



TECHNISCHE
UNIVERSITÄT
DARMSTADT

TECHNISCHE UNIVERSITÄT DARMSTADT
DEPARTMENT OF COMPUTER SCIENCE
CHAIR OF APPLIED CRYPTOGRAPHY

Master Thesis

Threshold Blind Schnorr Signature Scheme for Cryptographic Wallets

Philipp-Florens Lehwalder

31.03.2023

Supervisors: Prof. Sebastian Faust, Ph.D.
Andreas Erwig

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Philipp-Florens Lehwalder, die vorliegende Master Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

31. März 2023

Philipp-Florens Lehwalder

Abstract

Cryptocurrencies and DeFi have gained tremendous popularity in recent years, with the highest market capitalization to date of \$3 trillion [Coi23], and by their nature allow the participation of people from all social classes around the world.

The integral building block of cryptocurrencies are digital signatures, which are used to authorize transactions published on the blockchain by signing them with one's secret key. This secret key has to be securely stored in a wallet, whereas many users conveniently use custodial online wallets that manage their secret key for them. However, users have to trust the online wallet service to not misuse their key, and they face a loss of privacy as the service learns all their transactions to be sent.

In this thesis, we present the first threshold blind signature scheme based on Schnorr that solves these drawbacks. The combination of threshold and blind signatures allows the secret key to be shared among multiple servers, thereby distributing trust and obtaining signatures without the servers learning the content. We prove the security of our scheme by reducing it to the underlying blind Schnorr signature scheme of Fuchsbauer et al. [FPS19]. Additionally, we propose a threshold blind Schnorr signature construction based on the popular FROST scheme by Komlo and Goldberg [KG20]. Both schemes are directly compatible with various cryptocurrencies and, in general, with systems that support Schnorr, as the verification remains the same as for standard Schnorr signatures.

Contents

1	Introduction	1
1.1	Related Work	2
1.2	Outline	2
2	Preliminaries	3
2.1	Notations	3
2.2	Algebraic Group Model	3
2.3	One-More Discrete Logarithm Problem	4
2.4	The ROS Problem	5
2.4.1	ROS Attacks	6
2.4.2	Countermeasures	7
2.5	Building Blocks	9
2.5.1	Threshold Secret Sharing	9
2.5.2	Schnorr Signature Scheme	9
2.5.3	Equivocable Commitment Scheme	11
2.5.4	Non-Interactive Zero-Knowledge Proof of Knowledge	12
3	Threshold Signatures	15
3.1	Background	15
3.2	Definition	16
4	Blind Signatures	18
4.1	Background	18
4.2	Definition	19
4.3	Blind Schnorr Signatures	20
4.3.1	Original Blind Schnorr Signature Scheme	21
4.3.2	Clause Blind Schnorr Signature Scheme	22
4.3.3	Alternative Blind Schnorr Signature Schemes	23
5	Threshold Blind Signatures	25
5.1	Background	25
5.2	Definition	26
5.3	Our Construction	27
5.3.1	Proof of Security	29
5.3.2	On Drijvers' and Benmahouda's Attack	35
5.4	Blind FROST Construction	35
5.4.1	Proof of Security	39
5.4.2	Optimizations	40
6	Conclusion	41
6.1	Contributions	41
6.2	Future Work	41

1 Introduction

In cryptocurrencies, digital signatures play an essential role as they are used to validate the authorization and authenticity of transactions published on the blockchain and ultimately provide trust in the system. The secret key for signing and thus authorizing transactions, which is stored in a so-called *wallet*, is therefore directly associated with the control over the owned monetary assets. Loss of the secret key, on the other hand, results inevitably in loss of access to all assets, illustrating the immense importance of securely storing this key. Considering Bitcoin, according to a study of Chainalysis [Cha18], up to 3.7 million Bitcoins are lost due to lost secret keys which would value up to \$170 billion at the exchange rate on 1st January 2022. Similarly, if hackers get to know the secret key, they can create transactions transferring all funds to their own address as happened numerous times in the past, resulting in billions of dollars in damages alone in 2021 and 2022 [Beh21; Ben21; Gkr21; Jha22].

However, especially out of convenience reasons, many cryptocurrency users leave their funds on centralized exchanges or prefer to rely on *online wallets* such as the Coinbase Wallet¹ or Blockchain.com², the latter of which already has more than 85 million users by 2022 [Blo22]. In this case, we mostly speak about *custodial* wallets, where the user logs in via their username and password and the third party has the responsibility of managing the user's secret keys. While online wallets reduce the user's responsibility of securely storing the secret key, the user still has to trust the service to properly protect the key against attackers and that they do not misuse the key. In addition, the user's privacy is reduced because the service is always aware of the transaction the user is about to send and could also decide to refuse to sign it.

The first security-related drawbacks can be addressed by using *threshold cryptography* which enables distributing the trust from one central party to multiple independent parties. Threshold cryptography dates back to the 1980s/1990s, starting with the idea that a high-value secret key should not be held by a single entity and thus represent a single point of failure, but should be shared among multiple entities so that only a threshold number of them can reconstruct it [Bla79; Sha79; DF90]. Falling into this area, *threshold signatures* allow a group of participants, each with a share of the signing key, to create signatures only if a threshold number of them are involved. Therefore, in the context of online wallets, instead of a single server knowing the complete secret key, the key would be shared among n servers, and as long as less than t servers are corrupt, the attacker does not learn any information about the secret key.

The latter privacy-related drawbacks can be addressed by using *blind signatures*, which were first introduced by Chaum in 1982 [Cha82]. Blind signatures allow obtaining a signature on a message from a signer in a way that the signer does not see the message that has been signed. As an analogy in the real world for blind signatures, we can consider carbon-lined envelopes where the receiver puts their message inside the envelope and the signer then signs the envelope. When the user takes the message out of the envelope, it receives the signed message without the signer learning the content. For online wallets, blind signatures would guarantee the user that the service does not learn any information about the user's transactions when issuing the required signature.

The signature scheme that is used behind many cryptocurrencies is the Schnorr signature scheme that has been originally proposed by Schnorr in 1989/1991 [Sch89; Sch91], although a patent first hindered greater adoption of the scheme until it expired in 2008. While Schnorr is most prominently used in Bitcoin since the "Taproot" update, it is also employed in other cryptocurrencies such as Polkadot, Cardano, Ripple, or Monero in the form of EdDSA.

Besides counting to one of the most efficient signature schemes, Schnorr signatures are especially

¹<https://www.coinbase.com/wallet>

²<https://www.blockchain.com/wallet>

promising for cryptocurrencies since their linearity allows key- and signature aggregation, they are non-malleable (unlike ECDSA), and are compatible with distributed key generation, which is useful for threshold signatures [Pie20; Coi22].

By combining threshold cryptography and blind signatures for Schnorr we thus achieve enhanced security and stronger privacy guarantees for online wallets that are directly compatible with a variety of popular cryptocurrencies.

1.1 Related Work

Especially with the rise in the popularity of cryptocurrencies in the last few years, the demand for multi- and threshold signatures has increased. They enable a group of parties of a specified size to confirm a transaction, each with its own key (share), rather than only a single party having control over the assets. Since plenty of cryptocurrencies support the Schnorr signature scheme, which allows elegant signature aggregation, various multi- and threshold signature schemes based on Schnorr have been introduced [SS01; Max+19; NRS21; KG20; Gar+21; Ruf+22; Lin22]. However, attacks by Drivers et al. and Benhamouda et al. [Dri+19; Ben+22] revealed the insecurity of popular multi- and threshold Schnorr signature schemes, partly even allowing a forgery in polynomial time. Nevertheless, while not all schemes were affected, others could be fixed without significant compromises on efficiency.

The situation is similar for blind signatures; while there are plenty of schemes based on different assumptions such as RSA [CFN88; AF96; CKW04] or using bilinear pairings [Oka06; Bla+13; GG14], for schemes that build on Schnorr or Okamoto-Schnorr [CP92; AO00; Oka92; ZGP17], the ROS problem stated a known vulnerability as early work [Wag02] already revealed sub-exponential attacks. The recent attack published by Benhamouda et al. [Ben+22] then allowed polynomial-time attacks against the ROS problem. Using different techniques or building upon variants of the ROS problem, secure blind Schnorr schemes have been proposed by Fuchsbauer et al. [FPS19], Katz et al. [KLR21], Tessaro and Zhu [TZ22], and Fuchsbauer and Wolf [FW22].

The field of threshold blind signatures is still relatively sparse and mainly consists of constructions based on the Okamoto-Schnorr blind signature scheme and employ bilinear pairings [LJY02; KKL02; VZK03; CZL06; KM15]. So far, a threshold blind signature scheme based on plain Schnorr has not been proposed.

For more in-depth background on threshold-, blind-, and threshold blind signatures, each chapter begins with a prefacing background about the respective topic.

1.2 Outline

In Chapter 2, we provide preliminary material such as notations, complexity assumptions, and building blocks. In particular, we cover the ROS problem regarding attacks and impacts, countermeasures, and proposed variants.

In Chapter 3, we give some background for threshold signatures and present a comparison of existing threshold Schnorr signature schemes and provide formal definitions.

Chapter 4 addresses blind signatures by giving a background and comparing existing blind Schnorr signature schemes, providing formal definitions, and recapping the original blind Schnorr scheme by Chaum [CP92], and the clause blind Schnorr scheme by Fuchsbauer [FPS19]. Finally, we discuss the other blind Schnorr signature schemes for a threshold extension for our desired use case.

In the primary Chapter 5, we cover threshold blind signatures. We first give a background of already existing schemes and compare them with our two constructions, and provide formal definitions. We finally present our two threshold blind Schnorr constructions including proofs of security.

Lastly, Chapter 6 summarizes the main contributions of this thesis and concludes with a discussion for future work.

2 Preliminaries

In this chapter, we introduce notations we will use throughout this thesis and define different cryptographic problems with a focus on the ROS problem, which will be discussed in more detail. Finally, we define the building blocks upon which our threshold blind Schnorr signature scheme constructions are based.

2.1 Notations

The security parameter is denoted with $\lambda \in \mathbb{N}$ and is usually written as 1^λ .

A function f is called *negligible* if for all constants $c \geq 0$, there exists an integer x_1 such that for all $x \geq x_1$ it holds that $f(x) \leq \frac{1}{x^c}$. We typically name such a function $\text{negl}(x)$.

When assigning a value v to a variable x , we write $x \leftarrow v$. We similarly use $\leftarrow$ when assigning x with the output of a deterministic algorithm $\mathcal{A}$, i.e., $x \leftarrow \mathcal{A}$. If the algorithm $\mathcal{A}$ is probabilistic, we write $x \leftarrow \$ \mathcal{A}$. For a set $\mathcal{X}$, the same syntax as $x \leftarrow \$ \mathcal{X}$ is used to denote that x is assigned to a value sampled uniformly at random from $\mathcal{X}$. The size or order of a set $\mathcal{X}$ is indicated by $|\mathcal{X}|$.

For an integer n , the abbreviation $[n]$ refers to the ordered set $\{1, \dots, n\}$.

We abbreviate a probabilistic polynomial-time algorithm with PPT and a deterministic polynomial-time algorithm with DPT.

When running an algorithm $\mathcal{A}$ on some inputs $(x_1, \dots, x_n)$ while providing access to some oracle $\mathcal{O}$, we write $y \leftarrow \mathcal{A}^{\mathcal{O}}(x_1, \dots, x_n)$, where y denotes the output of $\mathcal{A}$.

When writing $(y_1, y_2) \leftarrow \langle \mathcal{A}(x_1), \mathcal{B}(x_2) \rangle$, we denote the execution of a protocol between algorithm $\mathcal{A}$ and $\mathcal{B}$ given x_1 and x_2 as inputs, respectively, and $\mathcal{A}$ outputs y_1 and $\mathcal{B}$ outputs y_2 .

With GrGen we refer to a PPT algorithm that takes the security parameter 1^λ as an input and outputs a group description $(p, \mathbb{G}, G)$ where p is a λ bit prime and $\mathbb{G}$ an abelian group of order p , and G a generator of the group. Throughout this thesis, we only will use multiplicative groups.

We use standard code-based security games for security definitions and proofs [Sho04]. A security game **Game** is played between an adversary $\mathcal{A}$ and an (implicit) challenger $\mathcal{C}$ who answers oracle queries of $\mathcal{A}$. It consists of the main procedure and arbitrary many oracle procedures. The main procedure initializes the game and runs the adversary $\mathcal{A}$ who can query the defined oracles and outputs some game-specific value. In the end, the game returns an output that is mostly a bit b such that we write $b \leftarrow \text{Game}^{\mathcal{A}}(\lambda)$ as the output of **Game** given the security parameter λ while playing with $\mathcal{A}$.

By the *view* for an adversary $\mathcal{A}$, we refer to all messages that $\mathcal{A}$ sees during the execution of a protocol. By $\mathcal{S}$, we denote the set of signers identified by their indices. The size of $\mathcal{S}$ or the number of signers is specified by the integer n . A single signer is written as S_i with $i \in \mathcal{S}$. The user who requests a signature from the signers is written as U .

We sometimes write non-interactive zero-knowledge proof instead of non-interactive zero-knowledge proof of knowledge for better readability.

For schemes having a setup algorithm that generates the public parameters pp , these public parameters are implicitly given as an input to each algorithm, even if not stated explicitly.

2.2 Algebraic Group Model

The algebraic group model (AGM) was proposed by Fuchsbaauer, Kiltz, and Loss [FKL18] and serves as an idealized computational model where all adversaries are algebraic. An algebraic algorithm is an algorithm that, for each group element it outputs, also outputs the representation of this element in terms of all group elements it has seen so far. We provide a formal definition below.

Definition 1 (Algebraic Group Model (AGM)). *Let $(p, \mathbb{G}, G)$ be a group description given as an output by GrGen. In the algebraic group model, we assume that all adversaries are algebraic, that is: For each group element $Z \in \mathbb{G}$ an adversary $\mathcal{A}_{alg}$ outputs, it also has to output a representation $(z_0, \dots, z_n) \in \mathbb{Z}_p^{n+1}$ such that $Z = \sum_{i=0}^n z_i X_i$ where $X_0, \dots, X_n \in \mathbb{G}^{n+1}$ are all elements $\mathcal{A}_{Alg}$ has received so far.*

The AGM lies between the standard model and the generic group model (GGM) [Nec94; Sho97], which is an even stronger idealization where the adversary only gets random labels for the group elements and has to query an oracle for performing group operations with them. In contrast, an algebraic adversary gets the actual group elements and can freely perform any group operations by itself but only has to provide a representation of the outputted element in the end.

2.3 One-More Discrete Logarithm Problem

We now provide a formal definition for the one-more discrete logarithm problem and begin with a brief recapitulation of the discrete logarithm problem.

The discrete logarithm problem is a fundamental assumption in public-key cryptography since it has been first introduced by Diffie and Hellman in their seminal work in 1976 [DH22].

Given a group $\mathbb{G}$ of order p , where p is a prime number of size λ , with a generator G , the discrete logarithm problem challenges to find a value $x \in \mathbb{Z}_p$ for a randomly sampled element $X \leftarrow \mathbb{G}$ such that $xG = X$. The formal game-based definition is given below.

Definition 2 (DL Assumption). *Let $\mathcal{A}$ be a PPT adversary and $\text{negl}(\lambda)$ a negligible function. Then, the discrete logarithm assumption holds if we have $\Pr[\text{DL}_l^{\mathcal{A}}(\lambda) = 1] \leq \text{negl}(\lambda)$ for the security game DL_l specified in Figure 2.1.*

Game $\text{DL}_l^{\mathcal{A}}(\lambda)$
1 : $(p, \mathbb{G}, G) \leftarrow \text{GrGen}(1^\lambda)$
2 : $x \leftarrow \mathbb{Z}_p; X \leftarrow xG$
3 : $x' \leftarrow \mathcal{A}(p, \mathbb{G}, G, X)$
4 : return $(x' = x)$

Figure 2.1: The DL game.

The class of "one-more" problems has been introduced by Bellare, Namprempre, Pointcheval, and Semanko in 2001 [Bel+01] allowing them to prove the security of Chaum's RSA blind signature scheme based on the one-more RSA inversion problem. In an updated version of the paper [Bel+03], they further defined the one-more discrete logarithm problem which had been used to prove the active and concurrent security of the Schnorr identification scheme [BP02]. Or, more recently, to prove the unforgeability of the blind Schnorr signature scheme [CP92] in the algebraic group model by Fuchsbauer et al. [FPS19]. In general, one-more problems provide the adversary with $k + 1$ challenges of a problem that is assumed to be *one-way* and ask the adversary to output all $k + 1$ solutions while allowing only k solution-oracle queries. The formal game-based definition influenced by Bauer et al. [BFP21] of the one-more discrete logarithm problem is given below:

Definition 3 (OMDL Assumption). *Let $\mathcal{A}$ be a PPT adversary and $\text{negl}(\lambda)$ a negligible function, then, the one-more discrete logarithm assumption holds, if $\Pr[\text{OMDL}_l^{\mathcal{A}}(\lambda) = 1] \leq \text{negl}(\lambda)$ for the security game OMDL_l specified in Figure 2.2.*

Game $\text{OMDL}_l^A(\lambda)$	Oracle $\text{Chal}()$
1 : $(p, \mathbb{G}, G) \leftarrow \$ \text{GrGen}(1^\lambda)$	1 : $x \leftarrow \$ \mathbb{Z}_p; X \leftarrow xG$
2 : $\vec{x} := (); q_s \leftarrow 0$	2 : $\vec{x} \leftarrow \vec{x} (x)$
3 : $\vec{x}' \leftarrow \$ \mathcal{A}^{\text{Chal}, \text{Dlog}}(p, \mathbb{G}, G)$	3 : return X
4 : return $(\vec{x}' = \vec{x} \wedge q_s \leq \vec{x})$	
	Oracle $\text{Dlog}(X)$
	1 : $q_s \leftarrow q_s + 1$
	2 : $x \leftarrow \log_G(X)$
	3 : return x

Figure 2.2: The OMDL game.

2.4 The ROS Problem

The ROS (Random inhomogeneities in an Overdetermined Solvable system of linear equations) problem was originally described by Schnorr [Sch01] in the context of blind Schnorr signatures. He ascertained that the unforgeability of the blind Schnorr [CP92] and the blind Okamoto-Schnorr signature scheme [Oka92] does not only rely on the hardness of the discrete logarithm problem but also on the ROS problem.

We now recall the textual description of the ROS problem and provide the game-based definition, which is mainly borrowed from Fuchsbauer et al. [FPS19].

Let p be a prime of length λ , $l \geq 1$ an integer, Ω a set for auxiliary inputs, and let $H_{ros} : \mathbb{Z}_p^l \times \Omega \rightarrow \mathbb{Z}_p$ be a random oracle. Then, the ROS problem challenges to find $l + 1$ vectors $\vec{\rho}_i \in \mathbb{Z}_p^l$ with auxiliary information $aux_i \in \Omega$ and a solution vector $\vec{c} \in \mathbb{Z}_p^l$ such that the following linear system holds for $i \in [l + 1]$:

$$\sum_{j=1}^l \rho_{i,j} c_j \equiv_p H_{ros}(\vec{\rho}_i, aux_i) \quad (2.1)$$

Definition 4 (ROS Problem). *The ROS problem is defined by the security game ROS_l^A depicted in Figure 2.3 with respect to an integer $l > 1$, where $H_{ros} : \mathbb{Z}_p^l \times \Omega \rightarrow \mathbb{Z}_p$ is a random oracle. The ROS assumption holds, if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{ROS}_l^A(\lambda) = 1] \leq \text{negl}(\lambda)$ for a negligible function $\text{negl}(\lambda)$.*

Game $\text{ROS}_l^A(\lambda)$	Oracle $H_{ros}(\vec{\rho}, aux_i)$
1 : $(p, \mathbb{G}, G) \leftarrow \$ \text{GrGen}(\lambda); T_{ros} := ()$	1 : if $T_{ros}(\vec{\rho}, aux_i) = \perp$ then
2 : $((\vec{\rho}_i, aux_i)_{i \in [l+1]}, \vec{c}) \leftarrow \$ \mathcal{A}^{H_{ros}}(p)$	2 : $T_{ros}(\vec{\rho}, aux_i) \leftarrow \$ \mathbb{Z}_p$
3 : return $(\forall i \neq j \in [l + 1] : (\vec{\rho}_i, aux_i) \neq (\vec{\rho}_j, aux_j) \wedge$	3 : fi
4 : $\forall i \in [l + 1] : \sum_{j=1}^l \rho_{i,j} c_j \equiv_p H_{ros}(\vec{\rho}_i, aux_i))$	4 : return $T_{ros}(\vec{\rho}, aux_i)$

Figure 2.3: The ROS problem.

In contrast to other formulations of the ROS problem, the definition from Fuchsbauer et al. additionally introduces an input $aux \in \Omega$ for the random oracle H_{ros} which is useful for the reduction of blind signatures as it can serve as an input for the message.

We now give a short intuition of how the ROS problem relates to the unforgeability of blind Schnorr signatures. Let $\mathcal{A}$ be an adversary that can efficiently solve the ROS problem, then, we can construct a

PPT adversary $\mathcal{B}$ breaking the one-more unforgeability of the original blind Schnorr signature scheme oBSS, which is given in Section 4.3.1:

$\mathcal{B}$ starts with initiating l concurrent signing sessions using the oracle $\text{OSign}_{\text{oBSS}}$ to receive $R_1, \dots, R_l$. Then, $\mathcal{B}$ runs $\mathcal{A}$ to output a solution $((\vec{\rho}_i, m_i)_{i \in [l+1]}, \vec{c})$ for the ROS problem corresponding to $\forall i \in [l+1] : \sum_{j=1}^l \rho_{i,j} c_j \equiv_p \text{H}(\sum_{j=1}^l \rho_{i,j} R_j, m_i)$.

Using the solution, $\mathcal{B}$ replies with $c_1, \dots, c_l$ as the challenges to $\text{OSign}_{\text{oBSS}}$ for closing the respective signing sessions while receiving $s_1, \dots, s_l$ as an answer.

Lastly, $\mathcal{B}$ computes $R_i^* \leftarrow \sum_{j=1}^l \rho_{i,j} R_j$ and $s_i^* \leftarrow \sum_{j=1}^l \rho_{i,j} s_j$ for $i \in [l+1]$ and outputs $(m_i, (R_i^*, s_i^*))_{i \in [l+1]}$ as the final forgery.

Correctness of the forgery follows from the fact that when $((\vec{\rho}_i, m_i)_{i \in [l+1]}, \vec{c})$ is a valid ROS solution, all tuples are unique and due to Equation 2.1, we have:

$$s_i^* G = \sum_{j=1}^l \rho_{i,j} s_j = \sum_{j=1}^l \rho_{i,j} (r_j + c_j x) G = R_i^* + \text{H}(R_i^*, m_i) X$$

Besides the initial Schnorr - and Okamoto-Schnorr blind signature scheme [CP92; Oka92], plenty of other blind signature variants had been published with their security based on the ROS problem, some of which are built on top of the two previously mentioned schemes.

For example, in the area of partially blind signatures, there are the schemes [AO00; Cho+05; Wu+07]. Regarding identity-based schemes, the blind signature scheme by Zhang et al. [ZK02] or the sign-encryption scheme by Yuen et al. [YW05] are based on the ROS problem. Other exemplary schemes that similarly rely on the ROS problem are, e.g., the blind group signature scheme by Chan [Cha+04], the proxy blind signature scheme by Kim et al. [KC06], and the first conditional blind signature scheme by Zacharakis et al. [ZGP17].

2.4.1 ROS Attacks

Only less than a year after Schnorr's publication of the ROS problem, Wagner [Wag02] showed a reduction of the ROS problem to a generalization of the birthday problem, the *k-sum problem*, which can be solved using Wagner's proposed *k-tree* algorithm in sub-exponential time. More specifically, with $k = l + 1$, the algorithm finds a solution with constant probability running in $O(k \cdot 2^{\lambda/(1+\lceil \log k \rceil)})$ time. When being able to run the algorithm with the number of dimensions set to $k = 2^{\sqrt{\lambda}-1}$, the complexity results to $O(k \cdot 2^{2\sqrt{\lambda}})$. Thus, Wagner's attack forced the existing schemes to increase the security parameter λ , corresponding to the number of p , to achieve an adequate security level compared to other blind signature schemes based on a different assumption.

More recently, Benhamouda et al. [Ben+22] presented a new attack that solves the ROS problem in polynomial time by expressing it as a subset-sum problem which can then be efficiently solved. To do so, however, it is required that $l \geq \lambda$, i.e., the number of parallel signing queries is at least equal to the bit size of p . For cases with $l < \lambda$, the authors present a generalized attack which is a combination of their algorithm with Wagner's *k-tree* algorithm that runs in sub-exponential time but still outperforms Wagner's attack by itself in certain settings. Needless to say, after the publication of Benhamouda's polynomial attack, all signature schemes with their security based on the ROS problem while not limiting the number of concurrent signing sessions had to be considered insecure.

Another recent attack related to the ROS problem has been published by Drijvers et al. [Dri+19] who specifically targeted two-round Schnorr multi-signature schemes (but which similarly affects threshold schemes). In principle, they showed how to break the unforgeability in sub-exponential time of various multi-signature schemes by reducing them to the *k-sum* problem which in turn can be solved using Wagner's *k-tree* algorithm. The requirement for their attack is that the adversary can either control the message m or select its own nonce R_i after seeing all other participants' nonces, which sum up to the group's nonce R used in the final signature. The problem can be informally described as follows: Given a value R^* and access to a random oracle H , the adversary has to find values m^* and R_i, m_i for $i \in [l]$ such that the following equation holds:

$$H(R^*, m^*) = \sum_{i=1}^l H(R_i, m_i) \quad (2.2)$$

2.4.2 Countermeasures

The recommended mitigation by Drijvers et al. [Dri+19] regarding their attack against multi-signature schemes is to add an initial round where all participants first have to commit to their individual nonces R_i before learning the other participants' nonces. A different approach without adding an extra round has been taken by Nick et al. [NRS21] with their MuSig2 scheme; there, each signer sends multiple nonces $R_{i,1} \dots R_{i,\nu}$ with $\nu \geq 2$ such that the resulting combined nonce $\hat{R}_i$ for each signer consists of a random linear combination $\hat{R}_i = \prod_{j=1}^{\nu} R_{i,j}^{b_j}$. The value b_j is then determined as the output of a random oracle given all nonces $R_{i,j}$ and the message m . This tweak prevents the adversary from solving the k -sum problem as the left-hand side of Equation 2.2, which is a combination of the individual signers' nonces, is not constant anymore. Moreover, Komlo and Goldberg [KG20] achieved only one communication round, plus a preprocessing stage, while preventing Drijvers attack with their FROST scheme. The core idea is similar to the one from Nick et al. in terms of removing the independence of the left - and right-hand side of Equation 2.2; which is done in FROST by including all inputs to the hash function on the right side of the equation to the computation of R^* using a so-called "binding value".

Regarding the existing ROS attacks, the most straightforward and effective countermeasure is to only allow sequential signing sessions or at least limit the number of concurrent signing sessions in relation to the desired security level. This measure prevents an adversary from constructing or solving the ROS equation as it does not even receive more than one nonce R from the signer (or not enough to solve the ROS problem efficiently). Of course, this approach is not realistic for scenarios where it is required to issue a large number of signatures simultaneously under the same key. However, in the context of online wallets, where we assume one key per user, issuing even only one signature for a transaction at a time is indeed reasonable.

To overcome the inherent weaknesses of the ROS problem, Fuchsbauer et al. [FPS19] and, more recently, Tessaro et al. [TZ22] proposed variants of the ROS problem, namely the *modified ROS (MROS) problem* and the *weighted fractional ROS (WFROS) problem* that are not affected by the above-mentioned attacks¹.

MROS Problem: In connection with the clause blind Schnorr signature scheme, Fuchsbauer, Plouviez, and Seurin [FPS19] proposed the modified ROS (MROS) problem that presumably exponentially increases the attack complexity. Informally, the MROS problem is about finding $l + 1$ tuples of two vectors $\vec{\rho}_{i,0}, \vec{\rho}_{i,1}$, and two solution vectors $\vec{c}_0, \vec{c}_1$ all in $\mathbb{Z}_p^l$ in the following way: For each $j \in [l]$, the adversary has to commit $c_{j,0}, c_{j,1}$ after which the game returns a randomly sampled bit $b_j \leftarrow \{0, 1\}$. This bit b_j then determines which vector's element $\rho_{i,b_j,j}$ and solution $c_{j,b}$ to be used in the MROS equation. Furthermore, the bit's complement $1 - b_j$ constraints that $\rho_{i,1-b_j,j} = 0$ for all $i \in [l + 1]$. This measure essentially forces the adversary to initially guess the l bits b_j for $j \in [l]$ by setting the ρ_i vectors accordingly and solve the standard ROS problem.

According to the assessment of Fuchsbauer, the attack complexity for the best attack against the MROS problem using Wagner's algorithm thus would be increased by the exponential factor of 2^{-l} , which results in $O(2^l \cdot (l + 1) 2^{\lambda/(1+\lfloor \log(l+1) \rfloor)})$.

It is obvious to see that this directly thwarts the polynomial-time attack by Benhamouda as the attack requires that $l \geq \lambda$.

On the other hand, as pointed out by Tessaro et al. [TZ22], the complexity still becomes sub-exponentially in $2^{O(\lambda/\log \lambda)}$ with the worst case l . This fact forces employing a larger security parameter

¹The attack by Wagner and the generalized attack by Benhamouda still can be applied to both variants but with an exponentially decreased success probability.

in practice, which is still reasonable considering the sub-exponential attacks against the original ROS problem.

We now recall the game-based definition borrowed from Fuchsbauer et al. [FPS19].

Definition 5 (Modified ROS Problem). *The modified ROS problem is defined by the security game MROS_l^A depicted in Figure 2.4 with respect to an integer $l > 1$, where $H_{\text{ros}} : \mathbb{Z}_p^l \times \mathbb{Z}_p^l \times \Omega \rightarrow \mathbb{Z}_p$ is a random oracle. The modified ROS assumption holds, if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{MROS}_l^A(\lambda) = 1] \leq \text{negl}(\lambda)$ for a negligible function $\text{negl}(\lambda)$.*

Game $\text{MROS}_l^A(\lambda)$	Oracle $H_{\text{ros}}(\vec{p}_0, \vec{p}_1, aux_i)$
1 : $(p, \mathbb{G}, G) \leftarrow \text{GrGen}(\lambda); T_{\text{ros}} := ()$	1 : if $T_{\text{ros}}(\vec{p}_0, \vec{p}_1, aux_i) = \perp$ then
2 : $(\vec{p}_{i,0}, \vec{p}_{i,1}, aux_i)_{i \in [l+1]} \leftarrow \mathcal{A}^{H_{\text{ros}}, \text{Select}}(p)$	2 : $T_{\text{ros}}(\vec{p}_0, \vec{p}_1, aux_i) \leftarrow \mathbb{Z}_p$
3 : return $(\forall i \neq i' \in [l+1] : (\vec{p}_{i,0}, \vec{p}_{i,1}, aux_i) \neq (\vec{p}_{i',0}, \vec{p}_{i',1}, aux_{i'}) \wedge$	3 : fi
4 : $\forall i \in [l+1] : \sum_{j=1}^l \rho_{i,b_j,j} c_j \equiv_p H_{\text{ros}}(\vec{p}_{i,0}, \vec{p}_{i,1}, aux_i) \wedge$	4 : return $T_{\text{ros}}(\vec{p}_0, \vec{p}_1, aux_i)$
5 : $\forall i \in [l+1] \forall j \in [l] : \rho_{i,1-b_j,j} = 0)$	<u>Oracle $\text{Select}(j, c'_0, c'_1)$</u>
	1 : $b \leftarrow \{0, 1\}$
	2 : $c_j \leftarrow c'_b$
	3 : return b

Figure 2.4: The MROS game.

WFROS Problem: The idea behind the weighted fractional ROS problem can be best conceived by considering the exemplary interactive Schnorr scheme by Tessaro and Zhu [TZ22], which the authors later on converted into a blind signature scheme. In the signature protocol, the signer S , having a secret key $x \in \mathbb{Z}_p$, communicates with the user U who requests a signature on a message m w.r.t. the public key $X = xG$ as follows:

1. The signer generates two nonces $r, y \leftarrow \mathbb{Z}_p$ and sends (R, Y) to the user, where $R \leftarrow rG$ and $Y \leftarrow yX$.
2. The user replies with the challenge $c \leftarrow H(R, Y, m)$.
3. The signer responds with the signature $\sigma := (R, s, y)$, where $s \leftarrow r + cxy$.

To verify the signature, one first computes $Y \leftarrow yX$ and $c \leftarrow H(R, Y, M)$, and then checks if $sG = R + cY$ holds.

Applying a ROS-style attack to the exemplary scheme would then result in solving the following modified ROS equation:

$$\forall i \in [l+1] : \sum_{j=1}^l \rho_{i,j} y_j c_j \equiv_p H_{\text{ros}}(\vec{\rho}_i, aux_i) \sum_{j=1}^l \rho_{i,j} y_j \quad (2.3)$$

The intuition given by Tessaro and Zhu on why this problem is much harder than the ROS problem is that $y_j X$ completely hides y_j which is only revealed later after the adversary has committed to c_j . Before this step, y_j consequently acts as a random and unknown weight in the sum. However, this intuition only holds in the generic group model and demands additional techniques in the algebraic group model.

The actual WFROS problem is defined in a more general form and is not recapitulated here for the sake of readability.

Another approach of avoiding ROS-based attacks for blind Schnorr signatures has been proposed by Fuchsbauer and Wolf [FW22] using non-interactive zero-knowledge proofs and public-key encryption.

The core idea is that the user commits to its blinding values and message before having received the nonce from the signer and to prove that the challenge has been indeed computed by using the committed values. Therefore, the adversarial user cannot freely adapt the resulting challenge after learning the nonces in the l parallel signing sessions to solve the ROS equation. Yet, this approach has two significant downsides. First, it requires an additional communication round for committing the encrypted blinding values and the message. And more importantly, it consequently requires an expensive zero-knowledge proof that the encrypted values have been included in the computation of the hash for the resulting challenge.

2.5 Building Blocks

2.5.1 Threshold Secret Sharing

The concept of threshold secret sharing was first introduced by Blakley and Shamir in 1979 [Bla79; Sha79]. A (t, n) -threshold secret sharing scheme allows sharing a secret among n parties such that only subgroups of at least t members can reconstruct the original secret. However, having less than t shares should not reveal any information about the secret.

We now briefly recall the (t, n) -secret sharing scheme SSS by Shamir [Sha79].

For the parameters $t, n \in \mathbb{N}$ with $1 \leq t \leq n$ and given a group description $(p, \mathbb{G}, G) \leftarrow \$ \text{GrGen}(1^\lambda)$ and a secret $s \in \mathbb{Z}_p$, the dealer proceeds as follows: The dealer first randomly chooses the coefficients $a_1, \dots, a_{t-1} \leftarrow \$ \mathbb{Z}_p$ and sets the polynomial $f(x) = a_0 + a_1x + \dots + a_{t-1}x^{t-1}$ with $a_0 = s$. Each share for a party P_i with $i \in [n]$ is then computed as $s_i \leftarrow f(i)$ and sent to P_i .

Using polynomial interpolation, any subgroup of at least t parties can now reconstruct the polynomial $f(x)$ with their shares and obtain the secret by computing $s \leftarrow f(0)$.

2.5.2 Schnorr Signature Scheme

A traditional signature scheme Π allows a signer to sign a message with its secret key such that anyone can verify the signature using the associated public key.

We provide the formal definitions below and recall the Schnorr signature scheme as an exemplary instantiation.

Definition 6 (Signature scheme Π). *A signature scheme Π over a message space $\mathcal{M}$ is a tuple of four algorithms $\Pi = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify})$ that are defined as follows:*

- $pp \leftarrow \$ \text{Setup}(1^\lambda)$: *The PPT algorithm Setup takes the security parameter 1^λ as an input and outputs the public parameters pp .*
- $(sk, pk) \leftarrow \$ \text{KeyGen}(pp)$: *On input the public parameters pp , the PPT algorithm KeyGen, outputs a secret key sk and a public key pk .*
- $\sigma \leftarrow \$ \text{Sign}(sk, m)$: *On input the secret key sk and a message $m \in \mathcal{M}$, the PPT algorithm Sign outputs a signature σ .*
- $0/1 \leftarrow \text{Verify}(pk, m, \sigma)$: *On input a public key pk , a message $m \in \mathcal{M}$, and a signature σ , the DPT algorithm Verify outputs 1 if σ is a valid signature on the message m with respect to pk or 0, otherwise.*

A signature scheme Π has to satisfy correctness and unforgeability, which we will now formally define. Correctness for signature schemes requires that all honestly generated signatures successfully verify.

Definition 7 (Correctness for Π). *A signature scheme Π satisfies correctness, if for all $\lambda \in \mathbb{N}$ and all messages $m \in \mathcal{M}$, it holds that:*

$$\Pr \left[\begin{array}{l} pp \leftarrow \$ \Pi.\text{Setup}(1^\lambda); \\ (sk, pk) \leftarrow \$ \Pi.\text{KeyGen}(pp); \quad : \quad \Pi.\text{Verify}(pk, m, \sigma) = 1 \\ \sigma \leftarrow \$ \Pi.\text{Sign}(sk, m) \end{array} \right] = 1$$

Unforgeability demands that an adversary, which is given a public key and access to a signing oracle, cannot produce a forgery, i.e., a fresh signature message pair, that verifies under the given public key. We capture the unforgeability property by existential unforgeability under a chosen-message attack (EUF-CMA).

Definition 8 (EUF-CMA for Π). *A signature scheme Π satisfies strong existential unforgeability under chosen-message attack if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{EUF-CMA}^{\mathcal{A}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$ for a negligible function $\text{negl}(\lambda)$ and the security game EUF-CMA defined in Figure 2.5.*

Game $\text{EUF-CMA}^{\mathcal{A}}(\lambda)$:

- **Setup Phase:** The challenger sets up the public parameters $pp \leftarrow \$ \Pi.\text{Setup}(1^\lambda)$ and a key-pair $(sk, pk) \leftarrow \$ \Pi.\text{KeyGen}(pp)$, and initializes a set $Q \leftarrow \emptyset$. Finally, it runs the adversary $\mathcal{A}$ on input the public key pk .
- **Online Phase:** $\mathcal{A}$ is given access to the oracle OSign_Π which is given a message $m \in \mathcal{M}$ and outputs a signature $\sigma \leftarrow \$ \Pi.\text{Sign}(sk, m)$. It also updates the set Q as $Q \leftarrow Q \cup \{(m, \sigma)\}$.
- **Output Phase:** $\mathcal{A}$ wins the game if it outputs a forgery (m^*, σ^*) such that $(m^*, \sigma^*) \notin Q$ and $\Pi.\text{Verify}(pk, m^*, \sigma^*) = 1$.

Figure 2.5: The security game for strong existential unforgeability under chosen-message attack for signature schemes.

We now recap one of the most popular signature schemes, the Schnorr signature scheme Schn introduced by Schnorr in 1989/1991 [Sch89; Sch91]:

Setup: The algorithm $\text{Schn.Setup}(1^\lambda)$ takes the security parameter 1^λ as an input and outputs the public parameters $pp := (p, \mathbb{G}, G, H)$, where $(p, \mathbb{G}, G)$ are obtained by running $\text{GrGen}(1^\lambda)$ and H is a hash function with $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$.

Key Generation: The algorithm $\text{Schn.KeyGen}(pp)$ first parses the public parameters as $(p, \mathbb{G}, G, H) := pp$, samples a random value $x \leftarrow \$ \mathbb{Z}_p$, calculates $X \leftarrow xG$, and outputs the secret key as $sk := x$ and the public key as $pk := X$.

Signature Generation: The algorithm $\text{Schn.Sign}(sk, m)$ first parses the secret key as $x := sk$ and then proceeds as follows:

1. A nonce is sampled as $r \leftarrow \$ \mathbb{Z}_p$.
2. A challenge is computed as $c \leftarrow H(R, m)$, where $R \leftarrow rG$.
3. The signature $\sigma := (R, s)$ is outputted, where $s \leftarrow r + cx$.

Verification: The algorithm $\text{Schn.Verify}(pk, m, \sigma)$ parses the public key as $X := pk$ and the signature as $(R, s) := \sigma$, computes $c \leftarrow H(R, m)$ and outputs 1 if $sG = R + cX$ holds and 0, otherwise.

Correctness trivially holds as $sG = (r + cx)G = R + cX = R + H(R, m)X$. The unforgeability of Schnorr signatures has been proven under the discrete logarithm assumption in the random oracle model by Pointcheval and Stern [PS96b].

2.5.3 Equivocable Commitment Scheme

A traditional *commitment scheme* is a two-phase protocol that allows a sender to *commit* to a value in the first phase and to *open* the commitment in the second phase so that the receiver can verify that this was indeed the value that was committed to in the first phase. An *equivocable commitment scheme* is a commitment scheme with a special setup algorithm providing a trapdoor that allows opening the commitment to an arbitrary value [DG03]. We now provide the formal definitions which are influenced by [RS21; CRR21].

Definition 9 (Equivocable Commitment Scheme). *An equivocable commitment scheme over a message space $\mathcal{M}$ is a tuple of five algorithms $EC = (\text{Gen}, \text{Com}, \text{Open}, \text{SimSetup}, \text{Equiv})$ that are defined as follows:*

- $\text{crs} \leftarrow \$ \text{Setup}(1^\lambda)$: On input the security parameter 1^λ , the PPT algorithm Setup outputs the common reference string crs .
- $(\text{co}, \text{op}) \leftarrow \$ \text{Com}(\text{crs}, m)$: On input a common reference string crs and a message $m \in \mathcal{M}$ to be committed on, the PPT algorithm Com outputs the commitment co and an opening value op .
- $m/\perp \leftarrow \text{Open}(\text{crs}, \text{co}, \text{op})$: On input a common reference string crs , a commitment co , and an opening value op , the DPT algorithm Open outputs either a message $m \in \mathcal{M}$ or $\perp$.
- $(\widetilde{\text{crs}}, \tau) \leftarrow \$ \text{SimSetup}(1^\lambda)$: On input the security parameter 1^λ , the PPT algorithm SimSetup outputs a common reference string $\widetilde{\text{crs}}$ and an equivocation trapdoor τ .
- $\widetilde{\text{op}} \leftarrow \$ \text{Equiv}(\tau, \text{co}, m)$: On input a trapdoor τ , a commitment co , and a message $m \in \mathcal{M}$, the PPT algorithm Equiv outputs an opening value $\widetilde{\text{op}}$.

As for traditional commitment schemes, an equivocable commitment scheme EC has to fulfill the properties *correctness*, *binding*, and *hiding*. Additionally, an equivocable commitment scheme needs to fulfill *equivocation correctness* and *equivocation indistinguishability*. We provide formal definitions below.

The first property, correctness, requires that honestly generated commitments open to the same message that has been committed to.

Definition 10 (Correctness for EC). *An (equivocable) commitment scheme is correct if for all $\lambda \in \mathbb{N}$ and all $m \in \mathcal{M}$ the following holds:*

$$\Pr \left[\text{crs} \leftarrow \$ EC.\text{Setup}(1^\lambda); (\text{co}, \text{op}) \leftarrow \$ EC.\text{Com}(\text{crs}, m) : EC.\text{Open}(\text{crs}, \text{co}, \text{op}) = m \right] = 1$$

Similarly to standard correctness, equivocation correctness ensures that equivocated commitments open successfully to the message specified when generating the opening value.

Definition 11 (Equivocation Correctness for EC). *An equivocable commitment scheme fulfills equivocation correctness if for all $\lambda \in \mathbb{N}$ and all $m, m' \in \mathcal{M}$ the following holds:*

$$\Pr \left[\begin{array}{l} (\widetilde{\text{crs}}, \tau) \leftarrow \$ EC.\text{SimSetup}(1^\lambda); \\ (\text{co}, \text{op}) \leftarrow \$ EC.\text{Com}(\widetilde{\text{crs}}, m); \quad : EC.\text{Open}(\widetilde{\text{crs}}, \text{co}, \widetilde{\text{op}}) = m' \\ \widetilde{\text{op}} \leftarrow \$ EC.\text{Equiv}(\tau, \text{co}, m') \end{array} \right] = 1$$

Intuitively, the binding property states that without knowledge of the trapdoor, it is hard to come up with a different opening value that opens the same commitment to a different message, i.e., the sender cannot change its committed value afterward.

Definition 12 (Binding for EC). *An (equivocable) commitment scheme is computationally binding if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{Binding}_{EC}^{\mathcal{A}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$ for a negligible function $\text{negl}(x)$ and the security game Binding_{EC} specified in Figure 2.6.*

Game $\text{Binding}_{\text{EC}}^{\mathcal{A}}(\lambda)$	Game $\text{Hiding}_{\text{Com}}^{\mathcal{A}}(\lambda)$
1 : $\text{crs} \leftarrow \$ \text{EC.Setup}(1^\lambda)$	1 : $\text{crs} \leftarrow \$ \text{EC.Setup}(1^\lambda)$
2 : $(\text{co}, \text{op}, \text{op}') \leftarrow \$ \mathcal{A}(\text{crs})$	2 : $(m_0, m_1) \leftarrow \$ \mathcal{A}(\text{crs})$
3 : $m \leftarrow \text{EC.Open}(\text{crs}, \text{co}, \text{op})$	3 : $b \leftarrow \$ \{0, 1\}$
4 : $m' \leftarrow \text{EC.Open}(\text{crs}, \text{co}, \text{op}')$	4 : $(\text{co}, \text{op}) \leftarrow \$ \text{EC.Com}(\text{crs}, m_b)$
5 : return $((m \neq m') \wedge (m, m' \in \mathcal{M}))$	5 : $b' \leftarrow \$ \mathcal{A}(\text{co})$
	6 : return $(b' = b)$

Figure 2.6: The security games for binding and hiding of (equivocable) commitment schemes.

The hiding property of (equivocable) commitment schemes ensures that it is infeasible to infer the committed value given a commitment.

Definition 13 (Hiding for EC). *An (equivocable) commitment scheme is computationally hiding if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{Hiding}_{\text{EC}}^{\mathcal{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$ for a negligible function $\text{negl}(\lambda)$ and the security game $\text{Hiding}_{\text{EC}}$ specified in Figure 2.6.*

The last requirement of equivocable commitment schemes is that honest and equivocable commitments are indistinguishable. This notion is captured with equivocation indistinguishability.

Definition 14 (Equivocation Indistinguishability for EC). *An equivocable commitment scheme satisfies equivocation indistinguishability if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{EqInd}_{\text{EC}}^{\mathcal{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$ for a negligible function $\text{negl}(\lambda)$ and the security game EqInd_{EC} specified in Figure 2.7.*

Game $\text{EqInd}_{\text{EC}}^{\mathcal{A}}(\lambda)$
1 : $m \leftarrow \$ \mathcal{A}(1^\lambda)$
2 : $\text{crs}_0 \leftarrow \$ \text{EC.Setup}(1^\lambda)$
3 : $(\text{crs}_1, \tau) \leftarrow \$ \text{EC.SimSetup}(1^\lambda)$
4 : $b \leftarrow \$ \{0, 1\}$
5 : $(\text{co}, \text{op}_0) \leftarrow \$ \text{EC.Com}(\text{crs}_b, m)$
6 : $\text{op}_1 \leftarrow \$ \text{EC.Equiv}(\tau, \text{co}, m)$
7 : $b' \leftarrow \$ \mathcal{A}(\text{crs}_b, \text{co}, \text{op}_b)$
8 : return $(b' = b)$

Figure 2.7: The security game for equivocation indistinguishability.

2.5.4 Non-Interactive Zero-Knowledge Proof of Knowledge

A non-interactive zero-knowledge proof system of knowledge NIZK allows a prover to non-interactively prove to a verifier that it knows a witness w for a statement ϕ satisfying a binary relation $\mathcal{R}$ without revealing any information about this witness. The language $\mathcal{L}_{\mathcal{R}}$ for a relation $\mathcal{R}$ is defined as the set of all statements for which a fulfilling witness exists, i.e., $\mathcal{L}_{\mathcal{R}} = \{\phi \mid \exists w \text{ s.t. } (\phi, w) \in \mathcal{R}\}$. The following definitions are based on Groth et al. and Haralambiev [GM17; Har11].

Definition 15 (NIZK Proof System). *A non-interactive zero-knowledge proof system is a tuple of six algorithms $\text{NIZK} = (\text{Setup}, \text{Prove}, \text{Verify}, \text{SimSetup}, \text{SimProve}, \text{Extract})$ that are defined as follows:*

- $\text{crs} \leftarrow \$ \text{Setup}(1^\lambda, \mathcal{R})$: On input the security parameter 1^λ and a relation $\mathcal{R}$, the PPT algorithm **Setup** generates a common reference string crs .

- $\pi \leftarrow \$ \text{Prove}(crs, \phi, w)$: On input the common reference string crs , a statement ϕ , and a witness w satisfying $(\phi, w) \in \mathcal{R}$, the PPT algorithm **Prove** outputs a proof π .
- $0/1 \leftarrow \text{Verify}(crs, \phi, \pi)$: On input the common reference string crs , a statement ϕ , and a proof π , the DPT algorithm **Verify** outputs 1 if the proof proves knowledge of a witness w s.t. $(\phi, w) \in \mathcal{R}$ and 0, otherwise.
- $(\widetilde{crs}, \tau, \xi) \leftarrow \$ \text{SimSetup}(1^\lambda, \mathcal{R})$: On input the security parameter 1^λ and a relation $\mathcal{R}$, the PPT algorithm **SimSetup** outputs a common reference string $\widetilde{crs}$, a simulation trapdoor τ , and an extraction trapdoor ξ .
- $\pi \leftarrow \$ \text{SimProve}(\tau, \phi)$: On input the simulation trapdoor τ and a statement ϕ , the PPT algorithm **SimProve** outputs a proof π .
- $w \leftarrow \text{Extract}(\xi, \phi, \pi)$: On input the extraction trapdoor ξ , a statement ϕ , and a proof π , the DPT algorithm **Extract** outputs the extracted witness w .

We demand that a NIZK proof system of knowledge has to fulfill the properties *perfect correctness*, *perfect soundness*, *zero-knowledge*, and *knowledge extraction* for which we provide the formal definitions below.

Correctness for NIZK proof systems demands that all honestly generated proofs verify, i.e., a prover can convince a verifier of a valid statement.

Definition 16 (Correctness for NIZK). *A NIZK scheme is perfectly correct if for all $\lambda \in \mathbb{N}$, all relations $\mathcal{R}$, and all $(\phi, w) \in \mathcal{R}$, the following holds:*

$$\Pr \left[crs \leftarrow \$ \text{NIZK.Setup}(1^\lambda, \mathcal{R}); \pi \leftarrow \$ \text{NIZK.Prove}(crs, \phi, w) : \text{NIZK.Verify}(crs, \phi, \pi) = 1 \right] = 1$$

Soundness means that it should be infeasible for an adversary to create a validating proof for a false statement.

Definition 17 (Soundness for NIZK). *A NIZK scheme is perfectly sound if for all $\lambda \in \mathbb{N}$, all relations $\mathcal{R}$, and all PPT adversaries $\mathcal{A}$, the following holds:*

$$\Pr \left[crs \leftarrow \$ \text{NIZK.Setup}(1^\lambda, \mathcal{R}); (\phi, \pi) \leftarrow \$ \mathcal{A}(crs) : (\phi \notin \mathcal{L}_{\mathcal{R}}) \wedge (\text{NIZK.Verify}(crs, \phi, \pi) = 1) \right] = 0$$

Intuitively, zero-knowledge requires that an adversary should not be able to distinguish real proofs from simulated proofs.

Definition 18 (Zero-Knowledge for NIZK). *A NIZK scheme satisfies zero knowledge if for all PPT adversaries $\mathcal{A}$ it holds that $\Pr[\text{ZeroK}_{\text{NIZK}}^{\mathcal{A}}(1^\lambda)] = \frac{1}{2} + \text{negl}(\lambda)$ for the security game $\text{ZeroK}_{\text{NIZK}}$ depicted in Figure 2.8 and a negligible function $\text{negl}(\lambda)$.*

Game $\text{ZeroK}_{\text{NIZK}}^{\mathcal{A}}(\lambda)$	Oracle $\text{OProve}(\phi_i, w_i)$
1 : $crs_0 \leftarrow \$ \text{NIZK.Setup}(1^\lambda, \mathcal{R})$	1 : assert $(\phi_i, w_i) \in \mathcal{R}$
2 : $crs_1, \tau \leftarrow \$ \text{NIZK.SimSetup}(1^\lambda, \mathcal{R})$	2 : if $b = 0$ then
3 : $b \leftarrow \$ \{0, 1\}$	3 : $\pi_i \leftarrow \$ \text{NIZK.Prove}(crs_0, \phi_i, w_i)$
4 : $b' \leftarrow \$ \mathcal{A}^{\text{OProve}}(crs_b)$	4 : else
5 : return $(b' = b)$	5 : $\pi_i \leftarrow \$ \text{NIZK.SimProve}(\tau, \phi_i)$
	6 : fi
	7 : return π_i

Figure 2.8: The security game for perfect zero knowledge of NIZK schemes.

Knowledge extraction ensures that the prover actually "knows" the witness which is shown by the fact that for every verifying proof, there exists an extractor that can extract a valid witness.

Definition 19 (Knowledge Extraction). *A NIZK scheme satisfies knowledge extraction if for all PPT adversaries $\mathcal{A}$, it holds that $\Pr[\text{KnExtr}_{\text{NIZK}}^{\mathcal{A}}(1^\lambda) = 1] = \text{negl}(\lambda)$ for the security game $\text{KnExtr}_{\text{NIZK}}$ depicted in Figure 2.9 and a negligible function $\text{negl}(\lambda)$.*

Game $\text{KnExtr}_{\text{NIZK}}^{\mathcal{A}}(\lambda)$
1 : $(crs, \tau, \xi) \leftarrow \$ \text{NIZK.SimSetup}(1^\lambda, \mathcal{R})$
2 : $(\phi, \pi) \leftarrow \$ \mathcal{A}(crs)$
3 : $w \leftarrow \text{NIZK.Extract}(\xi, \phi, \pi)$
4 : return $(\text{NIZK.Verify}(crs, \phi, \pi) \wedge (\phi, w) \notin \mathcal{R})$

Figure 2.9: The security game for knowledge extraction of NIZK schemes.

3 Threshold Signatures

In this chapter, we give some background about threshold signatures with a focus on schemes based on Schnorr and provide definitions for threshold signature schemes.

3.1 Background

In recent times, threshold signatures have received increasing attention due to their applicability for securing cryptocurrency wallets, where the loss of the secret key results in the loss of the coins' ownership. Generally speaking, threshold signatures eliminate the risk of a single point of failure in the context of signature creation by sharing the secret key among n participants such that only groups of size at least t (the threshold) can create a signature. The final signature can be verified as any other signature and is thus indistinguishable from a traditional signature that has been created by a single party.

Threshold signature schemes assume either an *honest* or *dishonest majority*, referring to whether security guarantees only hold if the majority of the participants behave honestly, or if no such assumption is required. In both cases, the adversary is trivially allowed to corrupt up to $t - 1$ participants; for an honest majority, it is further required that $t < \frac{1}{2} \cdot n$ holds.

The property *robustness* in the context of threshold signatures means that the honest signers are still able to create a signature even if the other signers deviate from the protocol; thus all robust schemes require an honest majority. One can consider *identifiable aborts* as an alternative for robustness in the dishonest majority setting. When a scheme supports identifiable aborts, it guarantees that misbehaving signers can be identified before the signing protocol aborts such that those signers can be excluded from future executions.

While the secret key can be shared by a trusted party, e.g., via Shamir's secret sharing [Sha79] or Pedersen's verifiable secret sharing (VSS) [Ped01], the shares can also be generated in a distributed fashion as with Gennaro's DKG [Gen+99], which is based upon Feldman's earlier approach [Fel87], without exposing the secret key.

If a threshold signature scheme supports *proactive* security such as, it has a mechanism to periodically update the key shares to prevent an adversary from reconstructing the secret key in case the adversary learns some secret shares over a period of time. Exemplary threshold signature schemes that achieve proactive security are [Her+97; Bol03; Can+20], and regarding Schnorr-based schemes, the scheme by Lindell [Lin22].

Existing constructions range from various underlying cryptosystems such as RSA [Sho00; SG02; De+94; Gen+96], ECSDA/DSA [Can+20; GG18; GGN16; LN18], BLS [Bol02; BDN18], or Schnorr [SS01; KG20; Lin22; Gar+21]. Since the bottleneck often lies in the signing phase with multiple signing rounds, which even becomes more costly with increasing participants, fewer communication rounds are favorable. Ideally, the signing phase is non-interactive, which, however, has been so far only achieved with pairing-based constructions such as [Bol02; BDN18; LYK07] that are still considered rather expensive.

Regarding Schnorr signatures, which are already widely used in cryptocurrencies, threshold variants are especially promising due to their linearity and efficient signature computation, and verification.

Already in 2001, Stinson and Stroh [SS01] proposed a threshold signature scheme and proved that it is as secure as the original Schnorr scheme. They use Gennaro's VSS for the key - and the signature generation, which further provides robustness assuming an honest majority. As a downside for the robustness, the protocol upholds a high communication overhead since each player has to perform one run of the VSS protocol for generating a shared nonce.

The popular FROST scheme by Komlo and Goldberg [KG20] only requires a single communication round in the signing phase by generating the nonces in a non-interactive preprocessing stage. In the original paper, the authors only proved their scheme with a heuristic to an interactive variant of FROST, later work [CKM21] presented a proof of FROST (albeit slightly modified) by reducing it to the DL and OMDL assumption in the AGM using intermediate assumptions.

In the work of Garillot et al. [Gar+21], the authors presented a stateless, deterministic threshold Schnorr signature scheme with verifiable nonce derivation in order to prevent nonce reuse or biasing. Verifiable nonce derivation is achieved with commitments of the PRF key in the key-generation phase and making use of zero-knowledge proofs from garbled circuits and oblivious transfers during the signing phase. Besides relatively expensive computations, the bottleneck for the communication results from the fact that each pair of the n participants have to run the deterministic nonce derivation protocol, once as the prover garbling the circuit and once as the verifier.

Lindell recently published a threshold Schnorr scheme [Lin22] representing a simple three-round construction that reduces its security to the assumption that the Schnorr signature scheme itself is secure resulting in a proof in the standard model, which is further UC-secure.

In Table 3.1, we compare different threshold Schnorr signature schemes in terms of assumptions, rounds, communication and computation complexity per signer, robustness, and whether it requires a preprocessing phase or zero-knowledge proofs (ZKPs). Regarding the communication and computation complexity, we assume a signature dealer in lieu of broadcasting messages to all signers and verifying and combining the partial signatures. The dealer enables us to consider broadcasting as a constant operation.

Table 3.1: Comparison of threshold Schnorr signature schemes.

Scheme	Assumptions	Rounds	Complexity		Robust	Preprocess	ZKPs
			Comm.	Comp.			
[SS01]	Schnorr, RO	3	$O(n)$	$O(n)$	Yes	No	No
[KG20]	DL, RO	1	$O(1)$	$O(n)^{\dagger_1}$	No	Yes	No
[Gar+21]	RO	$3^{\dagger_2}$	$O(n)$	$O(n)$	No	No	Yes
[Ruf+22]	OMDL, RO	1	$O(n^{t-1})$	$O(n^{t-1})$	Yes	Yes	No
[Lin22]	Schnorr	3	$O(1)$	$O(1)$	No	No	Yes $\dagger_3$

$\dagger_1$: Can be reduced to $O(1)$ using an optimization by Crites et al. [CKM21].

$\dagger_2$: Each pair of signers additionally has to execute a deterministic nonce derivation protocol.

$\dagger_3$: Note that the NIZK proof only consists of proving knowledge of a discrete logarithm which could be just instantiated via Schnorr.

3.2 Definition

We now provide the formal definition of a (t, n) -threshold signature scheme and their corresponding security notions.

Definition 20 (Threshold Signature TS). *A (t, n) -threshold signature scheme TS consists of a tuple of four algorithms $TS = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify})$ along with an associated message space $\mathcal{M}$ that are defined as follows:*

- $pp \leftarrow \$ \text{Setup}(1^\lambda)$: On input the security parameter 1^λ , the PPT algorithm **Setup** outputs the public parameters pp , which might include common reference strings used within the other algorithms.
- $(\{sk_i, vk_i\}_{i \in \mathcal{S}}, pk) \leftarrow \$ \text{KeyGen}(pp, t, n)$: On input the public parameters pp , and two integers $t, n \in \mathbb{N}$ with $1 \leq t \leq n$, the PPT algorithm **KeyGen** outputs a secret key share sk_i and a verification key share vk_i for every signer S_i and the common public key pk . The signers' secret key shares sk_i form a (t, n) -threshold sharing of the shared secret corresponding to the common public key pk .

Game $\text{Unforge}_{\text{TS}}^{\mathcal{A}}(\lambda)$:

- **Setup Phase:** The challenger sets up the public parameters $pp \leftarrow \$ \text{TS.Setup}(1^\lambda)$ and the key-pairs $(\{sk_i, vk_i\}_{i \in \mathcal{S}}, pk) \leftarrow \$ \text{TS.KeyGen}(pp, t, n)$ for $i \in \mathcal{S}$, and sets $\mathcal{V} := \{vk_i\}_{i \in \mathcal{S}}$. Also, it initializes an empty set $Q \leftarrow \emptyset$. Finally, it runs the adversary $\mathcal{A}$ on input the public key pk .
- **Corruption Phase:** $\mathcal{A}$ outputs a set $\mathcal{S}_C$ of signer indices it wants to corrupt, where $\mathcal{S}_C \subset \mathcal{S}$ and $|\mathcal{S}_C| \leq t - 1$. The challenger then provides $\mathcal{A}$ with the secret keys $\{sk_i\}_{i \in \mathcal{S}_C}$ of the corrupted parties.
- **Online Phase:** $\mathcal{A}$ is given access to the oracle OSign_{TS} which on input a message m executes the signing protocol on behalf of the honest signers. $\mathcal{A}$ is also allowed to interleave the various oracle calls arbitrarily. For each finished execution of the protocol, the set Q is updated as $Q \leftarrow Q \cup \{(m, \sigma)\}$.
- **Output Phase:** $\mathcal{A}$ wins the game if it outputs a forgery (m^*, σ^*) where the signature is valid and fresh, i.e., $\text{TS.Verify}(pk, m, \sigma^*) = 1$ and $(m^*, \sigma) \notin Q$.

Figure 3.1: The strong unforgeability security game for threshold signature schemes.

- $\sigma \leftarrow \$ \langle \text{Sign}(sk_1, \mathcal{V}, m), \dots, \text{Sign}(sk_n, \mathcal{V}, m) \rangle$: The interactive signing protocol between the PPT signer algorithms Sign . Each algorithm Sign takes the secret key share sk_i of a signer S_i with $i \in \mathcal{S}$, the set of all verification key shares $\mathcal{V} := \{vk_i\}_{i \in \mathcal{S}}$, and the message $m \in \mathcal{M}$ as an input. In the end, the protocol outputs a signature σ , which can equal $\perp$ on failure.
- $0/1 \leftarrow \text{Verify}(pk, m, \sigma)$: On input the public key pk , a signature σ , and a message $m \in \mathcal{M}$, the DPT algorithm Verify outputs 1 if the signature on the message m is valid with respect to pk and 0, otherwise.

Correctness for a threshold signature scheme TS means that all signatures that are generated by honestly following the signing protocol have to verify.

Definition 21 (Correctness for TS). A (t, n) -threshold signature scheme TS is correct, if for all $\lambda \in \mathbb{N}$ and all messages $m \in \mathcal{M}$ followed by running $pp \leftarrow \$ \text{Setup}(1^\lambda)$, $(\{sk_i, vk_i\}_{i \in \mathcal{S}}, pk) \leftarrow \$ \text{KeyGen}(pp, t, n)$ for $i \in \mathcal{S}$ with $\mathcal{V} := \{vk_i\}_{i \in \mathcal{S}}$, we have:

$$\Pr[\sigma \leftarrow \$ \langle \text{Sign}(sk_1, \mathcal{V}, m), \dots, \text{Sign}(sk_n, \mathcal{V}, m) \rangle : \text{Verify}(pk, m, \sigma) = 1] = 1$$

A threshold signature scheme TS is secure if it satisfies unforgeability. For threshold signature schemes, this notion demands that an adversary, who is given a public key and up to $t - 1$ secret keys while having access to a signing oracle, cannot produce a valid forgery. The formal definition is given below.

Definition 22 (Unforgeability for TS). A (t, n) -threshold signature scheme TS is said to be unforgeable if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{Unforge}_{\text{TS}}^{\mathcal{A}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$ for a negligible function $\text{negl}(\lambda)$ and the security game $\text{Unforge}_{\text{TS}}$ defined in Figure 3.1.

4 Blind Signatures

This chapter introduces blind signatures and recalls the original blind Schnorr signature scheme by [CP92] and the clause blind Schnorr signature scheme by Fuchsbauer, Plouviez, and Seuvrin [FPS19] on which our threshold blind Schnorr signature scheme is based on.

4.1 Background

The concept of blind signatures was first envisioned by Chaum [Cha82] in the context of privacy-preserving e-cash systems. A blind signature scheme allows a user to obtain a signature on a message from the signer while guaranteeing that the signed message and the final signature remain unknown to the signer. The unforgeability property demands that the user should not be able to create a signature without the help of the signer.

Possible applications range from classical ideas like e-cash systems [Cha82; CFN88; OO91; Bra93; Bel+09] or e-voting [Cha88; FOO92; Ohk+99] to modern privacy critical topics like anonymous credentials [CL01; BL13a; Fuc11], anonymous attestations [BCC04], e-healthcare systems [Vor+18; Sun+21], COVID contact tracing systems [BRS20; Dar+20], and VPN related services as already offered by Apple and Google [App21; Goo].

In the area of blockchains and cryptocurrencies, blind signatures enable, for example, blind coin swaps or trustless tumblers [Nic18; Hei+16; Liu+19], and, as showcased in this thesis, privacy-enhancing online wallets.

Regarding the provable security of blind signature schemes, only a few constructions with their security being based on generic assumptions in the standard model have been published [Gar+11; Bla+13; Haz+07; GG14; CKW04]. Moreover, they are mostly feasibility results and are relatively inefficient due to bilinear pairings, homomorphic encryption, or expensive zero-knowledge proofs. On the other hand, there exist a variety of efficient schemes proven in the random oracle model, e.g., [PS96a; PS00; Abe01; Bol03; HKL19], whereas some of them rely on non-standard assumptions or even require stronger idealizations like the algebraic- or generic group model [Cha+04; FPS19; KLX22; TZ22]. In addition, with the advent of post-quantum cryptography, several blind signature constructions based on post-quantum secure assumptions, mostly lattices, have been proposed [Agr+22; AEB20; Pap+19; Rüc10].

Most notably, the Schnorr blind signature scheme proposed by Chaum [CP92], which is considered to be one of the most efficient blind signature schemes, can neither be proven secure in the standard model nor in the random oracle model without any stronger assumptions [FS10; BL13b]. Only in the generic group model in addition to the random oracle, Schnorr and Jakobsen [SJ05] were able to prove the one-more unforgeability. Fuchsbauer et al. [FPS19] later presented a proof in the algebraic group model; a less idealized model than the generic group model. However, the security of the Schnorr and Okamoto-Schnorr blind signature schemes relies on the ROS problem, which has been shown to be solvable in sub-exponential time already in 2002 [Wag02] and even in polynomial time in an attack published by Benhamouda et al. [Ben+22]. To overcome this vulnerability, Fuchsbauer et al. [FPS19] proposed a "clause" blind Schnorr signature scheme that builds upon a novel variant of the ROS problem, the modified ROS problem (MROS). The weighted fractional ROS problem (WFROS) is another variant proposed by Tessaro and Zhu [TZ22] along with their blind Schnorr signature scheme.

As the applications of blind signatures are versatile and demand different features, various blind signature variants have been proposed such as fair -, partially -, proxy, or conditional blind signatures. Fair blind signatures were introduced by Stadler et al. [SPC95] to restrict the unconditional unlinkability

against illegal misuse, allowing an authority to link a signature with the corresponding signing session. Partially blind signatures were introduced by Abe and Fujisaki [AF96] and guarantee that a signature contains some common information on which the signer and user have agreed to. For e-cash systems, this common information may be the value of the issued coin. The first proxy blind signature scheme was presented by Lin and Jan [LJ00] that allows delegating the signing authority to another party, the *proxy*, which then can blindly sign messages on behalf of the original signer. Conditional blind signatures [ZGP17] are designated verifier signatures where the signer can specify whether a signature is valid or not, which however remains oblivious for the user.

In Table 4.1, we compare different Schnorr blind signature schemes in terms of assumptions, moves of interactions, complexity (communication and computation), whether they are concurrently secure, require computation (and verification) of zero-knowledge proofs (ZKPs), and if they are compatible with Schnorr. Regarding the moves, we omit considering an initial interaction by the user to trigger the signature issuing as implicitly done in other works when specifying the rounds. The last property applies if a scheme uses the same verification algorithm as the standard Schnorr scheme such that it can be used in systems that already use Schnorr signatures.

Table 4.1: Comparison of blind Schnorr signature schemes.

Scheme	Assumptions	Moves of Interaction	Complexity	Concurrency	ZKPs	Compatible w/ Schnorr
[CP92]	OMDL, ROS, RO, AGM	3	$O(1)$	No	No	Yes
[FPS19]	OMDL, MROS, RO, AGM	3	$O(1)$	Yes	No	Yes
[KLR21]	Schnorr, RO	7	$O(l)^{\dagger_1}$	Yes	No	No
[TZ22] 1) ^{†₂}	OMDL, WFROS, RO, GGM	3	$O(1)$	Yes	No	No
[TZ22] 3) ^{†₂}	DL, WFROS, RO, AGM	3	$O(1)$	Yes	No	No
[FW22]	Schnorr, DL	4	$O(1)$	Yes	Yes	Yes

^{†₁}: l denotes the number of signing executions.

^{†₂}: With 1) and 3), we refer to the first proposed scheme, defined as BS_1 , and the third scheme BS_3 , respectively.

4.2 Definition

We follow the definitions from [FPS19; HKL19] for blind signature schemes and their security notions.

Definition 23 (Blind Signature Scheme). *A blind signature scheme is a tuple of five algorithms $BS = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{User}, \text{Verify})$ along with an associated message space $\mathcal{M}$ that are defined as follows:*

- $pp \leftarrow \$ \text{Setup}(1^\lambda)$: On input the security parameter 1^λ , the PPT algorithm **Setup** outputs the public parameters pp .
- $(sk, pk) \leftarrow \$ \text{KeyGen}(pp)$: On input the public parameters pp , the PPT algorithm **KeyGen**, outputs a secret key sk and a public key pk .
- $(b, \sigma) \leftarrow \$ \langle \text{Sign}(sk), \text{User}(pk, m) \rangle$: The interactive, randomized signing protocol between the signer algorithm **Sign** and the user algorithm **User**. The algorithm **Sign** takes as input the secret key sk and outputs a bit b which equals 1 to indicate success or 0 to indicate failure. The algorithm **User** takes the public key pk and a message $m \in \mathcal{M}$, and eventually outputs a signature σ which can equal $\perp$ to indicate failure.

- $0/1 \leftarrow \text{Verify}(pk, m, \sigma)$: On input a public key pk , a signature σ and a message $m \in \mathcal{M}$, the DPT algorithm Verify outputs either 1 or 0 indicating a valid or invalid signature on the message m with respect to pk .

A blind signature scheme BS must be correct, which requires that all signatures generated by honestly following the signing protocol, verify.

Definition 24 (Correctness for BS). *A blind signature scheme BS is correct, if for all $\lambda \in \mathbb{N}$ and all messages $m \in \mathcal{M}$ followed by running $pp \leftarrow \$ \text{Setup}(1^\lambda)$, $(sk, pk) \leftarrow \$ \text{KeyGen}(pp)$, we have:*

$$\Pr[(b, \sigma) \leftarrow \langle \text{Sign}(sk), \text{User}(pk, m) \rangle : (b = 1 \wedge \text{Verify}(pk, m, \sigma))] = 1$$

A blind signature scheme BS is secure if it satisfies unforgeability and blindness. We provide the formal definitions below.

Unforgeability in the context of blind signatures demands that no adversarial user should be able to output k valid message-signature pairs for a given public key after having requested only $k - 1$ signatures from the signer.

Definition 25 (Unforgeability for BS). *A blind signature scheme BS is unforgeable if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{Unforge}_{\text{BS}}^{\mathcal{A}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$ for a negligible function $\text{negl}(\lambda)$ and the security game $\text{Unforge}_{\text{BS}}$ defined in Figure 4.1.*

Game $\text{Unforge}_{\text{BS}}^{\mathcal{A}}(\lambda)$:

- **Setup Phase:** The challenger sets up the public parameters $pp \leftarrow \$ \text{BS.Setup}(1^\lambda)$ and a key-pair $(sk, pk) \leftarrow \$ \text{BS.KeyGen}(pp)$, and initializes $q_s \leftarrow 0$. Finally, it runs the adversary $\mathcal{A}$ on input the public key pk .
- **Online Phase:** $\mathcal{A}$ is given access to the oracle OSign_{BS} which executes the signing protocol $\langle \text{BS.Sign}(sk), \cdot \rangle$ on behalf of the signer, where $\mathcal{A}$ plays the role of the user. $\mathcal{A}$ is also allowed to interleave the various oracle calls arbitrarily. For each finished execution of the signing protocol, q_s is increased as $q_s \leftarrow q_s + 1$.
- **Output Phase:** $\mathcal{A}$ wins the game if it outputs a forgery of unique pairs $(m_i^*, \sigma_i^*)_{i \in [k]}$ where all signatures are valid, i.e., $\text{BS.Verify}(m_i^*, \sigma_i^*, pk) = 1$ for $i \in [k]$, and it holds that $k > q_s$.

Figure 4.1: The security game for unforgeability of blind signature schemes.

Blindness demands that an adversarial signer cannot link executions of the signing protocol with the signed message or the resulting signature. This notion is captured by showing that an adversary cannot decide which message m_0, m_1 was signed first after the user has obtained a signature on both messages in random order.

Definition 26 (Blindness for BS). *A blind signature scheme is blind if for all PPT adversaries $\mathcal{A}$, it holds that $\Pr[\text{Blind}_{\text{BS}}^{\mathcal{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$, for a negligible function $\text{negl}(\lambda)$ and the security game Blind_{BS} defined in Figure 4.2. If this equation is changed to $\Pr[\text{Blind}_{\text{BS}}^{\mathcal{A}}(1^\lambda) = 1] = \frac{1}{2}$ and holds for all unbounded adversaries, the scheme is considered perfectly blind.*

4.3 Blind Schnorr Signatures

We now recapitulate the original blind Schnorr signature scheme oBSS by Chaum [CP92] and the modified scheme BSS by Fuchsbauer et al. [FPS19], which reduces to the modified ROS problem. We further recap the security properties of the BSS scheme as we will later reduce the unforgeability and blindness of our threshold blind Schnorr signature construction to this scheme.

Game $\text{Blind}_{\text{BS}}^{\mathcal{A}}(\lambda)$:

- **Setup Phase:** The challenger runs the adversary $\mathcal{A}$ given 1^λ as an input who outputs a public key pk and two messages m_0, m_1 . The challenger then samples a bit $b \leftarrow \{0, 1\}$.
- **Online Phase:** $\mathcal{A}$ is given access to the oracles $\text{OUser}_{\text{BS},b}$ and $\text{OUser}_{\text{BS},1-b}$ which can be queried only once. The oracles execute the protocol $\langle \cdot, \text{BS.User}(pk, m_b) \rangle$ and $\langle \cdot, \text{BS.User}(pk, m_{1-b}) \rangle$, respectively, where $\mathcal{A}$ plays the role of the signer. The adversary is allowed to arbitrarily interleave between the oracle queries. Only after both executions have been successfully finished, i.e., $\sigma_b \neq \perp$ and $\sigma_{1-b} \neq \perp$ with $\sigma_b \leftarrow \text{BS.User}(pk, m_b)$ and $\sigma_{1-b} \leftarrow \text{BS.User}(pk, m_{1-b})$, the last oracle replies with (σ_b, σ_{1-b}) . If one of the signatures is invalid, it replies with $(\perp, \perp)$.
- **Output Phase:** $\mathcal{A}$ wins the game, if it outputs a bit b' such that $b' = b$.

Figure 4.2: The security game for blindness of blind signature schemes.

4.3.1 Original Blind Schnorr Signature Scheme

The original blind Schnorr signature scheme oBSS introduced by Chaum [CP92] relies on the ROS problem and therefore should be considered insecure. Nevertheless, for giving an intuition on how the ROS problem affects blind Schnorr signature schemes and for illustrating the modification by Fuchsbauer, we recap it here:

Setup: The setup algorithm oBSS.Setup is the same as Schn.Setup .

Key Generation: The key generation algorithm oBSS.KeyGen is the same as Schn.KeyGen .

Signature Generation: The signing protocol $\langle \text{oBSS.Sign}(sk), \text{oBSS.User}(pk, m) \rangle$ between the signer S and the user U works as follows:

1. The signer samples a nonce $r \leftarrow \mathbb{Z}_p$ and sends $R \leftarrow rG$ to the user.
2. The user samples two blinding values $\alpha, \beta \leftarrow \mathbb{Z}_p$, computes $R' \leftarrow R + \alpha G + \beta X$. It then computes $c' \leftarrow H(R', m)$ and responds with the challenge $c \leftarrow c' + \beta \bmod p$.
3. The signer responds with $s \leftarrow r + cx$ and outputs 1.
4. The user checks if $sG = R + cX$ holds, then sets $s' \leftarrow s + \alpha$ and outputs the signature $\sigma := (R', s')$.

Verification: The verification algorithm oBSS.Verify is the same as Schn.Verify .

Correctness of oBSS holds since when verifying a given signature $\sigma = (R', s')$ we have:

$$\begin{aligned} s'G &= sG + \alpha G = (r + cx)G + \alpha G = R + cX + \alpha G \\ &= R + \alpha G + \beta X + c'X = R' + H(R', m)X \end{aligned}$$

Theorem 1 (Unforgeability). *Let $\mathcal{A}_{\text{alg}}$ be an algebraic adversary against the unforgeability of the oBSS scheme running in time at most τ and making at most q_s queries to $\text{OSign}_{\text{oBSS}}$ and q_h queries to the random oracle. Then, there exist an adversary $\mathcal{B}_{\text{omdl}}$ against the one-more discrete logarithm problem making at most q_s queries to the Dlog oracle and an adversary $\mathcal{B}_{\text{ros}}$ against the ROS problem making at most $q_h + q_s + 1$ random oracle queries, both running in time at most $\tau + O(q_h + q_s)$ such that it holds:*

$$\Pr[\text{Unforge}_{\text{oBSS}}^{\mathcal{A}_{alg}}(1^\lambda) = 1] \leq \Pr[\text{OMDL}^{\mathcal{B}_{omdl}}(1^\lambda) = 1] + \Pr[\text{ROS}_l^{\mathcal{B}_{ros}}(1^\lambda) = 1]$$

Theorem 2 (Blindness). *The oBSS scheme satisfies perfect blindness. That is, for all adversaries $\mathcal{A}$ against the blindness of oBSS, it holds that:*

$$\Pr[\text{Blind}_{\text{oBSS}}^{\mathcal{A}}(1^\lambda) = 1] = \frac{1}{2}$$

4.3.2 Clause Blind Schnorr Signature Scheme

The clause blind Schnorr signature scheme by Fuchsbauer et al. [FPS19] is a modification of the previous scheme where two sessions of the signing protocol are run in parallel, while the signer only finishes one of them (giving it the name "clause" from finishing $\text{Run}_1 \vee \text{Run}_2$).

Setup: The setup algorithm BSS.Setup is the same as Schn.Setup .

Key Generation: The key generation algorithm BSS.KeyGen is the same as Schn.KeyGen .

Signature Generation: The signing protocol $\langle \text{BSS.Sign}(sk), \text{BSS.User}(pk, m) \rangle$ between the signer S and the user U works as follows:

1. The signer samples two nonces $r_0, r_1 \leftarrow \mathbb{Z}_p$ and sends $R_0 \leftarrow r_0G$ and $R_1 \leftarrow r_1G$ to the user.
2. The user performs the following steps for $k \in \{0, 1\}$: Sample blinding values $\alpha_k, \beta_k \leftarrow \mathbb{Z}_p$, compute $R'_k \leftarrow R_k + \alpha_k G + \beta_k X$, and compute the challenge as $c'_k \leftarrow \text{H}(R'_k, m)$ and $c_k \leftarrow c'_k + \beta_k \bmod p$. The challenge tuple (c_0, c_1) is then sent back to the signer.
3. The signer samples a bit $b \leftarrow \{0, 1\}$ and answers with (b, s) , where $s \leftarrow r_b + c_b x$. Lastly, it outputs 1.
4. The user checks if $sG = R_b + c_b X$ holds, then sets $s' \leftarrow s + \alpha_b$ and outputs the signature $\sigma := (R'_b, s')$.

Verification: The verification algorithm BSS.Verify is the same as Schn.Verify .

The proof of correctness for BSS is identical to oBSS.

The clause blind Schnorr signature scheme BSS by Fuchsbauer et al. [FPS19] is unforgeable assuming the hardness of the one-more discrete logarithm problem and modified ROS problem and satisfies perfect blindness. Here, we only recapitulate the security properties; for details on the proof carried out in the random oracle and algebraic group model, we refer to their paper.

Theorem 3 (Unforgeability). *Let $\mathcal{A}_{alg}$ be an algebraic adversary against the unforgeability of the BSS scheme running in time at most τ and making at most q_s queries to $\text{OSign}_{\text{BSS}}$ and q_h queries to the random oracle. Then, there exist an adversary $\mathcal{B}_{omdl}$ against the OMDL problem making at most q_s queries to the Dlog oracle and an adversary $\mathcal{B}_{mros}$ against the modified ROS problem making at most $q_h + q_s + 1$ random oracle queries, both running in time at most $\tau + O(q_h + q_s)$ such that it holds:*

$$\Pr[\text{Unforge}_{\text{BSS}}^{\mathcal{A}_{alg}}(1^\lambda) = 1] \leq \Pr[\text{OMDL}^{\mathcal{B}_{omdl}}(1^\lambda) = 1] + \Pr[\text{MROS}_l^{\mathcal{B}_{mros}}(1^\lambda) = 1]$$

Theorem 4 (Blindness). *For all PPT adversaries $\mathcal{A}$ against the blindness of the BSS scheme, there exist two PPT adversaries $\mathcal{B}_1, \mathcal{B}_2$ against the blindness of the oBSS scheme such that it holds:*

$$\Pr[\text{Blind}_{\text{BSS}}^{\mathcal{A}}(1^\lambda) = 1] \leq 4 \cdot \left(\Pr[\text{Blind}_{\text{oBSS}}^{\mathcal{B}_1}(1^\lambda) = 1] + \Pr[\text{Blind}_{\text{oBSS}}^{\mathcal{B}_2}(1^\lambda) = 1] \right)$$

Since the original blind Schnorr signature scheme oBSS by Chaum [CP92] is perfectly blind, the blind Schnorr signature scheme BSS by Fuchsbauer et al. [FPS19] is also perfectly blind.

4.3.3 Alternative Blind Schnorr Signature Schemes

In this section, we discuss other blind Schnorr signature schemes regarding their applicability for converting them to a threshold construction in the context of cryptographic wallets.

Katz et al. [KLR21] presented a generic transform of "boosting" the security of certain blind signature schemes (including Schnorr) under logarithmically many executions to polynomial many executions while maintaining possible concurrent security. Their transformation builds upon the idea of Pointcheval [Poi98], who originally proposed their transformation for the Okamoto-Schnorr blind signature scheme. The core idea from Pointcheval is a cut-and-choose technique, where the user and the signer always start two parallel sessions and let the user initially commit to its randomness to be used for the challenges. The signer later chooses which of the two commitments the user has to reveal and only on successful verification replies with the final signature value. If the verification fails, the signer has to halt forever and refuse any future signing queries. The intuition behind the transform is that if the user behaves honestly, the simulator can extract the randomness of both commits and can simulate signing without invoking the signer of the underlying blind signature scheme. If the user misbehaves and tries to cheat in one session, the simulator has to invoke the signer for the session where it cannot extract the randomness. However, as the user can only succeed in cheating with probability $\frac{1}{2}$ in each session, the probability that the signer of the blind signature scheme is invoked super-logarithmic times is negligible.

Katz et al. improved Pointcheval's transform such that the signer still can continue signing after detecting a cheating user. Furthermore, if the underlying blind signature scheme supports concurrent security, the transformed scheme also maintains concurrent security, unlike to Pointcheval's transform. The downside is that the communication and computation overhead linearly increases with every signing session (or with each detected cheating attempt) leading to a 1-out-of- N cut-and-choose for the $N - 1$ st execution. For both approaches, the verification of the transformed scheme differs from the original blind signature scheme as it requires an additional step before running the underlying verification algorithm.

The applicability in practice is therefore questionable, especially for systems where many signatures are issued per time and by different users. With Pointcheval's technique, if one user would be caught cheating at some point, the signer(s) would stop issuing signatures in general, also to other users. As we can reasonably assume one user per public key for cryptographic wallets and do not necessarily require concurrent signing, the compromise (other than the different verification algorithm) could be tolerated. While the support for concurrency by Katz et al. is beneficial, a growing computation and communication effort per requested signature is highly ineligible in this context. Only using their alternative method of only increasing the required commitments per cheating attempt may be more realistic in practice. In a recent work by Chairattana et al. [Cha+22], the authors improved the communication complexity to grow only logarithmically with the number of sessions using randomness homomorphic commitments and puncturable pseudorandom functions. Still, the major downside for all "boosting" transformations remains that the verification is different than for standard Schnorr signatures such that it is incompatible with existing systems using Schnorr signatures as desired by our application for cryptographic wallets.

The blind Schnorr schemes by Tessaro and Zhu [TZ22], described in Section 2.4.2, build upon a variant of the ROS problem, the WFROS problem that is assumed to be harder than ROS and MROS, according to the authors.

Compared to the original blind Schnorr scheme, their computational and communication overhead is insignificant; it only consists of the signer sending another nonce $Y = yX$, where X is the public key, in the first round and its discrete logarithm y in the third round. This additional nonce is consequently also included by the user in the (un-)blinding computations and for signature verification. The result is that the final signature can be considered as a signature with respect to Y as the public key, such that the verification algorithm has to be slightly adapted. Unless the setting already expects a different public key for every new signature and includes it in the hash for the challenge, their scheme is not

compatible with existing systems employing Schnorr signatures. The authors prove the unforgeability of their scheme to the DL assumption in the generic group model.

In addition, Tessaro and Zhu also propose an alternative scheme that is secure in the algebraic group model, which further demands a changed public key with an additional group element compared to Schnorr. Without adding this extra group element in the public key, the scheme can only be reduced to the OMDL assumption as opposed to the weaker DL assumption.

The recently published blind Schnorr scheme by Fuchsbauer and Wolf [FW22], mentioned in Section 2.4.2, mitigates ROS attacks by leveraging zero-knowledge proofs over the hash function used for computing the challenge. Initially, the user commits to the blinding values and the message, it will use later on in the session, by sending them in encrypted form using a public-key encryption scheme. After having received the nonce from the signer, the user sends, along with the challenge, a zero-knowledge proof proving that the values committed in the previous step have been correctly used to compute the challenge. The obvious drawback is that this approach requires an additional communication round between the user and the signer(s) for the commitment step. Moreover, the zero-knowledge proof is computationally considerably expensive both in its generation and in its verification, the latter of which must be performed by all signers in a threshold setting.

Lastly, the clause blind Schnorr signature scheme by Fuchsbauer et al. [FPS19], described in Section 4.3.2, is based on another variant of the ROS problem, the MROS problem. Similar to Pointcheval [Poi98], we can think of that the signer and the user always begin two parallel signing sessions, whereas the signer decides which sessions will be completed. But in contrast to the boosting techniques, the scheme does not require any commitment by the user and thus remains only with two rounds. The scheme by Fuchsbauer further allows the same verification as for the original (blind) Schnorr signature scheme.

To conclude, we consider the clause blind Schnorr signature scheme by Fuchsbauer as the most efficient scheme that is on top directly compatible with existing Schnorr supporting systems and therefore also most suitable for cryptographic wallets for various cryptocurrencies.

5 Threshold Blind Signatures

In this chapter, we first introduce the definition of threshold blind signature schemes and their notions of security. Then, we present our threshold blind Schnorr signature scheme and prove the security of our construction.

5.1 Background

The first threshold blind signature scheme has been introduced by Juang and Lei in 1996 [JL96] based on the discrete logarithm but without any proof of security. Later, the first provably secure constructions were presented by Lei et al. [LJY02] and Kim et al. [KKL02] in 2002. Both schemes are based on the Okamoto-Schnorr blind signature scheme and reduce their security onto it. As the security of the Okamoto-Schnorr blind signature scheme has been broken due to the novel ROS attacks, we can thus consider both constructions as insecure. A robust threshold blind signature has been proposed by Vo et al. [VZK03], which is based on pairings and reduces its security to the hardness of the Gap-Diffie-Hellman assumption in the random oracle model. Another robust scheme that supports partial blindness was presented by Cao et al. [CZL06], which is based on the RSA assumption. A major drawback is that their scheme requires an always trusted party for the signing protocol. In 2015, Kutchra and Manulis [KM15] provided the first formalization of threshold blind signatures along with their pairing-based construction in the standard model. They employ techniques of the Okamoto-Schnorr blind signature scheme and various non-interactive zero-knowledge proofs. The rerandomizable property allows randomizing a given signature such that it is indistinguishable from the original signature. For privacy-preserving use cases, such as anonymous credentials, this eliminates the need for expensive zero-knowledge proofs in order to keep proofs about the signature unlinkable. However, so far no threshold blind signature scheme has been proposed that is based on plain Schnorr and allows unmodified verification of the issued signatures so that it can be seamlessly integrated into existing Schnorr-supporting systems.

In Table 5.1, we compare different threshold blind signature schemes in terms of assumptions, moves of interactions, complexity (communication and computation), and whether it requires pairings, zero-knowledge proofs (ZKPs), or a trusted party (TP). Regarding the complexity, as for threshold signatures regarding, we assume a signature dealer for replacing broadcasts and being responsible for the combination and verification of partial signatures.

Table 5.1: Comparison of threshold blind Schnorr signature schemes.

Scheme	Assumptions	Moves of Interactions	Complexity	Pairings	ZKPs	TP
[VZK03]	GDH, RO	2	$O(1)$	Yes	No	No
[CZL06]	RSA, RO	2	$O(1)$	No	No	Yes
[KM15]	CDH, DLin	2	$O(n)$	Yes	Yes	No
TBSS	OMDL, MROS, RO, AGM	3	$O(1)$	No	Yes ^{†₁}	No
BFROST	- ^{†₂}	3	$O(n)^{\dagger_3}$	No	Yes ^{†₁}	No

^{†₁}: Note that the NIZK proof only consists of proving knowledge of a discrete logarithm which could be just instantiated via Schnorr.

^{†₂}: Has no formal security proof yet.

^{†₃}: Can be reduced to $O(1)$ using an optimization by Crites et al. [CKM21].

5.2 Definition

A (t, n) -threshold blind signature scheme allows a user to obtain a signature on a message issued under a threshold-shared key while ensuring that the signers do not learn the message, and as long as less than t signers are corrupt, the scheme remains secure. We now provide the formal definition of a threshold blind signature scheme and its security properties as influenced by [FPS19; KLR21; KM15]. We note that we cannot adopt the already existing definition by Kuchta and Manulis [KM15] since their definition does not support concurrency between the signing rounds.

Definition 27 (Threshold Blind Signature TBS). *A (t, n) -threshold blind signature scheme is executed between a set of n signers S_i with $i \in \mathcal{S}$ and a user U and consists of a tuple of five algorithms $\text{TBS} = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{User}, \text{Verify})$ along with an associated message space $\mathcal{M}$ that are defined as follows:*

- $pp \leftarrow \$ \text{Setup}(1^\lambda)$: On input the security parameter 1^λ , the PPT algorithm **Setup** outputs the public parameters pp , which might include common reference strings used within the other algorithms.
- $(\{sk_i, vk_i\}_{i \in \mathcal{S}}, pk) \leftarrow \$ \text{KeyGen}(pp, t, n)$: On input the public parameters pp , and two integers $t, n \in \mathbb{N}$ with $1 \leq t \leq n$, the PPT algorithm **KeyGen**, outputs a secret key share sk_i and a verification key share vk_i for every signer S_i with $i \in \mathcal{S}$ and the common public key pk . The signers' secret key shares sk_i form a (t, n) -threshold sharing of the shared secret corresponding to the common public key pk .
- $(b, \sigma) \leftarrow \$ \langle \text{Sign}(sk_1, \mathcal{V}), \dots, \text{Sign}(sk_n, \mathcal{V}), \text{User}(pk, m) \rangle$: The interactive, randomized signing protocol between the signer algorithms **Sign** and the user algorithm **User**. The input of each **Sign** algorithm is a secret key sk_i of signer S_i with $i \in \mathcal{S}$ and the set of all verification keys $\mathcal{V} := \{vk_i\}_{i \in \mathcal{S}}$. The output of **Sign** is a bit b , which equals 1 to indicate success or 0 to indicate failure (to simplify the notation, the protocol only outputs a single bit b corresponding to the conjunction of all bits). The **User** algorithm takes as input the public key pk and a message $m \in \mathcal{M}$, and in the end outputs the signature σ , where $\sigma = \perp$ on failure.
- $0/1 \leftarrow \$ \text{Verify}(pk, m, \sigma)$: On input the public key pk , a signature σ , and a message $m \in \mathcal{M}$, the DPT algorithm **Verify** outputs 1 if the signature on the message m is valid with respect to pk and 0, otherwise.

A threshold blind signature scheme TBS has to satisfy correctness, which requires that all signatures generated by honestly following the signing protocol verify.

Definition 28 (Correctness for TBS). *A (t, n) -threshold blind signature scheme TBS is correct, if for all $\lambda \in \mathbb{N}$ and all messages $m \in \mathcal{M}$ followed by running $pp \leftarrow \text{Setup}(1^\lambda)$, $(\{sk_i, vk_i\}_{i \in \mathcal{S}}, pk) \leftarrow \text{KeyGen}(pp, t, n)$ for $i \in \mathcal{S}$ with the set of all verification keys $\mathcal{V} := \{vk_i\}_{i \in \mathcal{S}}$, we have:*

$$\Pr \left[(b, \sigma) \leftarrow \$ \langle \text{Sign}(sk_1, \mathcal{V}), \dots, \text{Sign}(sk_n, \mathcal{V}), \text{User}(pk, m) \rangle : \begin{array}{l} (b = 1 \\ \wedge \text{Verify}(pk, m, \sigma)) \end{array} \right] = 1$$

A threshold blind signature scheme TBS is secure if it satisfies unforgeability and blindness. We provide the formal definitions below.

Unforgeability for blind signatures follows the "one-more" paradigm in the sense that after the adversary finishes $k - 1$ executions of the signature issuing protocol, it should not be able to output k distinct and valid signatures. In the case of threshold blind signatures, the adversary is further allowed to corrupt up to $t - 1$ signers with whom it can participate in the signing sessions.

Definition 29 (Unforgeability for TBS). *A (t, n) -threshold blind signature scheme TBS is unforgeable if for all PPT adversaries $\mathcal{A}$, we have $\Pr[\text{Unforge}_{\text{TBS}}^{\mathcal{A}}(\lambda) = 1] \leq \text{negl}(\lambda)$ for a negligible function $\text{negl}(\lambda)$ and the security game $\text{Unforge}_{\text{TBS}}$ defined in Figure 5.1.*

Game $\text{Unforge}_{\text{TBS}}^{\mathcal{A}}(\lambda)$:

- **Setup Phase:** The challenger sets up the public parameters $pp \leftarrow \$ \text{TBS.Setup}(1^\lambda)$ and the key-pairs $(\{sk_i, vk_i\}_{i \in \mathcal{S}}, pk) \leftarrow \$ \text{TBS.KeyGen}(pp, t, n)$ for $i \in \mathcal{S}$, and sets $\mathcal{V} := \{vk_i\}_{i \in \mathcal{S}}$. Also, it initializes $q_s \leftarrow 0$. Finally, it runs the adversary $\mathcal{A}$ on input the public key pk .
- **Corruption Phase:** $\mathcal{A}$ outputs a set $\mathcal{S}_C$ of signer indices it wants to corrupt, where $\mathcal{S}_C \subset \mathcal{S}$ and $|\mathcal{S}_C| \leq t - 1$. The challenger then hands out the secret keys $\{sk_i\}_{i \in \mathcal{S}_C}$ to $\mathcal{A}$.
- **Online Phase:** $\mathcal{A}$ is given access to the oracle $\text{OSign}_{\text{TBS}}$ which executes the signing protocol on behalf of the honest signers together with the corrupt signers, where $\mathcal{A}$ plays the role of the user. $\mathcal{A}$ is also allowed to interleave the various oracle calls arbitrarily. For each finished execution of the signing protocol, q_s is increased as $q_s \leftarrow q_s + 1$.
- **Output Phase:** $\mathcal{A}$ wins the game if it outputs a forgery of unique pairs $(m_i^*, \sigma_i^*)_{i \in [k]}$ where all signatures are valid, i.e., $\text{TBS.Verify}(pk, m_i^*, \sigma_i^*) = 1$ for $i \in [k]$, and it holds that $k > q_s$.

Figure 5.1: The security game for one-more unforgeability of threshold blind signature schemes.

Blindness requires that the adversary, who is in control of all signers, should not be able to link executions of the signing protocol with the message that has been signed or the resulting signature.

Definition 30 (Blindness for TBS). *A (t, n) -threshold blind signature scheme is blind if for all PPT adversaries $\mathcal{A}$, it holds that $\Pr[\text{Blind}_{\text{TBS}}^{\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$, for a negligible function $\text{negl}(\lambda)$ and the security game $\text{Blind}_{\text{TBS}}$ defined in Figure 5.2. If this equation is changed to $\Pr[\text{Blind}_{\text{TBS}}^{\mathcal{A}}(1^\lambda) = 1] = \frac{1}{2}$ and holds for all unbounded adversaries, the scheme is considered perfectly blind.*

5.3 Our Construction

In this section, we present our (t, n) -threshold blind Schnorr signature scheme TBSS based on the blind Schnorr signature scheme by Fuchsbaauer et al. [FPS19] and the threshold Schnorr signature scheme by Lindell [Lin22], which uses equivocable commitments and zero-knowledge proofs. Our construction is secure in the presence of a dishonest majority and supports identifiable aborts, i.e., misbehaving participants can be identified and excluded from further signing sessions. Depending on the signing stage and the number of participants relating to t , the protocol can even continue in the case of misbehaving participants.

Setup: The algorithm $\text{TBSS.Setup}(1^\lambda)$ takes the security parameter 1^λ as an input and outputs the public parameters $pp := (p, \mathbb{G}, G, H, \text{crs}_{\text{EC}}, \text{crs}_{\text{ZK}})$, where:

The first three parameters correspond to a group description given by $(p, \mathbb{G}, G) \leftarrow \$ \text{GrGen}(1^\lambda)$ and H is a hash function with $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$. The last two parameters, crs_{EC} and crs_{ZK} are common reference strings for the equivocable commitment scheme EC and the non-interactive zero-knowledge proof system NIZK, respectively. They are in turn generated using their respective setup algorithm, i.e., $\text{crs}_{\text{EC}} \leftarrow \$ \text{EC.Setup}(1^\lambda)$ and $\text{crs}_{\text{ZK}} \leftarrow \$ \text{NIZK.Setup}(1^\lambda, \mathcal{R})$. The relation $\mathcal{R}$ we use for the NIZK scheme to prove knowledge over the discrete logarithm of a value $Y \in \mathbb{G}$ is defined as follows: $(\phi, w) = (Y, y) \in \mathcal{R}$ iff $Y = yG$, where $y \in \mathbb{Z}_p$.

Key Generation: The algorithm $\text{TBSS.KeyGen}(pp, t, n)$ takes the public parameters pp and two integers $t, n \in \mathbb{N}$ as an input, where $1 \leq t \leq n$. A trusted dealer samples a random $x \leftarrow \$ \mathbb{Z}_p$ and runs Shamir's secret sharing scheme SSS by first choosing a random polynomial $f(z) = a_0 + a_1z + \dots +$

Game $\text{Blind}_{\text{TBS}}^{\mathcal{A}}(\lambda)$:

- **Setup Phase:** The challenger runs the adversary $\mathcal{A}$ given 1^λ as an input who outputs a public key pk and two messages m_0, m_1 . The challenger then samples a bit $b \leftarrow \{0, 1\}$.
- **Online Phase:** $\mathcal{A}$ is given access to the oracles $\text{OUser}_{\text{TBS},b}$ and $\text{OUser}_{\text{TBS},1-b}$ which can be queried only once. The oracles execute the protocol $\langle \cdot, \text{TBS.User}(pk, m_b) \rangle$ and $\langle \cdot, \text{TBS.User}(pk, m_{1-b}) \rangle$, respectively, where $\mathcal{A}$ plays the role of the signers. The adversary is allowed to arbitrarily interleave between the oracle queries. Only after both executions have been successfully finished, i.e., $\sigma_b \neq \perp$ and $\sigma_{1-b} \neq \perp$ with $\sigma_b \leftarrow \text{TBS.User}(pk, m_b)$ and $\sigma_{1-b} \leftarrow \text{TBS.User}(pk, m_{1-b})$, the last oracle replies with (σ_b, σ_{1-b}) . If one of the signatures is invalid, it replies with $(\perp, \perp)$.
- **Output Phase:** $\mathcal{A}$ wins the game, if it outputs a bit b' such that $b' = b$.

Figure 5.2: The security game for blindness of threshold blind signature schemes.

$a_{t-1}z^{t-1}$ of degree $t-1$ with $a_0 = x$. The secret key share and verification key share for signer S_i with $i \in \mathcal{S}$ is then computed as $sk_i \leftarrow f(i)$ and $vk_i \leftarrow f(i)G$, respectively. Finally, the common public key is set to $pk \leftarrow xG$.

Instead of standard Shamir's secret sharing, the trusted dealer could also execute a verifiable secret sharing protocol for the key generation such as Pedersen's VSS scheme [Ped01].

Furthermore, the need for a trusted dealer can be eliminated by implementing the key generation algorithm in a distributed manner, where each participant contributes to the generation of the common secret. Possible distributed key generation protocols are, e.g., the DKG scheme by Pedersen [Ped91] or the robust DKG scheme by Gennaro et al. [Gen+99].

Signature Generation: The signing protocol $\langle \text{Sign}(sk_1, \mathcal{V}), \dots, \text{Sign}(sk_n, \mathcal{V}), \text{User}(pk, m) \rangle$ between the signers S_i with $i \in \mathcal{S}$ and the user U is depicted in Figure 5.3. Since only more than t signers have to participate in the signing protocol, we denote the set of active signers with $\mathcal{S}_0$, where $\mathcal{S}_0 \subseteq \mathcal{S}$. Each signer S_i is given its secret key sk_i , which it parses as $x_i := sk_i$, and the set of all verification keys $\{vk_1, \dots, vk_n\} := \mathcal{V}$, parsed as $X_i := vk_i$ as an input. The user's U input is the public key pk , which it parses as $X := pk$, and a message m . The protocol consists of an initialization step for the signers, two signing stages, and two user stages that are described below:

Initialization: Prior to the first signing stage, every signer converts its secret share x_i , contained in their secret key sk_i , and the verification keys $\{vk_i\}_{i \in \mathcal{S}_0}$ into additive form using the Lagrange coefficient for the set $\mathcal{S}_0$. Concretely, this is done by multiplying x_i and all vk_i with $\lambda_i = \prod_{\substack{j \in \mathcal{S}_0 \\ j \neq i}} \frac{j}{j-i}$.

Signing Stage 1: Each signer S_i begins with generating a secret nonce $r_{0,i} \leftarrow \mathbb{Z}_p$ and computes the public nonce as $R_{0,i} \leftarrow r_{0,i}G$. Then, S_i creates a commitment $co_{0,i}$ on $R_{0,i}$ using the equivocable commitment scheme EC and computes a zero-knowledge proof $\pi_{0,i}$ that proves knowledge the discrete logarithm of $R_{0,i}$ using the NIZK scheme. The commitment $co_{0,i}$ is then broadcasted to all other signers. When having received all other commitments, each signer reveals $(R_{0,i}, op_{0,i}, \pi_{0,i})$, where $op_{0,i}$ is the opening value of the commitment. Using this tuple, each signer verifies the commitments of the other signers and the proof. If they do not verify, the misbehaving signer is reported and excluded from the signer set $\mathcal{S}_0$. As long as it still holds that $|\mathcal{S}_0| \geq t$, the protocol continues after repeating the initialization step with the updated signing set, otherwise it aborts. The signers then proceed with computing R_0 as the sum of all individual $R_{0,i}$ values. They repeat

this process for the second secret nonce $r_{1,i}$ to end up with R_1 (note that the generation of R_0 and R_1 can be parallelized). Finally, they send (R_0, R_1) to the user.

User Stage 1: After having received (R_0, R_1) , the user *blinds* its message m as follows: First, the user samples random $\alpha_0, \beta_0 \leftarrow \mathbb{Z}_p$ values and computes $R'_0 \leftarrow R_0 + \alpha_0 G + \beta_0 X$. Then, the user computes the challenge using the hash function H as $c'_0 \leftarrow H(R'_0, m)$ and $c_0 \leftarrow c'_0 + \beta_0 \bmod p$. The user repeats this process for R_1 and ends up with the challenge tuple (c_0, c_1) which is sent back to the signers.

Signing Stage 2: After having received the challenges c_0, c_1 from the user, the signers collaboratively sample a bit to decide which R_0, R_1 and c_0, c_1 value they will use to create the signature. To do so, each signer chooses a random bit b_i , commits to it using EC, broadcasts the commitment to the other signers, and reveals the bit and opening value after having received all other commitments. Using the opening values, each signer verifies the commitments of the other signers, if they do not verify, the misbehaving signer is reported and the protocol aborts. The resulting bit b is calculated as the sum of all signers' bits b_i modulo 2. Having decided on the bit b , each signer computes and broadcasts its response s_i using the secret share x_i as $s_i \leftarrow r_{b,i} + c_b x_i \bmod p$. Before computing the final response s , the well-formedness of each response is checked with $s_j G = R_{b,j} + c_b X_j$ for $j \in \mathcal{S}_0$, while for an invalid s_j , signer S_j is reported and the protocol aborts. The final response s , which is computed as the sum of all s_i values, and the bit b are then sent back to the user. Lastly, the signers output 1 to indicate success.

User Stage 2: After having received the bit b and the value s , the user first validates the correctness of s by checking if $sG = R_b + c_b X$ holds. If this equation does not hold, the user aborts, otherwise the user then *unblinds* s by computing $s' \leftarrow s + \alpha_b$. Lastly, the user outputs the final signature $\sigma := (R'_b, s')$.

Verification: The verification algorithm $\text{TBSS.Verify}(pk, \sigma, m)$ first parses the public key as $X := pk$ and the signature as $(R, s) := \sigma$. Then, it computes $c \leftarrow H(R, m)$ and outputs 1 if $sG = R + cX$ holds and 0, otherwise.

Note that the verification works exactly the same as in the original Schnorr signature scheme (cp. Section 2.5.2).

We now show correctness of our construction.

Lemma 1 (Correctness). *Our (t, n) -threshold blind Schnorr signature scheme TBSS satisfies correctness; that is, if the signers and the user honestly follow the signing protocol, the resulting signature σ verifies with probability 1.*

Proof. Let $\sigma := (R'_b, s')$ be the signature on a message m that was output by the user in the protocol $\langle \text{TBSS.Sign}(sk_1, \mathcal{V}), \dots, \text{TBSS.Sign}(sk_n, \mathcal{V}), \text{TBSS.User}(pk, m) \rangle$. Since the secret key shares $sk_i := x_i$ provided by KeyGen are (t, n) -threshold shares of the common secret x corresponding to the public key $pk := X = xG$, and after the initialization step it holds that $\sum_{i \in \mathcal{S}_0} x_i = x$, we have:

$$\begin{aligned} s'G &= sG + \alpha_b G = \sum_{i \in \mathcal{S}_0} s_i G + \alpha_b G = \sum_{i \in \mathcal{S}_0} (r_{b,i} + c_b x_i) G + \alpha_b G \\ &= R_b + c_b X + \alpha_b G = R_b + \alpha_b G + \beta_b X + c'_b X = R'_b + H(R'_b, m) X \end{aligned}$$

□

5.3.1 Proof of Security

We now formally prove the security of our (t, n) -threshold blind Schnorr signature in terms of unforgeability and blindness by reducing it to the security of the underlying blind Schnorr signature scheme by Fuchsbaauer et al. [FPS19].

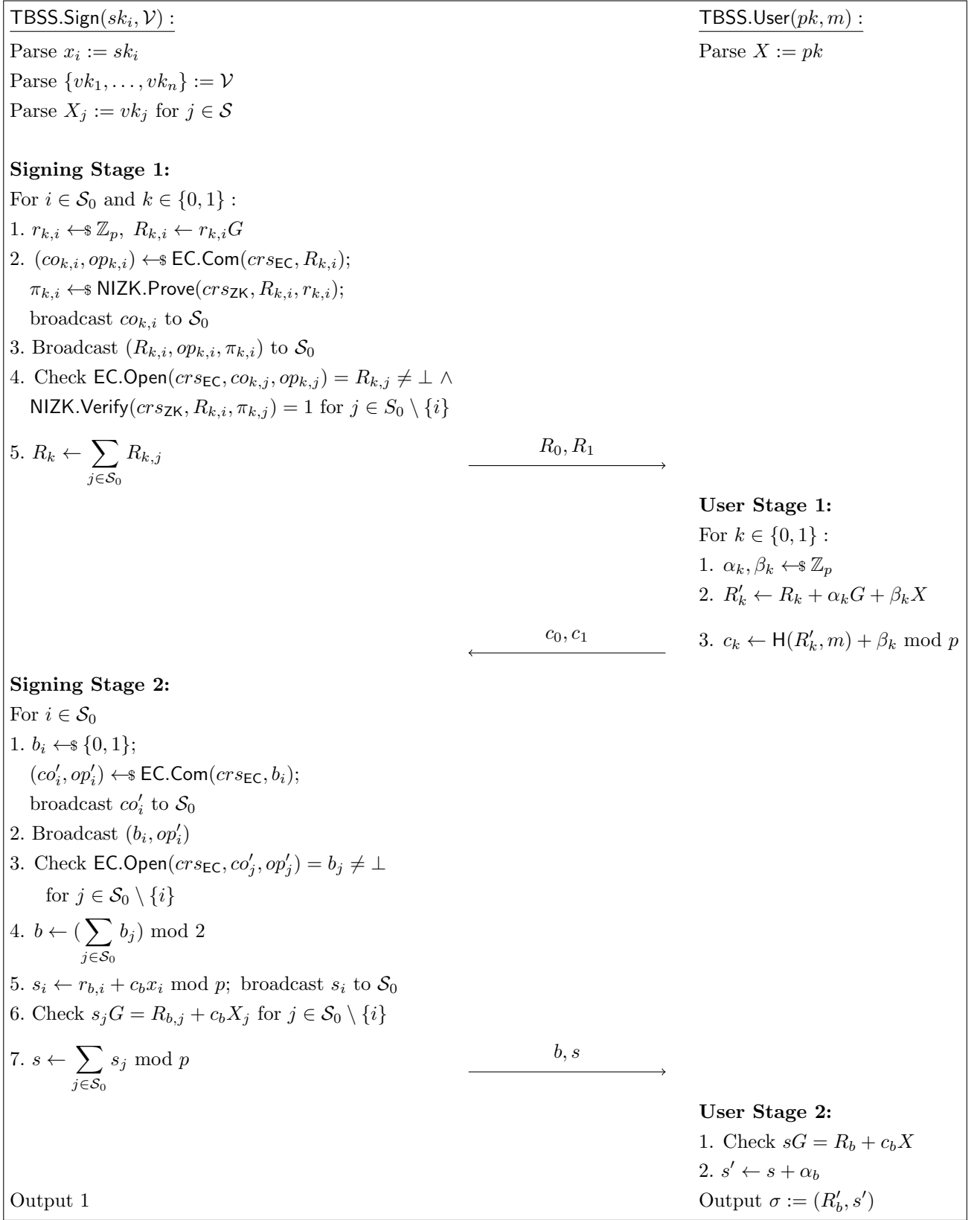


Figure 5.3: The signing protocol of our TBSS construction

Theorem 5 (Unforgeability). *Our (t, n) -threshold blind Schnorr signature scheme TBSS is unforgeable assuming the soundness property of the NIZK scheme, the security of the EC scheme, and the unforgeability of the blind Schnorr signature scheme BSS. Let $\mathcal{A}$ be a PPT adversary against the unforgeability of the TBSS scheme, then there exists an PPT adversary $\mathcal{B}$ against the unforgeability of the BSS scheme such that for a negligible function $\text{negl}(\lambda)$, it holds that:*

$$\Pr[\text{Unforge}_{\text{TBSS}}^{\mathcal{A}}(1^\lambda) = 1] \leq \Pr[\text{Unforge}_{\text{BSS}}^{\mathcal{B}}(1^\lambda) = 1] + \text{negl}(\lambda)$$

We first begin with providing the proof idea that shows the final simulation of the adversary $\mathcal{B}$ playing against the $\text{Unforge}_{\text{BSS}}$ game by internally using the adversary $\mathcal{A}$ who plays against the $\text{Unforge}_{\text{TBSS}}$ game. $\mathcal{B}$, therefore, has to simulate the oracle $\text{OSign}_{\text{TBSS}}$ while having access to $\text{OSign}_{\text{BSS}}$ such that $\text{OSign}_{\text{TBSS}}$ behaves exactly as in the security game $\text{Unforge}_{\text{TBSS}}$. Additionally, since $\mathcal{A}$ is allowed to participate in the signing sessions with up to $t - 1$ corrupted signers, the view in these sessions also needs to be indistinguishable as in a real execution of the $\text{Unforge}_{\text{TBSS}}$ game.

Initially, $\mathcal{B}$ receives the public key $pk := X$ from its challenger $\mathcal{C}_{\text{BSS}}$ and is granted access to the oracle $\text{OSign}_{\text{BSS}}$.

In the setup phase, $\mathcal{B}$ prepares the common reference strings using the simulated setup algorithms for the equivocable commitment scheme EC and for the non-interactive zero-knowledge proof system NIZK. That is, the common reference string crs_{EC} is given by $(crs_{\text{EC}}, \tau_{\text{EC}}) \leftarrow \$ \text{EC.SimSetup}(1^\lambda)$ and crs_{ZK} is given by $(crs_{\text{ZK}}, \tau_{\text{NIZK}}, \xi) \leftarrow \$ \text{NIZK.SimSetup}(1^\lambda)$.

When $\mathcal{B}$ provides $\mathcal{A}$ with the given public key pk , $\mathcal{A}$ outputs a set $\mathcal{S}_C$ of at most $t - 1$ indices of signers it wants to corrupt.

After having received $\mathcal{S}_C$, $\mathcal{B}$ generates the key-pairs (sk_i, vk_i) for $i \in \mathcal{S}$ in a way such that the combination of the verification key shares vk_i equals the public key X provided by the challenger $\mathcal{C}_{\text{BSS}}$. For the sake of simplicity of the proof, we assume a fixed signing set $\mathcal{S}_0$ that will be used throughout the signing sessions and contains all corrupt signers. However, it is also possible to conduct the simulation with arbitrary signing sets, e.g., where the adversary varies with the number of participating corrupt signers, analogously as shown in the proof by Lindell [Lin22].

First, $\mathcal{B}$ chooses a random index of an honest signer $i' \leftarrow \$ \mathcal{S}_0 \setminus \mathcal{S}_C$ and sets $x_i \leftarrow \$ \mathbb{Z}_p$ and $X_i \leftarrow x_i G$ for $i \in \mathcal{S}_0 \setminus \{i'\}$. The verification key of $S_{i'}$ is then patched as $X_{i'}' \leftarrow (X - \sum_{i \in \mathcal{S}_0 \setminus \{i'\}} X_i \lambda_i) / \lambda_{i'}$ where λ_i denotes the Lagrange coefficient for i in the set $\mathcal{S}_0$. Note that $\mathcal{B}$ does not have the secret share $x_{i'}$ of $S_{i'}$ since it does not know the discrete logarithm of $X_{i'}$. The key-pairs $(sk_i, vk_i) := (x_i, X_i)$ for the corrupted parties $i \in \mathcal{S}_C$ are then given to $\mathcal{A}$.

Handling $\text{OSign}_{\text{TBSS}}$ Queries: We divide the execution of $\langle \text{TBSS.Sig}(sk_1, \mathcal{V}), \dots, \text{TBSS.Sig}(sk_n, \mathcal{V}), \cdot \rangle$ into two phases corresponding to **Signing Phase 1** and **Signing Phase 2** in the protocol. This division analogously applies to BSS. The protocol begins with the first phase, which ends with outputting the nonces (R_0, R_1) . The second phase begins when receiving the challenge tuple (c_0, c_1) from the user and outputs the tuple (b, s) .

Simulating Signing Phase 1: When $\mathcal{A}$ queries $\text{OSign}_{\text{TBSS}}$, $\mathcal{B}$ queries the oracle $\text{OSign}_{\text{BSS}}$ and receives the nonce tuple (R_0, R_1) as a response. Then, $\mathcal{B}$ has to simulate performing **Signing Stage 1** of the signing protocol together with the corrupted parties of $\mathcal{A}$. The goal of $\mathcal{B}$ is that the simulation outputs the exact same tuple (R_0, R_1) while the view for $\mathcal{A}$ during the execution is indistinguishable from a real execution. We now describe how $\mathcal{B}$ behaves in this simulation:

- Perform the Steps 1 - 3 as in the protocol.
- After having received all $(R_{k,i}, op_{k,i}, \pi_{k,i})$ tuples from the corrupted signers, $\mathcal{B}$ calculates $R_{k,i'}^* \leftarrow R_k - \sum_{i \in \mathcal{S}_0 \setminus \{i'\}} R_{k,i}$. Also, it extracts $r_{k,i} \leftarrow \text{NIZK.Extract}(\xi, R_{k,i}, \pi_{k,i})$ from all corrupted signers S_i with $i \in \mathcal{S}_C$. Then, $\mathcal{A}$ is rewinded to Step 1.
- $\mathcal{B}$ computes a new opening value $op_{k,i'}^* \leftarrow \$ \text{Equiv}(\tau_{\text{EC}}, co_{k,i'}, R_{k,i'}^*)$ such that the commitment $co_{k,i'}$ opens to $R_{k,i'}^*$. The new proof is simulated as $\pi_{k,i'}^* \leftarrow \$ \text{SimProve}(\tau_{\text{ZK}}, R_{k,i'}^*)$. All other values stay the same.

- Continue with the Steps 3 - 5 exactly as in the protocol. Since the commitments and proofs of all honest signers correctly verify, this stage will successfully return R_k .

The nonce tuple (R_0, R_1) is then sent to the user played by $\mathcal{A}$.

Simulating Signing Phase 2: When receiving the challenge tuple (c_0, c_1) , $\mathcal{B}$ forwards the tuple to its execution of $\text{OSign}_{\text{BSS}}$ and receives (b, s) . Consequently, $\mathcal{B}$ has to simulate performing **Signing Stage 2** such that it results with the same output (b, s) . This simulation works as follows:

- Follow Steps 1-2 as in the protocol.
- After having received all bits b_i from the corrupted signers, $\mathcal{B}$ computes $b' \leftarrow (\sum_{i \in \mathcal{S}_0} b_i) \bmod 2$. If $b' = b$, the simulation continues, otherwise $\mathcal{A}$ is rewinded to Step 1 and the bit of $S_{i'}$ is flipped to $b_{i'}^* \leftarrow \neg b_{i'}$. Hence, $\mathcal{B}$ also needs to update the opening as $op_{i'}^* \leftarrow \text{Equiv}(\tau_{\text{EC}}, co_{i'}, b_{i'}^*)$.
- In Step 5, all s_i values of the honest parties are computed as in the protocol except for $s_{i'}$ which is calculated such that the sum of all values results in s . Therefore, $\mathcal{B}$ first calculates $s_i \leftarrow r_{b,i} + c_b x_i$ for $i \in \mathcal{S}_C$. This step is possible since $\mathcal{B}$ knows all secret shares x_i of the corrupted parties and knows all nonce shares $r_{b,i}$ due to the extraction of the zero-knowledge proofs. Then, it patches $s_{i'} \leftarrow s - \sum_{i \in \mathcal{S}_0 \setminus \{i'\}} s_i$. Note that even if $\mathcal{B}$ would first receive all s_i values from the corrupted signers before having to decide on its $s_{i'}$, we theoretically could omit the zero-knowledge proofs for the nonces. Yet, if $\mathcal{A}$ sends incorrect s_i values, this would lead to $s_{i'}$ being incorrect too and thus making the simulation distinguishable.
- Perform Steps 6-7 as in the protocol such that it ends with the output (b, s) for the signers.

Finally, the tuple (b, s) is sent to the user, completing the signing protocol for the oracle query.

When $\mathcal{A}$ outputs a forgery $(m_i^*, \sigma_i^*)_{i \in [k]}$, $\mathcal{B}$ outputs the same to its challenger $\mathcal{C}_{\text{BSS}}$, which is trivially a valid forgery in $\text{Unforge}_{\text{BSS}}$ iff it is a valid forgery in $\text{Unforge}_{\text{TBSS}}$.

Now, we will provide the formal proof using the sequence-of-games technique by Shoup [Sho04].

Proof. In order to prove Theorem 5, we create indistinguishable transitions from the original security game $\text{Unforge}_{\text{TBSS}}$ game played by the PPT adversary $\mathcal{A}$ to eventually enable a reduction to the $\text{Unforge}_{\text{BSS}}$ game played by the PPT adversary $\mathcal{B}$. Without loss of generality, we assume a fixed signing set $\mathcal{S}_0$ throughout the signing sessions. Analogously as shown by Lindell [Lin22], the proof can be extended to support arbitrary signing sets.

Game G_0 : The first game corresponds to the original game $\text{Unforge}_{\text{TBSS}}$. The only addition is that the game samples a random honest signer index $i' \leftarrow \mathcal{S}_0 \setminus \mathcal{S}_C$, where $\mathcal{S}_C$ denotes the set of at most $t - 1$ signers $\mathcal{A}$ wants to corrupt. By definition: $\Pr[\text{Unforge}_{\text{TBSS}}^{\mathcal{A}}(\lambda)] = \Pr[G_0^{\mathcal{A}}(\lambda)]$.

Game G_1 : This game works exactly as the previous game with the modification that the common reference string crs_{EC} for the EC scheme is generated by $(crs_{\text{EC}}, \tau_{\text{EC}}) \leftarrow \text{EC.SimSetup}(1^\lambda)$, which provides the game with the equivocation trapdoor τ_{EC} . Due to the equivocation indistinguishability of the EC scheme, $\mathcal{A}$ only has a negligible advantage of distinguishing the distributions $\{crs: crs \leftarrow \text{EC.Setup}(1^\lambda)\}$ and $\{\widetilde{crs}: (\widetilde{crs}, \tau) \leftarrow \text{EC.SimSetup}(1^\lambda)\}$. Therefore, we have: $\Pr[G_0^{\mathcal{A}}(\lambda)] \leq \Pr[G_1^{\mathcal{A}}(\lambda)] + \nu_1(\lambda)$, for a negligible function $\nu_1(\lambda)$.

Game G_2 : This game works like the previous game but the common reference string crs_{ZK} for the NIZK scheme is, similar to crs_{EC} , generated by the simulated setup algorithm as $(crs_{\text{ZK}}, \tau_{\text{ZK}}, \xi) \leftarrow \text{NIZK.SimSetup}(1^\lambda, \mathcal{R})$. Therefore, the game has access to the simulation trapdoor τ_{ZK} and to the

extraction trapdoor ξ . Since the zero-knowledge property of the NIZK scheme guarantees that the distributions $\{crs: crs \leftarrow \$ \text{NIZK.Setup}(1^\lambda, \mathcal{R})\}$ and $\{\widetilde{crs}: (\widetilde{crs}, \tau, \xi) \leftarrow \$ \text{NIZK.SimSetup}(1^\lambda, \mathcal{R})\}$ are only distinguishable with a negligible probability, we have: $\Pr[G_1^A(\lambda)] \leq \Pr[G_2^A(\lambda)] + \nu_2(\lambda)$, for a negligible function $\nu_2(\lambda)$.

Game G_3 : This game is the same as the last game except that in **Signing Stage 1**, the opening of signer $S_{i'}$ is calculated using the **EC.Equiv** algorithm as $op_{k,i'}^* \leftarrow \$ \text{Equiv}(\tau_{\text{EC}}, co_{k,i'}, R_{k,i'})$. Due to the equivocation indistinguishability of the EC scheme, the opening $op_{k,i'}^*$ is indistinguishable from a real opening except with a negligible probability. Therefore, it holds that $\Pr[G_2^A(\lambda) = 1] \leq \Pr[G_3^A(\lambda) = 1] + \nu_3(\lambda)$, for a negligible function $\nu_3(\lambda)$.

Game G_4 : This game is the same as the last game but with another modification in **Signing Stage 1**, where the proof of signer $S_{i'}$ is simulated as $\pi_{k,i'}^* \leftarrow \$ \text{SimProve}(\tau_{\text{ZK}}, R_{k,i'})$. The zero-knowledge property of the NIZK scheme ensures that the simulated proof $\pi_{k,i'}^*$ is indistinguishable from a real proof except with a negligible probability. Therefore, we have $\Pr[G_3^A(\lambda) = 1] \leq \Pr[G_4^A(\lambda) = 1] + \nu_4(\lambda)$, for a negligible function $\nu_4(\lambda)$.

Game G_5 : This game proceeds exactly the same as the last game but with the following difference in the **Signing Stage 1**: For every proof $\pi_{k,i}$ with $i \in \mathcal{S}_C$ it received from the corrupt signers, the game tries to extract the witness, which corresponds to the discrete logarithm of $R_{k,i}$, as $r_{k,i} \leftarrow \text{NIZK.Extract}(\xi, R_{k,i}, \pi_{k,i})$. If $r_{k,i}G \neq R_{k,i}$, the game aborts. However, due to the knowledge extractability of the NIZK scheme, this event can only happen with negligible probability. Consequently, it holds that $\Pr[G_4^A(\lambda) = 1] \leq \Pr[G_5^A(\lambda) = 1] + \nu_5(\lambda)$, for a negligible function $\nu_5(\lambda)$.

Game G_6 : This game works as the previous game but now the **Signing Stage 2** is modified as follows: At the beginning of the stage, the game samples a random bit $b \leftarrow \$ \{0, 1\}$. After it has received all bits b_i , it checks whether $b = (\sum_{i \in \mathcal{S}_0} b_i)$. If it is true, the game proceeds just like in the protocol, otherwise, the game behaves as follows: First, $\mathcal{A}$ is rewinded to the beginning of the stage. Then, it flips the bit $b_{i'} \leftarrow \neg b_{i'}$ of the signer $S_{i'}$ and creates a new opening value $op_{i'}^* \leftarrow \$ \text{Equiv}(\tau_{\text{EC}}, co_{i'}, b_{i'})$ for the commitment $co_{i'}$ such that it opens to the updated bit. The equivocation indistinguishability ensures that every PPT adversary only has a negligible advantage of distinguishing the opening $op_{i'}^*$ from a real opening. Following this fact, we have $\Pr[G_5^A(\lambda) = 1] \leq \Pr[G_6^A(\lambda) = 1] + \nu_5(\lambda)$, for a negligible function $\nu_5(\lambda)$.

Game G_7 : This last game behaves similarly to the previous game except for the following modifications: Prior to the protocol, the game begins executing the protocol $\langle \text{BSS.Sign}(sk), \cdot \rangle$ as a black box to receive the nonces (R_0, R_1) , where $sk := x$. Note that since the game has all individual secret keys sk_i (esp. x_i), it can trivially reconstruct x .

In **Signing Stage 1**, after the game has received all $R_{k,i}$ values for $i \in \mathcal{S}_C$, it updates $R_{k,i'} \leftarrow R_k - \sum_{i \in \mathcal{S}_0 \setminus \{i'\}} R_i \bmod p$ for $k \in \{0, 1\}$ and rewinds the adversary $\mathcal{A}$ to the beginning of the stage. Note that $R_{k,i'}$ is still identically distributed as before since R_k is chosen uniformly at random.

In **Signing Stage 2**, the game continues the execution of $\langle \text{BSS.Sign}(sk), \cdot \rangle$ by sending (c_0, c_1) to the signer and subsequently receiving (b, s) . Then, instead of randomly sampling a bit b as introduced in game G_6 , this game uses the received bit b . Furthermore, it will adapt $s_{i'}$ by first calculating $s_i \leftarrow r_{b,i} + c_b x_i \bmod p$ for $i \in \mathcal{S}_C$ and then setting $s_{i'} \leftarrow s - \sum_{i \in \mathcal{S}_0 \setminus \{i'\}} s_i$ such that the sum of all s_i values results in s . The value $s_{i'}$ is also the same value signer $S_{i'}$ would have sent and thus this change is indistinguishable and s results in a valid signature.

As the changes are indistinguishable, we have: $\Pr[G_6^A(\lambda) = 1] = \Pr[G_7^A(\lambda) = 1]$.

We can now construct an adversary $\mathcal{B}$ against the $\text{Unforge}_{\text{BSS}}$ game while internally simulating the game G_7 to $\mathcal{A}$: Given the public parameters $(p, \mathbb{G}, G, H, crs_{\text{EC}}, crs_{\text{ZK}}) := pp$ and the public key $X := pk$ from the challenger $\mathcal{C}_{\text{BSS}}$ of the $\text{Unforge}_{\text{BSS}}$ game, $\mathcal{B}$ generates all key-pairs (sk_i, vk_i) for $i \in \mathcal{S}$ as in

the original protocol but uses the provided public key pk . Also, it patches the verification key share of signer $S_{i'}$ to $X'_i \leftarrow (X - \sum_{i \in \mathcal{S}_0 \setminus \{i'\}} X_i \lambda_i) / \lambda_{i'}$ where λ_i denotes the Lagrange coefficient for i in the set $\mathcal{S}_0$.

On every new $\text{OSign}_{\text{TBSS}}$ query, $\mathcal{B}$ also queries its oracle $\text{OSign}_{\text{BSS}}$. After $\mathcal{B}$ receives the tuple (R_0, R_1) during the execution of $\text{OSign}_{\text{BSS}}$, it uses them in **Signing Stage 1** instead of generating them on its own. In **Signing Stage 2**, which begins with $\mathcal{A}$ sending the challenge tuple (c_0, c_1) to the signers, $\mathcal{B}$ forwards the tuple to the execution of its oracle $\text{OSign}_{\text{BSS}}$ to receive (b, s) . Consequently, in **Signing Stage 2**, $\mathcal{B}$ will similarly use the bit b and the signature value s instead of generating them itself. Observe that the patched $s_{i'}$ then exactly corresponds to the value that the real signer $S_{i'}$ would have sent when in possession of the secret share $x_{i'}$ and $r_{k,i'}$ such that $x = \sum_{i \in \mathcal{S}_0} x_i$ and $r_k = \sum_{i \in \mathcal{S}_0} r_{k,i}$, as similarly shown by Lindell [Lin22]. The reason for this follows from the construction of $s'_{i'}$:

$$s_{i'} = s - \sum_{i \in \mathcal{S}_0 \setminus \{i'\}} s_i = r_b + c_b x - \left(\sum_{i \in \mathcal{S}_0 \setminus \{i'\}} r_{b,i} + \sum_{i \in \mathcal{S}_0 \setminus \{i'\}} c_b x_i \right) = r_{b,i'} + c_b x_{i'}$$

Finally, when $\mathcal{A}$ outputs a forgery $((m_i^*, \sigma_i^*), \dots, (m_k^*, \sigma_k^*))$, $\mathcal{B}$ outputs the same to its challenger $\mathcal{C}_{\text{BSS}}$. It is trivial to see that when $\mathcal{A}$ wins, i.e., the forgery is valid, this also directly yields a valid forgery in the $\text{Unforge}_{\text{BSS}}$ game, so $\mathcal{B}$ wins likewise.

Since the transition from the first game G_0 , corresponding to the original security game $\text{Unforge}_{\text{TBSS}}$, to the last game G_7 is indistinguishable except for a sum of negligible probabilities, which still remains negligible and is represented by the function $\text{negl}(\lambda) = \sum_{i=1}^5 \nu_i(\lambda)$, the following equation holds:

$$\begin{aligned} \Pr[\text{Unforge}_{\text{TBSS}}^{\mathcal{A}}(1^\lambda) = 1] &= \Pr[G_0^{\mathcal{A}}(1^\lambda) = 1] \\ &\leq \Pr[G_7^{\mathcal{A}}(1^\lambda) = 1] + \text{negl}(\lambda) \\ &= \Pr[\text{Unforge}_{\text{BSS}}^{\mathcal{B}}(1^\lambda) = 1] + \text{negl}(\lambda) \end{aligned}$$

□

Theorem 6 (Blindness). *Our (t, n) -threshold blind Schnorr signature scheme TBSS satisfies perfect blindness assuming that the underlying blind Schnorr signature scheme BSS is perfectly blind. Let $\mathcal{A}$ be an adversary against the blindness of the TBSS scheme, then there exists an adversary $\mathcal{B}$ against the blindness of the BSS scheme such that the following equation holds:*

$$\Pr[\text{Blind}_{\mathcal{A}, \text{TBSS}}(1^\lambda) = 1] = \Pr[\text{Blind}_{\mathcal{B}, \text{BSS}}(1^\lambda) = 1]$$

Proof. The intuition behind the proof is that the user side of the signature protocol stays the same as in the non-threshold setting. For the blindness property, it does not matter whether there exists only one signer or multiple signers where each of them having a threshold share of the secret key. This can also be observed when considering the blindness security games for (threshold) blind signatures $\text{Blind}_{\mathcal{A}, \text{TBSS}}$ and $\text{Blind}_{\mathcal{A}, \text{BSS}}$ which are identical since the signers' side is represented by the adversary who, in the threshold setting, is in control of all signers' secret key shares and thus can be considered as a single entity; as in a blind signature scheme.

We now show how we construct an adversary $\mathcal{B}$ playing against the $\text{Blind}_{\text{BSS}}^{\mathcal{B}}$ game who internally uses an adversary $\mathcal{A}$ against the $\text{Blind}_{\text{TBSS}}^{\mathcal{A}}$ game:

The simulation is straightforward, when $\mathcal{A}$ outputs a public key pk and two messages m_0, m_1 , $\mathcal{B}$ forwards them to its challenger $\mathcal{C}_{\text{BSS}}$ of game $\text{Blind}_{\text{BSS}}^{\mathcal{A}}$. When $\mathcal{A}$ queries the oracle $\text{OUser}_{\text{TBSS}, b}$ or $\text{OUser}_{\text{TBSS}, b-1}$, $\mathcal{B}$ likewise queries its oracles $\text{OUser}_{\text{BSS}, b}$ or $\text{OUser}_{\text{BSS}, b-1}$ and forwards the exchanged messages during the protocol. In both schemes, the messages sent to the user algorithm TBSS.User or

BSS.User are the nonce tuple (R_0, R_1) and the signature tuple (b, s) . The message sent by BSS.User is the challenge tuple (c_0, c_1) which is forwarded to $\mathcal{A}$ in the respective protocol. After both protocols have been successfully finished, the oracle of $\text{Blind}_{\text{BSS}}^{\mathcal{B}}$ outputs the signatures (σ_b, σ_{b-1}) that $\mathcal{B}$ finally sends to $\mathcal{A}$.

When $\mathcal{A}$ outputs a bit b' , $\mathcal{B}$ outputs the same bit to $\mathcal{C}_{\text{BSS}}$. Since the simulation is perfect, the success probability for $\mathcal{B}$ in $\text{Blind}_{\text{BSS}}^{\mathcal{B}}$ is exactly the same as for $\mathcal{A}$ in $\text{Blind}_{\text{TBSS}}^{\mathcal{A}}$. $\square$

5.3.2 On Drijvers' and Benmahouda's Attack

While the vulnerability of blind Schnorr signatures regarding ROS attacks has been circumvented by the underlying blind Schnorr scheme by Fuchsbauer et al. [FPS19] through relying on the modified ROS problem, threshold Schnorr signatures are likewise susceptible. As shown by Drijvers et al. [Dri+19] and Benmahouda et al. [Ben+22] the unforgeability of various two-round multi- and threshold signature schemes can be broken by performing a ROS-like attack. While Drijvers et al. showed how to reduce the security of exemplary multi-signature schemes to the k -sum problem, Benmahouda et al. directly targeted the ROS problem, as discussed in Section 2.4.1, both problems can be solved in sub-exponential or even polynomial time. Both attacks have in common that the adversary, who is part of the signers, has to be able to choose its own nonce R_i after learning the nonces of the other participants. By participating in l parallel signing sessions, the adversary can adaptively choose its nonces to compute the challenges in beforehand to solve a specific equation formulated as the k -sum - or the ROS problem. Finding a solution then yields a successful forgery with $l + 1$ signatures.

This attack is prevented in the threshold Schnorr scheme by [Lin22], on which our construction is based, by having each signer commit its nonce R_i first and revealing it only after having received all other commitments.

However, in the case of (threshold) blind Schnorr signatures, the signers cannot compute the challenge on their own because the user computes the challenge and sends them to the signers. And without applying additional techniques that restrict the user on adapting the challenge after having received the nonce from the signers, an adversarial user can always employ a ROS attack to break the unforgeability. Therefore, the clause modification from Fuchsbauer et al. [FPS19] or similar approaches as from Tessaro et al. [TZ22] are inevitable for (threshold) blind Schnorr signatures.

The reason why our construction TBSS still requires a commitment phase for the nonces is that it allows us to conduct the unforgeability proof, where the honest signers adapt their committed nonces using the equivocation trapdoor. When doing so after having learned all other nonces, this allows the simulation to control the final nonce and thus achieve the reduction to the underlying blind Schnorr signature scheme.

5.4 Blind FROST Construction

The scheme FROST, short for Flexible Round-Optimized Schnorr Threshold Signatures, by Komlo and Goldberg [KG20], is one of the most popular state-of-the-art threshold Schnorr signature schemes, which also motivated many follow-up publications [CKM21; Gon+21; BTZ22; Ruf+22].

The natural question arises if our threshold blind Schnorr signature construction could also be built by using FROST instead of Lindell's threshold signature scheme [Lin22]. We give the affirmative answer by introducing the construction BFROST that has fewer communication rounds for the signers compared to TBSS but in turn, requires a (non-interactive) preprocessing phase.

Setup: The setup algorithm BFROST.Setup is the same as TBSS.Setup except that it outputs two hash functions H_1, H_2 instead of one. Following, given 1^λ as an input, the algorithm outputs the public parameters as $pp := (p, \mathbb{G}, G, H_1, H_2, \text{crSec}, \text{crSzK})$

Key Generation: The key generation algorithm BFROST.KeyGen works the same as TBSS.KeyGen .

Signature Generation: The signing protocol $\langle \text{Sign}(sk_1, \mathcal{V}), \dots, \text{Sign}(sk_n, \mathcal{V}), \text{User}(pk, m) \rangle$ between the signers S_i is depicted in Figure 5.5 and includes a preprocessing phase for the signers, depicted in Figure 5.4. We denote the set of active signers during the signature generation phase with $\mathcal{S}_0 \subseteq \mathcal{S}$, where $|\mathcal{S}_0| \geq t$. Each signer S_i is given its secret key sk_i , parsed as $x_i := sk_i$, and the set of all verification keys $\{vk_1, \dots, vk_n\} := \mathcal{V}$, parsed as $X_i := vk_i$ as an input. The user's U input is the public key pk , which it parses as $X := pk$, and a message m . The protocol consists of a preprocessing phase for the signers, which can be executed in advance and does not depend on the message to be signed, an initialization step for the signers, two signing stages, and two user stages that are described below:

Preprocess: The preprocessing phase is executed by the signers prior to the signing stages and ends with outputting a tuple of two public lists (L_0, L_1) or rather a reference to them. The lists contain the public nonces of the signers, i.e., $L_k = ((D_{k,1}, E_{k,1}), \dots, (D_{k,n}, E_{k,n}))$ for $k \in \{0, 1\}$, where each signer S_i stores the corresponding discrete logarithms $(d_{k,i}, e_{k,i})$ (the secret nonces) locally. During the preprocessing phase, all signers also have to submit a zero-knowledge proof of knowledge for the discrete logarithms $(d_{k,i}, e_{k,i})$. Those proofs are broadcasted to all other signers and have to be verified before proceeding with the signature generation protocol. Signers that submitted invalid proofs are reported and their public nonces are removed from the lists.

We stress that this phase does not depend on the message to be signed and is non-interactive since all steps can be performed asynchronously, also the published zero-knowledge proofs can be verified at any time (but before beginning the signature generation phase). Therefore, the preprocessing phase can be run arbitrarily many times to have more available nonces in preparation for future signature generations.

Initialization: Same as in TBSS.

Signing Stage 1: Initially, the signers agree on the nonce pairs taken from the preprocessing lists (L_0, L_1) they want to use, concretely, they decide on a new list $B_k := (D_{k,j}, E_{k,j})_{j \in \mathcal{S}_0}$ for $k \in \{0, 1\}$. Then, they compute the *binding values* as $\rho_{j,0} \leftarrow H_1(j, 0, B_0)$ for $j \in \mathcal{S}_0$ and compute the group's nonce as $R_0 \leftarrow \sum_{j \in \mathcal{S}_0} (D_{0,j} + E_{0,j} \rho_{0,j})$ and repeat these steps for B_1 . The final two group nonces (R_0, R_1) are sent to the user.

User Stage 1: Same as in TBSS.

Signing Stage 2: After receiving the challenges c_0, c_1 from the user, the signers collaboratively sample a bit b deciding which R_0, R_1 and c_0, c_1 will be used for the signature. Each signer first chooses a random bit b_i , commits to it using the equivocable commitment scheme EC, broadcasts the commitment to the other signers, and reveals the bit and opening value after receiving all other commitments. Using the opening values, each signer verifies the commitments of the other signers, misbehaving signers are reported and the protocol aborts. The resulting bit b is calculated as the sum of all signers' bits b_i modulo 2. Each signer then computes and broadcasts its signature value s_i using the secret key share x_i and the secret nonces $(d_{b,i}, e_{b,i})$ as $s_i \leftarrow d_{b,i} + e_{b,i} \rho_{b,i} + c_b x_i \bmod p$. Before computing the final response s , the well-formedness of each signature value is checked with $s_j G = R_{b,j} + c_b X_j$ for $j \in \mathcal{S}_0$, where, $R_{b,j} \leftarrow D_{b,j} + E_{b,j} \rho_{b,j}$, while for an invalid s_j , signer S_j is reported and the protocol aborts. The final response s , which is computed as the sum of all s_i values, and the bit b are then sent back to the user. Lastly, the signers output 1 to indicate success.

User Stage 2: Same as in TBSS, where the user ends up with outputting the signature $\sigma := (R'_b, s')$.

BFROST Preprocessing Phase

Let L_0, L_1 be public lists. For $i \in \mathcal{S}$ and $k \in \{0, 1\}$:

1. $(d_{k,i}, e_{k,i}) \leftarrow \mathbb{Z}_p \times \mathbb{Z}_p$ and store locally
 2. $(D_{k,i}, E_{k,i}) \leftarrow (d_{k,i}G, e_{k,i}G)$ and append to $L_k[i]$
 3. $(\pi_{k,i}, \mu_{k,i}) \leftarrow \text{NIZK.Prove}(crs_{ZK}, D_{k,i}, d_{k,i}), \text{NIZK.Prove}(crs_{ZK}, E_{k,i}, e_{k,i})$
and broadcast to $\mathcal{S}$
 4. Verify $(\pi_{k,j}, \mu_{k,j})$, for $j \in \mathcal{S} \setminus \{i\}$ and remove $(D_{k,j}, E_{k,j})$ from L_k if invalid
- Output (L_0, L_1)

Figure 5.4: The BFROST Preprocessing Stage.

Verify: The verification algorithm `BFROST.Verify` works the same as `TBSS.Verify`.

The apparent compromise we had to take is to create the binding value $\rho_{k,i}$ without including the message m as it remains unknown for the signers in blind signatures. In FROST, including the message in the binding value prevents an adversary from combining responses for different messages within an attack similar to the one from Drijvers et al. [Dri+19]. For blind signatures, an adversarial user thus can still adapt m and the modified nonce R'_k to perform a ROS attack despite of the binding value. Therefore, the clause modification from Fuchsbaauer et al. [FPS19] is still indispensable to protect against such attacks.

Another change to FROST is that we require that each participant submits a zero-knowledge proof of knowledge for $(b_{k,i}, d_{k,i})$ during the preprocessing phase. In the second signing stage, this enables us to compute all signature shares s_i in advance to patch $s_{i'}$ (similar to the proof of TBSS) such that the summed value s results in the signature received from the reduction.

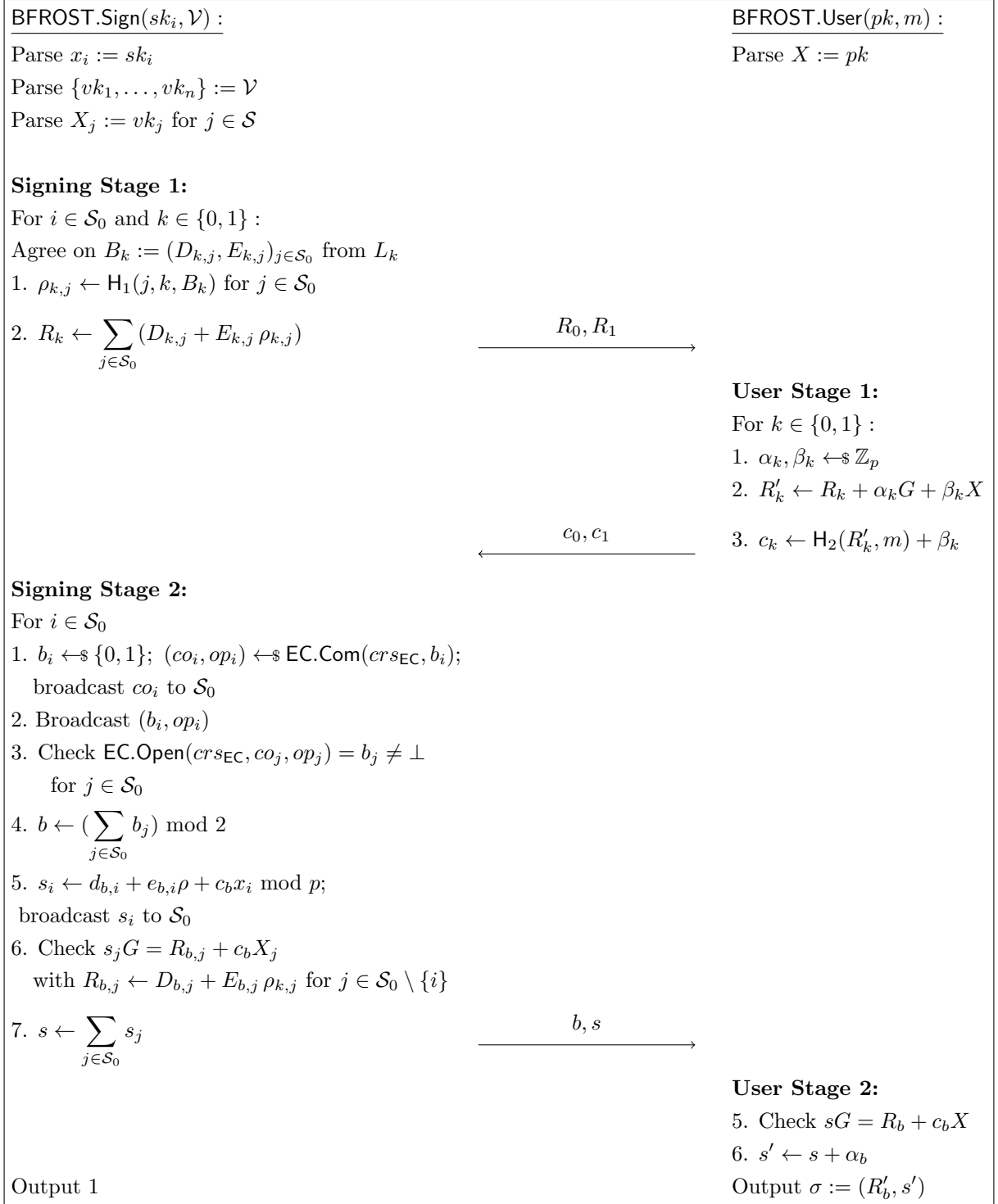


Figure 5.5: The BFROST Signing Protocol.

5.4.1 Proof of Security

The originally proposed proof of FROST by Komlo and Goldberg [KG20] is carried out by first proving the security of a variant of FROST, called FROST-Interactive, under the DL assumption. The essential difference between FROST and FROST-Interactive is that the binding value ρ_i is computed with a verifiable random function (VRF), which requires one additional communication round during the signing process and an interactive preprocessing stage with zero-knowledge proofs. The authors then heuristically show that the replacement of the VRF with a hash function, as in FROST, does not alter the security.

In the follow-up work by Crites et al. [CKM21], the authors present a security proof of a slightly optimized variant of FROST using new, immediate assumptions, namely the Schnorr knowledge of exponent assumption (Schnorr-KoE) and the binonce Schnorr computational assumption (Bischnorr). While the Schnorr-KoE assumption has been reduced to the hardness of the DL problem in the algebraic group model, the Bischnorr computational assumption is implied by the OMDL assumption in the random oracle model. The Schnorr-KoE assumption is mainly concerned with extracting the discrete logarithm of a verifying Schnorr signature, which is leveraged to extract the adversary's secret shares during the distributed key generation protocol. The latter Bischnorr computational assumption is oriented to the actual signing protocol of the FROST scheme and allows reducing the complexity of proving FROST directly considering multiple parties and rewinding or forking techniques.

A different proof of the same optimized version of FROST has been conducted by Bellare et al. [BTZ22]. They did not include FROST's distributed key generation protocol in their proof setting but only considered the signing protocol. By first defining a hierarchy of security notions, which gets less strict about declaring a forgery as trivial, they showed how the optimized version falls under a stronger notion. The proof of Bellare et al. does not rely on an intermediate assumption, instead, it reduces the unforgeability directly to the OMDL assumption in the random oracle model using rewinding.

For our blind FROST construction BFROST, we see two general approaches for performing a proof of unforgeability, which we will now informally introduce.

VRF Heuristic: The first approach works similarly to the original proof of FROST using a heuristic of an interactive version of the scheme where the binding value $\rho_{k,i}$ is computed using a verifiable random function (VRF). The computation of the binding value is then done as $\rho_{k,i} \leftarrow a_{k,i} + b_{k,i} \mathbf{H}(i, k, B)$, where the nonces $A_{k,i} = a_{k,i}G, B_{k,i} = b_{k,i}G$ are also published in the preprocessing phase along with a zero-knowledge proof of knowledge of $a_{k,i}, b_{k,i}$. This change requires that each signer first shares its binding value with the other signers before being able to compute the challenge. Due to the zero-knowledge proofs, however, the simulator is able to compute all binding values prior to the corrupt parties. In the proof of FROST, the simulator then programs the random oracle for the challenge such that it allows a reduction to the DL problem after having forked the adversary.

In BFROST, we would instead program the random oracle for the binding value of a random honest signer $S_{i'}$ so that the resulting group nonce R_k equals the one received from the reduction to BSS. The rest of the proof remains analogous to the proof of TBSS.

Komlo and Goldberg finally heuristically show that the replacement of the VRF with a hash function does not alter the security of the scheme.

Intermediate Assumption: The second approach would build upon an intermediate assumption in a similar manner as in the proof of FROST by Crites et al. [CKM21] to achieve a direct reduction to the MROS- and OMDL problem (direct in the sense of not reducing it to the underlying scheme BSS). The relevant intermediate assumption by Crites et al. is the binonce Schnorr computational assumption (Bischnorr), which is oriented to the signing protocol of FROST. The intermediate assumption allows simplifying the proof by separating it from parts that consider interactions between different signers and parts that require rewinding or forking techniques. A proof without introducing an intermediate

assumption as done by Bellare et al. [BTZ22] should also be possible but may be more involved.

The possible security game of the intermediate assumption consists of three main oracles, namely **ONonce**, **OSelect**, and **OSign**.

The **ONonce** oracle returns a tuple consisting of a session index and public nonces $(l, (D_0, E_0), (D_1, E_1))$.

The **OSelect** oracle works the same as in the **MROS** game, which is given a session index l and a challenge tuple (c_0, c_1) . It internally samples a bit b_j and sets $c_j \leftarrow c_{b_j}$.

The **OSign** oracle is queried with a signer index $i \in \mathcal{S}_0$, a session index l , and a list of nonces $((D_{j,0}, E_{j,0}), (D_{j,1}, E_{j,1}))$ for $j \in \mathcal{S}_0$. It checks whether **OSelect** has been queried for session l , if the nonces for signer S_i have been given as an output by the oracle **ONonce**, and then returns (b_j, s_i) , where s_i is the signature value from signer S_i calculated like in **BFROST**.

Besides the three oracles, an adversary is given access to hash functions H_1 and H_2 that are modeled as random oracles.

The adversary wins if it outputs k unique and valid signature pairs (m_i, σ_i) and it holds that $k > q_s$, where q_s is the number of **OSign** queries.

One can consider this security game as a combination of the Bischnorr - and the **MROS** game. While Crites et al. reduced the Bischnorr assumption to the **OMDL** assumption in the random oracle model, Fuchsbauer et al. reduced the clause blind Schnorr signature scheme **BSS**, on which the **BFROST** scheme is based, to the **OMDL** and the **MROS** assumption in the algebraic group model. We expect that **BFROST** can be reduced to the assumption represented by this security game and that this assumption in turn can be reduced to the **OMDL**- and **MROS** assumption in the algebraic group model.

5.4.2 Optimizations

The presented **BFROST** scheme is constructed with the intention of deviating as little as possible from the original **FROST** scheme so that systems that already support **FROST** signatures only need to make a few changes to be able to issue blind signatures.

However, this imposes computational and communication overhead due to the inclusion of parts of **FROST** that are not necessary for the security of the resulting blind signature scheme.

In particular, as the binding value does not have the task of preventing Drijvers - or **ROS** attacks in **BFROST**, it should suffice that the signers only generate 2 nonces $(R_{0,i}, R_{1,i})$ per signing session or preprocessing run instead of the 4 nonces $((D_{i,0}, E_{i,0}), (D_{i,1}, E_{i,1}))$.

Furthermore, when proving **BFROST** based on the second approach instead of reducing it to **BSS** using the **VRF** heuristic, we should be able to carry out three additional optimizations.

First, we do not need zero-knowledge proofs for the published nonces during the preprocessing phase anymore. Since we do not reduce **BFROST** to the underlying **BSS** scheme, where we receive the final signature value s from the reduction and then patch the $s_{i'}$ value of signer $S_{i'}$, we get access to the **Dlog** oracle from the **OMDL** game. Using the oracle, the simulator is able to calculate the correct $s_{i'}$ value without knowledge of $S_{i'}$'s secret share or of s_j for $j \in \mathcal{S}_C$ (for which it would need the secret nonces of the corrupt parties). The simulator does so by querying the **Dlog** oracle on $R_{b,i'} + c_b X_{i'}$.

Second, we can optimize the computation of the binding value as proposed by Crites et al. [CKM21]. Instead of having an individual binding value $\rho_{k,i}$ per signer, it stays the same for all signers and is computed as $\rho_k \leftarrow H_1(k, B_k)$. Then, the multiplications for each signer are reduced from the number of signers $|\mathcal{S}_0|$ to only 1.

Lastly, along with the previous optimization we can let the user generate the group's nonces (R_0, R_1) itself by giving the user access to the preprocessing lists (L_0, L_1) . The protocol would therefore begin with the user sending the challenge tuple (c_0, c_1) to the signers. Note that in practice, the user would probably also include a set of indices such that the signers know which nonces to use for the protocol. This optimization reduces the moves of interactions from 3 to only 2.

6 Conclusion

6.1 Contributions

As the main contribution of this thesis, we presented the construction of the first threshold blind Schnorr signature scheme (cp. Section 5.3), which is directly compatible with a variety of blockchain systems as the verification remains the same as standard Schnorr. The threshold blind Schnorr signature scheme has been shown to be provably secure if the underlying blind Schnorr signature scheme of Fuchsbaauer et al. [FPS19] is secure, which holds for the unforgeability and the perfect blindness. We additionally proposed another alternative construction based on FROST (cp. Section 5.4), which makes use of a preprocessing phase to reduce the number of signer rounds in the online phase. Furthermore, we gave a formal definition of threshold blind signatures (cp. Section 5.2) that captures concurrent security. This aspect is crucial for threshold and blind signature schemes based on Schnorr due to ROS (and ROS-related) attacks that were discussed in Section 2.4.

6.2 Future Work

Since recent attacks [Ben+22] have once again highlighted the insecurity of the ROS problem, there is a high demand for effective countermeasures and secure variants. Within those variants, we have the MROS and the WFROS problem. However, these novel assumptions have not yet undergone extensive cryptanalysis yet, which is necessary to build confidence in them.

Besides Schnorr, ECDSA counts as one of the most employed signature schemes in blockchain networks. Therefore, it could be of interest to see whether threshold blind constructions are also possible and how they would compare in terms of efficiency.

From the point of view of provable security, a proof of the blind FROST scheme BFROST is still open and would be particularly valuable for the optimized variant. Similarly, it could be valuable to prove the threshold blind Schnorr signature scheme TBSS directly to the OMDL and MROS assumption as the commit and reveal phase for the signers probably wouldn't be necessary anymore. Additionally, it would be worth exploring the practical aspects of implementing the proposed constructions for online wallets including challenges and opportunities for large-scale deployments.

Bibliography

- [Bla79] George Robert Blakley. “Safeguarding cryptographic keys”. In: *Managing Requirements Knowledge, International Workshop on*. IEEE Computer Society. 1979, pp. 313–313.
- [Sha79] Adi Shamir. “How to share a secret”. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.
- [Cha82] David Chaum. “Blind Signatures for Untraceable Payments”. In: *Annual International Cryptology Conference*. 1982.
- [Fel87] Paul Feldman. “A practical scheme for non-interactive verifiable secret sharing”. In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE. 1987, pp. 427–438.
- [Cha88] David Chaum. “Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA”. In: *Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1988, pp. 177–182.
- [CFN88] David Chaum, Amos Fiat, and Moni Naor. “Untraceable electronic cash”. In: *Conference on the Theory and Application of Cryptography*. Springer. 1988, pp. 319–327.
- [Sch89] Claus-Peter Schnorr. “Efficient identification and signatures for smart cards”. In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, pp. 239–252.
- [DF90] Yvo Desmedt and Yair Frankel. “Threshold cryptosystems”. In: *Advances in Cryptology — CRYPTO’89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 307–315. ISBN: 978-0-387-34805-6.
- [OO91] Tatsuaki Okamoto and Kazuo Ohta. “Universal electronic cash”. In: *Annual international cryptology conference*. Springer. 1991, pp. 324–337.
- [Ped91] Torben Pryds Pedersen. “A threshold cryptosystem without a trusted party”. In: *Advances in Cryptology—EUROCRYPT’91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*. Springer. 1991, pp. 522–526.
- [Sch91] Claus-Peter Schnorr. “Efficient signature generation by smart cards”. In: *Journal of cryptology* 4.3 (1991), pp. 161–174.
- [CP92] David Chaum and Torben Pryds Pedersen. “Wallet databases with observers”. In: *Annual international cryptology conference*. Springer. 1992, pp. 89–105.
- [FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. “A practical secret voting scheme for large scale elections”. In: *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1992, pp. 244–251.
- [Oka92] Tatsuaki Okamoto. “Provably secure and practical identification schemes and corresponding signature schemes”. In: *Annual international cryptology conference*. Springer. 1992, pp. 31–53.
- [Bra93] Stefan Brands. “Untraceable off-line cash in wallet with observers”. In: *Annual international cryptology conference*. Springer. 1993, pp. 302–318.
- [De +94] Alfredo De Santis et al. “How to share a function securely”. In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. 1994, pp. 522–533.
- [Nec94] Vassiliy Ilyich Nechaev. “Complexity of a determinate algorithm for the discrete logarithm”. In: *Mathematical Notes* 55.2 (1994), pp. 165–172.

- [SPC95] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. “Fair blind signatures”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1995, pp. 209–219.
- [AF96] Masayuki Abe and Eiichiro Fujisaki. “How to date blind signatures”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 1996, pp. 244–251.
- [Gen+96] Rosario Gennaro et al. “Robust and efficient sharing of RSA functions”. In: *Annual International Cryptology Conference*. Springer. 1996, pp. 157–172.
- [JL96] Wen-Shenq Juang and Chin-Laung Lei. “Blind threshold signatures based on discrete logarithm”. In: *Annual Asian Computing Science Conference*. Springer. 1996, pp. 172–181.
- [PS96a] David Pointcheval and Jacques Stern. “Provably secure blind signature schemes”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 1996, pp. 252–265.
- [PS96b] David Pointcheval and Jacques Stern. “Security proofs for signature schemes”. In: *International conference on the theory and applications of cryptographic techniques*. Springer. 1996, pp. 387–398.
- [Her+97] Amir Herzberg et al. “Proactive public key and signature systems”. In: *Proceedings of the 4th ACM Conference on Computer and Communications Security*. 1997, pp. 100–110.
- [Sho97] Victor Shoup. “Lower bounds for discrete logarithms and related problems”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1997, pp. 256–266.
- [Poi98] David Pointcheval. “Strengthened security for blind signatures”. In: *Eurocrypt*. Vol. 1403. 1998, pp. 391–405.
- [Gen+99] Rosario Gennaro et al. “Secure distributed key generation for discrete-log based cryptosystems”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1999, pp. 295–310.
- [Ohk+99] Miyako Ohkubo et al. “An improvement on a practical secret voting scheme”. In: *International Workshop on Information Security*. Springer. 1999, pp. 225–234.
- [AO00] Masayuki Abe and Tatsuaki Okamoto. “Provably secure partially blind signatures”. In: *Annual International Cryptology Conference*. Springer. 2000, pp. 271–286.
- [LJ00] WD Lin and JK Jan. “A security personal learning tools using a proxy blind signature scheme”. In: *Proceedings of International Conference on Chinese Language Computing*. Illinois, USA. 2000, pp. 273–277.
- [PS00] David Pointcheval and Jacques Stern. “Security arguments for digital signatures and blind signatures”. In: *Journal of cryptology* 13.3 (2000), pp. 361–396.
- [Sho00] Victor Shoup. “Practical threshold signatures”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2000, pp. 207–220.
- [Abe01] Masayuki Abe. “A secure three-move blind signature scheme for polynomially many signatures”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2001, pp. 136–151.
- [Bel+01] Mihir Bellare et al. “The power of RSA inversion oracles and the security of Chaum’s RSA-based blind signature scheme”. In: *International Conference on Financial Cryptography*. Springer. 2001, pp. 319–338.
- [CL01] Jan Camenisch and Anna Lysyanskaya. “An efficient system for non-transferable anonymous credentials with optional anonymity revocation”. In: *International conference on the theory and applications of cryptographic techniques*. Springer. 2001, pp. 93–118.

- [Ped01] Torben Pryds Pedersen. “Non-interactive and information-theoretic secure verifiable secret sharing”. In: *Advances in Cryptology—CRYPTO’91: Proceedings*. Springer. 2001, pp. 129–140.
- [Sch01] Claus Peter Schnorr. “Security of blind discrete log signatures against interactive attacks”. In: *International Conference on Information and Communications Security*. Springer. 2001, pp. 1–12.
- [SS01] Douglas R Stinson and Reto Strobl. “Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates”. In: *Australasian Conference on Information Security and Privacy*. Springer. 2001, pp. 417–434.
- [BP02] Mihir Bellare and Adriana Palacio. “GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks”. In: *Annual International Cryptology Conference*. Springer. 2002, pp. 162–177.
- [Bol02] Alexandra Boldyreva. “Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme”. In: *Public Key Cryptography—PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6*. Springer. 2002, pp. 31–46.
- [KKL02] Jinho Kim, Kwangjo Kim, and Chulsoo Lee. “An efficient and provably secure threshold blind signature”. In: *International Conference on Information Security and Cryptology*. Springer. 2002, pp. 318–327.
- [LJY02] Chin-Laung Lei, Wen-Shenq Juang, and Pei-Ling Yu. “Provably secure blind threshold signatures based on discrete logarithm”. In: *Journal of Information Science and Engineering* 18.1 (2002), pp. 23–39.
- [SG02] Victor Shoup and Rosario Gennaro. “Securing threshold cryptosystems against chosen ciphertext attack”. In: *Journal of Cryptology* 15.2 (2002), pp. 75–96.
- [Wag02] David Wagner. “A generalized birthday problem”. In: *Annual International Cryptology Conference*. Springer. 2002, pp. 288–304.
- [ZK02] Fangguo Zhang and Kwangjo Kim. “ID-based blind signature and ring signature from pairings”. In: *International conference on the theory and application of cryptology and information security*. Springer. 2002, pp. 533–547.
- [Bel+03] Mihir Bellare et al. “The One-More-RSA-Inversion Problems and the Security of Chaum’s Blind Signature Scheme.” In: *Journal of Cryptology* 16.3 (2003).
- [Bol03] Alexandra Boldyreva. “Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme”. In: *International Workshop on Public Key Cryptography*. Springer. 2003, pp. 31–46.
- [DG03] Ivan Damgard and Jens Groth. “Non-Interactive and Reusable Non-Malleable Commitment Schemes”. In: *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*. STOC ’03. San Diego, CA, USA: Association for Computing Machinery, 2003, pp. 426–437. ISBN: 1581136749. DOI: 10.1145/780542.780605. URL: <https://doi.org/10.1145/780542.780605>.
- [VZK03] Duc Liem Vo, Fangguo Zhang, and Kwangjo Kim. “A new threshold blind signature scheme from pairings”. In: *SCIS2003*. SCIS. 2003, pp. 233–238.
- [BCC04] Ernie Brickell, Jan Camenisch, and Liqun Chen. “Direct anonymous attestation”. In: *Proceedings of the 11th ACM conference on Computer and communications security*. 2004, pp. 132–145.
- [CKW04] Jan Camenisch, Maciej Koprowski, and Bodgan Warinschi. “Efficient blind signatures without random oracles”. In: *International Conference on Security in Communication Networks*. Springer. 2004, pp. 134–148.

- [Cha+04] Tony K Chan et al. “Blind spontaneous anonymous group signatures for ad hoc groups”. In: *European Workshop on Security in Ad-hoc and Sensor Networks*. Springer. 2004, pp. 82–94.
- [Sho04] Victor Shoup. “Sequences of games: a tool for taming complexity in security proofs”. In: *cryptology eprint archive* (2004).
- [Cho+05] Sherman SM Chow et al. “Two improved partially blind signature schemes from bilinear pairings”. In: *Australasian Conference on Information Security and Privacy*. Springer. 2005, pp. 316–328.
- [SJ05] Claus-Peter Schnorr and Markus Jakobsson. “Security of Discrete Log Cryptosystems in the Random Oracle and the Generic Model”. In: 2005.
- [YW05] Tsz Hon Yuen and Victor K Wei. “Fast and proven secure blind identity-based signcryption from pairings”. In: *Cryptographers’ Track at the RSA Conference*. Springer. 2005, pp. 305–322.
- [CZL06] Zhenfu Cao, Haojin Zhu, and Rongxing Lu. “Provably secure robust threshold partial blind signature”. In: *Science in China Series F: Information Sciences* 49.5 (2006), pp. 604–615.
- [KC06] Young-Seol Kim and Jik-Hyun Chang. “Provably secure proxy blind signature scheme”. In: *Eighth IEEE International Symposium on Multimedia (ISM’06)*. IEEE. 2006, pp. 998–1003.
- [Oka06] Tatsuaki Okamoto. “Efficient blind and partially blind signatures without random oracles”. In: *Theory of Cryptography Conference*. Springer. 2006, pp. 80–99.
- [Haz+07] Carmit Hazay et al. “Concurrently-secure blind signatures without random oracles or setup assumptions”. In: *Theory of Cryptography Conference*. Springer. 2007, pp. 323–341.
- [LYK07] Jin Li, Tsz Hon Yuen, and Kwangjo Kim. “Practical threshold signatures without random oracles”. In: *International Conference on Provable Security*. Springer. 2007, pp. 198–207.
- [Wu+07] Qianhong Wu et al. “Efficient partially blind signatures with provable security”. In: *International Conference on Computational Science and Its Applications*. Springer. 2007, pp. 1096–1105.
- [Bel+09] Mira Belenkiy et al. “Compact e-cash and simulatable VRFs revisited”. In: *International Conference on Pairing-Based Cryptography*. Springer. 2009, pp. 114–131.
- [FS10] Marc Fischlin and Dominique Schröder. “On the impossibility of three-move blind signature schemes”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2010, pp. 197–215.
- [Rüc10] Markus Rückert. “Lattice-based blind signatures”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2010, pp. 413–430.
- [Fuc11] Georg Fuchsbauer. “Commuting signatures and verifiable encryption”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2011, pp. 224–245.
- [Gar+11] Sanjam Garg et al. “Round optimal blind signatures”. In: *Annual Cryptology Conference*. Springer. 2011, pp. 630–648.
- [Har11] Kristiyan Haralambiev. “Efficient Cryptographic Primitives for Non-Interactive Zero-Knowledge Proofs and Applications”. AAI3466893. PhD thesis. USA, 2011. ISBN: 9781124807942.
- [BL13a] Foteini Baldimtsi and Anna Lysyanskaya. “Anonymous credentials light”. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 1087–1098.

- [BL13b] Foteini Baldimtsi and Anna Lysyanskaya. “On the security of one-witness blind signature schemes”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2013, pp. 82–99.
- [Bla+13] Olivier Blazy et al. “Short blind signatures”. In: *Journal of computer security* 21.5 (2013), pp. 627–661.
- [GG14] Sanjam Garg and Divya Gupta. “Efficient round optimal blind signatures”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2014, pp. 477–495.
- [KM15] Veronika Kuchta and Mark Manulis. “Rerandomizable Threshold Blind Signatures”. In: *Trusted Systems*. Ed. by Moti Yung, Liehuang Zhu, and Yanjiang Yang. Cham: Springer International Publishing, 2015, pp. 70–89. ISBN: 978-3-319-27998-5.
- [GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. “Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security”. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2016, pp. 156–174.
- [Hei+16] Ethan Heilman et al. “Tumblebit: An untrusted bitcoin-compatible anonymous payment hub”. In: *Cryptology ePrint Archive* (2016).
- [GM17] Jens Groth and Mary Maller. “Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs”. In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Cham: Springer International Publishing, 2017, pp. 581–612. ISBN: 978-3-319-63715-0.
- [ZGP17] Alexandros Zacharakis, Panagiotis Grontas, and Aris Pagourtzis. “Conditional blind signatures”. In: *Cryptology ePrint Archive* (2017).
- [BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. “Compact multi-signatures for smaller blockchains”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2018, pp. 435–464.
- [Cha18] Chainalysis. *Bitcoin’s \$30 billion sell-off*. June 2018. URL: <https://blog.chainalysis.com/reports/money-supply/> (visited on 11/15/2022).
- [FKL18] Georg Fuchsbaauer, Eike Kiltz, and Julian Loss. “The algebraic group model and its applications”. In: *Annual International Cryptology Conference*. Springer. 2018, pp. 33–62.
- [GG18] Rosario Gennaro and Steven Goldfeder. “Fast multiparty threshold ECDSA with fast trustless setup”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1179–1194.
- [LN18] Yehuda Lindell and Ariel Nof. “Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1837–1854.
- [Nic18] Jonas Nick. *Blind Signatures in Scriptless Scripts*. July 2018. URL: <https://jonasnick.github.io/blog/2018/07/31/blind-signatures-in-scriptless-scripts/> (visited on 12/14/2022).
- [Vor+18] Jayneel Vora et al. “Blind signatures based secured e-healthcare system”. In: *2018 International conference on computer, information and telecommunication systems (CITS)*. IEEE. 2018, pp. 1–5.
- [Dri+19] Manu Drijvers et al. “On the security of two-round multi-signatures”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1084–1101.
- [FPS19] Georg Fuchsbaauer, Antoine Plouviez, and Yannick Seurin. *Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model*. Cryptology ePrint Archive, Paper 2019/877. <https://eprint.iacr.org/2019/877>. 2019. URL: <https://eprint.iacr.org/2019/877>.

- [HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. “A modular treatment of blind signatures from identification schemes”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 345–375.
- [Liu+19] Yi Liu et al. “TumbleBit++: A Comprehensive Privacy Protocol Providing Anonymity and Amount-Invisibility”. In: *International Conference on Provable Security*. Springer. 2019, pp. 339–346.
- [Max+19] Gregory Maxwell et al. “Simple schnorr multi-signatures with applications to bitcoin”. In: *Designs, Codes and Cryptography* 87.9 (2019), pp. 2139–2164.
- [Pap+19] Dimitrios Papachristoudis et al. “Leakage-resilient lattice-based partially blind signatures”. In: *IET Information Security* 13.6 (2019), pp. 670–684.
- [AEB20] Nabil Alkeilani Alkadri, Rachid El Bansarkhani, and Johannes Buchmann. “BLAZE: practical lattice-based blind signatures for privacy-preserving applications”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2020, pp. 484–502.
- [BRS20] Samuel Brack, Leonie Reichert, and Björn Scheuermann. “Caudht: decentralized contact tracing using a DHT and blind signatures”. In: *2020 IEEE 45th Conference on Local Computer Networks (LCN)*. IEEE. 2020, pp. 337–340.
- [Can+20] Ran Canetti et al. “UC non-interactive, proactive, threshold ECDSA with identifiable aborts”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1769–1787.
- [Dar+20] Aaqib Bashir Dar et al. “Applicability of mobile contact tracing in fighting pandemic (COVID-19): Issues, challenges and solutions”. In: *Computer Science Review* 38 (2020), p. 100307.
- [KG20] Chelsea Komlo and Ian Goldberg. “FROST: flexible round-optimized Schnorr threshold signatures”. In: *International Conference on Selected Areas in Cryptography*. Springer. 2020, pp. 34–65.
- [Pie20] Tim Ruffing Pieter Wuille Jonas Nick. *Schnorr Signatures for secp256k1*. Jan. 2020. URL: <https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki> (visited on 11/15/2022).
- [App21] Apple. *iCloud Private Relay*. Dec. 2021. URL: https://www.apple.com/privacy/docs/iCloud_Private_Relay_Overview_Dec2021.PDF (visited on 12/14/2022).
- [BFP21] Balthazar Bauer, Georg Fuchsbaauer, and Antoine Plouviez. “The one-more discrete logarithm assumption in the generic group model”. In: *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV 27*. Springer. 2021, pp. 587–617.
- [Beh21] Rob Behnke. *Explained: The Bxh Exchange Hack (October 2021)*. Dec. 2021. URL: <https://halborn.com/explained-the-bxh-exchange-hack-october-2021/> (visited on 11/15/2022).
- [Ben21] Jeff Benson. *Hacker steals \$140 million from Polygon Gaming Platform, NFT marketplace Vulcan forged*. Dec. 2021. URL: <https://decrypt.co/88177/hacker-steals-140-million-polygon-gaming-platform-nft-marketplace-vulcan-forged> (visited on 11/15/2022).
- [CRR21] Geoffroy Couteau, AW Roscoe, and Peter YA Ryan. “Partially-fair computation from timed-release encryption and oblivious transfer”. In: *Australasian Conference on Information Security and Privacy*. Springer. 2021, pp. 330–349.
- [CKM21] Elizabeth Crites, Chelsea Komlo, and Mary Maller. “How to prove schnorr assuming schnorr: Security of multi-and threshold signatures”. In: *Cryptology ePrint Archive* (2021).

-
- [Gar+21] François Garillot et al. “Threshold schnorr with stateless deterministic signing from standard assumptions”. In: *Annual International Cryptology Conference*. Springer. 2021, pp. 127–156.
- [Gkr21] Eliza Gkritsi. *Crypto Exchange BitMart hacked with losses estimated at \$196M*. Dec. 2021. URL: <https://www.coindesk.com/tech/2021/12/06/bitmart-ceo-says-stolen-private-key-behind-196m-hack/> (visited on 11/15/2022).
- [Gon+21] Alonso González et al. “Identifiable Cheating Entity Flexible Round-Optimized Schnorr Threshold (ICE FROST) Signature Protocol”. In: *Cryptology ePrint Archive* (2021).
- [KLR21] Jonathan Katz, Julian Loss, and Michael Rosenberg. “Boosting the security of blind signature schemes”. In: *Advances in Cryptology—ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV* 27. Springer. 2021, pp. 468–492.
- [NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. “MuSig2: simple two-round Schnorr multi-signatures”. In: *Annual International Cryptology Conference*. Springer. 2021, pp. 189–221.
- [RS21] Lior Rotem and Gil Segev. *Non-Malleable Vector Commitments via Local Equivocability*. Cryptology ePrint Archive, Paper 2021/1207. <https://eprint.iacr.org/2021/1207>. 2021. URL: <https://eprint.iacr.org/2021/1207>.
- [Sun+21] Yi Sun et al. “PMRSS: privacy-preserving medical record searching scheme for intelligent diagnosis in IoT healthcare”. In: *IEEE Transactions on Industrial Informatics* 18.3 (2021), pp. 1981–1990.
- [Agr+22] Shweta Agrawal et al. “Practical, round-optimal lattice-based blind signatures”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 39–53.
- [BTZ22] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. “Stronger security for non-interactive threshold signatures: BLS and FROST”. In: *Cryptology ePrint Archive* (2022).
- [Ben+22] Fabrice Benhamouda et al. “On the (in) security of ROS”. In: *Journal of Cryptology* 35.4 (2022), pp. 1–25.
- [Blo22] Blockchain.com. *Blockchain.com Wallets*. 2022. URL: <https://www.blockchain.com/explorer/charts/my-wallet-n-users> (visited on 11/15/2022).
- [Cha+22] Rutchathon Chairattana-Apirom et al. “PI-Cut-Choo and Friends: Compact Blind Signatures via Parallel Instance Cut-and-Choose and More”. In: *Advances in Cryptology—CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part III*. Springer. 2022, pp. 3–31.
- [Coi22] Coinbase. *Why Digital Signatures are essential for blockchains*. Jan. 2022. URL: <https://www.coinbase.com/en/cloud/discover/dev-foundations/digital-signatures> (visited on 11/15/2022).
- [DH22] Whitfield Diffie and Martin E Hellman. “New directions in cryptography”. In: *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*. 2022, pp. 365–390.
- [FW22] Georg Fuchsbauer and Mathias Wolf. “(Concurrently Secure) Blind Schnorr from Schnorr”. In: *Cryptology ePrint Archive* (2022).
- [Jha22] Prashant Jha. *The aftermath of Axie Infinity’s \$650m Ronin Bridge hack*. Apr. 2022. URL: <https://cointelegraph.com/news/the-aftermath-of-axie-infinity-s-650m-ronin-bridge-hack> (visited on 11/15/2022).
- [KLX22] Julia Kastner, Julian Loss, and Jiayu Xu. “On pairing-free blind signature schemes in the algebraic group model”. In: *IACR International Conference on Public-Key Cryptography*. Springer. 2022, pp. 468–497.
-

- [Lin22] Yehuda Lindell. *Simple Three-Round Multiparty Schnorr Signing with Full Simulatability*. Cryptology ePrint Archive, Paper 2022/374. <https://eprint.iacr.org/2022/374>. 2022. URL: <https://eprint.iacr.org/2022/374>.
- [Ruf+22] Tim Ruffing et al. “ROAST: Robust asynchronous Schnorr threshold signatures”. In: *Cryptology ePrint Archive* (2022).
- [TZ22] Stefano Tessaro and Chenzhi Zhu. “Short pairing-free blind signatures with exponential security”. In: *Cryptology ePrint Archive* (2022).
- [Coi23] Coingecko. *Global Cryptocurrency Market Cap Charts*. 2023. URL: <https://www.coingecko.com/en/global-charts> (visited on 03/09/2023).
- [Goo] Google. *VPN by Google One*. URL: <https://one.google.com/about/vpn/howitworks> (visited on 12/14/2022).