the challenger for $\mathcal{A}$'s $\mathcal{O}_S$-oracle query $(i', *, *)$ satisfying $i' = \widehat{i}$.

It is straightforward to see that

$$\Pr[\text{Game}_3 = 1] = \Pr[\mathsf{Evt}_1] + \Pr[\mathsf{Evt}_2].$$

If both $\Pr[\mathsf{Evt}_1]$ and $\Pr[\mathsf{Evt}_2]$ are negligible, then we can conclude this proof. So what remains is to prove that both $\Pr[\mathsf{Evt}_1]$ and $\Pr[\mathsf{Evt}_2]$ are negligible

Case 1: If $\Pr[\mathsf{Evt}_1]$ is non-negligible:

In this case, we construct a PPT adversary $\mathcal{B}$ attacking the existential unforgeability of Sig as follows.

On receiving $(\mathsf{pp}_{\mathsf{Sig}}, pk)$ from the challenger of the existential unforgeability game, $\mathcal{B}$ firstly samples $\widehat{i} \leftarrow [\mathsf{poly}(\lambda)]$ uniformly at random, and generates $\mathsf{pp}_{\mathsf{MDVS}}$ and $\mathsf{pp}_{\mathsf{Commit}}$ via invoking $\mathsf{MDVS.Setup}$ and $\mathsf{Commit.Setup}$, respectively. $\mathcal{B}$ also runs $\mathsf{PRF.KG}$ and $\mathsf{MDVS.SignKG}$ to generate $k_{\widehat{i}}$ and $(\mathsf{ssk}_{\mathsf{MDVS},\widehat{i}}, \mathsf{spk}_{\mathsf{MDVS},\widehat{i}})$. Thus, the public key of the signer $\widehat{i}$ is $\mathsf{spk}_{\widehat{i}} = (\mathsf{spk}_{\mathsf{Sig},\widehat{i}} = pk, \mathsf{spk}_{\mathsf{MDVS},\widehat{i}})$, and the secret key of the signer $\widehat{i}$ is $\mathsf{ssk}_{\widehat{i}} = (k_{\widehat{i}}, \mathsf{ssk}_{\mathsf{Sig},\widehat{i}} = \bot, \mathsf{ssk}_{\mathsf{MDVS},\widehat{i}})$. Then, $\mathcal{B}$ initializes an empty set Q', sends $\mathsf{pp} = (\mathsf{pp}_{\mathsf{MDVS}}, \mathsf{pp}_{\mathsf{Sig}}, \mathsf{pp}_{\mathsf{Commit}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

- $\mathcal{O}_{SK}, \mathcal{O}_{VK}, \mathcal{O}_{SPK}, \mathcal{O}_{VPK}, \mathcal{O}_V$: $\mathcal{B}$ answers these oracle queries as in Game_3.
- $\mathcal{O}_S(i', S', m')$: When $i' \neq \widehat{i}$, $\mathcal{B}$ proceeds as in Game_3. When $i' = \widehat{i}$, $\mathcal{B}$ checks whether there is some tuple $(S', m', \sigma', \pi') \in Q'$ or not. If so, $\mathcal{B}$ return σ' to $\mathcal{A}$ directly. Otherwise, $\mathcal{B}$ generates a signature as follows. $\mathcal{B}$ firstly obtains the public keys $\{\mathsf{vpk}_j \leftarrow \mathcal{O}_{VPK}(j)\}_{j \in S'}$. Then, it computes $\sigma'_{\mathsf{MDVS}} \leftarrow \mathsf{MDVS.Sign}(\mathsf{pp}_{\mathsf{MDVS}}, \mathsf{ssk}_{\mathsf{MDVS},\widehat{i}}, \{\mathsf{vpk}_j\}_{j \in S'}, m')$. After that, $\mathcal{B}$ queries its own signing oracle on $(\mathsf{spk}_{\widehat{i}}, \sigma'_{\mathsf{MDVS}})$, and then obtain a signature σ'_{Sig} as a response. Subsequently, $\mathcal{B}$ computes $r'_{\mathsf{Commit}} \leftarrow \mathsf{PRF.Eval}(k_{\widehat{i}}, (\mathsf{spk}_{\widehat{i}}, \sigma'_{\mathsf{MDVS}}), 1)$ and $\mathsf{com}' \leftarrow \mathsf{Commit.com}(\mathsf{pp}_{\mathsf{Commit}}, (\mathsf{spk}_{\widehat{i}}, \{\mathsf{vpk}_j\}_{j \in S'}, \sigma'_{\mathsf{Sig}}); r'_{\mathsf{Commit}})$. Finally, $\mathcal{B}$ sets $\sigma' = (\sigma'_{\mathsf{MDVS}}, \mathsf{com}')$ and $Q' \leftarrow Q' \cup \{(S', m', \sigma', \pi' = (\sigma'_{\mathsf{Sig}}, r'_{\mathsf{Commit}}))\}$, returns σ' to $\mathcal{A}$.
- $\mathcal{O}_{Clm}(i', S', \sigma')$: When $i' \neq \widehat{i}$, $\mathcal{B}$ proceeds as in Game_3. When $i' = \widehat{i}$, $\mathcal{B}$ checks whether there is some tuple $(S', m', \sigma', \pi') \in Q'$ or not. If so (i.e., σ' is the response corresponding to the query $(\widehat{i}, S', m')$ to $\mathcal{O}_S$), $\mathcal{B}$ return π' to $\mathcal{A}$ directly. Otherwise, $\mathcal{B}$ returns $\bot$ to $\mathcal{A}$.

Receiving $\mathcal{A}$'s final output $(i^*, S^*, m^*, \sigma^*, \pi^*)$, $\mathcal{B}$ checks whether $i^* = \widehat{i}$. If not, $\mathcal{B}$ returns uniformly sampled $(m_{\text{rand}}, \sigma_{\text{rand}})$ as its final output. Otherwise, $\mathcal{B}$ parses $\sigma^* = (\sigma^*_{\mathsf{MDVS}}, \mathsf{com}^*)$ and $\pi^* = (r^*_{\mathsf{Commit}}, \sigma^*_{\mathsf{Sig}})$, and returns $(m_{\text{sig}} = (\mathsf{spk}_{\widehat{i}}, \sigma^*_{\mathsf{MDVS}}), \sigma_{\text{sig}} = \sigma^*_{\mathsf{Sig}})$ as its final output.

That is the construction of $\mathcal{B}$.

It is clear that $\mathcal{B}$ perfectly simulates Game_3 for $\mathcal{A}$. Note that $\mathcal{A}$ wins only when $\mathsf{ClmVer}(\mathsf{pp}, \mathsf{spk}_{\widehat{i}}, \{\mathsf{vpk}_j\}_{j\in S^*}, \sigma^*, \pi^*) = 1$, which implies that $\mathsf{Sig.Verify}(\mathsf{pp}_{\mathsf{Sig}}, \mathsf{spk}_{\mathsf{Sig},\widehat{i}}, (\mathsf{spk}_{\widehat{i}}, \sigma^*_{\mathsf{MDVS}}), \sigma^*_{\mathsf{Sig}}) = 1$. On the other hand, Evt_1 requires that $\sigma^* = (\sigma^*_{\mathsf{MDVS}}, \mathsf{com}^*)$ is not included in any response generated for $\mathcal{A}$'s $\mathcal{O}_S$-oracle query $(\widehat{i}, *, *)$. In other words, $m_{\text{sig}} = (\mathsf{spk}_{\widehat{i}}, \sigma^*_{\mathsf{MDVS}})$ has never been queried to $\mathcal{B}$'s own signing oracle. Hence, we have

$$\mathbf{Adv}^{\text{unforg}}_{\mathsf{Sig},\mathcal{B}}(\lambda) \geq \Pr[\mathsf{Evt}_1].$$

Note that we have assumed that $\Pr[\mathsf{Evt}_1]$ is non-negligible, so $\mathbf{Adv}^{\text{unf}}_{\mathsf{Sig},\mathcal{B}}(\lambda)$ is also non-negligible, contradicting the existential unforgeability of Sig.

Case 2: If $\Pr[\mathsf{Evt}_2]$ is non-negligible:

Let q be the number of queries made by $\mathcal{A}$ to the oracle $\mathcal{O}_S$. For each $\theta \in [q]$, let $\mathsf{Evt}_2^{(\theta)}$ denote the event that Evt_2 occurs and σ^* (from $\mathcal{A}$'s final output) is included in some response generated by the challenger for $\mathcal{A}$'s θ^{th} $\mathcal{O}_S$-oracle query.

Since $\Pr[\mathsf{Evt}_2]$ is non-negligible, there must be some $\widehat{\theta} \in [q]$ such that $\Pr[\mathsf{Evt}_2^{(\widehat{\theta})}]$ is non-negligible.

Now, we construct a PPT adversary $\mathcal{B}'$ attacking the existential unforgeability of Sig. This adversary $\mathcal{B}'$ is identical to the above adversary $\mathcal{B}$ (described in Case 1), except for the following modifications:

On $\mathcal{A}$'s $\widehat{\theta}^{th}$ query (i', S', m') to $\mathcal{O}_S$, $\mathcal{B}'$ proceeds as the aforementioned $\mathcal{B}$ does, except that when $i' = \widehat{i}$, during the generation of σ', $\mathcal{B}'$ uniformly samples σ'_{Sig} at random, instead of querying its own signing oracle on $(\mathsf{spk}_{\widehat{i}}, \sigma'_{\mathsf{MDVS}})$ to obtain σ'_{Sig}.

That is the construction of $\mathcal{B}'$.

By the hiding property of Commit, as long as the oracle $\mathcal{O}_{Clm}$ is not queried on $(\widehat{i}, S', \sigma')$ by $\mathcal{A}$, where σ' is the response of $\mathcal{A}$'s $\widehat{\theta}^{th}$ query $(\widehat{i}, S', m')$ to $\mathcal{O}_S$, $\mathcal{B}'$'s simulated game and Game_3 are indistinguishable, from $\mathcal{A}$'s point of view. On the other hand, if this $(\widehat{i}, S', \sigma')$ is queried on by $\mathcal{A}$ to $\mathcal{O}_{Clm}$, $\mathsf{Evt}_2^{(\widehat{\theta})}$ is impossible to occur. Hence, the probability that $\mathsf{Evt}_2^{(\widehat{\theta})}$ occurs in $\mathcal{B}'$'s simulation is equivalent to the probability that $\mathsf{Evt}_2^{(\widehat{\theta})}$ occurs in Game_3. Note that when $\mathsf{Evt}_2^{(\widehat{\theta})}$ occurs, $\mathcal{B}'$ successfully finding a valid forgery, i.e., the message $m_{\text{sig}} = (\mathsf{spk}_{\widehat{i}}, \sigma^*_{\mathsf{MDVS}})$ has never been queried to $\mathcal{B}'$'s own signing oracle. So we obtain

$$\mathbf{Adv}^{\text{unforg}}_{\mathsf{Sig},\mathcal{B}'}(\lambda) \geq \Pr[\mathsf{Evt}_2^{(\widehat{\theta})}].$$

Since $\Pr[\mathsf{Evt}_2^{(\widehat{\theta})}]$ is non-negligible, so $\mathbf{Adv}^{\text{unforg}}_{\mathsf{Sig},\mathcal{B}'}(\lambda)$ is also non-negligible, contradicting the existential unforgeability of Sig. □

□

H A more efficient DDW construction

In this section, we delve into a particular case within the MDDW framework, namely DDW, characterized by the existence of only one designated verifier (i.e, $|S| = 1$). We propose a more efficient construction by leveraging a concrete DVS scheme.

H.1 DVS construction

The DVS scheme is shown in Fig. 16, which was firstly proposed in [LV05] and extended from [SBWP03].

$\underline{\mathsf{Setup}(1^\lambda)}$:	$\underline{\mathsf{KG}(\mathsf{pp})}$:
$(e, \mathbb{G}, \mathbb{G}_T, g, p) \leftarrow \mathsf{GenG}(1^\lambda)$	$sk \leftarrow \mathbb{Z}_p^*$, $pk := g^{sk}$
Choose hash functions H_0: $\{0,1\}^* \to \mathbb{G}$	Return (pk, sk)
H_1: $\{0,1\}^* \to \{0,1\}^l$ $\quad /\!/ l = \mathsf{poly}(\lambda)$	
Return $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, p, \mathsf{H}_0, \mathsf{H}_1)$	$\underline{\mathsf{Verify}(\mathsf{pp}, \mathsf{spk}, \mathsf{vsk}, m, \sigma)}$:
	$(r, s) \leftarrow \sigma$, $h \leftarrow \mathsf{H}_0(m, r)$
$\underline{\mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}, \mathsf{vpk}, m)}$:	If $(s \neq \mathsf{H}_1(e(\mathsf{spk}, h^{\mathsf{vsk}})))$: Return 0
$r \leftarrow \{0,1\}^*$, $h \leftarrow \mathsf{H}_0(m, r)$	Return 1
$s \leftarrow \mathsf{H}_1(e(\mathsf{vpk}, h^{\mathsf{ssk}}))$	
Return $\sigma := (r, s)$	

Fig. 16 Algorithms for a concrete and efficient pseudorandom designated-verifier signature DVS

The correctness is trivial, so we omit the analysis of it here. Since there is only one designated verifier, we do not need to consider consistency and there is only one kind of off-the-record property. For the analysis of the existential unforgeability and off-the-record property, please refer to [SBWP03,LV05].

In addition, we will prove that the scheme supports pseudorandomness (please refer to the definition in Sec. H.3). In other words, without the secret key of the designated verifier, it is difficult to distinguish the output of the DVS scheme with a random string.

Then, we have the following theorem.

Theorem 6. *If* H_0 *and* H_1 *are collision resistant hash functions, the DVS scheme* DVS *above is existentially unforgeable, off-the-record and pseudorandom in the random oracle model, under the CBDH, DBDH and GBDH assumptions.*

We postpone the proof of Theorem 6 to Appendix H.3. In fact, the DVS scheme in Fig. 16 supports pseudorandomness for free. Then, when constructing DDW using the framework in Sec. 4, we can eliminate some steps, which we will elaborate on later.

H.2 Some concrete DDWs and their comparison

Schemes. Here we will introduce some concrete DDWs.

Scheme I. In Sec. 4.1, we introduce a framework that shows how to construct a MDDW from a MDVS. When adopting the MDVS [AYSZ14], which is recalled in Appendix. C, and letting $|S| = 1$, we can obtain a DDW.

Scheme II. On the other hands, incorporating the DVS scheme shown in Fig. 16 into the MDDW framework in Sec. 4.1, we can obtain another DDW scheme.

Scheme III. Scheme III can be viewed as an improvement of Scheme II. We could further reduce the size of the DVS, when plugging it into our MDDW framework in Sec. 4.1. The details are as follows.

In Fig. 16, the DVS signature contains a random string r and a hash value s. When plugging the DVS into our framework, we can adopt the hash of previously signed message (i.e., in DDW, the previously signed message could be the prompt or the some tokens just output by the model) to generate the random string r instead. In this way, the signature could only contain a hash value s.

In addition, since the DVS supports pseudorandomness, we do not need to perform XOR computation of the signature and the tokens before sampling new tokens. More exactly, codes in Line 7 of Algorithm 14 and codes in Line 8 of Algorithm 15 are removed. Then, the algorithms use the signature output by the DVS directly to proceed the computation.

Algorithm 14 $\mathsf{WatMar}(\mathsf{pp}, \mathsf{wsk}_i, \mathsf{dpk}_j, \boldsymbol{p})$

1: $\mathsf{ssk}_i \leftarrow \mathsf{wsk}_i$
2: $\mathsf{vpk}_j \leftarrow \mathsf{dpk}_j$
3: $\boldsymbol{t} \leftarrow \epsilon$
4: **while** $|\boldsymbol{t}| + \ell + \ell \cdot \mathsf{len}_{\mathrm{sig}} < n$ **do**
5: $\quad \boldsymbol{t} \leftarrow \boldsymbol{t} || \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
6: $\quad \hat{\boldsymbol{\sigma}} \quad \boldsymbol{\sigma} \leftarrow \mathsf{DVS.Sign}(\mathsf{pp}, \mathsf{ssk}_i, \mathsf{vpk}_j, H_1(\boldsymbol{t}[-\ell]))$
7: $\quad \sout{\boldsymbol{\sigma} \leftarrow \hat{\boldsymbol{\sigma}} \oplus H_2(\boldsymbol{t}[-\ell])}$
8: $\quad \boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \epsilon, \ \boldsymbol{m} \leftarrow \epsilon$
9: $\quad$ **while** $\boldsymbol{\sigma} \neq \epsilon$ **do**
10: $\quad\quad \sigma \leftarrow \boldsymbol{\sigma}[1], \ \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[2:]$
11: $\quad\quad \boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
12: $\quad\quad$ **while** $H_3(\boldsymbol{m} || \boldsymbol{x} || \boldsymbol{\sigma}_{\mathrm{prev}}) \neq \sigma$ **do**
13: $\quad\quad\quad \boldsymbol{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{p}, \boldsymbol{t})$
14: $\quad\quad \boldsymbol{m} \leftarrow \boldsymbol{m} || \boldsymbol{x}, \ \boldsymbol{t} \leftarrow \boldsymbol{t} || \boldsymbol{x}$
15: $\quad\quad \boldsymbol{\sigma}_{\mathrm{prev}} \leftarrow \boldsymbol{\sigma}_{\mathrm{prev}} || \sigma$
16: **if** $|\boldsymbol{t}| < n$ **then**
17: $\quad \boldsymbol{t} \leftarrow \boldsymbol{t} || \mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{p}, \boldsymbol{t})$
18: **return** $\boldsymbol{t}$

Algorithm 15 $\mathsf{Detect}(\mathsf{pp}, \mathsf{wpk}_i, \mathsf{dsk}_j, \boldsymbol{t})$

$\mathsf{spk}_i \leftarrow \mathsf{wpk}_i$
$\mathsf{vsk}_j \leftarrow \mathsf{dsk}_j$
for $\mu \in \{1, 2, \cdots, n - (\ell + \ell \cdot \mathsf{len}_{\mathrm{sig}})\}$ **do**
  $\widetilde{m} \leftarrow H_1(\boldsymbol{t}[\mu : \mu + \ell - 1])$, $\boldsymbol{\sigma} \leftarrow \epsilon$, $\boldsymbol{m} \leftarrow \epsilon$
  for $\varphi \in [\mathsf{len}_{\mathrm{sig}}]$ **do**
    $\boldsymbol{m} \leftarrow \boldsymbol{m} || \boldsymbol{t}[\mu + \varphi \cdot \ell : \mu + \varphi \cdot \ell + \ell - 1]$
    $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} || H_3(\boldsymbol{m} || \boldsymbol{\sigma})$
  ~~$\hat{\boldsymbol{\sigma}} \leftarrow \boldsymbol{\sigma} \oplus H_2(\boldsymbol{t}[\mu : \mu + \ell - 1])$~~
  if $\mathsf{DVS.Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_j, \widetilde{m}, \quad \hat{\boldsymbol{\sigma}} \quad \boldsymbol{\sigma}) = 1$ **then**
    return 1
return 0

Security analysis. It is clear that DDW of Scheme I and Scheme II are secure (i.e., satisfying completeness, consistency, soundness, distortion-freeness, robustness and off-the-record), since the two schemes adopt the framework in Sec. 4.1.

As for Scheme III, it is very similar to Scheme II, except that the XOR computation is removed. The changes on framework would have no influence on these security properties, except distortion-freeness. In addition, the DVS in Fig. 16 supports pseudorandomness. Since the hash function H_1 in the framework of Sec. 4.1 serves as a random oracle. Thus, the result of XOR computation is also pseudorandom, which is essentially equivalent to the output of DVS. Thus, Scheme III also has distortion-freeness.

Comparison. We compare the above three DDWs in terms of the size of a DVS signature, the time of inserting a DVS signature and the time of verifying a DVS signature.

Table 1: Comparison of some DDWs (from the view of one DVS signature)

Schemes	Scheme I	Scheme II	Scheme III
Size	$4 \times \lvert\mathbb{Z}_p^*\rvert$	$\lvert\mathbb{Z}_p^*\rvert + l$	l
Insertion	$1 \times \mathsf{Sca} + 3 \times \mathsf{Exp}$	$1 \times \mathsf{Exp} + 1 \times \mathsf{Bil}$	$1 \times \mathsf{Exp} + 1 \times \mathsf{Bil}$
Verification	$2 \times \mathsf{Sca} + 4 \times \mathsf{Exp}$	$1 \times \mathsf{Exp} + 1 \times \mathsf{Bil}$	$1 \times \mathsf{Exp} + 1 \times \mathsf{Bil}$

When considering the same security level, it should hold that $l \approx |\mathbb{Z}_p^*|$. For simplicity, we assume that Scheme I works over $\mathbb{G}$, and Scheme II and III works over a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Then, Sca means one time of the scalar computation over $\mathbb{G}$, Exp means one time of the exponential computation over $\mathbb{G}$, and bil means one time of the bilinear map computation.

In Table 1, it holds that $l \approx |\mathbb{Z}_p^*|$ when we consider the same security level. So, from the view of size of one DVS signature, it is clear that Scheme III outperforms the others. It means that given a fixed length of the output of the model, Scheme III can inserts more DVS signatures into the output of the model. Therefore, it achieves a stronger soundness.

$\underline{\mathbf{G}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda)}$:
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}^{(b)}_{S\text{-}chl},\mathcal{O}_V}(\mathsf{pp})$
where $\mathcal{O}_{SK},\mathcal{O}_{VK},\mathcal{O}_{SPK},\mathcal{O}_{VPK},\mathcal{O}_V$ are defined as in Fig. 3 with $|S| = 1$,
$\mathcal{O}^{(0)}_{S\text{-}chl}(i,j,m)$ outputs $\mathcal{O}_S(i,j,m)$ ($\mathcal{O}_S$ is also defined in Fig. 3 with input $S = \{j\}$),
$\mathcal{O}^{(1)}_{S\text{-}chl}(i,j,m)$ outputs a uniformly sampled $\sigma \leftarrow \mathcal{SG}$,
let Q denote the set of query-response (i.e., (i,j,m,σ)) of $\mathcal{O}^{(b)}_{S\text{-}chl}$,
all queries j to $\mathcal{O}_{VK}$ should satisfy $(*,j,*,*) \notin Q$,
all queries (i,j,m,σ) to $\mathcal{O}_V$ should satisfy $(i,j,m,\sigma) \notin Q$,
and all queries (i,j,m) to $\mathcal{O}^{(b)}_{S\text{-}chl}$ should satisfy j is not queried to $\mathcal{O}_{VK}$.
If $b' = b$, then return 1,
Return 0

Fig. 17 Game for defining pseudorandomness of DVS

Note that Sca means one time of the scalar computation over $\mathbb{G}$, Exp means one time of the exponential computation over $\mathbb{G}$, and bil means one time of the bilinear map computation. From Table 1, it is hard to determine which scheme would run in the least time, since it depends more on the actual implementation (a empirical comparison can be found in Sec. 5). However, if we only focus on Scheme II and Scheme III, we can conclude that Scheme III is more efficient than Scheme II, in terms of inserting one DVS signature and verifying one DVS signature, since Scheme III does not requires the XOR computation.

H.3 Proof of Theorem 6

As analysis in the aforementioned section, we omit the analysis of the existential unforgeability and off-the-record property, since you can find it in [SBWP03,LV05]. Therefore, we only focus on the pseudorandomness here.
Definition of Pseudurandomness. The definition of pseudorandomness is as follows.

Definition 22. (Pseudorandomness). *We say that* DVS *is pseudorandom, if for all PPT adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda) = |\Pr[\mathbf{G}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{G}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda)$ *defined in Fig. 17.*

Proof for pseudorandomness. We prove the pseudorandomness in the random oracle model, so we add a random oracle $\mathcal{O}_{RO}$, which inputs a string and outputs a random string. We assume that every time calling signing algorithm Sign, Sign would query $str = e(\mathsf{vpk}, h^{\mathsf{ssk}})$ to the random oracle $\mathcal{O}_{RO}$, where $h = \mathsf{H}_0(m,r)$.

Then, we prove the pseudorandomness of the DVS scheme in Fig. 16 in a sequence of games.

$\underline{\mathbf{G}_0}$: On receiving the security parameter λ, the game initializes the public parameter $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and a bit $b \leftarrow \{0,1\}$.

Then, the challenger answers the queries as follows.

- $\mathcal{O}_{RO}(str)$: On receiving a string str, it finds str in its table L_{ro}. If (str, y) in the table, then it returns y. Otherwise, it samples a random string y, inserts (str, y) into L_{ro}, and returns y.
- $\mathcal{O}_{SK}(i)$: On receiving an index i, it calls $\mathsf{SignKG}(\mathsf{pp})$ to generate a key pair $(\mathsf{spk}_i, \mathsf{ssk}_i)$, and store the key pair. Finally, it returns $(\mathsf{spk}_i, \mathsf{ssk}_i)$.
- $\mathcal{O}_{VK}(j)$: On receiving an index j, it proceeds as follows. If $(*, j, *, *) \in Q$, where Q is the query-response set of $\mathcal{O}^{(b)}_{S-chl}$, then it aborts. Otherwise, the oracle would return a key pair in the following way. If j has been queried, then search $(\mathsf{vpk}_j, \mathsf{vsk}_j)$. Else, it calls $\mathsf{VerKG}(\mathsf{pp})$ to generate a key pair $(\mathsf{vpk}_j, \mathsf{vsk}_j)$, and store the key pair. Finally, it returns $(\mathsf{vpk}_j, \mathsf{vsk}_j)$.
- $\mathcal{O}_{SPK}(i)$: On receiving a index i, it calls $\mathcal{O}_{SK}(i)$ to obtain the key pair $(\mathsf{spk}_i, \mathsf{ssk}_i)$, and then returns spk_i.
- $\mathcal{O}_{VPK}(j)$: On receiving a index j, it calls $\mathcal{O}_{VK}(j)$ to obtain the key pair $(\mathsf{vpk}_j, \mathsf{vsk}_j)$, and then returns vpk_j.
- $\mathcal{O}^{(b)}_{S-chl}(i, j, m)$: On receiving (i, j, m), it proceeds as follows. If j has been queried to $\mathcal{O}_{VK}$, then it aborts. Otherwise, it calls $\mathsf{Sign}(\mathsf{pp}, \mathsf{ssk}_i, \mathsf{vpk}_j, m)$ to generate the sigma σ, where ssk_i is output by $\mathcal{O}_{SK}(i)$ and vpk_j is output by $\mathcal{O}_{VPK}(j)$. Then it sets $Q \leftarrow \{(i, j, m, \sigma)\} \cup Q$ and returns σ.
- $\mathcal{O}_V(i, j, m, \sigma)$: On receiving (i, j, m, σ), it proceeds as follows. If $(i, j, m, \sigma) \in Q$, then it aborts. Otherwise, it outputs $b \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_j, m, \sigma)$, where spk_i is output by $\mathcal{O}_{SPK}(i)$, vsk_j is output by $\mathcal{O}_{VK}(j)$.

Finally, $\mathcal{A}$ outputs a bit b'.

It is clear that $\mathbf{G}_0$ is identical to $\mathbf{G}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda)$ when $b = 0$ in Fig. 17. Thus we have

$$\Pr[\mathbf{G}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda) = 1 | b = 0] = \Pr[\mathbf{G}_0 = 1].$$

$\underline{\mathbf{G}_1}$: $\mathbf{G}_1$ is similar to $\mathbf{G}_0$, except when answering the $\mathcal{O}_V$ queries on (i, j, m, σ), $\mathbf{G}_1$ proceeds as follows.

- $\mathcal{O}_V(i, j, m, \sigma)$: On receiving (i, j, m, σ), it proceeds as follows. If $(i, j, m, \sigma) \in Q$, then it aborts. Otherwise, the oracle proceeds:
  - If i has not been queried to $\mathcal{O}_{SK}$ and j has not been queried to $\mathcal{O}_{VK}$, then return 0.
  - If i has been queried to $\mathcal{O}_{SK}$, then parse $(r, s) \leftarrow \sigma$, and compute $h = \mathsf{H}_0(m, r)$, $str = e(\mathsf{vpk}_j, h^{\mathsf{ssk}})$. If str has not been queried to $\mathcal{O}_{RO}$, then return 0. Otherwise, check if $s = L_{(ro)}(str)$. If equal, return 1, otherwise 0.
  - If j has been queried to $\mathcal{O}_{VK}$, then parse $(r, s) \leftarrow \sigma$, and compute $h = \mathsf{H}_0(m, r)$, $str = e(\mathsf{spk}_k, h^{\mathsf{vsk}})$. If str has not been queried to $\mathcal{O}_{RO}$, then return 0. Otherwise, check if $s = L_{(ro)}(str)$. If equal, return 1, otherwise 0.

Let evt denote the event, that for a query (i, j, m, σ) to $\mathcal{O}_V$, it holds that $(i, j, m, \sigma) \notin Q$ and $\mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_k, m, \sigma) = 1$, where i has not been queried to $\mathcal{O}_{SK}$ by the adversary and j has not been queried to $\mathcal{O}_{VK}$ by the adversary.

Then, we have

$$|\Pr[\mathbf{G}_1 = 1] - \Pr[\mathbf{G}_0 = 1]| = \Pr[\mathsf{evt}].$$

If $\Pr[\mathsf{evt}] \leq \mathsf{negl}(\lambda)$, then $|\Pr[\mathbf{G}_1 = 1] - \Pr[\mathbf{G}_0 = 1]| \leq \mathsf{negl}(\lambda)$. In the following, we show that $\Pr[\mathsf{evt}] \leq \mathsf{negl}(\lambda)$.

Case 1 : If for all (i', j', m') that have been queried to $\mathcal{O}^{(b)}_{S-chl}$, it holds $(i, j, m) \neq (i', j', m')$, then we say it breaks the existential unforgeability game, of which the successful probability is negligible.

Case 2 : If for all (i', j', m') that have been queried to $\mathcal{O}^{(b)}_{S-chl}$, there exists one (denoted as $(\tilde{i}, \tilde{j}, \tilde{m})$) such that $(i, j, m) = (\tilde{i}, \tilde{j}, \tilde{m})$, then it implies that $\sigma \neq \tilde{\sigma}$, where $\tilde{\sigma}$ is output by $\mathcal{O}^{(b)}_{S-chl}(\tilde{m})$. We parse $\sigma = (r, s)$ and $\tilde{\sigma} = (\tilde{r}, \tilde{s})$.

- If $r = \tilde{r}$, then it holds that $s \neq \tilde{s}$. However, since $(\tilde{i}, \tilde{j}, \tilde{m})$ is queried to $\mathcal{O}^{(b)}_{S-chl}$, which invokes Sign to generate the sigma. It obtains $\tilde{s}$, by querying $\tilde{str} = e(\mathsf{vpk}_{\tilde{j}}, (\tilde{h})^{\mathsf{ssk}_{\tilde{i}}})$ to random oracle $\mathcal{O}_{RO}$, where $\tilde{h} = \mathsf{H}_0(\tilde{m}, \tilde{r}) = \mathsf{H}_0(m, r) = h$. Thus, when the verification algorithm Verify queries $str = e(\mathsf{spk}_i, h^{\mathsf{vsk}_j}) = e(\mathsf{vpk}_j, h^{\mathsf{ssk}_i}) = e(\mathsf{vpk}_{\tilde{j}}, (\tilde{h})^{\mathsf{ssk}_{\tilde{i}}}) = \tilde{str}$ to the random oracle $\mathcal{O}_{RO}$, it would get $s = \tilde{s}$, which is contradictory to $s \neq \tilde{s}$. Thus, the assumption is not held.
- If $r \neq \tilde{r}$, then it with overwhelming probability that $h = \mathsf{H}_0(m, r)$ is not equal to any $h' = \mathsf{H}_0(m', r')$, where m' is the any message queried to $\mathcal{O}^{(b)}_{S-chl}$ and r' is the corresponding output, since H_0 is a collision-resistant hash function. Thus, $(\mathsf{spk}_i, \mathsf{vpk}_j, h) \neq (\mathsf{spk}_{i'}, \mathsf{vpk}_{j'}, h')$ where (i', j', m') is queried to $\mathcal{O}^{(b)}_{S-chl}$. Note that $str = e(\mathsf{vpk}_j, h^{\mathsf{ssk}_i})$, so $(\mathsf{spk}_i, \mathsf{vpk}_j, h, str)$ is a DBDH tuple. If str has not been queried to $\mathcal{O}_{RO}$, then it with negligible probability that $\mathsf{H}_1(str) = s$ is held, since the random oracle $\mathcal{O}_{RO}$ would return a random string for str. Then, we analyze the probability that A succeeds in querying str to $\mathcal{O}_{RO}$, given $(\mathsf{spk}_i, \mathsf{vpk}_j, h)$. Supposing that A can succeeds in query str with non-negligible probability, we construct another adversary $\mathcal{B}$ to break the CBDH assumption.

  Here is the construction of $\mathcal{B}$. After receiving (X, Y, Z) from the challenger of CBDH, the adversary $\mathcal{B}$ firstly answers the oracle queries from $\mathcal{A}$, mostly just as the challenger in $\mathbf{G}_1$ does. There are some minor difference
  - Assuming that $\mathcal{A}$ issues at most q_0 queries to $\mathcal{O}^{(b)}_{S-chl}$, $\mathcal{B}$ chooses one of them $(\tilde{i}, \tilde{j}, \tilde{m})$, and sets $\mathsf{spk}_{\tilde{i}} = X$, $\mathsf{vpk}_{\tilde{j}} = Y$. To answer this query, $\mathcal{B}$ randomly chooses a random string $\tilde{r}$ and sets $\mathsf{H}_0(\tilde{r}, \tilde{m}) = \tilde{h} = g^{\alpha}$ (here, H_0 also serves as a random oracle). Then, $\mathcal{B}$ computes $\tilde{str} = e(\mathsf{spk}_{\tilde{i}}, \mathsf{vpk}_{\tilde{j}})^{\alpha}$ and queries $\tilde{str}$ to the oracle $\mathcal{O}_{RO}$, obtaining $\tilde{s}$. Finally, $\mathcal{B}$ returns $\tilde{\sigma} = (\tilde{r}, \tilde{s})$.

* When $\mathcal{A}$ issues queries to $\mathcal{O}_{RO}$ for H_0, if there are some queries (assume that they are q_1 such queries) in the form of $(*, \tilde{m})$, then $\mathcal{B}$ chooses one of them (e.g., $(r', \tilde{m})$) and sets $\mathsf{H}_0(r', \tilde{m}) = Z$.

After $\mathcal{A}$ issues a query (i, j, m, σ) to $\mathcal{O}_V$, which makes case 2 happens, if $(i, j, m) = (\tilde{i}, \tilde{j}, \tilde{m})$ and $\mathsf{H}_0(r, \tilde{m}) = Z$ where $\sigma = (r, s)$, then $\mathcal{B}$ finds the query in L_{ro} such that $\mathsf{H}_1(str) = s$. Finally, $\mathcal{B}$ outputs str as its output. That is construction of $\mathcal{B}$.

Then, we analyze the probability of $\mathcal{B}$.

$$\Pr[\mathcal{B} \text{ succeeds}] \geq \frac{1}{q_0 \cdot q_1} \Pr[\text{Case 2 happens}].$$

If Pr[Case 2 happens] is non-negligible, then Pr[$\mathcal{B}$ succeeds] is also non-negligible, which is contradictory to that CBDH is thought a hard problem. Thus, the assumption is not held, so Pr[Case 2 happens] is negligible.

In all, the probability of Case 2 is negligible.

Therefore, $\Pr[\mathsf{evt}] \leq \mathsf{negl}(\lambda)$, which implies that

$$|\Pr[\mathbf{G}_1 = 1] - \Pr[\mathbf{G}_0 = 1]| \leq \mathsf{negl}(\lambda).$$

$\underline{\mathbf{G}_2}$: $\mathbf{G}_2$ is similar to $\mathbf{G}_1$, except when answering the $\mathcal{O}^{(b)}_{S-chl}$ queries on m, it proceeds as follows.

– $\mathcal{O}^{(b)}_{S-chl}(i, j, m)$: On receiving (i, j, m), it proceeds as follows. If j has been queried to $\mathcal{O}_{VK}$, then it aborts. Otherwise, it randomly chooses $\sigma := (r, s) \leftarrow \{0,1\}^* \times \{0,1\}^l$, sets $Q \leftarrow \{(i, j, m, \sigma)\} \cup Q$, and returns σ.

Suppose that the adversary makes ℓ queries to oracle $\mathcal{O}^{(b)}_{S-chl}$. Denote $\mathbf{G}_{2,k}$ ($k \in \{0, 1, \cdots, \ell\}$) as the game, where for the first k queries to oracle $\mathcal{O}^{(b)}_{S-chl}$, the oracle $\mathcal{O}^{(b)}_{S-chl}$ proceeds these queries as it does in $\mathbf{G}_2$, and then for the left $(\ell - k)$ queries to oracle $\mathcal{O}^{(b)}_{S-chl}$, the oracle $\mathcal{O}^{(b)}_{S-chl}$ proceeds these queries as it does in $\mathbf{G}_1$. Thus, $\mathbf{G}_{2,0} = \mathbf{G}_1$ and $\mathbf{G}_{2,\ell} = \mathbf{G}_2$. We have the following lemma.

Lemma 6. *For every $k \in [\ell]$, it holds that $|\Pr[\mathbf{G}_{2,k-1} = 1] - \Pr[\mathbf{G}_{2,k} = 1]| \leq \mathsf{negl}(\lambda)$.*

Proof (of Lemma 6). We denote the ℓ queries as $((i_1, j_1, m_1), \ldots, (i_\ell, j_\ell, m_\ell))$. Then, we show that distinguishing between $\mathbf{G}_{2,k-1}$ and $\mathbf{G}_{2,k}$ is difficult.

In $\mathbf{G}_{2,k-1}$, when querying (i_k, j_k, m_k) to $\mathcal{O}^{(b)}_{S-chl}$, the oracle samples a random string $r_k \leftarrow \{0,1\}^*$, computes $h_k \leftarrow \mathsf{H}_0(m_k, r_k)$ and $str_k = e(\mathsf{vpk}_{j_k}, (h_k)^{\mathsf{ssk}_{i_k}})$, queries str_k to the random oracle $\mathcal{O}_{RO}$, obtaining s_k and setting $\sigma_k = (r_k, s_k)$, sets $Q \leftarrow \{(i_k, j_k, m_k, \sigma_k)\} \cup Q$ and returns σ_k.

Then, we define another game $\mathbf{G}'$, which is similar to $\mathbf{G}_{2,k-1}$, except that $\mathbf{G}'$ queries a random string str' to the random oracle $\mathcal{O}_{RO}$, obtaining s_k (i.e., the random oracle assigns the same s_k to the random oracle query).

It is clear the only difference is that $str_k = e(\mathsf{vpk}_{j_k}, (h_k)^{\mathsf{ssk}_{i_k}})$ and $str' \leftarrow \{0,1\}^{|str_k|}$. Given $(g, \mathsf{spk}_{i_k}, \mathsf{vpk}_{j_k})$, distinguishing str_k and str' can be reduced to DBDH problem, which is thought a hard problem. Thus, we have $|\Pr[\mathbf{G}_{2,k-1} = 1] - \Pr[\mathbf{G}' = 1]| \leq \mathsf{negl}(\lambda)$.

Note that, the distribution of a string output by the random oracle $\mathcal{O}_{RO}$ on a random query str', is equivalent to the uniform distribution. Thus, $\mathbf{G}'$ is identical to $\mathbf{G}_{2,k}$. Therefore, we have $\Pr[\mathbf{G}' = 1] = \Pr[\mathbf{G}_{2,k} = 1]$.

Therefore, it holds that $|\Pr[\mathbf{G}_{2,k-1} = 1] - \Pr[\mathbf{G}_{2,k} = 1]| \leq \mathsf{negl}(\lambda)$.

Applying Lemma 6, we have $|\Pr[\mathbf{G}_1 = 1] - \Pr[\mathbf{G}_2 = 1]| = |\Pr[\mathbf{G}_{2,0} = 1] - \Pr[\mathbf{G}_\ell = 1]| \leq \mathsf{negl}(\lambda)$.

$\underline{\mathbf{G}_3}$: $\mathbf{G}_3$ is similar to $\mathbf{G}_2$, except when answering the queries (i, j, m, σ) to $\mathcal{O}_V$, it proceeds as follows.

- $\mathcal{O}_V(i, j, m, \sigma)$: On receiving (i, j, m, σ), if $(i, j, m, \sigma) \in Q$, it aborts. Otherwise, it returns $b \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{spk}_i, \mathsf{vsk}_j, m, \sigma)$.

Note that the analysis of the indistinguishability between $\mathbf{G}_2$ and $\mathbf{G}_3$ is similar to that between $\mathbf{G}_0$ and $\mathbf{G}_1$. Thus, we have

$$|\Pr[\mathbf{G}_3 = 1] - \Pr[\mathbf{G}_2 = 1]| \leq \mathsf{negl}(\lambda).$$

In fact, $\mathbf{G}_3$ is identical to the game $\mathbf{G}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda)$ when $b = 1$.

Therefore, we have $|\Pr[\mathbf{G}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda) = 1 | b = 0] - \Pr[\mathbf{G}^{\text{ps-rand}}_{\mathsf{DVS},\mathcal{A}}(\lambda) = 1 | b = 1]| = |\Pr[\mathbf{G}_0 = 1] - \Pr[\mathbf{G}_3 = 1]| \leq \mathsf{negl}(\lambda)$, which implies that the DVS scheme DVS constructed in Fig. 16 is pseudorandom.