

BMuSig2: Schnorr-Compatible Blind Multi-Signatures

Kanchan Bisht
Indian Institute of Technology
Hyderabad, India
cs20resch11003@iith.ac.in

Keerthi Aiswarya Varshini
Indian Institute of Technology
Hyderabad, India
keerthi.av@prjt.cse.iith.ac.in

Shivam Sethi
Indian Institute of Technology
Hyderabad, India
cs24mtech12021@iith.ac.in

Maria Francis
Indian Institute of Technology
Hyderabad, India
mariaf@cse.iith.ac.in

R. Kabaleeshwaran
Indian Institute of Information
Technology Design and
Manufacturing Kurnool, India
kabaleesh@iiitk.ac.in

Abstract

Blind signatures and multi-signatures are well-known primitives, but blind multi-signatures (BMS), which combine both these primitives, were only recently formalized by Karantaidou et al. (CCS'24). A BMS scheme allows a user to obtain a compact signature on a common hidden message from a group of signers such that even if the signers collude, they cannot learn the message or link the final signature to any particular interaction. In this paper, we introduce BMuSig2, a 2-round concurrently secure blind multi-signature scheme whose signatures and verification match standard Schnorr signatures. This design enables systems using Schnorr signatures to adopt BMuSig2 as a drop-in replacement, requiring changes only to the issuance phase, while leaving verification unchanged. BMuSig2 builds on MuSig2 multi-signatures (CRYPTO'21) and integrates techniques from a recent blind signature scheme (CRYPTO'24) that leverages non-interactive zero-knowledge (NIZK) arguments and public-key encryption (PKE) to achieve concurrent security. We formally prove the security of BMuSig2 by relying on the unforgeability of MuSig2 and the security of the underlying NIZK and PKE components. We also provide a proof-of-concept implementation to demonstrate its practical efficiency.

CCS Concepts

• Security and privacy → Cryptography.

Keywords

Blind multi-signatures, Concurrent security, Schnorr-compatible

ACM Reference Format:

Kanchan Bisht, Keerthi Aiswarya Varshini, Shivam Sethi, Maria Francis, and R. Kabaleeshwaran. 2026. BMuSig2: Schnorr-Compatible Blind Multi-Signatures. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '26)*, June 1–5, 2026, Bangalore, India. ACM, New York, NY, USA, 18 pages. <https://doi.org/10.1145/3779208.3805986>

1 Introduction

Blind signatures [6] enable a user to obtain a signature on a message unknown to the signer, ensuring that the signer learns nothing about the message and cannot later link the signature to the signing session. Initially introduced for applications such as *e-voting* [7] and *anonymous e-cash* [6], blind signatures now serve as a core primitive for a wide range of privacy-preserving systems. They underpin *anonymous credential systems* like U-Prove (by Microsoft) [25] and Idemix (by IBM) [22], and *anonymous token systems* such as Privacy Pass [16] and Private State Tokens (by Google) [1]. They have also been integrated into blockchains to sign smart contracts for *anonymous transactions* [14, 15].

Multi-Signatures [3, 23] allow a group of signers to jointly generate a single signature on a common message. This significantly reduces storage and communication overhead, making them ideal for distributed applications such as blockchains, which require consensus or authorization from multiple entities (e.g., validators, stakeholders), where the final signature must reflect the approval of all entities involved. Moreover, in applications such as *cryptocurrencies* that already support digital signatures, multi-signatures can serve as a drop-in replacement for the (single-signer) signatures.

Blind Multi-Signatures (BMS), formalized recently by Karantaidou et al. [17] for the *decentralized issuance of anonymous tokens*, combine the functionalities of blind signatures and multi-signatures. BMS allow a user to interact with a group of signers to obtain a compact signature on a hidden common message. Formally, their security is defined by two properties: (i) *blindness*, which guarantees that colluding signers cannot link the final signature to any of their interactions with the user, and (ii) *one-more unforgeability* (OMUF), which ensures that an adversary interacting with the signers l times cannot generate more than l valid forgeries. Conceptually, a BMS resembles an (n, n) -threshold blind signature, but unlike threshold variants, signers can generate their keys independently. Threshold blind signatures, in contrast, require coordinated distributed key generation, making it harder to support dynamic signing sets.

Applications of blind signatures that naturally extend to a multi-authority setting can directly benefit from blind multi-signatures. This includes, for example, e-voting systems with multiple authorities (e.g., election commission officers, tallying authorities, auditors, etc.) who may not fully trust one another. BMS is also a useful primitive for applications like i) *Decentralized Autonomous Organizations* (DAOs)—where multiple governors or council members of a DAO



This work is licensed under a Creative Commons Attribution 4.0 International License. *ASIA CCS '26, Bangalore, India*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2356-8/26/06
<https://doi.org/10.1145/3779208.3805986>

jointly issue anonymous voting tokens to eligible participants, ii) *Decentralized Identity Systems* (DIDs)—where multiple authorities (e.g., government agencies, employers, banks) jointly issue anonymous credentials attesting to a user’s identity or attributes, and iii) *Federated E-cash*—where multiple financial institutions jointly issue anonymous digital tokens representing denominations, preventing unilateral minting and ensuring distributed control over the monetary base.

Constructing a BMS presents several technical challenges that require careful consideration. In particular, for interactive signing protocols—where multiple rounds of interaction occur between the signer and the user—an adversary may initiate several concurrent signing sessions to mount so-called “ROS attacks” [4] and forge signatures. If a scheme is vulnerable to such attacks, the signer is forced to process signing sessions sequentially, which makes the protocol prone to denial-of-service (DoS) attacks and thus impractical for real-world deployment. Consequently, naively extending a blind signature to a multi-signer setting, or incorporating blindness into a multi-signature scheme, can result in constructions that lack concurrent security if not rigorously analyzed. To address this, we propose a “Schnorr-compatible” BMS construction that is concurrently secure. By “Schnorr-compatible” we mean that the final signature is a standard Schnorr signature [26], verifiable via the usual Schnorr verification with a single aggregated verification key that represents all the signers.

1.1 Related Work

Schnorr Signatures. The Schnorr signature scheme [26] is defined over a cyclic group $\mathbb{G}$ of prime order q with generator g , and relies on a cryptographic hash function H_{sig} . A signer has the signing/verification key pair (x, X) , where $x \in \mathbb{Z}_q$ and $X = g^x \in \mathbb{G}$. To sign a message $m \in \{0, 1\}^*$, the signer first selects a random $r \in \mathbb{Z}_q$ and computes the nonce $R := g^r$, the challenge $c := H_{\text{sig}}(X, R, m) \in \mathbb{Z}_q$, and the response $s := r + c \cdot x \bmod q$. The resulting signature is the pair $\sigma = (R, s)$. To verify a signature, a verifier, given (R, s) and the verification key X , recomputes the challenge $c = H_{\text{sig}}(X, R, m)$ and checks whether $g^s \stackrel{?}{=} R \cdot X^c$.

Schnorr signatures were initially proven secure under the discrete logarithm assumption (DL) in the random oracle model (ROM). To avoid the loss from rewinding, Fuchsbaauer et al. [11] gave tighter proofs in the algebraic group model (AGM) with ROM. The standardization organization NIST has included Schnorr signatures, in the form of EdDSA¹, for standardization. They are considered efficient alternatives to RSA and ECDSA, as they offer smaller signature sizes and improved efficiency compared to RSA, while achieving performance comparable to ECDSA. A major advantage of Schnorr signatures is their inherent support for practical, “Schnorr-compatible” multi-signature schemes. Their compactness and efficiency make them inexpensive to store on blockchains. Moreover, since verification follows the standard Schnorr algorithm, the additional complexity is confined to the signers, while verifiers can perform the usual verification without any added overhead. Consequently, Schnorr signatures are widely deployed in major blockchains, such as Bitcoin [32], Bitcoin Cash, Litecoin, and Polkadot, and—through variants like EdDSA—in Monero, Zcash, and Cardano [12].

Schnorr-based Blind Signatures. Blind Schnorr signatures have undergone several refinements to achieve concurrent security. The original blind Schnorr protocol [8], which is a 2-round (3-move) protocol, was proven unforgeable in the Generic Group Model (GGM) and ROM, assuming the hardness of the so-called “ROS problem”. However, Wagner [30] later presented a subexponential-time attack, showing that ROS is weaker than conjectured. Benhamouda et al. [4] further proposed a polynomial-time algorithm for solving ROS, showing that an adversary running polynomially many concurrent signing sessions (specifically when no. of sessions $l \geq \log q$) can efficiently forge signatures. To address this, Fuchsbaauer et al. [11] introduced the *Clause Blind Schnorr* scheme, in which the signer and user engage in two parallel signing sessions, and the signer completes only one chosen at random. They prove unforgeability in AGM and ROM under the one-more discrete logarithm (OMDL) assumption and the hardness of the “modified ROS problem” (mROS). Although mROS is harder than ROS, it can still be solved in subexponential time, thereby limiting the scheme’s concurrent security.

Subsequent works [9, 29] proposed alternative constructions that deviate from the standard Schnorr structure to achieve concurrent security. Tessaro and Zhou [29] introduced non-linearity in their blind signatures to mitigate ROS attacks and proved their unforgeability relying on a new ROS variant—“Weighted Fractional ROS (WFROS)”. Building on this, Snowblind [9] presented a pairing-free blind signature scheme and its threshold variant that achieves exponential security under the hardness of “one-dimensional ROS”.

Most recently, Fuchsbaauer and Wolf [12] proposed a 2-round (4-move) “Schnorr-compatible” blind signature scheme that achieves concurrent security. The scheme leverages a non-interactive zero-knowledge (NIZK) argument, and its unforgeability relies on the soundness of the NIZK and the security of standard Schnorr signatures, for the underlying group and hash function families.

Schnorr-based Multi-Signatures. One of the challenges in extending signatures to multi-signer settings is the “rogue-key attacks” [3, 18, 19], where an adversary can manipulate the verification key based on the honest signers’ verification keys. These attacks can be mitigated either via the *knowledge-of-secret-key* (KOSK) model, which requires each signer to provide a zero-knowledge proof of knowledge (ZKPoK) for their verification key, or via the *plain public-key* (PPK) model, where verification keys can be aggregated without the need to check their validity.

MuSig [20] and its variants, MuSig-DN [24] and MuSig2 [23], are the only “Schnorr-compatible” multi-signature schemes that are concurrently secure under the PPK model. To achieve concurrent security, MuSig introduced an initial commitment round, where each signer commits to its nonce before seeing the nonces of other signers. This results in a 3-move protocol, where the first round can be preprocessed without the message. MuSig-DN reduces the protocol to 2 moves using deterministic nonces, but requires a heavy ZKPoK, making it less efficient in practice. MuSig2 further improved efficiency and security by providing a 2-move protocol, where each signer sends multiple nonces and the final aggregated nonce is computed as a linear combination of the nonces with coefficients derived from a hash function. PP-MuSig2 [31] extends MuSig2 to a privacy-preserving multi-signature scheme that hides both the identities and the number of signers.

¹<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf>

Table 1: Comparison of BMuSig2 and BM_SB

Property	BMuSig2 (Our work)	BM_SB [17]
No. of rounds	2 rounds (4 moves)	3 rounds (5 moves)
Issuance cost	Higher (uses NIZK)	Lower (no NIZK)
Verification cost	Constant	Linear in no. of signers
Schnorr-compatible	Yes	No
Key aggregation	Yes	No
Concurrent security	Yes	Yes
Security model	PPK (stronger)	KOSK (weaker)

Blind Multi-Signatures. To the best of our knowledge, the only practical blind multi-signature (BMS) constructions are due to Karantaidou et al. [17]. Their work introduces a pairing-based variant, BM_BLS, constructed from BLS signatures [5]. To improve the verification efficiency, they further proposed a pairing-free variant, BM_SB, derived from the Snowblind threshold blind signature scheme [9]. However, BM_SB requires each signer to provide a proof of possession of its verification key, which in turn prevents key aggregation. They proved the concurrent security of BM_BLS based on the DL assumption in ROM, and the concurrent security of BM_SB based on the DL assumption in the AGM+ROM setting.

1.2 Our Contribution

We present BMuSig2, the first practical, concurrently secure, Schnorr-compatible blind multi-signature scheme. BMuSig2 is a 2-round (4-move) protocol, improving upon the only other practical pairing-free BMS construction, BM_SB [17], which requires five moves. While the signing phase of BMuSig2 is slightly more expensive due to the use of NIZK arguments, it outputs standard Schnorr signatures that can be verified under a single aggregated verification key, resulting in a verification cost independent of the number of signers. In contrast, BM_SB does not support key aggregation and requires verification against each individual verification key.

Moreover, BMuSig2 is secure in the plain public-key (PPK) model, where verification keys require no proof-of-possession, whereas BM_SB relies on the stronger KOSK assumption, which requires every signer to provide such proofs to prevent rogue-key attacks. Together, these properties make BMuSig2 a drop-in replacement for systems already using Schnorr signatures, as only the issuance phase is modified while verification remains unchanged. Its signer-independent verification cost also makes it an efficient choice for practical applications, such as anonymous token verification over blockchain. We summarize this in Table 1.

We now briefly summarize our technical contributions.

Blind Multi-Signatures (BMS). In Sec. 3, we provide a rigorous definition of a 2-round (4-move) blind multi-signature scheme and formally define its security properties: *blindness* and *one-more unforgeability*.

Blind MuSig2 (BMuSig2). In Sec. 4, we present the BMuSig2 scheme, which builds upon the MuSig2 multi-signatures. In MuSig2, concurrent security is achieved by having each signer sample multiple nonces and derive the final aggregated nonce as their linear combination using a nonce coefficient, which is publicly computable via the hash of individual nonces of all the signers. This design prevents ROS-style attacks on MuSig2. If an adversary attempts to

control the aggregated nonce by choosing its nonce after seeing the honest signer’s nonce, the nonce coefficient changes accordingly, thereby modifying the honest signer’s effective nonce contribution. Consequently, the honest signer’s contribution taken across all sessions is not constant, which is required to mount a ROS-style forgery via Wagner’s algorithm.

A naive approach to construct a BMS scheme from MuSig2 is to introduce blindness as in the standard blind Schnorr signatures. However, as shown in Sec. 4, this approach does not achieve concurrent security. To lift MuSig2 to a concurrently secure BMS, we adapt the NIZK arguments from Fuchsbaauer et al. [12] for concurrently secure blind signatures. The core idea is that, in the first round, the user commits to their secret randomness (blinding factors) and the message before receiving the nonces of the signers. In the second round, the user provides NIZK arguments that prove the challenge sent to the signers is consistent with the committed values. This prevents ROS attacks in blind-signature settings, where an adversary could otherwise adaptively choose its randomness after seeing the nonces of the signers to generate a forgery.

In BMuSig2, blindness introduces an additional complication: the nonce coefficient—which is publicly computable in MuSig2—is no longer derivable by the signers in the blind setting. Only the user observes all the nonces and therefore only they can compute this coefficient and forward it to the signers. Therefore, in the second round, the user must provide an NIZK argument that simultaneously proves that (i) the challenge is consistent with the committed blinding factors and message, and (ii) the nonce coefficient is correctly computed from the received nonces.

Security. BMuSig2 uses a public-key encryption scheme PKE for the “commitment” in the first round and a NIZK argument system NArg for the proof in the second round. We establish the concurrent security of BMuSig2 by directly reducing it to the security of these underlying primitives. In particular, we show that blindness follows from the zero-knowledge property of the NIZK argument system (NArg) and the CPA security of the public-key encryption scheme (PKE), while one-more unforgeability follows from the soundness of NArg and the unforgeability of the MuSig2 signatures over the chosen group and hash function families.

Implementation and Evaluation. In Sec. 5, we present a proof-of-concept (PoC) of BMuSig2 for anonymous tokens, accompanied by a generic smart-contract library for on-chain verification of the tokens on Ethereum. Our evaluations show that BMuSig2 achieves practical and efficient verification while significantly reducing the token and verification key sizes compared to the baseline BM_SB.

2 Preliminaries

Notations. For $n \in \mathbb{N}^+$, we write $[n]$ to represent the set $\{1, \dots, n\}$. We use $x := y$ to denote the assignment of value x to variable y , while “=” to denote either reassignment of a variable or equality, depending on the context. The notation $x \equiv_q y$ is shorthand for $x \equiv y \pmod q$. We use $x \xleftarrow{\$} X$ to denote that x is sampled uniformly at random from the set X .

We denote lists by $\vec{a}$. The empty list is written as $\vec{a} := []$. Appending an element x to a list $\vec{a}$ is written as $\vec{a} := \vec{a} \parallel x$. Let $|\vec{a}|$ be the size of a list, then $\vec{a}_j$ denotes the j^{th} element of the list, or ε if $j \notin [|\vec{a}|]$. We denote tuples by $t := (a, \dots, z)$, and the i^{th} element of

t is given as $t[i]$, which equals ε if it is empty. If the tuple $(a, \dots, z)$ can be parsed from t in polynomial time, we write $(a, \dots, z) : \subseteq t$.

All algorithms are assumed to run in probabilistic polynomial time (PPT) in the (implicit) security parameter λ (often represented in unary as 1^λ). The adversaries are considered efficient by default, i.e., they are also modelled as PPT algorithms.

A function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ is called negligible, if for every constant $c > 0$, $\exists k_0$ such that $f(k) < 1/k^c$, $\forall k \geq k_0$. We use $\text{negl}(\lambda)$ to denote a negligible function, in the security parameter λ .

An m -round ($2m$ or $2m + 1$ -move) algorithm is an interactive protocol in which two (or more) parties exchange messages. A move is a single message sent by one party, while a round consists of two consecutive moves—one from each party—capturing a single query–response interaction. Some protocols conclude with an additional final one-sided message, resulting in $2m + 1$ total moves.

Definition 2.1 (Group Generation Algorithm). A group generation algorithm GGen is a PPT algorithm which outputs $(q, \mathbb{G}, g) \leftarrow \text{GGen}(1^\lambda)$ based on the security parameter λ , such that $\mathbb{G}$ is a group of prime order q , where $\lceil \log_2 q \rceil = \lambda$, and g is a generator of $\mathbb{G}$.

Definition 2.2 (Hash Function Generator). A (target-range) hash function generator HGen is a PPT algorithm that on input a number $n \in \mathbb{N}^+$ and returns the description of a function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_n$.

We use the notation $H_1, \dots, H_m \leftarrow \text{HGen}(n)$ to indicate that m hash functions are generated independently by HGen .

2.1 Non-Interactive Zero-Knowledge Argument

We follow the formal definitions of non-interactive zero-knowledge (NIZK) argument systems as presented in [12].

A NIZK system for a parameterized relation $R : \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}$ is defined for parameters par_R , a statement θ , and a witness w such that $R(\text{par}_R, \theta, w) = 1$. A NIZK system consists of the following PPT algorithms:

- $\text{NArg.Rel}(1^\lambda) \rightarrow \text{par}_R$: On input security parameter λ , outputs relation parameters par_R .
- $\text{NArg.Setup}(\text{par}_R) \rightarrow (crs, \tau)$: Outputs a common reference string (crs) and a simulation trapdoor τ based on par_R .
- $\text{NArg.Prove}(crs, \theta, w) \rightarrow \pi$: On input crs , a statement θ and witness w , outputs a proof π .
- $\text{NArg.Ver}(crs, \theta, \pi) \rightarrow \{0, 1\}$: On input crs , statement θ and proof π , verifies π and outputs 1 (accept) or 0 (reject).
- $\text{NArg.SimProve}(crs, \tau, \theta) \rightarrow \pi$: On input crs , statement θ and simulation trapdoor τ , outputs simulated proof π .

Security. A NIZK argument system must satisfy *correctness*, *computational soundness*, and *zero-knowledge* properties.

A NIZK argument system NArg is (perfectly) correct if any honestly generated proof from a valid witness for a true statement is always accepted by the verifier.

Definition 2.3 (Computational Soundness). The NIZK argument system $\text{NArg}[R]$ is (adaptively) computationally sound if for every adversary A , $\text{Adv}_{\text{NArg}[R], A}^{\text{SND}}(\lambda) := \Pr[\text{SND}_{\text{NArg}[R]}^A(\lambda)] = \text{negl}(\lambda)$, where the game SND is defined as:

Game $\text{SND}_{\text{NArg}[R]}^A(\lambda)$

```

 $\text{par}_R \leftarrow \text{NArg.Rel}(1^\lambda); (crs, \tau) \leftarrow \text{NArg.Setup}(\text{par}_R);$ 
 $(\theta, \pi) \leftarrow A(crs);$ 
return  $(\text{NArg.Ver}(crs, \theta, \pi) = 1 \wedge \forall w \in \{0, 1\}^* : R(\text{par}_R, \theta, w) = 0).$ 

```

Definition 2.4 (Zero-Knowledge). The NIZK argument system $\text{NArg}[R]$ is (computationally) zero-knowledge if for every adversary A , $\text{Adv}_{\text{NArg}[R], A}^{\text{ZK}}(\lambda) := |\Pr[\text{ZK}_{\text{NArg}[R]}^{A, 0}(\lambda)] - \Pr[\text{ZK}_{\text{NArg}[R]}^{A, 1}(\lambda)]| = \text{negl}(\lambda)$, where the game ZK is defined by:

Game $\text{ZK}_{\text{NArg}[R]}^{A, b}(\lambda)$

```

 $\text{par}_R \leftarrow \text{NArg.Rel}(1^\lambda);$ 
 $(crs, \tau) \leftarrow \text{NArg.Setup}(\text{par}_R);$ 
 $b' \leftarrow A^{\text{Prove}}(crs);$ 
return  $b'.$ 

Oracle  $\text{Prove}(\theta, w)$ 
if  $R(\text{par}_R, \theta, w) = 0$  return  $\perp$ ;
 $\pi_0 \leftarrow \text{NArg.Prove}(crs, \theta, w);$ 
 $\pi_1 \leftarrow \text{NArg.SimProve}(crs, \tau, \theta);$ 
return  $\pi_b.$ 

```

2.2 Public-Key Encryption

A public-key encryption consists of the following PPT algorithms.

- $\text{PKE.Setup}(1^\lambda) \rightarrow \text{par}$: On input security parameter λ , outputs public parameters par .
- $\text{PKE.KeyGen}(\text{par}) \rightarrow (ek, dk)$: On input par , outputs an encryption key ek and decryption key dk , where ek defines the message, randomness and ciphertext space $(\mathcal{M}_{ek}, \mathcal{R}_{ek}, \mathcal{C}_{ek})$.
- $\text{PKE.Enc}(ek, M; \rho) \rightarrow C$: On input an encryption key ek , message $M \in \mathcal{M}_{ek}$, and randomness $\rho \in \mathcal{R}_{ek}$, outputs a ciphertext $C \in \mathcal{C}_{ek}$.
- $\text{PKE.Dec}(dk, C) \rightarrow M$: On input the decryption key dk and ciphertext C , outputs a message M or $\perp$ if decryption fails.

Security. A PKE scheme must satisfy *correctness* and *CPA-security*. A public-key encryption scheme PKE is (perfectly) correct if, for every key pair generated and every valid message, encrypting the message with the public key and then decrypting with the secret key always returns the original message.

Definition 2.5 (CPA-Security). A public-key encryption scheme PKE is *secure against chosen-plaintext attacks* (*CPA-secure*) if for all adversaries A , $\text{Adv}_{\text{PKE}, A}^{\text{CPA}}(\lambda) := |\Pr[\text{CPA}_{\text{PKE}}^{A, 0}(\lambda)] - \Pr[\text{CPA}_{\text{PKE}}^{A, 1}(\lambda)]| = \text{negl}(\lambda)$, where the game CPA is defined as follows:

Game $\text{CPA}_{\text{PKE}}^{A, b}(\lambda)$

```

 $(ek, dk) \leftarrow \text{PKE.KeyGen}(1^\lambda);$ 
 $b' \leftarrow A^{\text{Enc}}(ek);$ 
return  $(b = b').$ 

Oracle  $\text{Enc}(M_0, M_1)$ 
 $C \leftarrow \text{PKE.Enc}(ek, M_b);$ 
return  $C.$ 

```

2.3 The ROS Problem

We adopt the following definition of the ROS (Random inhomogeneous, Overdetermined, Solvable system of linear equations) problem from [11]. Let q be a prime and $H_{\text{ros}} : \mathbb{Z}_q^l \times \Omega \rightarrow \mathbb{Z}_q$ be the random oracle taking inputs a vector $\vec{\rho} = (\rho_j)_{j \in [l]} \in \mathbb{Z}_q^l$ and auxiliary information $\text{aux} \in \Omega$. An adversary A is given oracle access to H_{ros} . Its goal is to output $l + 1$ distinct pairs $(\vec{\rho}_i, \text{aux}_i)_{i \in [l+1]}$ together with solutions $(c_j)_{j \in [l]} \in \mathbb{Z}_q^l$ that satisfy the linear system $\sum_{j=1}^l \rho_{i,j} c_j \equiv_q H_{\text{ros}}(\vec{\rho}_i, \text{aux}_i)$, $i \in [l+1]$.²

²The original definition of the ROS problem by Schnorr [27] sets $\Omega := \emptyset$. Allowing a more general choice of Ω constitutes a natural extension of the model and does not seem to impact the computational hardness of the problem.

Let $\text{Pgen}(1^\lambda)$ be a parameter-generation algorithm that, on input the security parameter λ , outputs an odd prime q of bit-length $\lambda = \lceil \log q \rceil$. The theorem below follows from [4, Theorem 1].

THEOREM 2.6 (ROS ATTACK). *If $l \geq \lambda$, there exists a PPT adversary A that solves the ROS problem relative to Pgen in dimension l .*

Fuchsbauer et al. [11] used this ROS solver A to create a reduction B that breaks the one-more unforgeability of blind Schnorr signatures. The reduction B begins by initiating l parallel signing sessions with the signer to obtain nonces $\{R_j\}_{j \in [l]}$, and then runs A as a subroutine. When A queries $H_{\text{ros}}(\vec{r}, (\alpha, \beta, m))$, for $m \in \{0, 1\}^*$ and $\alpha, \beta \in \mathbb{Z}_p$, the reduction B computes $R := g^\alpha X^\beta \prod_{j=1}^l R_j^{\rho_j}$ and responds with $H_{\text{sig}}(X, R, m) + \beta$. This mapping ensures that each ROS query corresponds to a signing transcript. Consequently, any ROS solution returned by A can be used by B to forge a valid blind Schnorr signature, thereby contradicting one-more unforgeability.

2.4 Blind Signatures

A blind signature scheme is an interactive protocol between a signer and a user that allows the user to obtain a signature on a message hidden from the signer, such that the signer cannot later link the resulting signature to the user. A blind signature scheme BS consists of four PPT algorithms. (Setup , KeyGen , Sign , Ver). The syntax of a blind signature scheme is defined in Appendix A.

Security. A blind signature scheme must satisfy *correctness*, *blindness* and *one-more unforgeability* (OMUF). *Correctness* ensures that an honest execution of the protocol always outputs a valid signature. *Blindness* ensures that a signature cannot be linked to its corresponding signing session, even by the signer. *One-more unforgeability* (OMUF) requires that any adversary engaging in at most ℓ signing sessions cannot output $\ell + 1$ valid signatures. In the *existential* variant (EOMUF), these $\ell + 1$ signatures must be on distinct messages, whereas in (strong) OMUF, no distinctness condition on the messages is required.

2.5 Multi-Signatures

Multi-signature schemes enable a group of signers $\{S_j\}_{j \in [n]}$ (each possessing their own signing/verification key pair) to collaboratively generate a compact signature σ on a message m . Some schemes additionally support *public-key aggregation*, which allows for a single, compact representation of verification keys of all signers. In this work, we focus on multi-signatures which allow for public-key aggregation. A multi-signature scheme MS consists of the following PPT algorithms. We use the index $i \in [n]$ to denote the local signer S_i executing a protocol step, and use $j \in [n]$ when referring to all signers collectively.

- $\text{MS.Setup}(1^\lambda) \rightarrow (par)$: On input the security parameter λ , outputs the public parameters par .
- $\text{MS.KeyGen}(par) \rightarrow (sk_i, vk_i)$: Outputs a signing-verification (sk_i, vk_i) key pair for the signer.
- $\text{MS.KeyAgg}(\hat{K} := \{vk_j\}_{j=1}^n) \rightarrow awk$: Given a set of verification keys $\hat{K} = \{vk_1, \dots, vk_n\}$, outputs an aggregated verification key awk .
- $\text{MS.Sign}(\{S_i(sk_i, \hat{K}, m)\}_{i \in [n]}) \rightarrow \sigma$: An interactive protocol among the signers, each using its signing key sk_i , the set

of verification keys $\hat{K}$, and message m to jointly generate a signature σ .

- $\text{MS.Ver}(awk, m, \sigma) \rightarrow \{0, 1\}$: Outputs 1 if σ is a valid signature for m under the aggregated verification key awk .

Security. A multi-signature scheme must satisfy *correctness* and *unforgeability*. *Correctness* ensures that honest signers jointly produce a valid signature on the message. *Unforgeability* in multi-signatures is captured by two standard notions: existential unforgeability under chosen-message attacks (EUF-CMA) and strong existential unforgeability under chosen-message attacks (SUF-CMA) [21]. EUF-CMA requires that, even if an adversary corrupts all but one signer with verification key vk^* , it cannot output a valid signature for $(\hat{K}, m)$ such that $vk^* \in \hat{K}$ and $(\hat{K}, m)$ was not previously queried to the signing oracle. SUF-CMA strengthens this notion by requiring that, for any pair $(\hat{K}, m)$, the adversary cannot produce more valid signatures than those returned by the signing oracle. In particular, even a new valid signature on a previously queried pair constitutes a valid forgery.

2.6 Multi-Signature Scheme: MuSig2

MuSig2 [23] is a concurrently secure, two-move multi-signature scheme built on standard Schnorr signatures. We define MuSig2 with respect to a group-generation algorithm GGen (Def. 2.1) that outputs a group of prime order q , a hash generation algorithm HGen (Def. 2.2), which outputs three hash functions $H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}}$ mapping to $\mathbb{Z}_q$, and an integer v , which specifies the number of nonces sent by each signer.

Formally, MuSig2 consists of the following PPT algorithms.

- $\text{MuSig2.Setup}(1^\lambda) \rightarrow par$: On input security parameter λ , runs $(\mathbb{G}, q, g) \leftarrow \text{GrGen}(1^\lambda)$, $H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}} \leftarrow \text{HGen}(q)$, and outputs public parameters $par := ((\mathbb{G}, q, g), H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}})$.
- $\text{MuSig2.KeyGen}(par) \rightarrow (sk_i, vk_i)$: On input par , signer S_i samples $x_i \xleftarrow{\$} \mathbb{Z}_q$, computes $X_i := g^{x_i}$, and outputs the signing and verification key as $(sk_i := x_i, vk_i := X_i)$.
- $\text{MuSig2.KeyAgg}(\hat{K} := \{X_j\}_{j \in [n]}) \rightarrow X$: On input the set of verification keys $\hat{K}$, computes the key-aggregation coefficients $a_i := H_{\text{agg}}(\hat{K}, X_i)$ (for every $i \in [n]$), and outputs the aggregated verification key as $X := \prod_{i=1}^n X_i^{a_i}$.
- $\text{MuSig2.Sign} = (\text{Sign}_1, \text{SignAgg}_1, \text{Sign}_2, \text{SignAgg}_2)$: This algorithm proceeds in 1-round (2-move).

First move

- $\text{MuSig2.Sign}_1(\cdot) \rightarrow pst_i$: Signer S_i samples $(r_{i,1}, \dots, r_{i,v}) \xleftarrow{\$} \mathbb{Z}_q$, computes the nonces $(R_{i,1}, \dots, R_{i,v}) := (g^{r_{i,1}}, \dots, g^{r_{i,v}})$ and outputs an internal state $st_i := (r_{i,1}, \dots, r_{i,v})$ together with a public state $pst_i := (R_{i,1}, \dots, R_{i,v})$.
- $\text{MuSig2.SignAgg}_1(\{pst_j\}_{j \in [n]}) \rightarrow pst$: The aggregator takes the individual nonces $\{(R_{i,1}, \dots, R_{i,v}) : pst_j\}_{j \in [n]}$ of all the signers, computes $\forall t \in [v], R_t := \prod_{j=1}^n R_{j,t}$ and outputs the public state $pst := (R_1, \dots, R_v)$.

Second move

- $\text{MuSig2.Sign}_2(\hat{K}, st_i, pst, sk_i, m) \rightarrow \sigma_i$: Let $m \in \{0, 1\}^*$ be the message and X denote the aggregated verification key from $\text{MuSig2.KeyAgg}(\hat{K})$. The signer S_i takes $(R_1, \dots, R_v) \subseteq pst$, computes $b := H_{\text{non}}(X, R_1, \dots, R_v, m)$ and forms the aggregated nonce $R := \prod_{t=1}^v R_t^{b^{t-1}}$.

Using R , it defines the challenge $c := H_{\text{sig}}(X, R, m)$, takes its internal state $(r_{i,1}, \dots, r_{i,v}) \subseteq st_i$ and signing key $sk_i := x_i$ to compute $s_i := c a_i x_i + \sum_{t=1}^v r_{i,t} b^{t-1} \bmod q$, and outputs the partial signature $\sigma_i := (R, s_i)$.

- **MuSig2.SignAgg2** $(m, \{\sigma_j\}_{j \in [n]}) \rightarrow \sigma$: The aggregator then takes the partial signatures $\{\sigma_i := (R, s_i)\}_{i \in [n]}$ of all the signers, computes $s := \sum_{j=1}^n s_j \bmod q$ and outputs the final signature as $\sigma := (R, s)$.
- **MuSig2.Ver** $(X, m, \sigma) \rightarrow \{0/1\}$: On input an aggregated verification key X , a message m , and a signature $\sigma = (R, s)$, the verifier computes $c := H_{\text{sig}}(X, R, m)$ and outputs 1 if $g^s = R X^c$.

Note: MuSig2 is a *stateful* scheme, i.e. after executing Sign1 with a particular signing state, a signer must ensure that the same state is never reused for another execution of Sign1. Reusing the state would allow an adversary to recover the signer's secret key trivially. *Security.* Correctness follows directly from the honest execution of the protocol. The two-nonce ($v = 2$) variant of MuSig2 achieves EUF-CMA security under the algebraic one-more discrete logarithm (AOMDL) assumption in the AGM+ROM model [23]. The four-nonce ($v = 4$) variant achieves SUF-CMA security under the AOMDL assumption in ROM [21].

2.6.1 Unforgeability of MuSig2. We briefly describe the EUF-CMA and SUF-CMA security notions for MuSig2, which underlie the security reductions for our construction. A formal definition of the games is provided in Appendix B.

For multi-signatures, an adversary $\mathcal{U}$ is assumed to have control over all except one honest signer. A key pair (sk_1, vk_1) is generated for the honest signer, and $\mathcal{U}$ is given the verification key vk_1 . $\mathcal{U}$ may initiate an arbitrary number of (concurrent) signing sessions with the honest signer. $\mathcal{U}$ has access to the MSIGN₁ and MSIGN₂ oracles, which simulate the Sign₁ and Sign₂ oracles of MuSig2 using the honest signer's signing key sk_1 .

In EUF-CMA game, $\mathcal{U}$ eventually outputs a verification key set $\hat{K}^*$, a message m^* , and a signature σ^* as a forgery. The game returns 1 (i.e. $\mathcal{U}$ wins) if (i) $\mathcal{U}$ never issued a MSIGN₂ query with the same verification key set $\hat{K}^*$ and message m^* , (ii) $vk_1 \in \hat{K}^*$, and (iii) the forgery is valid, i.e. $\text{Ver}(\text{KeyAgg}(\hat{K}^*), m^*, \sigma^*) = 1$.

In SUF-CMA game, $\mathcal{U}$ eventually outputs a verification key set $\hat{K}^*$, a message m^* , and f' signatures $(\sigma_{\ell'})_{\ell' \in [f']}$ as a forgery. The game returns 1 (i.e. $\mathcal{U}$ wins) if (i) $(\hat{K}^*, m^*)$ was queried to MSIGN₂ fewer than f' times (ii) $vk_1 \in \hat{K}^*$, and (iii) all f' signatures are valid $\forall \ell' \in [f'] : \text{Ver}(\text{KeyAgg}(\hat{K}^*), m^*, \sigma_{\ell'}) = 1$.

We treat MuSig2 as a secure building block in our construction. Consequently, our scheme is *stateful*, and its security analysis relies on the following assumption.

Assumption 1 (MuSig2 Unforgeability). There exists a group generator algorithm GGen and a hash-function generator HGen such that the MuSig2 scheme is unforgeable (EUF-CMA/SUF-CMA). In particular, for every adversary $\mathcal{U}$,

$$\text{Adv}_{\text{MuSig2}[\text{GGen}, \text{HGen}], \mathcal{U}}^{\text{UF}}(\lambda) \leq \text{negl}(\lambda).$$

Note. We directly assume the unforgeability of MuSig2, because all known security proofs of MuSig2 are carried out in the ROM, but the NArg relation $R_{\text{B MuSig2}}$ defined in 1 depends on the concrete hash function used in the scheme. In the ROM, this hash function would

be replaced by a random function, for which efficient proofs are not possible. Therefore, we assume MuSig2 is secure under a specific hash instantiation. This approach is analogous to Fuchsbaumer–Wolf [12], who likewise assume the security of Schnorr signatures under a concrete hash function.

3 Blind Multi-Signatures

A blind multi-signature scheme allows a user $\mathcal{U}$ to interact with multiple signers $\{\mathcal{S}_j\}_{j \in [n]}$ in order to obtain a compact signature on a message m hidden from all the signers. For concreteness, we focus on schemes with 2-round (4-move) interactive protocols. A blind multi-signature scheme BMS consists of the following PPT algorithms.

- **BMS.Setup** $(1^\lambda) \rightarrow (par)$: On input security parameter λ , outputs the public parameters par .
- **BMS.KeyGen** $(par) \rightarrow (sk_i, vk_i)$: Outputs a signing and verification key pair (sk_i, vk_i) for the signer $\mathcal{S}_i$.
- **BMS.KeyAgg** $(\hat{K} := \{vk_j\}_{j \in [n]}) \rightarrow avk$: Given a set of verification keys $\hat{K}$, outputs an aggregated verification key avk .
- **BMS.Sign** $(\langle \mathcal{U}(\hat{K}, m), \{\mathcal{S}_i(\hat{K}, sk_i)\}_{i \in [n]} \rangle \rightarrow \perp/\sigma$: Let $m \in \{0, 1\}^*$ be the message and $\hat{K}$ denote the set of verification keys of the n signers. The signing protocol between the user $\mathcal{U}$ and signers $\{\mathcal{S}_i\}_{i \in [n]}$ on a message m is a 2-round (4-move) interactive protocol which proceeds as follows.

First Round

- **User0** $(\hat{K}, m) \rightarrow (st_{\mathcal{U},0}, pst_{\mathcal{U},0})$: The user generates an internal state $st_{\mathcal{U},0}$ and outputs a public state $pst_{\mathcal{U},0}$.
- **Sign1** $(sk_i, \hat{K}, pst_{\mathcal{U},0}) \rightarrow (st_{\mathcal{S}_i}, pst_{\mathcal{S}_i})$: Each signer $\mathcal{S}_i$ uses its signing key sk_i to compute a secret state $st_{\mathcal{S}_i}$ and a public response $pst_{\mathcal{S}_i}$.

Second Round

- **User1** $(st_{\mathcal{U},0}, \{pst_{\mathcal{S}_j}\}_{j \in [n]}) \rightarrow (st_{\mathcal{U},1}, pst_{\mathcal{U},1})$: The user updates its internal state to $st_{\mathcal{U},1}$ based on all signer responses and outputs a public response $pst_{\mathcal{U},1}$.
- **Sign2** $(st_{\mathcal{S}_i}, pst_{\mathcal{U},1}) \rightarrow (b, \sigma_i)$: Each signer $\mathcal{S}_i$ uses its state $st_{\mathcal{S}_i}$ and the user's second-round message $pst_{\mathcal{U},1}$ to compute a partial signature σ_i . It also outputs a status bit $b \in \{0, 1\}$ indicating whether it successfully completed the computation ($b = 1$) or not ($b = 0$).
- **User2** $(st_{\mathcal{U},1}, \{\sigma_j\}_{j \in [n]}) \rightarrow \perp/\sigma$: The user takes its internal state $st_{\mathcal{U},1}$ and partial signatures $\{\sigma_j\}_{j \in [n]}$ from all the signers as input, and outputs either the aggregated signature σ or $\perp$ if the protocol fails.
- **BMS.Ver** $(avk, m, \sigma) \rightarrow \{0, 1\}$: Outputs 1 if σ is a valid signature on m under the aggregated verification key avk .

3.1 Security of BMS

A blind multi-signature scheme BMS must satisfy the standard security properties of *correctness*, *one-more unforgeability* (OMUF), and *blindness*. *Correctness* guarantees that an honest execution of the protocol always produces a valid signature for the intended message. We now formalize the EOMUF, (strong) OMUF and blindness security notions for a two-round BMS scheme in detail.

3.1.1 One-More Unforgeability. Fig. 1 presents the EOMUF and OMUF games, which formally capture the unforgeability of blind

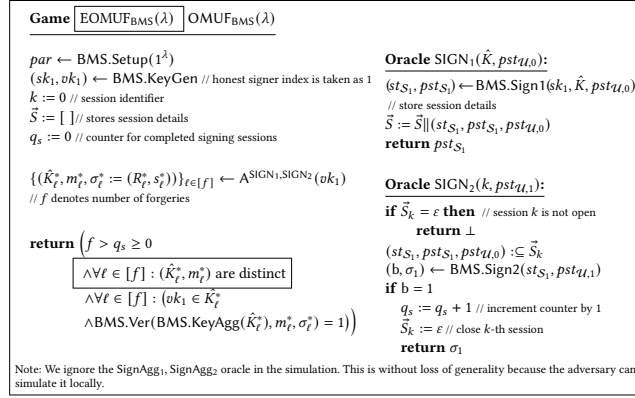


Figure 1: EOMUF and OMUF games for BMS. The solid box denotes the additional condition required only for EOMUF.

multi-signatures. We consider the security in plain public key (PPK) model (In the KOSK model, the adversary must also provide the secret key for any verification key it outputs).

In this game, the adversary A controls all except one honest signer. A key pair (sk_1, vk_1) is generated for the honest signer, and the adversary A is given the verification key vk_1 . The adversary A may initiate an arbitrary number of concurrent signing sessions with the honest signer. Formally, A has access to two signing oracles SIGN_1 and SIGN_2 , which simulate the honest signer's behaviour in the two phases of the interactive signing protocol (Sign1 and Sign2 respectively) using sk_1 .

Finally, the adversary A outputs a non-empty forgery set $\{(\hat{K}_t^*, m_t^*, \sigma_t^* := (R_t^*, s_t^*))\}_{t \in [f]}$. In OMUF, the game returns 1, i.e., A wins if (i) the number of forgeries f is greater than the number of SIGN_2 queries q_s , (ii) each forgery includes the honest signer's verification key vk_1 in the verification key set $\hat{K}_t^*$, and (iii) the forgery is valid, i.e., $\forall \ell \in [f], \text{BMS.Ver}(\text{BMS.KeyAgg}(\hat{K}_t^*), m_t^*, \sigma_t^*) = 1$. In EOMUF, in addition to the above conditions, it is required that (iv) $\forall \ell \in [f], (\hat{K}_t^*, m_t^*)$ are distinct.

Note: The adversary may initiate a signing session via SIGN_1 oracle without ever completing it with SIGN_2 oracle. Consequently, any forgery produced after invoking only SIGN_1 is still valid. Therefore, the total no. of forgeries must exceed the no. of completed SIGN_2 queries, denoted by q_s , since these may correspond to signatures directly generated via honest signer interactions.

Definition 3.1 (One-More Unforgeability (OMUF))³. A blind multi-signature scheme BMS satisfies OMUF if for all adversaries A

$$\text{Adv}_{\text{BMS}, A}^{\text{OMUF}}(\lambda) := \Pr [\text{OMUF}_{\text{BMS}}^A(\lambda)] \leq \text{negl}(\lambda),$$

where OMUF_{BMS} game is defined in Fig. 1.

3.1.2 Blindness. Fig. 2 presents the blindness game for the blind multi-signatures. In this game, the adversary A controls all n corrupted signers and is represented by two algorithms (A_1, A_2) . In the first phase, A_1 outputs a vector of verification keys $\hat{K}$ and two challenge messages (m_0, m_1) . Next, the challenger samples a bit $b \in \{0, 1\}$ and runs two concurrent signing sessions $k \in \{0, 1\}$:

³An analogous definition holds for EOMUF by replacing the OMUF game with the corresponding EOMUF game.

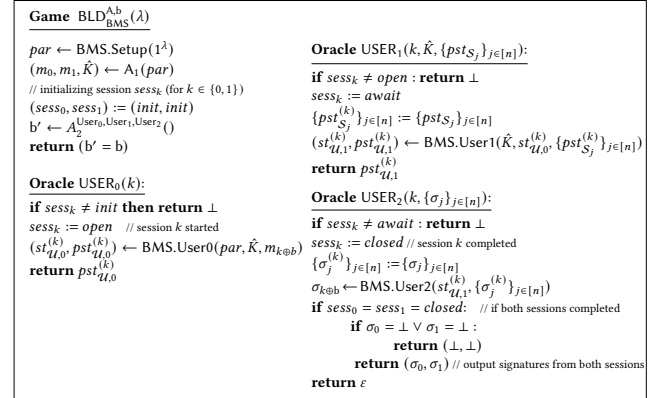


Figure 2: Blindness game for BMS.

session 0 signs m_b and session 1 signs m_{1-b} . The honest user's behaviour in each session is simulated by the oracles USER_0 , USER_1 , and USER_2 . If either session fails, the game outputs $(\perp, \perp)$. Otherwise, once both sessions complete successfully, the adversary A receives the final signatures σ_0 and σ_1 on messages m_0 and m_1 , respectively. At the end, A outputs a guess bit b' and wins the game if $b' = b$. Hence, any non-negligible advantage in guessing b means that A can determine the session corresponding to the signature, thereby violating blindness.

Definition 3.2 (Blindness). A blind multi-signature scheme BMS satisfies blindness if for all adversaries A

$$\text{Adv}_{\text{PBS}, A}^{\text{BLD}}(\lambda) := \Pr [\text{BLD}_{\text{BMS}}^{A,1}(\lambda)] - \Pr [\text{BLD}_{\text{BMS}}^{A,0}(\lambda)] \leq \text{negl}(\lambda),$$

where $\text{BLD}_{\text{BMS}}^{A,b}$ (for $b \in \{0, 1\}$) is the game defined in Fig. 2.

4 Blind Multi-Signature Scheme: BMuSig2

In this section, we present a Schnorr-type blind multi-signature scheme BMuSig2 based on the MuSig2 construction (Sec. 2.6).

Naive (insecure) construction. We begin by describing a naive attempt to construct a BMS scheme by directly incorporating Schnorr-style blindness into MuSig2. We later show that this construction is vulnerable to ROS attack. Let $((\mathbb{G}, q, g), H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}})$ be the public parameters, (x_i, X_i) be the signing-verification key pair of the signer S_i , and $\hat{K} := \{X_i\}_{i \in [n]}$ be the verification key set. The signing algorithm is a 2-round (3-move) interactive protocol between the user $\mathcal{U}$ and the signers $\{S_i\}_{i \in [n]}$ which proceeds as follows.

- $\text{Sign1}(sk_i := x_i, \hat{K}) \rightarrow (st_{S_i}, pst_{S_i})$: Each signer S_i samples $(r_{i,1}, \dots, r_{i,v}) \xleftarrow{\$} \mathbb{Z}_q$ and computes the nonces $(R_{i,1} := g^{r_{i,1}}, \dots, R_{i,v} := g^{r_{i,v}})$. It stores the internal state $st_{S_i} := (r_{i,1}, \dots, r_{i,v})$ and sends $pst_{S_i} := (R_{i,1}, \dots, R_{i,v})$ to the user.
- $\text{User1}(m, \{pst_{S_j}\}_{j \in [n]}) \rightarrow (st_{\mathcal{U},1}, pst_{\mathcal{U},1})$: The user collects the individual nonces $\{(R_{j,1}, \dots, R_{j,v}) : \subseteq pst_{S_j}\}_{j \in [n]}$ from all the signers and computes $\forall t \in [v], R_t := \prod_{j=1}^n R_{j,t}$. It computes key-aggregation coefficients $a_j := H_{\text{agg}}(\hat{K}, X_j)$ ($\forall j \in [n]$), the aggregated verification key $X := \prod_{j=1}^n X_j^{a_j}$, and the nonce coefficient $b' := H_{\text{non}}(X, R_1, \dots, R_v, m)$.

The user samples $\alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_q^3$ and first randomizes the nonce coefficient $b := b' + \gamma$, then forms the aggregated nonce $R :=$

$\prod_{t=1}^v R_t^{b^{t-1}}$, and randomizes it as $R' := g^\alpha X^\beta \cdot R$. It computes the challenge $c' := H_{\text{sig}}(X, R', m)$, $c := c' + \beta$ stores the internal state $st_{\mathcal{U},1} := (\alpha, \beta, \gamma)$, and sends $pst_{\mathcal{U},1} := (R, b, c)$ to all signers.

- $\text{Sign2}(st_{\mathcal{S}_i}, pst_{\mathcal{U},1}) \rightarrow \sigma_i$: Each signer $\mathcal{S}_i$ computes $a_i := H_{\text{agg}}(\hat{K}, X_i)$, and uses its state $(r_{i,1}, \dots, r_{i,v}) \subseteq st_{\mathcal{S}_i}$ and the user's message $(R, b, c) \subseteq pst_{\mathcal{U},1}$ to compute $s_i := ca_i x_i + \sum_{t=1}^v r_{i,t} b^{t-1} \bmod q$. It then sends $\sigma_i := (R, s_i)$ to the user.
- $\text{User2}(st_{\mathcal{U},1}, \{\sigma_j\}_{j \in [n]}) \rightarrow \sigma$: The user takes its internal state $(\alpha, \beta, \gamma) \subseteq st_{\mathcal{U},1}$ and partial signatures $\{(R, s_j) \subseteq \sigma_j\}_{j \in [n]}$ from all the signers as input, computes $s := \sum_{j=1}^n s_j \bmod q$, $s' = s + \alpha$ and outputs the final signature as $\sigma := (R', s')$.

The signature $\sigma := (R', s')$ is valid according to MuSig2.Ver since

$$\begin{aligned} g^{s'} &= g^\alpha \cdot g^{\sum_{j=1}^n (c a_j x_j + \sum_{t=1}^v r_{j,t} b^{t-1})} = g^\alpha \cdot \left(\prod_{j=1}^n (g^{x_j})^{a_j} \right)^c \cdot \prod_{t=1}^v R_t^{b^{t-1}} \\ &= g^\alpha \cdot X^c \cdot R = g^\alpha \cdot X^{c'+\beta} \cdot R \quad (\text{since } c = c' + \beta), \\ &= (g^\alpha X^\beta R) \cdot X^{c'} = R' \cdot X^{c'}. \end{aligned}$$

We now show that this scheme is insecure in the concurrent setting by adapting the attack strategy on blind Schnorr signatures from [11]. Let B be an adversary in the OMUF_{BMS} game that corrupts all but one signer. Without loss of generality, we focus on the case where $n = 2$, signer $\mathcal{S}_1$ is honest and signer $\mathcal{S}_2$ is controlled by the adversary. Let $(sk_1 := x_1, vk_1 := X_1)$ be the honest signer's key pair and $(sk_2 := x_2, vk_2 := X_2)$ be the adversarial signer's key pair. Let $X \leftarrow \text{MuSig2.KeyAgg}(\hat{K} := \{X_1, X_2\})$ be the aggregated verification key.

The adversary B initiates l parallel signing sessions on messages $m_1, m_2, \dots, m_l$ with $\mathcal{S}_1$ by querying $\text{SIGN}_1(\hat{K} := \{X_1, X_2\})$ and receives the corresponding nonces $(R_{1,1}^{(k)} := g^{r_{1,1}^{(k)}}, \dots, R_{1,v}^{(k)} := g^{r_{1,v}^{(k)}})$, for $k \in [l]$. Since B controls $\mathcal{S}_2$, it can freely set its nonce as $(R_{2,1}^{(k)} := g^{r_{2,1}^{(k)}}, \dots, R_{2,v}^{(k)} := g^{r_{2,v}^{(k)}})$. For $k \in [l]$, B sets $\forall t \in [v]$, $R_t^{(k)} := R_{1,t}^{(k)} R_{2,t}^{(k)}$ and computes randomised nonce coefficient $b_k = H_{\text{non}}(X, R_1^{(k)}, \dots, R_v^{(k)}, m_k) + \gamma_k$ where $\gamma_k \in \mathbb{Z}_q$. Finally, it computes the aggregated nonce as $R^{(k)} := \prod_{t=1}^v (R_t^{(k)})^{b_k^{t-1}}$.

Next, the adversary B uses the ROS solver (defined in Sec. 2.3) to find $l+1$ tuples $(\tilde{\rho}_i, aux_i := \alpha_i, \beta_i, m_i^*) \in \mathbb{Z}_q^l \times \mathbb{Z}_q \times \mathbb{Z}_q \times \{0, 1\}^*$ and a vector $(c_1, \dots, c_l) \in \mathbb{Z}_q^l$ such that $\sum_{k=1}^l \rho_{i,k} c_k = H_{\text{ros}}(\rho_i, aux_i) = c_i^* + \beta_i$, $\forall i \in [l+1]$, where $\rho_{i,k}$ denotes the k^{th} component of the vector $\tilde{\rho}_i$, and

$$c_i^* := H_{\text{sig}}\left(X, g^{\alpha_i} X^{\beta_i} \cdot \prod_{k=1}^l (R^{(k)})^{\rho_{i,k}}, m_i^*\right) \bmod q,$$

for some message m_i^* and blinding factors α_i, β_i .

Next, for $k \in [l]$, the adversary B closes the signing session k with $\mathcal{S}_1$ by querying $\text{SIGN}_2(k, (R^{(k)}, b_k, c_k))$ oracle to obtain $s_1^{(k)} := c_k a_1 x_1 + \sum_{t=1}^v r_{1,t}^{(k)} b_k^{t-1}$, where $a_1 := H_{\text{agg}}(\hat{K}, X_1)$. Since B controls $\mathcal{S}_2$, it can compute $s_2^{(k)} := c_k a_2 x_2 + \sum_{t=1}^v r_{2,t}^{(k)} b_k^{t-1}$, where $a_2 := H_{\text{agg}}(\hat{K}, X_2)$ and can compute the signature $(R^{(k)}, s^{(k)})$, where $s^{(k)} := s_1^{(k)} + s_2^{(k)} \bmod q$. Finally, for $i \in [l+1]$, the adversary

outputs $l+1$ forgeries⁴ $(\hat{K}_i^*, m_i^*, \sigma_i^* := (R_i^*, s_i^*))$, where $\forall i \in [l+1]$, $\hat{K}_i^* = \{X_1, X_2\}$ and

$$R_i^* := g^{\alpha_i} X^{\beta_i} \cdot \prod_{k=1}^l (R^{(k)})^{\rho_{i,k}} \text{ and } s_i^* := \sum_{k=1}^l \rho_{i,k} s_k + \alpha_i.$$

It is now easy to verify that for all $i \in [l+1]$, $\sigma_i^* = (R_i^*, s_i^*)$ is a valid signature for m_i^* .

$$\begin{aligned} g^{s_i^*} &= g^{\sum_{k=1}^l \rho_{i,k} \left((c_k a_1 x_1 + \sum_{t=1}^v r_{1,t}^{(k)} b_k^{t-1}) + (c_k a_2 x_2 + \sum_{t=1}^v r_{2,t}^{(k)} b_k^{t-1}) \right) + \alpha_i} \\ &= g^{\alpha_i} \prod_{k=1}^l g^{\rho_{i,k} \left(\sum_{t=1}^v (r_{1,t}^{(k)} + r_{2,t}^{(k)}) b_k^{t-1} \right)} \cdot g^{\left(\sum_{k=1}^l \rho_{i,k} c_k \right) (a_1 x_1 + a_2 x_2)} \\ &= g^{\alpha_i} \prod_{k=1}^l \left(\prod_{t=1}^v (R_t^{(k)})^{b_k^{t-1}} \right)^{\rho_{i,k}} \cdot g^{(c_i^* + \beta_i) (a_1 x_1 + a_2 x_2)} \\ &= (g^{\alpha_i} X^{\beta_i} \prod_{k=1}^l (R^{(k)})^{\rho_{i,k}}) X^{c_i^*} = R_i^* X^{c_i^*}. \end{aligned}$$

There are two main issues with the above construction.

- (1) The main reason for the attack is that, after initiating l concurrent signing sessions and receiving the nonces sent by the honest signer, the adversary can adaptively choose, for each $k \in [l]$, the blinding factors, the messages, the vectors $\tilde{\rho}_k \in \mathbb{Z}_q^l$, and the coefficients $c_1, \dots, c_l \in \mathbb{Z}_q$ using a ROS solver. Thus, instead of following the protocol to compute each challenge honestly, the adversary completes every session with the ROS-derived c_k values and obtains the corresponding signatures $(R^{(k)}, s^{(k)})$, for all $k \in [l]$. Since the adversary chooses the blinding factors and challenges to satisfy the ROS relations, it can then construct forgeries by taking appropriate linear combinations of the individual session signatures using $\tilde{\rho}_i$ vectors. In summary, the attack succeeds because the user can select the blinding factors and messages after observing the signer's nonces, thereby circumventing honest challenge generation.

- (2) Even if the previous issue is addressed, a fundamental problem remains. The EUF-CMA security of MuSig2 in the concurrent setting relies on all signers computing the nonce coefficient b deterministically from public inputs, ensuring each session uses a unique, correctly derived b . In the blind variant, however, a masked b is provided by the user, and the signers cannot verify its correctness. This allows a malicious user to choose b arbitrarily across sessions, enabling attacks analogous to those prevented in standard MuSig2.

Without a mechanism to enforce the correct computation of b , any reduction from the EUF-CMA security of MuSig2 to OMUF security of the blind multisignature fails. Since our construction's security relies on MuSig2's established EUF-CMA security, it is therefore essential that users cannot bias or manipulate b in any signing session.

To achieve concurrent security, we enforce the following:

⁴Since the adversary controls all the other signers, it is straightforward to adapt the attack to generate forgeries with different aggregated verification key as long as the verification-key set contains X_1 . For simplicity, we present all forgeries under a single such set.

- The user first commits to the blinding factors and the message before receiving the signers' nonces, and later proves that the resulting challenge is consistent with the committed values. This prevents the user from adaptively choosing blinding factors and challenges based on the ROS solution.
- The user also proves to the signers that the nonce coefficient b is computed honestly.

4.1 BMuSig2 Construction

We present BMuSig2 to address the limitations of the naive approach. The modified signing protocol is outlined below.

- The protocol begins with the user sending a commitment C of the message m and the blinding factors α, β before receiving the nonces of the signers. Concretely, the user computes $C = \text{PKE}.\text{Enc}(ek, (m, \alpha, \beta, \gamma); \rho)$, where PKE is a CPA secure public-key encryption scheme, ek is the encryption key, and ρ is the randomness used for encryption.
- Each signer S_i responds with their nonce tuple $(R_{i,1}, \dots, R_{i,v})$.
- The user collects nonces $\{(R_{j,1}, \dots, R_{j,v})\}_{j \in [n]}$ from the signers and computes the following similar to naive protocol:
 - for $t \in [v]$, $R_t := \prod_{j \in [n]} R_{j,t}$.
 - nonce coefficient $b := H_{\text{non}}(X, R_1, \dots, R_v, m) + \gamma \bmod q$,
 - aggregated nonce $R := \prod_{t=1}^v R_t^{b^{t-1}}$, and randomized aggregated nonce $R' := R \cdot g^\alpha X^\beta$,
 - $c := H_{\text{sig}}(X, R', m) + \beta \bmod q$,
 and additionally generates a NIZK argument π proving the consistency of b and c with the committed values. For this NIZK argument NArg, we formally define the following parameterized relation R_{BMuSig2} with parameters $par_R := (q, \mathbb{G}, g, H_{\text{non}}, H_{\text{sig}})$.

$$R_{\text{BMuSig2}}(par_R, \theta := (X, R_1, \dots, R_v, b, R, c, C, ek), w := (m, \alpha, \beta, \gamma, \rho)) :$$

$$\begin{aligned} R' &:= R \cdot g^\alpha X^\beta // R' \text{ is the blinded form of } R \\ \wedge b &\equiv_q H_{\text{non}}(X, R_1, \dots, R_v, m) + \gamma // b \text{ is computed correctly} \\ \wedge c &\equiv_q H_{\text{sig}}(X, R', m) + \beta // c \text{ is computed correctly} \\ \wedge \text{PKE}.\text{Enc}(ek, (m, \alpha, \beta); \rho) &= C // C \text{ encrypts witnesses under } ek \end{aligned} \quad (1)$$

The user sends $(R_1, \dots, R_v, b, c, \pi)$ to all the signers.

- Each signer S_i verifies π , computes their partial signature $s_i := ca_i x_i + \sum_{t=1}^v r_{i,t} b^{t-1} \bmod q$ and returns s_i to the user.
- The user collects all the partial signatures $\{s_j\}_{j \in [n]}$, computes $s = \sum_{j=1}^n s_j$, and outputs the final aggregate signature $\sigma := (R', s' = s + \alpha)$.

BMuSig2 combines three core components—MuSig2 multisignatures, a public key encryption scheme PKE, and a NIZK argument system NArg. Formally, we define BMuSig2 as a tuple of five PPT algorithms: (Setup, KeyGen, KeyAgg, Sign, Ver). Fig. 3 presents the complete protocol as an interactive execution between the user $\mathcal{U}$ and the signers $\{S_j\}_{j \in [n]}$.

4.2 Security of BMuSig2

4.2.1 One-More Unforgeability of BMuSig2. We prove the unforgeability of BMuSig2 (Fig. 3) under two instantiations. The two-nonce ($v = 2$) variant achieves EOMUF security assuming the soundness

of the underlying NIZK proof system and the EUF-CMA security of MuSig2, whereas the four-nonce variant ($v = 4$) achieves (strong) OMUF security assuming the soundness of the NIZK proof system and the SUF-CMA security of MuSig2.

THEOREM 4.1. ⁵ *Let GGen and HGen be a group and hash-function generation algorithms, respectively; PKE be a public-key encryption scheme; MuSig2[GGen, HGen] be the multi-signature scheme instantiated with GGen and HGen; and let NArg[R_{BMuSig2}] be a non-interactive argument system for the relation R_{BMuSig2} . Then, for any adversary A playing Game OMUF (defined in Fig. 8) against the scheme BMuSig2[GGen, HGen, PKE, NArg], and completing at most q_s signing sessions via the SIGN_2 oracle, there exist PPT algorithms U and S such that*

- U is an adversary in the SUF-CMA game against the unforgeability of MuSig2[GGen, HGen], and
- S is an adversary in the SND game against the soundness of NArg[R_{BMuSig2}],

$$\text{Adv}_{\text{BMuSig2}, A}^{\text{OMUF}}(\lambda) \leq \text{Adv}_{\text{MuSig2}[\text{GGen}, \text{HGen}], U}^{\text{SUF-CMA}}(\lambda) + \text{Adv}_{\text{NArg}[\text{R}_{\text{BMuSig2}}], S}^{\text{SND}}(\lambda).$$

PROOF SKETCH. The detailed proof of Theorem 4.1 is given in Appendix C. Here we summarize the main ideas.

At a high level, the goal of the proof is to reduce the OMUF of BMuSig2 to the SUF-CMA security of MuSig2. Formally, the proof proceeds via a sequence of games hops defined in Fig. 8. We begin with Game G_0 , which is the original OMUF game $\text{OMUF}_{\text{BMuSig2}}^A$ played with adversary A . In Game G_1 , the challenger decrypts each ciphertext C submitted by A to obtain (m, α, β) and checks their consistency with the challenge c and nonce coefficient b . If any checks fail, the game aborts. Using the perfect correctness of the PKE scheme, we show that any adversary triggering an abort can be used to construct an adversary that breaks the soundness of NArg[R_{BMuSig2}]. Finally, in Game G_2 , we show that any adversary that still wins the OMUF game must necessarily produce a valid forgery for MuSig2, thereby violating the SUF-CMA security of MuSig2. Concretely, given the honest signer's verification key $vk_1 = X_1$, the reduction samples crs and ek honestly, and simulates the signing oracles for the OMUF adversary A . For each signing session initiated by A with ciphertext C , the reduction decrypts C to obtain (m, α, β) , queries its own MuSig2 signing oracle to obtain a signature $(\bar{R}, \bar{s})$ on m , and returns the simulated response $(R := \bar{R} \cdot g^{-\alpha} X^{-\beta}, s := \bar{s} - \alpha)$ to A .

OMUF security. If A is an OMUF adversary that outputs f valid BMuSig2 forgeries $\{(\hat{K}_\ell, m_\ell^*, \sigma_\ell^*)\}_{\ell \in [f]}$, with f strictly greater than the number of completed signing sessions; then there exists a tuple $(\hat{K}^*, m^*)$ for which the number of forged signatures f' exceeds the number of signing queries made on $(\hat{K}^*, m^*)$. Hence, at least one of the f' signatures was not obtained from the signing oracle. Therefore all the f' forgeries containing this tuple: $(\hat{K}^*, m^*, (\sigma_{\ell'}^*)_{\ell' \in [f']})$ constitutes a successful forgery for MuSig2 under SUF-CMA. $\square$

Note: By a similar argument, if A is an EOMUF adversary that outputs f valid BMuSig2 forgeries $\{(\hat{K}_\ell, m_\ell^*, \sigma_\ell^*)\}_{\ell \in [f]}$, with f strictly greater than the number of completed signing sessions; then there exists at

⁵An analogous theorem holds for EOMUF security by replacing the OMUF game with the EOMUF game and the SUF-CMA game of MuSig2 with its EUF-CMA game.

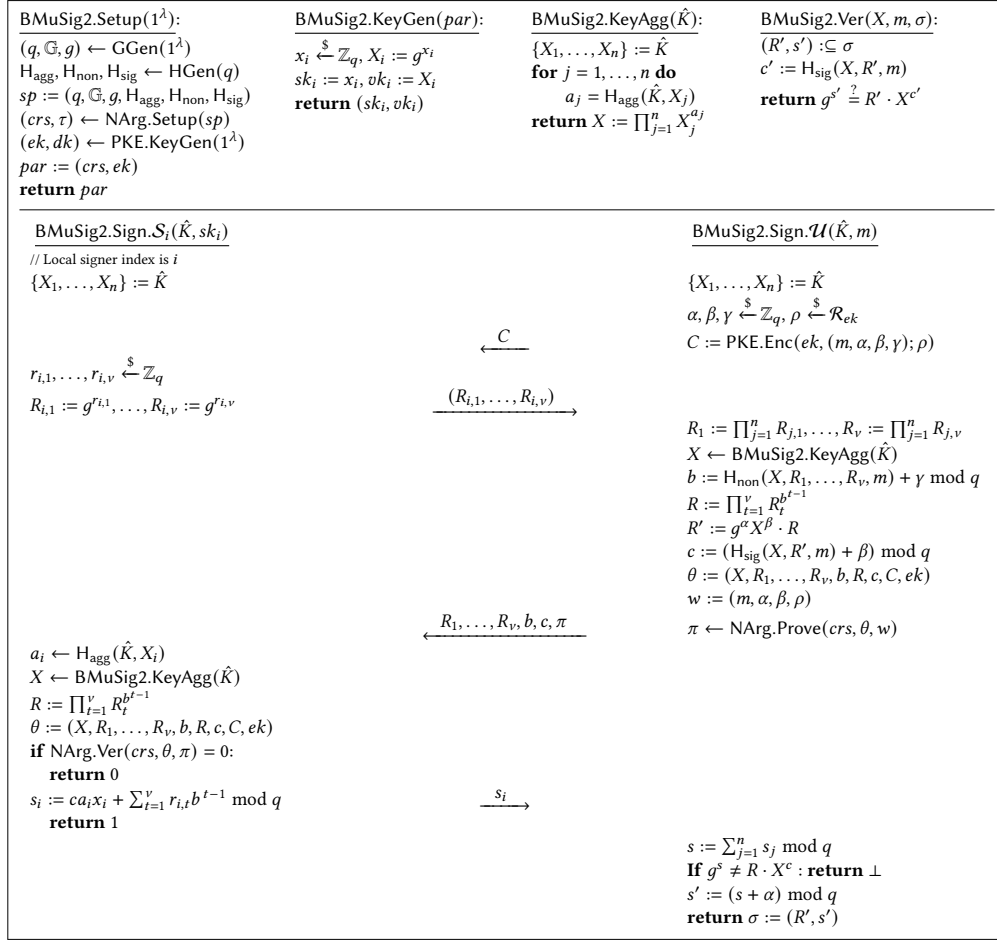


Figure 3: BMuSig2: Blind Multi-Signature scheme

least one tuple $(\hat{K}^*, m^*, \sigma^*)$ in this set that was never produced by the signing oracle, and therefore constitutes a successful forgery for MuSig2 under EUF-CMA.

4.2.2 Blindness of BMuSig2. We prove that BMuSig2 (Fig. 3), satisfies blindness as defined in Def.3.2. The proof relies on the reductions to the security of its underlying building blocks—the zero-knowledge property of NArg and the CPA-security of PKE scheme. The argument applies identically to both nonce variants ($v \in \{2, 4\}$), as only the number of nonces differs and this does not affect the signer’s view or the indistinguishability analysis.

THEOREM 4.2. *Let GGen and HGen be the group and hash generation algorithms respectively; PKE be a public-key encryption scheme; and NArg[R_{BMuSig2}] be an argument system for the relation R_{BMuSig2} . Then for any adversary A playing Game BLD (defined in Fig. 11) against the scheme BMuSig2[GGen, HGen, PKE, NArg], there exist PPT adversary algorithms*

- Z_0, Z_1 against the zero-knowledge of NArg[R_{BMuSig2}], and
- C_0, C_1 against the CPA-security of PKE,

$$\begin{aligned} \text{Adv}_{\text{BMuSig2}, A}^{\text{BLD}}(\lambda) &\leq \text{Adv}_{\text{NArg}[R_{\text{BMuSig2}}], Z_0}^{\text{ZK}}(\lambda) + \text{Adv}_{\text{NArg}[R_{\text{BMuSig2}}], Z_1}^{\text{ZK}}(\lambda) \\ &\quad + \text{Adv}_{\text{PKE}, C_0}^{\text{CPA}}(\lambda) + \text{Adv}_{\text{PKE}, C_1}^{\text{CPA}}(\lambda). \end{aligned}$$

PROOF SKETCH. The detailed proof of Theorem 4.2 is given in Appendix D. Here we summarize the main ideas.

The proof proceeds with a sequence of games hops defined in Fig. 11. We start with Game G_0 , which is the original blindness game $\text{BLD}_{\text{BMuSig2}}^{A, b}$ played with adversary A, for an arbitrarily fixed $b \in \{0, 1\}$. In the game G_1 , we replace the user’s proofs π (in both signing sessions) by simulated proofs. The zero-knowledge property of NArg[R_{BMuSig2}] ensures that the adversary’s view in G_1 and G_0 is computationally indistinguishable. In Game G_2 , we replace the user’s ciphertexts C (in both the sessions) with encryptions of a fixed message. The CPA security of the PKE scheme guarantees that the adversary’s view in G_1 and G_2 is computationally indistinguishable. Finally, for game G_2 , we show that the adversary’s view is independent of bit b , which completes the proof. $\square$

5 Implementation and Evaluations

We present a proof-of-concept (PoC) implementation⁶ of the four-nonce, OMUF-secure BMuSig2 variant in Rust. Inspired by [17], we focus on the decentralized issuance of anonymous tokens using blind multisignatures. The setting has a set of issuers (signers), a user, and a service provider (verifier). The issuers provide an anonymous token to a user whose identity is typically known to them during issuance. The user can later present this token to a service provider, who can authenticate its validity and provide services to the user. Blockchain applications can issue these tokens off-chain while supporting on-chain verification. To demonstrate practicality, we implemented a token verification smart contract in Solidity. We also compare the performance of BMuSig2 with the existing blind multisignature BM_SB from [17].

We implemented and benchmarked the results on an x86_64 Linux workstation equipped with an Intel(R) Xeon(R) W-2133 CPU (3.60 GHz, 6 cores / 12 threads) and 32 GiB RAM, running Linux Mint 22.2 (kernel 6.8.0-87-generic). The software stack consists of Rust 1.88, Circom 2.1.x for circuit compilation—with a secp256k1 variant via circom-secp—and the Arkworks ecosystem (v0.5.0) for cryptographic primitives. For on-chain verification, we rely on Solidity smart contracts compiled using Foundry. All benchmarks were executed over 10 independent trials, and we report average values across these runs. Rust co-routines were used to simulate communication between the user and the issuers, thereby excluding network latency. The issuance costs consider both the user’s computation and the aggregate computation done by all the issuers.

We adopt two instantiation scenarios defined by Fuchsbauer et al. [12] to set the parameters of our BMuSig2 scheme—A) a setup in which the group and hash function are chosen to optimize compatibility and efficiency with the NArg system, and B) a blockchain-oriented setup that extends standard Schnorr signature groups and hash functions, such as those used in Bitcoin, to the multisigner setting. Following [12], we benchmark the computationally expensive component, NArg, using Circom circuits extended from their implementation, and refer to their work for the complete NArg instantiation details. As in their instantiation, scenario A uses the trusted-setup zk-SNARK Groth16 [13], while scenario B uses the transparent-setup NIZK Spartan [28]; the PKE component is instantiated with DHIES [2] key encapsulation of ElGamal [10] with an additive one-time pad for the data encapsulation.

Table 2 summarizes the parameter instantiations for BMuSig2 and the baseline BM_SB scheme [17]. For BMuSig2, scenario A uses the *BabyJubJub* (BJB) curve with the signature hash H_{sig} instantiated using Poseidon and the proving system instantiated using Groth16, while scenario B uses the secp256k1 curve with H_{sig} instantiated as SHA-256 and Spartan as the proving system. In both the scenarios, H_{non} is instantiated as Poseidon to optimize NArg, and H_{agg} is instantiated to match H_{sig} for consistency. BM_SB is instantiated only with secp256k1 and SHA-256, as it does not rely on NIZKs or curve-specific optimizations, and a single instantiation suffices for evaluation.

Table 3 compares the NIZK performance for the two BMuSig2 instantiations. BMuSig2 (A) is much more efficient with $\approx 7 \times$ fewer

Table 2: Instantiations for BMuSig2 and BM_SB [17]

Parameter	BMuSig2		BM_SB [17]
	A	B	
Signature Curve	BJB	secp256k1	secp256k1
Signature Hash	Poseidon	SHA-256	SHA-256
Proving System	Groth16	Spartan	—

Table 3: NIZK Performance of BMuSig2 (A/B)

Metric	BMuSig2 (A)	BMuSig2 (B)
No. of constraints	20031	131072
Proof generation time	3.22 s	72.62 s
Proof verification time	54.41 ms	2.81 s
Proof size (compressed)	128 B	27395 B

Table 4: Token and verification key sizes, assuming n issuers.

Scheme	Token Size			Verification Key Size		
	$\mathbb{G}$	$\mathbb{Z}_q$	Total (Bytes)	$\mathbb{G}$	$\mathbb{Z}_q$	Total (Bytes)
BMuSig2	1	1	96	1	0	64
BM_SB [17]	1	2	128	n	0	$64n$

Table 5: Communication overhead for token issuance between the user and a single issuer as a function of n issuers.

Scheme	Round	$\mathcal{U} \rightarrow \mathcal{S}$			$\mathcal{S} \rightarrow \mathcal{U}$		Bytes Exchanged	Total
		$\mathbb{G}$	$\mathbb{Z}_q$	π	$\mathbb{G}$	$\mathbb{Z}_q$		
BMuSig2	1	1	1	0	4	0	352	$704 + \pi$
	2	4	2	1	0	1	$352 + \pi$	
BM_SB [17]	1	0	n	NA	2	1	$32n + 160$	$64n + 224$
	2	0	$n - 1$	NA	0	2	$32n + 32$	
	3	0	0	NA	0	1	32	

constraints. It achieves significantly faster proof generation and verification, with a more compact proof size than BMuSig2 (B).

We now analyze token (signature) sizes, verification key sizes, and communication costs. Throughout this section, we let n denote the number of issuers. Each field element is 32 bytes, and each group element consists of two field elements, resulting in 64 bytes.

Table 4 shows the sizes of tokens and their corresponding verification key/keys. A BMuSig2 token $\sigma = (R, s) \in \mathbb{G} \times \mathbb{Z}_q$ is 96 bytes, compared to 128 bytes for a BM_SB token $\sigma = (R, y, z) \in \mathbb{G} \times \mathbb{Z}_q \times \mathbb{Z}_q$. For signature verification, BMuSig2 requires only a single aggregated verification key—a group element of size 64 bytes, whereas for BM_SB, it scales with the number of issuers, n . Moreover, the verification keys in BM_SB must also be accompanied by their corresponding proofs of possession (not included in the table).

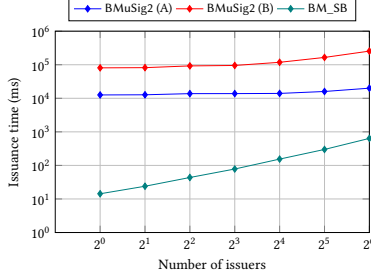
Table 5 summarizes the communication overhead between a user and a single issuer as a function of n . The two schemes differ primarily in how the total communication scales with n . We write $\mathcal{U} \rightarrow \mathcal{S}$ and $\mathcal{S} \rightarrow \mathcal{U}$ to denote the data sent from the user to the issuer and vice versa. For BM_SB, the communication grows linearly in n , i.e., $64n + 224$ bytes. This makes BM_SB efficient for small issuer sets, but increasingly costly as n grows. In contrast, the communication cost of BMuSig2 is independent of n , and is given by $704 + \pi$ bytes, where π denotes the size of the NIZK proof.

In our implementations, group elements are compressed. On the secp256k1 curve, a single group element compresses to 33 bytes, while on the BJB curve it compresses to 32 bytes. Consequently, the communication overhead for BM_SB is $64n + 162$ bytes, while

⁶<https://github.com/shivamsethi98/bmusig2>

Table 6: Execution times for issuance and verification of a single token issued by 8 issuers.

Metric	BMuSig2		BM_SB [17]
	A	B	
Issuance	13.80 s	95.21 s	0.079 s
Verification	4.51 ms	3.22 ms	14.60 ms
Token size (compressed)	64 B	65 B	97 B

**Figure 4: Execution times for issuance of a single token.**

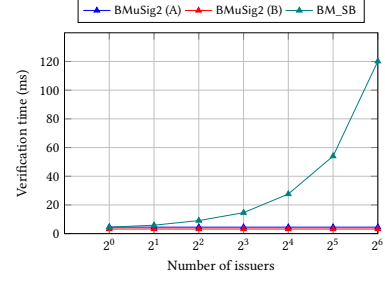
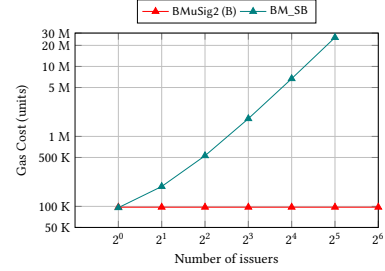
for BMuSig2 it is $416 + \pi$ bytes for instantiation A ($\pi = 128$ B) and $425 + \pi$ bytes for instantiation B ($\pi \approx 27$ kB). BMuSig2 (A) becomes more bandwidth-efficient than BM_SB for issuer sets of size, $n \geq 6$. In contrast, BM_SB is more communication-efficient than BMuSig2 (B) for small issuer sets (roughly $n \leq 432$). Overall, BM_SB is preferable for small to moderate n , whereas BMuSig2 provides an n -independent alternative that is better for compact proofs or when the system has many issuers.

Table 6 compares issuance and verification time and token size of BMuSig2 (A and B) with BM_SB w.r.t. 8 issuers. BMuSig2 has a high issuance cost (13.80 s and 95.21 s) as expected due to the NIZK component, but verifies faster (≈ 3 –4 ms) and produces smaller tokens (64–65 bytes) compared to BM_SB, which requires 14.60 ms for verification and produces tokens of size 97 bytes. Overall, BMuSig2 provides compact tokens and efficient verification at the cost of slower issuance.

Fig. 4 compares the issuance times of a single token for BMuSig2 (A and B) and BM_SB versus the number of issuers. BMuSig2 (A and B) exhibit high latencies (12.59–254.8 s), scaling poorly due to the overhead of NIZK proof generation and verification. In contrast, BM_SB verification scales roughly proportionally with the no. of issuers and is significantly faster, reaching only 0.6 s for 64 issuers.

Fig. 5 compares the verification times of a single token for BMuSig2 (A and B) and BM_SB w.r.t the number of issuers n . Both instantiations of BMuSig2 (A and B) exhibit constant verification times (4.51 ms and 3.22 ms, respectively), and is independent of n . In contrast, verification time of BM_SB increases with n , reaching 120.15 ms for 64 issuers. This indicates that BMuSig2 provides efficient and scalable verification.

Figure 6 compares the Ethereum gas cost for on-chain token verification as a function of the number of issuers n . Since the BJB curve is not natively supported on Ethereum, we report only the verification costs for BMuSig2 (B) and BM_SB over the secp256k1 curve. The gas cost for BMuSig2 (B) remains approximately constant at 97K units—independent of n —as verification relies on a single aggregated verification key. In contrast, the cost of BM_SB

**Figure 5: Execution times for verification of a single token.****Figure 6: Ethereum gas cost for token verification.**

grows sharply with n , reaching ≈ 26 M units for 32 issuers and it exceeds Ethereum's 30M block gas limit for larger issuer sets. For a representative case of 8 issuers, BMuSig2 (B) requires ≈ 97 K gas whereas BM_SB requires ≈ 529 K gas. Consequently, within a 30M gas block, BMuSig2 (B) supports roughly 308 token verifications, while BM_SB supports only about 56 token verifications.

Acknowledgments

We are grateful to Prof. Santanu Sarkar (IIT Madras) for pointing out that the blindness property does not hold (in the published version) due to the deterministic nonce coefficient b . We have resolved this issue in the revised version by masking b , while keeping the overall construction unchanged.

References

- [1] [n. d.]. Private State Tokens | Privacy Sandbox — [privacysandbox.google.com](https://privacysandbox.google.com/protectations/private-state-tokens). <https://privacysandbox.google.com/protectations/private-state-tokens>. [Accessed 11-11-2025].
- [2] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. 1999. DHAES: An Encryption Scheme Based on the Diffie-Hellman Problem. *IACR Cryptol. ePrint Arch.* 1999 (1999), 7.
- [3] Mihir Bellare and Gregory Neven. 2006. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and Communications Security*. 390–399.
- [4] Fabrice Benhamouda, Tancrede Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. 2022. On the (in) security of ROS. *Journal of Cryptology* 35, 4 (2022), 25.
- [5] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *International conference on the theory and application of cryptography and information security (ASIACRYPT 2001)*. Springer, 514–532.
- [6] David Chaum. 1983. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto 82*. Springer, 199–203.
- [7] David Chaum. 1988. Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA. In *Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 177–182.
- [8] David Chaum and Torben Pridy Pedersen. 1992. Wallet databases with observers. In *Annual International Cryptology Conference*. Springer, 89–105.

- [9] Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. 2023. Snowblind: A threshold blind signature in pairing-free groups. In *Annual International Cryptology Conference*. Springer, 710–742.
- [10] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.
- [11] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. 2020. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 63–95.
- [12] Georg Fuchsbauer and Mathias Wolf. 2024. Concurrently secure blind schnorr signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 124–160.
- [13] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 305–326.
- [14] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2017. Tumblebit: An untrusted Bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*.
- [15] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. 2016. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In *International Conference on Financial Cryptography and Data Security*. Springer, 43–60.
- [16] Scott Hendrickson, Jana Iyengar, Tommy Pauly, Steven Valdez, and Christopher A Wood. 2021. *Private access tokens*. Technical Report. Internet-Draft draft-private-access-tokens-01, Internet Engineering Task Force.
- [17] Ioanna Karantaidou, Omar Renawi, Foteini Baldimtsi, Nikolaos Kamarinakis, Jonathan Katz, and Julian Loss. 2024. Blind multisignatures for anonymous tokens with decentralized issuance. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 1508–1522.
- [18] Susan K Langford. 1996. Weaknesses in some threshold cryptosystems. In *Annual International Cryptology Conference*. Springer, 74–82.
- [19] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. 2006. Sequential aggregate signatures and multisignatures without random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 465–485.
- [20] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. 2019. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography* 87, 9 (2019), 2139–2164.
- [21] Sela Navot and Stefano Tessaro. 2024. One-more unforgeability for multi- and threshold signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 429–462.
- [22] Gregory Neven. 2011. IBM identity mixer (idemix). In *NIST Meeting on Privacy Enhancing Technology, Zurich, Switzerland*. 8–9.
- [23] Jonas Nick, Tim Ruffing, and Yannick Seurin. 2021. MuSig2: Simple two-round Schnorr multi-signatures. In *Annual International Cryptology Conference*. Springer, 189–221.
- [24] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. 2020. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1717–1731.
- [25] Christian Paquin. 2011. U-prove technology overview v1. 1. *Microsoft Corporation Draft Revision 1* (2011), 13.
- [26] Claus-Peter Schnorr. 1989. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*. Springer, 239–252.
- [27] Claus Peter Schnorr. 2001. Security of blind discrete log signatures against interactive attacks. In *International Conference on Information and Communications Security*. Springer, 1–12.
- [28] Srinath Setty. 2020. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*. Springer, 704–737.
- [29] Stefano Tessaro and Chenzhi Zhu. 2022. Short pairing-free blind signatures with exponential security. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 782–811.
- [30] David Wagner. 2002. A generalized birthday problem. In *Annual International Cryptology Conference*. Springer, 288–304.
- [31] Xiaoyang Wei, Shuai Han, and Shengli Liu. 2025. Optimized Privacy-Preserving Multi-signatures from Discrete Logarithm Assumption. In *European Symposium on Research in Computer Security*. Springer, 379–399.
- [32] Pieter Wuille, Jonas Nick, and Tim Ruffing. 2020. BIP 340: Schnorr signatures for secp256k1. *GitHub*. Jan 19 (2020).

A Syntax of Blind Signature

A blind signature scheme BS consists of following PPT algorithms:

- BS.Setup(1^λ) $\rightarrow$ (par): On input security parameter λ , outputs the public parameters par .

- BS.KeyGen(par) $\rightarrow$ (sk, vk): Generates a signing–verification key pair (sk, vk) for the signer.
- BS.Sign($(\mathcal{U}(m, vk), \mathcal{S}(sk))$) $\rightarrow \sigma/\perp$: An interactive signing protocol between a user $\mathcal{U}$ with message m and vk , and a signer $\mathcal{S}$ with sk . The protocol ends with a signature σ for user on success, and $\perp$ otherwise.
- BS.Ver(vk, m, σ) $\rightarrow \{0, 1\}$: Run by a verifier; outputs 1 if σ is a valid signature for m under vk .

B Unforgeability games for MuSig2

Fig. 7 presents the EUF-CMA and SUF-CMA games that formally captures the *unforgeability* of MuSig2. We consider an adversary $\mathcal{U}$ to have control over all except one honest signer. A key pair (sk_1, vk_1) is generated for the honest signer, and $\mathcal{U}$ is given the verification key vk_1 . $\mathcal{U}$ may initiate an arbitrary number of (concurrent) signing sessions with the honest signer. Formally, $\mathcal{U}$ has access to the MSIGN₁ and MSIGN₂ oracles, which simulate the Sign₁ and Sign₂ oracles of MuSig2 using the honest signer’s signing key sk_1 . $\mathcal{U}$ may query the MSIGN₁ oracle with the same verification key set $\hat{K}$ containing vk_1 , as well as the same message m to multiple MSIGN₁ queries, thereby obtaining signing sessions in which the honest signer participates multiple times. The MSIGN₂ oracle expects as input an aggregate value pst , purported to be the aggregation of all signers’ outputs from the previous round. This delegates the computation of SignAgg₁ to the adversary, reflecting that any aggregator (if present) is considered untrusted. Consequently, we do not provide explicit oracles for SignAgg₁ or SignAgg₂, as they involve no secret inputs and can be executed locally by $\mathcal{U}$.

C One-More Unforgeability of BMuSig2

This section gives the full proof of Theorem 4.1. We prove the one-more unforgeability by considering two variants of the notion: OMUF and EOMUF. We present the proof in detail for the OMUF variant (where $v = 4$) and additionally indicate (as a note) how the argument differs for the EOMUF variant (where $v = 2$).

PROOF. We proceed by a sequence of games specified in Fig. 8.

Game G_0 : This is the original OMUF_{BMuSig2} game from Fig. 8 (without the colored boxes). The system parameters and keys are generated using BMuSig.Setup and BMuSig.Keygen. The adversary $\mathcal{A}$ is given the verification key of an honest signer, $vk_1 := X_1$, and has access to two oracles, SIGN₁ and SIGN₂. $\mathcal{A}$ may concurrently engage in multiple signing sessions to obtain blind multi-signatures on messages of its choice. At the end of the game, $\mathcal{A}$ outputs a set of f forgeries $\{(\hat{K}_\ell^*, m_\ell^*, \sigma_\ell^* := (R_\ell^*, s_\ell^*))\}_{\ell \in [f]}$. $\mathcal{A}$ wins if: (i) $f > q_s \geq 0$, where q_s is the number sessions completed via SIGN₂ oracle queries; (ii) for every $\ell \in [f]$, vk_1 is included in $\hat{K}_\ell^*$; and (iii) all forgeries are valid, i.e., $\forall \ell \in [f]$, $\text{BMuSig2.Ver}(\text{BMuSig2.KeyAgg}(\hat{K}_\ell^*), m_\ell^*, \sigma_\ell^*) = 1$.

Game G_1 : In game G_1 , we construct a reduction from the soundness of NArg. We introduce three lists $\vec{m}$, $\vec{\alpha}$, $\vec{\beta}$ and $\vec{\gamma}$. We modify the SIGN₁ oracle so that, on input $(\hat{K}, C)$, it decrypts C to obtain (m, α, β) and appends these values to the respective lists.

On each call to the SIGN₂ oracle with input $(k, \hat{K}, pst, b, c, \pi)$, we check if for the decrypted values at index k , the event E_1 occurs, i.e. $c \neq H_{\text{sig}}(X, R', \vec{m}_k) + \vec{\beta}_k \bmod q$ or $b \neq H_{\text{non}}(X, R'_1, \dots, R'_v, \vec{m}_k) + \vec{\gamma}_k$, where $R' := R \cdot g^{\vec{\alpha}_k X \vec{\beta}_k}$, $R'_1 := R_1 \cdot g^{\vec{\alpha}_k X \vec{\beta}_k}$, and $R'_t := R_t$ ($\forall t \in$

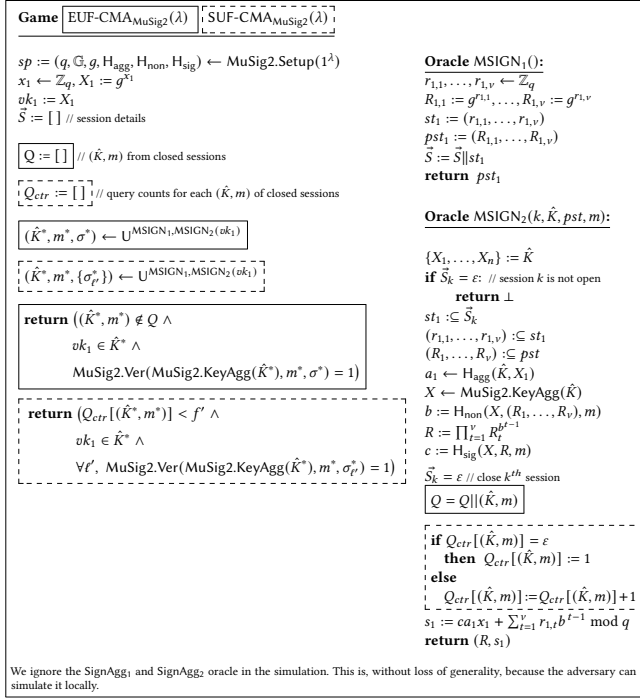


Figure 7: Unforgeability games for MuSig2. EUF-CMA contains all but the dashed box; SUF-CMA contains all but the solid box.

$\{2, \dots, v\}$). In this case, we abort the game and return 0.

Reduction from the Soundness of NArg. We show that the difference between $\text{Adv}_{\text{B MuSig2}, A}^{\text{OMUF}}(\lambda)$ and $\text{Adv}_A^{G_1}(\lambda)$ is bounded by the advantage of an adversary S in winning the soundness game SND against $\text{NArg}[\text{R}_{\text{B MuSig2}}]$, where S outputs the statement/proof pair whenever G_1 aborts (Fig. 9). Concretely,

$$\Pr[G_0(A)] \leq \Pr[G_1(A)] + \text{Adv}_{\text{NArg}, S}^{\text{SND}}(\lambda).$$

PROOF. By the definition of the SND game for NArg over $\text{R}_{\text{B MuSig2}}$, the reduction S receives $crs \leftarrow \text{NArg.Setup}(par_R)$ with par_R generated by MuSig2.Setup , and thus perfectly simulates G_1 for A until abort. In SIGN_2 , S checks whether, for a valid statement/proof pair (θ, π) , the extracted (m, α, β) violates $c \equiv H_{\text{sig}}(X, R', m) + \beta \bmod q$ or $b \equiv H_{\text{non}}(X, R'_1, \dots, R'_v, m) + \gamma \bmod q$, and returns (θ, π) if either check fails. We show that when this happens, S wins the SND game.

Let S reaches E_1 on a call to SIGN_2 with input $(k, \hat{K}, pst, b, c, \pi)$ and $(\theta := (X, R_1, \dots, R_v, b, R, c, ek), \pi)$. Since π is accepting for θ , it suffices to show that θ is not a valid statement. Hence, S successfully breaks the soundness of NArg. We argue this by contradiction. Assume θ is valid, then there exists a witness $w' = (m', \alpha', \beta', \gamma', \rho')$ s.t. $\text{R}_{\text{B MuSig2}}(par_R, \theta, w') = 1$. Therefore,

$$b \equiv_q H_{\text{non}}(X, R'_1, \dots, R'_v, m') + \gamma' \wedge c \equiv_q H_{\text{sig}}(X, R', m') + \beta' \wedge \text{PKE.Enc}(ek, (m', \alpha', \beta', \gamma'); \rho') = C. \quad (2)$$

By perfect correctness of PKE together with the first clause it implies that $m' = m$, $\alpha' = \alpha$, $\beta' = \beta$ and $\gamma' = \gamma$. But by the definition

of S , we have $c \not\equiv_q H_{\text{sig}}(X, R', m) + \beta$ or $b \not\equiv_q H_{\text{non}}(X, R'_1, \dots, R'_v, m) + \gamma$, which is a contradiction to Eqn. 2. This means that whenever E_1 occurs in G_1 then S wins the SND game and hence

$$\text{Adv}_{\text{B MuSig2}, A}^{\text{OMUF}}(\lambda) \leq \text{Adv}_{\text{NArg}[\text{R}_{\text{B MuSig2}}], S}^{\text{SND}}(\lambda) + \Pr[G_1^A(\lambda) = 1]. \quad (3)$$

□

Game G_2 : For game G_2 , we construct a reduction from the SUF-CMA security of MuSig2 scheme.

First, we introduce a counter list Q_{ctr} that records, for each tuple $(\hat{K}, m)$, the number of completed signing queries. We then modify the SIGN_1 oracle as follows. After decrypting the ciphertext C and obtaining $(m, \alpha, \beta, \gamma)$, the oracle calls MuSig2.Sign_1 to obtain $pst_1 = (\bar{R}_{1,1}, \dots, \bar{R}_{1,v})$. Next, we set the random nonces contributed by the honest signer as $R_{1,1} := \bar{R}_{1,1} \cdot g^{-\alpha} X^{-\beta}$ and $R_{1,t} := \bar{R}_{1,t}$ ($\forall t \in \{2, \dots, v\}$). Since $(\bar{R}_{1,1}, \dots, \bar{R}_{1,v})$ returned by MuSig2.Sign_1 are uniformly random group elements, it follows that the resulting tuple $pst_1 = (R_{1,1}, \dots, R_{1,v})$ is also uniformly distributed. Thus, the simulation up to this point is perfect. We also modify the SIGN_2 oracle. The oracle invokes MuSig2.Sign_2 on input $(\hat{K}, \tilde{s}_1, pst, sk_1, m)$, to obtain $(\bar{R}, \bar{s}_1)$. It then computes $s_1 := (\bar{s}_1 - \alpha) \bmod q$ and returns s_1 as the response. Additionally, we update the count of queries completed for $(\hat{K}, m)$ in $Q_{ctr}[(\hat{K}, m)]$ to record that MuSig2.Sign_2 was queried for $(\hat{K}, m)$. This ensures that in the final reduction to the SUF-CMA security of MuSig2, the adversary's forgery does not correspond to the signatures that the reduction itself previously obtained from the MSIGN_2 oracle.

In game G_2 , the adversary receives simulated values $(R_{1,1}, \dots, R_{1,v})$ and s_1 , which are distributed exactly as in G_1 . Hence, the adversary's view is indistinguishable from that in G_1 , and thus

$$\Pr[G_2^A(\lambda)] = \Pr[G_1^A(\lambda)]. \quad (4)$$

Reduction from the SUF-CMA security of MuSig2. Finally, we construct an adversary U (Fig. 10) that succeeds in winning the SUF-CMA game against the $\text{MuSig2}[\text{GGen}, \text{HGen}]$ scheme with probability $\Pr[G_2^A(\lambda)]$. By the definition of SUF-CMA of MuSig2, U receives the public parameters sp and the honest signer verification key $X_1 := vk$ as input and has access to the signing oracles MSIGN_1 and MSIGN_2 . U uses sp and X_1 to complete the setup of B MuSig2 . Note that to simulate G_2 for A , U embeds its challenge verification key X_1 into the verification key for B MuSig2 . The corresponding secret key is not required since U on each SIGN_1 and SIGN_2 query by A forwards the call to its signing oracles MSIGN_1 and MSIGN_2 , respectively. Therefore, the simulation is perfect.

We argue that if the OMUF adversary A wins G_2 by outputting the forgery set $F := \{(\hat{K}_\ell^*, m_\ell^*, \sigma_\ell^*)\}_{\ell \in [f]}$, then it must contain at least one valid forgery for SUF-CMA adversary U . We know that for a valid OMUF forgery, $f > q_s$. We now group the forgeries in F by their tuples $(\hat{K}, m)$ and maintain a counter list F_{ctr} that records, for each pair $(\hat{K}, m)$ appearing in F , how many forgeries correspond to it. Recall that $Q_{ctr}[(\hat{K}, m)]$ stores the no. of completed signing queries made for that tuple.

If $F_{ctr}[(\hat{K}, m)] \leq Q_{ctr}[(\hat{K}, m)]$ for all $(\hat{K}, m)$, then summing over all tuples gives $f \leq q_s$, contradicting $f > q_s$. Hence, there exists some $(\hat{K}^*, m^*)$ such that $Q_{ctr}[(\hat{K}^*, m^*)] = \varepsilon$ or $F_{ctr}[(\hat{K}^*, m^*)] > Q_{ctr}[(\hat{K}^*, m^*)]$. Let $I = \{\ell' \in [f] : (\hat{K}_{\ell'}^*, m_{\ell'}^*) = (\hat{K}^*, m^*)\}$. Then at least one signature in $\{(\sigma_{\ell'}^*)_{\ell' \in I}\}$ was never returned by the

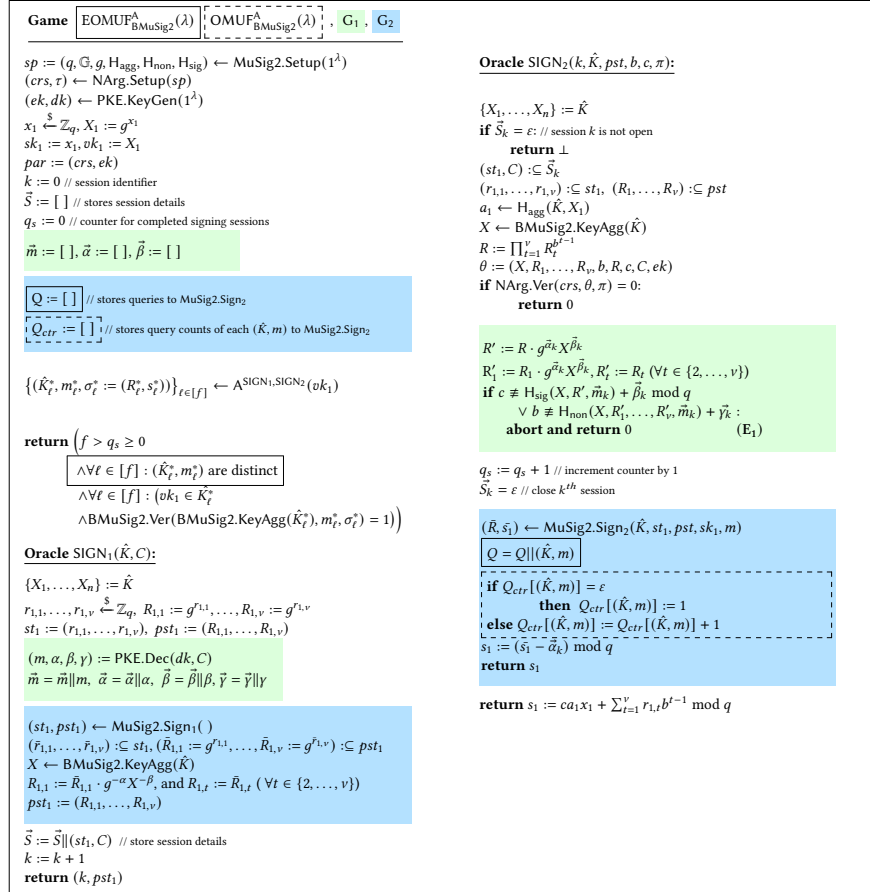


Figure 8: The OMUF (excluding the solid box) and EOMUF (excluding the dashed box) games with the hybrids used in the proof of Theorem 4.1. G_0 is the original game, G_1 adds the green box, and G_2 further adds the blue box.

MSign_2 oracle. Hence the tuple $(\hat{K}^*, m^*, (\sigma_{\ell'}^*)_{\ell' \in [I]})$ constitutes a valid forgery for MuSig2 . Therefore,

$$\Pr[G_2^A(\lambda)] \leq \text{Adv}_{\text{MuSig2}[\text{GGen}, \text{HGen}], \text{U}}^{\text{SUF-CMA}}(\lambda). \quad (5)$$

Eqn. 5, together with Eqns. 3 and 4 concludes the proof. $\square$

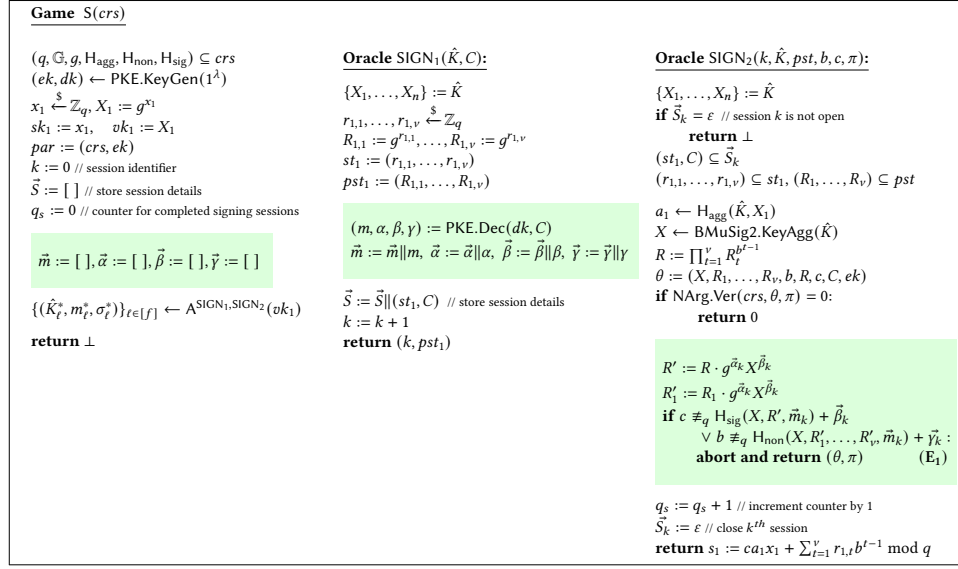
Note: The proof of EOMUF security proceeds analogously to that of OMUF and follows the same sequence of hybrid games. The game $\text{EOMUF}_{\text{BMuSig2}}$ differs from $\text{OMUF}_{\text{BMuSig2}}$ only in that the adversary is required to output forgeries corresponding to distinct tuples $(\hat{K}_\ell^*, m_\ell^*)$. The game G_1 remains unchanged. In G_2 , we replace the counter Q_{ctr} with a list Q that records every queried pair $(\hat{K}, m)$ upon completion of its signing query. Suppose an EOMUF adversary A wins in G_2 and it outputs the forgery set $F := \{(\hat{K}_\ell^*, m_\ell^*, \sigma_\ell^* = (R_\ell^*, s_\ell^*))\}_{\ell \in [f]}$. Since the tuples $(\hat{K}_\ell^*, m_\ell^*)$ are distinct and $f > |Q| = q_s$, there exists at least one pair $(\hat{K}_\ell^*, m_\ell^*) \notin Q$. The reduction therefore outputs the corresponding element in $F \setminus Q$ as a valid EUF-CMA forgery for U .

D Blindness of BMuSig2

This section gives the proof of Theorem 4.2.

PROOF. We proceed by a sequence of games specified in Fig. 11. **Game G_0 :** This is the original blindness game $\text{BLD}_{\text{BMuSig2}}$ from Fig. 11 (without the colored boxes). The adversary A participates as the n signers and consists of two algorithms (A_1, A_2) . First, A_1 outputs a set of verification keys $\hat{K}$ together with two messages (m_0, m_1) . Next, the challenger selects a random bit $b \in \{0, 1\}$ and runs two concurrent signing sessions ($k = 0, 1$) for message m_b and m_{1-b} with A_2 . The user's behaviour in these sessions is modelled through the three oracles USER_0 , USER_1 and USER_2 . If one or both sessions fail to output a (valid) signature, the game outputs $(\perp, \perp)$. Otherwise, if A closes both sessions successfully, the game outputs the resulting signatures (σ_0, σ_1) . At the end A outputs a bit b' and wins the game if $b' = b$.

Game G_1 : In Game G_1 , we modify the USER_1 oracle so that, instead of invoking NArg.Prove on θ , it uses NArg.SimProve to generate a simulated proof. We claim that this modification is computationally indistinguishable from G_0 due to the zero-knowledge property of $\text{NArg}[\text{R}_{\text{BMuSig2}}]$. To prove this for $b \in \{0, 1\}$, we construct the adversary Z_b (Fig. 12) against the zero-knowledge game of $\text{NArg}[\text{R}_{\text{BMuSig2}}]$, where Z_b simulates the game BLD_b for A . From the definition of the ZK game, Z_b receives as input a common

Figure 9: S playing against the soundness of $\text{NArg}[\text{RBMuSig2}]$.

reference string crs generated by NArg.Setup and MuSig2.Setup . Using this, Z_b simulates the BLD_b^b game for A. It answers each USER_1 query of A by invoking its Prove oracle to generate a proof π . When A_2 outputs its decision bit b' , Z_b returns b' to its ZK challenger. By definition of the adversary Z_b , $\Pr[\text{ZK}_{\text{NArg}[\text{RBMuSig2}]}^{Z_b,0}]$ and $\Pr[\text{ZK}_{\text{NArg}[\text{RBMuSig2}]}^{Z_b,1}]$ denote the probabilities that Z_b wins the ZK game when the challenger provides an actual proof and a simulated proof, respectively. Therefore, we have $\Pr[\text{ZK}_{\text{NArg}[\text{RBMuSig2}]}^{Z_b,0}] = \Pr[\text{BLD}_{\text{BMuSig2}}^{A,b}]$ and $\Pr[\text{ZK}_{\text{NArg}[\text{RBMuSig2}]}^{Z_b,1}] = \Pr[G_1^{A,b}]$.

We define the ZK advantage of Z_b as,

$$\begin{aligned} \text{Adv}_{\text{NArg}[\text{RBMuSig2}], Z_b}^{\text{ZK}} &:= \left| \Pr[\text{ZK}_{\text{NArg}[\text{RBMuSig2}]}^{Z_b,0}] - \Pr[\text{ZK}_{\text{NArg}[\text{RBMuSig2}]}^{Z_b,1}] \right| \\ &= \left| \Pr[\text{BLD}_{\text{BMuSig2}}^{A,b}] - \Pr[G_1^{A,b}] \right|. \end{aligned}$$

Applying the triangle inequality, this implies

$$\begin{aligned} \text{Adv}_{\text{BMuSig2}, A}^{\text{BLD}} &= \left| \Pr[\text{BLD}_{\text{BMuSig2}}^{A,1}] - \Pr[\text{BLD}_{\text{BMuSig2}}^{A,0}] \right| \\ &= \left| \Pr[\text{BLD}_{\text{BMuSig2}}^{A,1}] - \Pr[G_1^{A,1}] + \Pr[G_1^{A,1}] - \Pr[\text{BLD}_{\text{BMuSig2}}^{A,0}] \right| \\ &\quad + \left| \Pr[G_1^{A,0}] - \Pr[G_1^{A,1}] \right| \\ &\leq \text{Adv}_{\text{NArg}[\text{RBMuSig2}], Z_1}^{\text{ZK}} + \left| \Pr[G_1^{A,1}] - \Pr[G_1^{A,0}] \right| + \text{Adv}_{\text{NArg}[\text{RBMuSig2}], Z_0}^{\text{ZK}}. \end{aligned} \tag{6}$$

Game G_2 : In Game G_2 we modify the USER_0 oracle to encrypt an arbitrary fixed message $\tilde{m}$ and $\alpha = 0, \beta = 0, \gamma = 0$ instead of the chosen message. We claim that the game G_2 is computationally indistinguishable from the game G_1 due to the CPA-security of PKE. To prove this, for $b \in \{0, 1\}$, we define an adversary algorithm C_b (Fig. 13) that plays the CPA game for the PKE scheme. From the definition of CPA, C_b receives the encryption key ek , extracts the security parameter 1^λ , and generates (sp, crs) to simulate $G_1^{A,b}$. On a call

to $\text{USER}_0(k)$, C_b sets $M_0 := (\tilde{m}, 0, 0, 0)$, $M_1 := (m_{k \oplus b}, \alpha_k, \beta_k, \gamma_k)$, and queries its encryption oracle on (M_0, M_1) , and forwards the received ciphertext C_k to A_2 . (Note: C_b does not know the randomness used to generate C_k but this is sufficient since proofs in USER_1 are simulated and do not require the witness). By definition of C_b , $\Pr[\text{CPA}_{\text{PKE}}^{C_b,0}]$ and $\Pr[\text{CPA}_{\text{PKE}}^{C_b,1}]$ denote the probabilities that C_b wins the CPA game when the challenger encrypts an actual message and a fixed message, respectively. Therefore we have $\Pr[\text{CPA}_{\text{PKE}}^{C_b,0}] = \Pr[G_2^{A,b}]$ and $\Pr[\text{CPA}_{\text{PKE}}^{C_b,1}] = \Pr[G_1^{A,b}]$. We define the CPA advantage of C_b as

$$\text{Adv}_{\text{PKE}, C_b}^{\text{CPA}} := \left| \Pr[\text{CPA}_{\text{PKE}}^{C_b,1}] - \Pr[\text{CPA}_{\text{PKE}}^{C_b,0}] \right| = \left| \Pr[G_1^{A,b}] - \Pr[G_2^{A,b}] \right|.$$

Applying the triangle inequality, this implies

$$\begin{aligned} \left| \Pr[G_1^{A,1}] - \Pr[G_1^{A,0}] \right| &= \left| \Pr[G_1^{A,1}] - \Pr[G_2^{A,1}] + \Pr[G_2^{A,1}] - \Pr[G_2^{A,0}] \right. \\ &\quad \left. + \Pr[G_2^{A,0}] - \Pr[G_1^{A,0}] \right| \\ &\leq \text{Adv}_{\text{PKE}, C_1}^{\text{CPA}} + \left| \Pr[G_2^{A,1}] - \Pr[G_2^{A,0}] \right| + \text{Adv}_{\text{PKE}, C_0}^{\text{CPA}}. \end{aligned} \tag{7}$$

Proving blindness from game G_2 : The signer's view after the successful completion of the two signing sessions $k \in \{0, 1\}$ consists of *i*) parameters (crs, ek) , *ii*) signatures with their corresponding messages, $(m_0, \sigma_0 := (R'_0, s'_0))$ and $(m_1, \sigma_1 := (R'_1, s'_1))$, and *iii*) transcripts T_k corresponding to the n signers, where T_k is defined as

$$T_k := (C_k, (R_{j,1}^{(k)}, \dots, R_{j,v}^{(k)}), (R_1^{(k)}, \dots, R_v^{(k)}), R_k, b_k, c_k, \pi, \{s_j^{(k)}\}),$$

where $k \in \{0, 1\}$ and $j \in [n]$. Since the ciphertext C_k is an encryption of a fixed message with randomness ρ_k , and π is a simulated proof, neither reveals any information about the bit b . Therefore, we exclude them from our analysis of transcripts and user states.

For $i \in \{0, 1\}$, we define the view of adversary (signer) A as $V_{k,i}(A) := (T_k, m_i, \sigma_i)$. First, we show that for all $k, i \in \{0, 1\}$, the view $V_{k,i}(A)$ determines a valid user state $st := (\alpha_{k,i}, \beta_{k,i}, \gamma_{k,i})$.



Figure 10: U playing against SUF-CMA (excluding the solid box) / EUF-CMA (excluding the dashed box) security of MuSig2.

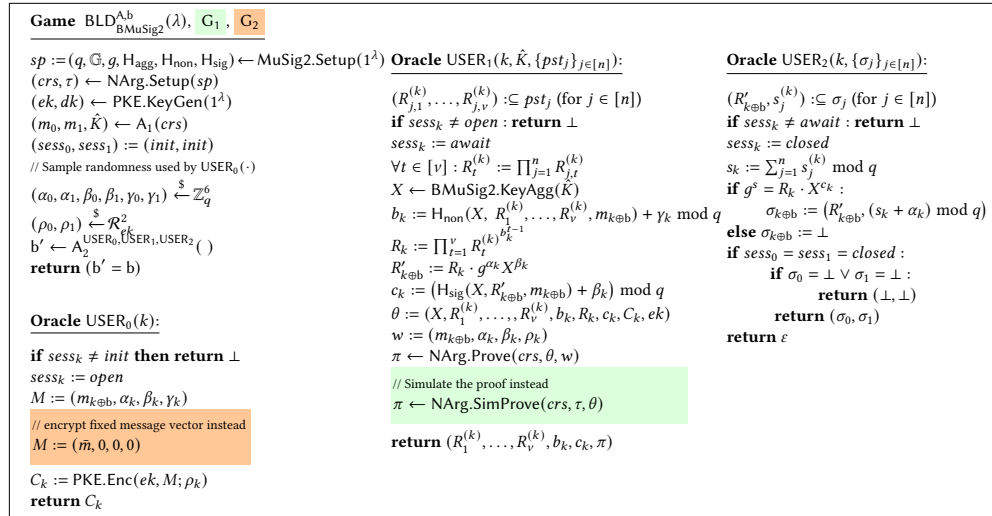


Figure 11: Blindness game for BMuSig2 and hybrid games used in the proof of Theorem 4.2.

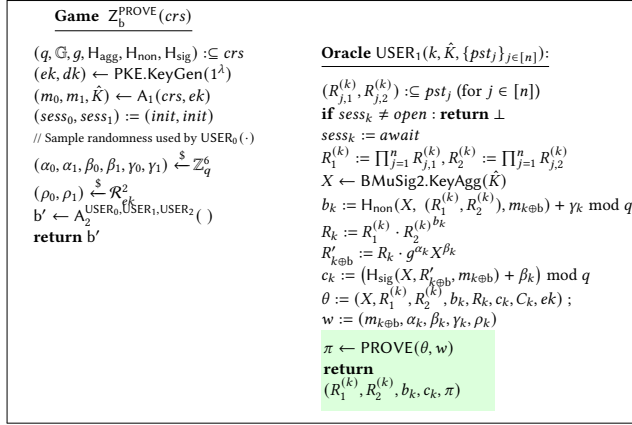


Figure 12: The game Z_b playing against zero-knowledge of the $\text{NArg}[\text{R}_{\text{BMuSig2}}]$. The $\text{USER}_0, \text{USER}_2$ oracles simulated to A_2 are same as G_0 .

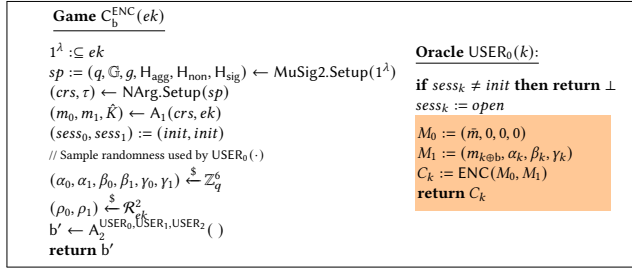


Figure 13: The game C_b playing against CPA security of PKE. The $\text{USER}_1, \text{USER}_2$ oracles simulated to A_2 are same as G_1 .

This st^7 represents the internal values the user would have used if session k corresponds to message m_i . We can construct it as follows,

$$\begin{aligned}
 \alpha_{k,i} &:= s'_i - s_k \bmod q, \\
 \beta_{k,i} &:= c_k - H_{\text{sig}}(X, R'_i, m_i) \bmod q, \\
 \gamma_{k,i} &:= b_k - H_{\text{non}}(X, R_1^{(k)}, \dots, R_v^{(k)}, m_i) \bmod q.
 \end{aligned} \tag{8}$$

where $s_k = \sum_{j \in [n]} s_j^{(k)}$. Next, we have to show that these values are uniformly distributed. The uniformity of $\alpha_{k,i}$ follows from the uniformity of s'_i which is computed as $(s_i + \alpha_i)$, where α_i is the real blinding factor used during the experiment. Similarly, the uniformity of $\beta_{k,i}$ and $\gamma_{k,i}$ follows from c_k and b_k which consists of the real blinding factor β_k and γ_k in the experiment respectively.

Now, we can define $R'_{k,i} := \prod_{t=1}^v R_t^{(k) b_t^{t-1}} \cdot g^{\alpha_{k,i}} X^{\beta_{k,i}}$, $s_{k,i} := s_k - \alpha_{k,i}$, $c_{k,i} := H_{\text{sig}}(X, R'_{k,i}, m_i)$. Finally, we show that for all $k, i \in \{0, 1\}$, the user state st defines a valid signature (R'_i, s'_i) and hash c'_i . From the validity of signatures $\sigma_i = (R'_i, s'_i)$, we have

$$g^{s'_i} = R'_i \cdot X^{c'_i} \Rightarrow R'_i = g^{s'_i} \cdot X^{-c'_i}, \tag{9}$$

where $c'_i = H_{\text{sig}}(X, R'_i, m_i)$. Now substitute $s'_i = \alpha_{k,i} + s_k$ and $c'_i = c_k - \beta_{k,i}$ from Eqn. 8 in Eqn. 9 to obtain

$$R'_i = g^{\alpha_{k,i} + s_k} \cdot X^{-c_k + \beta_{k,i}}. \tag{10}$$

We assume A closes both signing sessions $k \in \{0, 1\}$ successfully, and in this case, T_k is a valid transcript. Therefore we have,

$$g^{s_k} = R_k \cdot X^{c_k}, \text{ where } R_k = \prod_{t=1}^v R_t^{(k) b_t^{t-1}}. \tag{11}$$

We substitute the above value for g^{s_k} in Eqn.10 to obtain

$$\begin{aligned}
 R'_i &= g^{\alpha_{k,i}} \cdot (R_k \cdot X^{c_k}) \cdot X^{-c_k + \beta_{k,i}} = g^{\alpha_{k,i}} \cdot R_k \cdot X^{\beta_{k,i}} \\
 &= R_k \cdot g^{\alpha_{k,i}} X^{\beta_{k,i}} = \prod_{t=1}^v R_t^{(k) b_t^{t-1}} \cdot g^{\alpha_{k,i}} X^{\beta_{k,i}} = R'_{k,i}.
 \end{aligned}$$

We can see that R'_i is determined by the user state. Consequently, $s'_i = s_k - \alpha_{k,i} \bmod q$ and $c'_i = H_{\text{sig}}(X, R'_{k,i}, m_i)$. This means both transcripts T_0 and T_1 can define the signature (R'_i, s'_i) . Hence, the advantage in distinguishing G_2 with $b = 0$ from G_2 with $b = 1$ is

$$\left| \Pr[G_2^{\Lambda,1}] - \Pr[G_2^{\Lambda,0}] \right| = 0. \tag{12}$$

Eqn. 12, together with Eqns. 6 and 7 concludes the proof. $\square$

⁷Note that the user state is determined via $st := (\alpha_{k,i}, \beta_{k,i}, \gamma_{k,i})$ because all the other values on the user side are fixed (given the transcripts and the message-signature pair).