BLS Threshold Blind Signatures

January 15, 2022

Abstract

Threshold Blind Signatures are useful for federated e-cash [1] and voting [2] schemes, making these potentially more robust and secure. Both blind BLS signatures and threshold BLS signatures were described by Boldyreva [3] but no combination of these two aspects was formally analyzed so far. We describe a threshold blind signature scheme built from Boldyreva's blind and threshold schemes and prove it secure under the One More co-Computational Diffie-Hellman assumption in the random oracle model.

Contents

1. Introduction

Blind signature protocols allow a user to acquire a signature on a message without revealing the message to the signer and are a key building block for privacy preserving applications. They were originally invented by Chaum to construct an e-cash scheme [1] but later other schemes based on blind signatures like electronic voting [2] were invented.

It might be unituitive why a signer would sign a message without knowing it, so we will shortly explain the basic idea behind Chaumian e-cash. Let there be a bank with a signing key. Each signature with this key is worth 1\$. To acquire e-cash tokens a user sends x\$ to the bank and draws x random tokens $t_1, \ldots, t_x$. The bank will then engage in x rounds of the blind signature scheme resulting in the user receiving signatures σ_i for all t_i. Each pair (t_i, σ_i) is now worth 1\$ and can be redeemed at the bank. Since the bank does not learn t_i during the signing protocol the issuance and redemption are unlinkable and the signed tokens are thus anonymous.

One apparent problem with such a scheme is the bank being a single point of failure. Both its integrity and availability can be easily attacked by attacking one site or entity. Moreover the bank itself could easily "print" more tokens than it has collateral and become fractional reserve. The blind nature of the signatures makes audits much harder.

In such a high stakes setting it is desirable to split the trust among multiple sites or even entities to increase robustness and reduce the required trust. To do so a (t, n)-threshold blind signature scheme is required, meaning that any t of n signers can together create a signature but not $t - 1$ or fewer.

Multiple threshold blind signature schemes are described in the literature, but none are optimal for said use case. Early schemes are based on the discrete logarithm assumption [4, 5] or RSA [6], but lack a proper security model or are known to be insecure under strong adversary models. Kim et al.'s scheme [5] is based on Okamoto-Schnorr blind signatures, which are only secure for less than $\log(\lambda)$ parallel executions [7]. There also exist pairing based schemes [8, 9], but these are not optimized for real-world use cases. Vo et al. [8] base their scheme on a threshold scheme described by Boldyreva [3]. Boldyreva also describes a blind signature scheme, but no threshold blind signature scheme.

1.1. Contribution

We construct a pairing-based threshold blind signature scheme from Boldyreva's schemes [3]. Our scheme is similar to that by Vo, Zhang and Kim [8] but uses a slightly different and more efficient way of blinding messages and is proven secure for more practical pairings, which enables easy implementation using widespread pairing curves like BLS12-381. We prove our scheme $\mathcal{TBS}$-$\mathcal{BLS}$ secure under the One More co-Computational

Diffie-Hellman assumption in the Random Oracle Model. We also include more detailed proofs for Boldyreva's threshold signature scheme.

1.2. Structure

The paper is structured as follows: section 2.1 defines custom notation used throughout the paper, section 2.2 introduces pairings and cryptographic assumptions about these, the following sections of chapter 2 are dedicated to security definitions of threshold signature schemes (section 2.6) and threshold blind signature schemes (section 2.7). Section 2.8 describes one possible protocol instantiation of a threshold blind signature scheme following the definition from section 2.7. Chapter 3 recounts the threshold signature scheme by Boldyreva [3] and proves it secure. Chapter 4 describes our threshold blind signature scheme and proves it secure. Chapter 5 points out further research topics arising from our work.

2. Preliminaries

2.1. Notation

Throughout the paper we use the following custom notation:

$[f(\cdot)]$: the support $[f(\cdot)]$ of algorithm or protocol $f(\cdot)$ is defined as the set of possible outputs of f: $[f(\cdot)] = \{x \mid P[f(\cdot) = x] > 0\}$

$\texttt{take}(t, S)$: choose any $S' \subseteq S$ such that $|S'| = t$. Any, possibly random, subset may be chosen.

$x \leftarrow\$\, X$: choose an element $x \in X$ uniformly at random.

When referring to indexed variables like σ_i we assume that the value is actually tagged with the index i and said tag is accessible. This means σ_i can be thought of as a pair (i, σ_i), making i accessible to the algorithms working with σ_i. While this is a non-standard notation we do this to improve readability.

Some of the adversaries defined in this paper are interactive and consist of multiple sub-algorithms $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$. These are assumed to be stateful, i.e. any $\mathcal{A}_i$ has access to all inputs and internal state of $\mathcal{A}_j$ for all $j \leq i$.

2.2. Cryptographic Assumptions

For the rest of this paper, we will assume $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_T$ to be additive cyclic groups of prime order q and $G_1 \in \mathcal{G}_1$ and $G_2 \in \mathcal{G}_2$ fixed generators of these.

Definition 1 (Pairing). *A pairing $e : \mathcal{G}_1 \times \mathcal{G}_2 \rightarrow \mathcal{G}_T$ is a function with the following properties:*

- *Let $E_1 \in \mathcal{G}_1$ and $E_2 \in \mathcal{G}_2$ be group elements, then for any scalars a and b the following holds: $e(a \cdot E_1, b \cdot E_2) = a \cdot b \cdot e(E_1, E_2)$.*
- *e is not degenerate, meaning that $e(G_1, G_2)$ outputs a generator of $\mathcal{G}_T$.*
- *e is efficiently computeable.*

Definition 2 (coCDH [10]). *The co-computational Diffie-Hellman (coCDH) problem is hard for a group pair $(\mathcal{G}_1, \mathcal{G}_2)$ if for every polynomial time adversary $\mathcal{A}$, to which a isomorphism oracle $\varphi : \mathcal{G}_2 \rightarrow \mathcal{G}_1$ is given, the following holds true:*

$$Pr[\mathcal{A}^{\varphi}(G_2, aG_2, H) = aH \mid a \leftarrow\$\, \mathbb{Z}_q; H \leftarrow\$\, \mathcal{G}_1] = \mathsf{negl}(\lambda)$$

We note that we will need the isomorphism oracle φ only for security proofs and not schemes themselves. For the most desirable type of pairings (type 3) [11] we assume that there is no efficient way of calculating φ, but many schemes, including BLS [12], are not known to be provable secure without the reduction having access to φ. So we will assume such an oracle in our reductions as well.

Definition 3 (coDH tuple)**.** *A co-Diffie-Hellman tuple is a triple* $(A \in \mathcal{G}_1, B \in \mathcal{G}_2, C \in \mathcal{G}_1)$ *such that* $e(A, B) = e(C, G_2)$*.*

A generalization of coCDH is the One More co-Computational Diffie-Hellman problem (OMcoCDH), which allows the adversary more choice over the target elements for which to solve the coCDH problem. We use the game $\texttt{OMcoCDH}^{\varphi}_{\mathcal{A}}(1^\lambda)$ (fig. 2.1) to formally define OMcoCDH. Informally the game proceeds as follows:

- A challenger chooses a random scalar $x \in \mathbb{Z}_q$ and calculates $X = x \cdot G_2$.
- The adversary $\mathcal{A}$ is given:
  - The group element $X \in \mathcal{G}_2$.
  - A target oracle $\mathcal{O}_\mathrm{T}$, which returns random elements of $\mathcal{G}_1$ and keeps track of them.
  - A coCDH oracle $\mathcal{O}_\mathrm{DH}$, which returns a solution $Z = x \cdot Y$ to the computational co-DH problem for X and any other group element $Y \in \mathcal{G}_1$, such that (X, Y, Z) is a valid co-Diffie-Hellman tuple. The oracle counts its invocations in q_DH.
  - An isomorphism oracle $\varphi : \mathcal{G}_2 \to \mathcal{G}_1$ just like for coCDH.
- $\mathcal{A}$ returns a set DH of tuples (Y_i, Z_i) that form valid co-Diffie-Hellman tuples with X. $\mathcal{A}$ wins if $|DH| > q_\mathrm{DH}$ and all Y_i were generated by the target oracle $\mathcal{O}_\mathrm{T}$.

Definition 4 (OMcoCDH [13][1])**.** *We call the One More co-Computational Diffie Hellman (OMcoCDH) problem hard on a group if there exists no polynomial time adversary* $\mathcal{A}$ *that is able to win game* $\texttt{OMcoCDH}^{\varphi}_{\mathcal{A}}(1^\lambda)$ *(fig. 2.1) with more than negligible probability.*

2.3. Signature Schemes

A signature scheme $\mathcal{S} = (\texttt{Keygen}, \texttt{Sign}, \texttt{Verify})$ is a tuple of three algorithms:

$(sk, pk) \leftarrow \texttt{Keygen}(1^\lambda)$**:** An algorithm that takes a security parameter as input and creates a new private/public key pair (sk, pk)

$\sigma \leftarrow \texttt{Sign}(sk, m)$**:** An algorithm that takes a message m and a private key sk as arguments and returns a signature σ

[1] It is called OMcoGDH by Jarecki et al. [13] but we note that the two assumptions are essentially the same on the type of pairing we are working with.

```
$\mathtt{OMcoCDH}^{\varphi}_{\mathcal{A}}(1^\lambda)$
$x \leftarrow\!\!\$\ \mathbb{Z}_q$
$X := x \cdot G_2$
$DH := \mathcal{A}^{\mathcal{O}_\mathrm{T}, \mathcal{O}_\mathrm{DH}, \varphi}(X)$
return $|DH| > q_\mathrm{DH} \land \forall\ (Y, Z) \in DH :\ Y \in T \land x \cdot Y = Z$

$\mathcal{O}_\mathrm{T}()$
$Y \leftarrow\!\!\$\ \mathcal{G}_1$
$T := T \cup \{Y\}$
return $Y$

$\mathcal{O}_\mathrm{DH}(Y)$
$q_\mathrm{DH} := q_\mathrm{DH} + 1$
return $x \cdot Y$
```

Figure 2.1.: Game $\mathtt{OMcoCDH}^{\varphi}_{\mathcal{A}}(1^\lambda)$ used to define OMcoCDH. Note that the isomorphism oracle $\varphi : \mathcal{G}_2 \to \mathcal{G}_1$ given to $\mathtt{OMcoCDH}^{\varphi}_{\mathcal{A}}(1^\lambda)$ is forwarded to $\mathcal{A}$.

```
$\mathtt{EUF\text{-}CMA}^{\mathcal{S}}_{\mathcal{A}}(1^\lambda)$
$(sk, pk) := \mathcal{S}.\mathtt{Keygen}(1^\lambda)$
$(m, \sigma) := \mathcal{A}^{\mathcal{O}_\mathrm{sig}}(pk)$
return $m \notin M_{\mathcal{O}_\mathrm{sig}} \land \mathcal{S}.\mathtt{Verify}(pk, m, \sigma) = 1$

$\mathcal{O}_\mathrm{sig}(m)$
$M_{\mathcal{O}_\mathrm{sig}} := M_{\mathcal{O}_\mathrm{sig}} \cup \{m\}$
return $\mathcal{S}.\mathtt{Sign}(sk, m)$
```

Figure 2.2.: Game $\mathtt{EUF\text{-}CMA}^{\mathcal{S}}_{\mathcal{A}}(1^\lambda)$ used to define EUF-CMA.

$\{0,1\} \leftarrow \mathtt{Verify}(pk, m, \sigma)$: A deterministic algorithm to verify that a signature σ is valid for a given message m and public key pk

We consider $\mathcal{S}$ a secure signature scheme if it is both correct and unforgeable.

Definition 5 (Correctness)**.** *A signature scheme $\mathcal{S}$ is called correct if the following holds true:*

$$\forall\ m \in \{0,1\}^*,\ (sk, pk) \in [\mathcal{S}.\mathtt{Keygen}(1^\lambda)] :\ \mathcal{S}.\mathtt{Verify}(pk, m, \mathcal{S}.\mathtt{Sign}(sk, m)) = 1$$

To formally model unforgeability we use the security notion of existential unforgeability under chosen message attack (EUF-CMA). This means that an attacker can request signatures for arbitrary messages but shall not be able to produce a new message-signature pair.

Definition 6 (EUF-CMA)**.** *A signature scheme $\mathcal{S}$ is called existentially unforgeable under chosen message attack (EUF-CMA) if there exists no polynomial time adversary $\mathcal{A}$ that can win game $\mathtt{EUF\text{-}CMA}^{\mathcal{S}}_{\mathcal{A}}(1^\lambda)$ (fig. 2.2) with more than negligible probability.*

2.4. BLS Signature Scheme

Boneh et al. [12] describe a signature scheme $\mathcal{BLS} = (\mathtt{Keygen}, \mathtt{Sign}, \mathtt{Verify})$ (fig. 2.3) based on the coCDH problem on pairing groups.

```
$\mathcal{BLS}$.Keygen($1^\lambda$)
    $x \leftarrow\!\$\, \mathbb{Z}_q$
    return $(x, x \cdot G_2)$
```

```
$\mathcal{BLS}$.Sign($x, m$)
    $\sigma := x \cdot H_1(m)$
    return $\sigma$
```

```
$\mathcal{BLS}$.Verify($P, m, \sigma$)
    // Verify that $(P, m, \sigma)$ is a valid DH tuple
    return $\sigma \in \mathcal{G}_1 \wedge e(H_1(m), P) = e(\sigma, G_2)$
```

Figure 2.3.: The $\mathcal{BLS}$ signature scheme algorithms.

A signature σ is a group element of $\mathcal{G}_1$ that forms a valid coDH-tuple with the public key $P = x \cdot G_2$ and the message hashed to group $\mathcal{G}_1$ using $H_1(\cdot)$. Clearly such a tuple is easy to construct with knowledge of the secret x, but hard otherwise if coCDH problem is hard for the group pair $(\mathcal{G}_1, \mathcal{G}_2)$. The concrete algorithms of $\mathcal{BLS}$ are defined as follows:

$(x, P) \leftarrow$ `Keygen`(1^λ)**:** A private key $x \leftarrow\!\$\, \mathbb{Z}_q$ is chosen at random. The public key $P = x \cdot G_2$ can be derived by multiplying it with the generator G_2.

$\sigma \leftarrow$ `Sign`(x, m)**:** Signing consists of hashing the message to the first curve of the pairing and multiplying the result with the private key to create the signature $\sigma = x \cdot H_1(m) \in \mathcal{G}_1$.

$\{0, 1\} \leftarrow$ `Verify`(P, m, σ)**:** Verifying is done by checking that $\sigma \in \mathcal{G}_1$ and that the public key P, the hash of the message $H_1(m)$ and the signature σ form a valid coDH tuple.

Note that we assume the public key to be valid, i.e. $P \in \mathcal{G}_2$, which we expect to be ensured externally (e.g. on key exchange).

The scheme is proven existentially unforgeable under chosen message attack (EUF-CMA) by Boneh et al. [12] under the coCDH assumption.

Shacham [14] also proves uniqueness of $\mathcal{BLS}$ signatures.

Lemma 1 ($\mathcal{BLS}$ uniqueness [14])**.** *For every message m and public key P there exists exactly one valid $\mathcal{BLS}$ signature σ.*

2.5. Secret Sharing

We call $(x_1, \ldots, x_n) \xrightarrow{t} x$ a (t, n)-secret sharing of x if there is an efficient algorithm `Interpolate` to calculate x from any $S \subseteq \{x_1, \ldots, x_n\}$ with $|S| = t$ but doing so with any S with $t - 1$ or less elements is intractable.

One well-known such scheme is Shamir secret sharing [15]. It is defined through a random polynomial $f(x) = a_0 + a_1 \cdot x + \ldots a_{t-1} \cdot x^{t-1}$ over a finite field where the shared secret is the first coefficient $a_0 = f(0)$. The shares are defined as the evaluation of f such that $x_i = f(i)$. This allows to recover f from any t shares by interpolating the polynomial using Lagrange's method. In fig. 2.4 we describe the interpolation of a single value at place $\hat{x}$, reconstructed from t points on the polynomial using Lagrange's method.

The shared secret x can be recovered by interpolating at place $\hat{x} = 0$. By interpolating the value at any other $\hat{x}$ the result is the share $\hat{x}$ of the secret.

```
Interpolate({(x_1, y_1), ..., (x_t, y_t)}, x̂)
    return $\sum_{i \in \{1,\dots,t\}} \gamma_i(\hat{x}, x_1, \dots, x_t) \cdot y_i$

$\gamma_i(\hat{x}, x_1, \dots, x_t)$
    return $\prod_{\substack{k \in \{1,\dots,t\} \\ k \neq i}} \frac{\hat{x} - x_k}{x_i - x_k}$
```

Figure 2.4.: Shamir secret sharing interpolation algorithm that interpolates place x of the polynomial defined by t shares using Lagrange's method. γ_i is the Lagrange coefficient.

Note that the shared secret x itself can be used as a point on the polynomial as $(0, x)$ when interpolating.

2.6. Threshold Signature Schemes

A (t, n)-threshold signature scheme[2] $\mathcal{TS} = (\texttt{Keygen}, \texttt{Sign}, \texttt{ShareVer}, \texttt{Combine}, \texttt{Verify})$ allows the generation of signatures by cooperation of any t out of n signers.

$(sk_1, \dots, sk_n, pk_1, \dots, pk_n, pk) \leftarrow \texttt{Keygen}(1^\lambda, t, n)$**:** An algorithm run by a trusted dealer that takes a security parameter 1^λ, a threshold t and a total number of signers n as arguments and generates the signers' secret key shares, public key shares and the aggregate public key. We also assume n and t to be implicitly available to all other algorithms.

$\sigma_i \leftarrow \texttt{Sign}(sk_i, m)$**:** An algorithm run by every signer i that takes the private key share sk_i and message m as arguments and outputs a signature share σ_i.

$\{0, 1\} \leftarrow \texttt{ShareVer}(pk_i, m, \sigma_i)$**:** An algorithm to verify that a given signature share is valid for message m under the corresponding public key share pk_i.

$\sigma \leftarrow \texttt{Combine}(\Sigma)$**:** An algorithm that takes a set $\Sigma \subseteq \{\sigma_i \mid i \in \{1, \dots, n\}\}$ with $|\Sigma'| \geq t$ of valid signature shares and outputs a signature σ.

$\{0, 1\} \leftarrow \texttt{Verify}(pk, m, \sigma)$**:** An algorithm to verify that a signature σ is valid for a given message m and public key pk.

In an implementation the trusted dealer may be replaced by a distributed key generation protocol (DKG), one such protocol is described in appendix C.

In our description of the scheme we use the concept of signing sessions. Each session signs exactly one message and every signer will receive the same message as input in one session. That means that $\mathcal{TS}.\texttt{Sign}$ has to be called with the same m by all honest signers in one session.

This has to be guaranteed externally, e.g. by a reliable broadcast or other byzantine fault tolerant (BFT) [16] consensus protocol as shown in section 2.8. The notion of

[2]We only consider schemes which create signature shares that can later be combined by any party. There may exist others that cannot be modeled this way but are not relevant to this work.

sessions is very useful for e.g. e-cash applications where, to start such a session, money has to be sent. If a proof of having sent money could be used to acquire signature shares for different messages that would lead to an attack described in appendix A. We note that a threshold e-cash scheme would be most likely built with a BFT consensus protocol anyway, which would allow agreeing on one message per session at little additional cost.

In the context of a threshold signature scheme we need to re-define unforgeability. Instead of forging a signature at all being impossible for an adversary, this now has to hold for an adversary that controls up to $t-1$ signers. We define (t,n)-EUF-CMA to describe this notion of threshold unforgeability.

We use game $(t,n)\text{-}\mathtt{EUF\text{-}CMA}_{\mathcal{A}}^{\mathcal{TS}}(1^\lambda, t, n)$ to define (t,n)-EUF-CMA, it proceeds as follows:

1. The challenger generates the private key shares $sk_1, \dots, sk_n$, the corresponding public key shares $pk_1, \dots, pk_n$ and the public key pk using $\mathcal{TS}.\mathtt{Keygen}$.
2. $\mathcal{A}$ is given the first $t-1$ secret keys[3] $sk_1, \dots, sk_{t-1}$, all public key shares $pk_1, \dots, pk_n$ and the aggregate public key pk. $\mathcal{A}$ also has access to a signature oracle $\mathcal{O}_{\text{sig}}$ which generates the signature shares of all honest parties for a particular message and saves all signed messages in set M_{sig}.

   $\mathcal{A}$ runs up to q_s signing sessions through $\mathcal{O}_{\text{sig}}$ which records all signed messages in M_{sig}. The oracle takes a message m and returns the signature shares of all honest parties.
3. $\mathcal{A}$ wins if it can present a valid signature for a message not in M_{sig}.

The oracle $\mathcal{O}_{\text{sig}}$ takes one message and returns signature shares for all honest parties. This is a strong requirement but it is justifies by our intended application in federated (e-cash) settings where there most likely already is a consensus algorithm which can be used to ensure that. If the requirement of all signers signing the same message during one session was not met an attack described in appendix A would be possible A protocol ensuring this requirement is described in section 2.8.

It also appears as if the oracle returning all honest signers' signature shares implies the requirement of all signers being online in a real implementation for unforgeability to hold. This is not the case. Giving $\mathcal{A}$ all signature share merely makes it stronger. It could always simulate less signers being online by dropping signature shares. Thus this game is without loss of generality equivalent to one where the adversary can choose from which honest party to acquire the signature shares.

Definition 7 ((t,n)-EUF-CMA)**.** *We call a threshold signature scheme $\mathcal{TS}$ (t,n)-existentially unforgeable under chosen message attack ((t,n)-EUF-CMA) if there exists no polynomial time adversary $\mathcal{A}$ that can win game $(t,n)\text{-}\mathtt{EUF\text{-}CMA}_{\mathcal{A}}^{\mathcal{TS}}(1^\lambda)$ (fig. 2.5) with more than negligible probability.*

[3]We note that for the schemes we will consider this is without loss of generality and equivalent to $\mathcal{A}$ being allowed to choose up to $t-1$ signers to compromise.

```
$(t,n)$-EUF-CMA$_{\mathcal{A}}^{\mathcal{TS}}(1^\lambda)$
$(sk_1,\dots,sk_n,pk_1,\dots,pk_n,pk) := \mathcal{TS}.\mathtt{Keygen}(1^\lambda,t,n)$
$(m,\sigma) := \mathcal{A}^{\mathcal{O}_{\mathrm{sig}}}(pk,pk_1,\dots,pk_n,sk_1,\dots,sk_{t-1})$
return $m \notin M_{\mathrm{sig}} \wedge \mathcal{TS}.\mathtt{Verify}(P,m,\sigma) = 1$

$\mathcal{O}_{\mathrm{sig}}(m)$
$M_{\mathrm{sig}} := M_{\mathrm{sig}} \cup \{m\}$
return $\{\mathcal{TS}.\mathtt{Sign}(x_i,m) \mid i \in \{t,\dots,n\}\}$
```

Figure 2.5.: Game (t,n)-$\mathtt{EUF\text{-}CMA}_{\mathcal{A}}^{\mathcal{TS}}(1^\lambda)$ used to define (t,n)-EUF-CMA.

The security of $\mathcal{TS}$ is not only dependent on its unforgeability, but also its robustness. Robustness describes the ability of a scheme to function in the presence of malicious parties. Since multiple, potentially malicious, parties need to cooperate to create a threshold signature it is an essential property for such a scheme. We use the notion of (t,n)-robustness to describe robustness in the presence of t malicious parties out of n.

We use game (t,n)-$\mathtt{robustness}_{\mathcal{TS}}^{\mathcal{A}}(pk,pk_1,\dots,pk_n,sk_1,\dots,sk_n)$ (fig. 2.6) to define (t,n)-robustness for $\mathcal{TS}$, it proceeds as follows:

1. $\mathcal{A}$ is given the secret key shares $sk_1,\dots,sk_{t-1}$, all public key shares $pk_1,\dots,pk_n$ and the aggregate public key pk. It also has access to a signing oracle $\mathcal{O}_{\mathrm{sig}}$.

   $\mathcal{A}$ chooses a message $m \in \{0,1\}^*$ and produces up to $t-1$ signature shares $\Sigma_{\mathcal{A}} \subseteq \{\sigma_i \mid i \in \{1,\dots,t-1\}\}$. Both m and $\Sigma_{\mathcal{A}}$ are returned.

2. The honest signers' signature shares $\Sigma_C = \{\mathtt{Sign}(sk_i,m) \mid i \in \{t,\dots,n\}\}$ are computed by the challenger. $\mathcal{A}$'s signature shares are filtered to receive $\Sigma_{\mathcal{A}}^{\mathrm{val}} = \{\sigma_i \mid \sigma_i \in \Sigma_{\mathcal{A}} : \mathtt{ShareVer}(pk,m,\sigma_i)\}$. These are then combined to one signature $\sigma = \mathtt{Combine}(\Sigma_{\mathcal{A}}^{\mathrm{val}} \cup \Sigma_C)$.

3. The game returns $\mathtt{Verify}(pk,m,\sigma) = 0$. $\mathcal{A}$ wins if the verification fails and the game returns 1.

Definition 8 ((t,n)-robustness $\mathcal{TS}$)**.** *We call a threshold signature scheme $\mathcal{TS}$ (t,n)-robust if $P[(t,n)$-$\mathtt{robustness}_{\mathcal{TS}}^{\mathcal{A}}(pk,pk_1,\dots,pk_n,sk_1,\dots,sk_n) = 1]$ is negligible for all polynomial time adversaries $\mathcal{A}$ and all possible key sets $(sk_1,\dots,sk_n,pk_1,\dots,pk_n,pk) \in [\mathcal{TS}.\mathtt{Keygen}(1^\lambda,t,n)]$.*

We note that unforgeability and robustness are in conflict with each other. While (t,n)-EUF-CMA is (informally speaking) more secure for larger t because it can tolerate more adversarial peers, robustness can only be achieved up to a certain point, namely as long as $t < n/2$. For larger t there would simply not be enough honest signers to combine the shares without breaking (t,n)-EUF-CMA, which states that at least t shares are needed.

```
$(t,n)\text{-}\mathtt{robustness}^{\mathcal{A}}_{\mathcal{TS}}(pk, pk_1, \dots, pk_n, sk_1, \dots, sk_n)$
$(m, \Sigma_{\mathcal{A}}) := \mathcal{A}^{\mathcal{O}_{\text{sig}}}(pk, pk_1, \dots, pk_n, sk_1, \dots, sk_t)$
**if** $|\Sigma_{\mathcal{A}}| > t - 1 \lor \exists\, \sigma_i \in \Sigma_{\mathcal{A}} : i \in \{t, \dots, n\}$ **then**
  **return** $0$
**endif**
$\Sigma_C := \{\mathcal{TS}.\mathtt{Sign}(sk_i, m) \mid i \in \{t, \dots, n\}\}$
$\Sigma^{\text{ver}}_{\mathcal{A}} := \{\sigma_i \mid \sigma_i \in \Sigma_{\mathcal{A}} : \mathtt{ShareVer}(pk, m, \sigma_i)\}$
$\sigma := \mathcal{TS}.\mathtt{Combine}(\Sigma^{\text{ver}}_{\mathcal{A}} \cup \Sigma_C)$
**return** $\mathcal{TS}.\mathtt{Verify}(pk, m, \sigma) = 0$
```

Figure 2.6.: Game $(t,n)\text{-}\mathtt{robustness}^{\mathcal{A}}_{\mathcal{TS}}$ used to define (t,n)-robustness for $\mathcal{TS}$ schemes. Oracle $\mathcal{O}_{\text{sig}}$ is defined as in fig. 2.5.

2.7. Threshold Blind Signature Schemes

A threshold blind signature scheme $\mathcal{TBS} = (\mathtt{Keygen}, \mathtt{Blind}, \mathtt{Sign}, \mathtt{ShareVer}, \mathtt{Combine}, \mathtt{Unblind}, \mathtt{Verify})$ allows a user $\mathcal{U}$ to acquire a blind signature for a message $m \in \{0,1\}^*$ from a group of n signers as long as t of these are honest.

$(sk_1, \dots, sk_n, pk_1, \dots, pk_n, pk) \leftarrow \mathtt{Keygen}(1^\lambda, t, n)$: An algorithm that generates the secret key shares for all signers, the corresponding public key shares $pk_1, \dots, pk_n$ and the aggregate public key pk. We assume n and t to be implicitly available to all other algorithms.

$(m', \beta) \leftarrow \mathtt{Blind}(m)$: An algorithm run by the user $\mathcal{U}$ that takes a message m as an argument and produces a corresponding blinded message m' and a blinding key β. The blinded message m' is handed to the signers while β is kept for later unblinding of the signature.

$\sigma'_i \leftarrow \mathtt{Sign}(sk_i, m')$: An algorithm run by each signer i to create a blind signature share σ'_i for blinded message m' using the private key share sk_i.

$\{0,1\} \leftarrow \mathtt{ShareVer}(pk_i, m', \sigma'_i)$: An algorithm to verify that a given blind signature share σ'_i is valid for the blinded message m' under the respective public key share pk_i.

$\sigma' \leftarrow \mathtt{Combine}(\Sigma')$: An algorithm that takes a set of valid blind signature shares $\Sigma' \subseteq \{\sigma'_i \mid i \in \{1, \dots, n\}\}$ with $|\Sigma'| \geq t$ and outputs a combined blind signature σ'.

$\sigma \leftarrow \mathtt{Unblind}(\sigma', \beta, pk)$: An algorithm run by $\mathcal{U}$ that takes a blinded signature σ', a blinding key β and a public key pk and unblinds the blind signature σ' to obtain σ.

$\{0,1\} \leftarrow \mathtt{Verify}(pk, m, \sigma)$: An algorithm to verify that a signature σ is valid for a given message m and public key pk.

There are three security goals for a threshold blind signature scheme: unforgeability, robustness and blindness. We adapt the unforgeability and blindness definition from Kuchta et al. [17].

```
$(t,n)$-$\texttt{OMF-CMA}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$
$(sk_1, \dots, sk_n, pk_1, \dots, pk_n, pk) := \mathcal{TBS}.\texttt{Keygen}_i(1^\lambda, n, t)$
$\{(m_1, \sigma_1), \dots, (m_s, \sigma_s)\} := \mathcal{A}^{\mathcal{O}_{\text{bsig}}}(1^\lambda, pk, pk_1, \dots, pk_n, sk_1, \dots, sk_{t-1})$
return $s = q_s + 1 \ \wedge \ \forall i, j$ where $i \neq j : (m_i, \sigma_i) \neq (m_j, \sigma_j)$
  $\wedge \ \forall \ i \in \{1, \dots, s\} : \ \mathcal{TBS}.\texttt{Verify}(pk, m_i, \sigma_i) = 1$

        $\mathcal{O}_{\text{bsig}}(m')$
        $q_s := q_s + 1$
        return $\{\mathcal{TBS}.\texttt{Sign}(sk_i, m') \mid i \in \{t, \dots, n\}\}$
```

Figure 2.7.: Game (t,n)-$\texttt{OMF-CMA}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$ used to define (t,n)-OMF-CMA.

To define unforgeability in the blind threshold setting, we cannot use (t,n)-EUF-CMA anymore since the messages that are being signed are unknown to the signers, so that there cannot be a notion of the honest parties having signed a certain message. Instead we use the threshold version of one more forgery under chosen message attack ((t,n)-OMF-CMA). This means that an attacker controlling up to $t-1$ out of n signers cannot produce more signatures than they started signing sessions for.

We use game (t,n)-$\texttt{OMF-CMA}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$ (fig. 2.7) to define (t,n)-OMF-CMA, which proceeds as follows:

1. The keys $(sk_1, \dots, sk_n, pk_1, \dots, pk_n, pk) = \mathcal{TBS}.\texttt{Keygen}(1^\lambda, t, n)$ are generated by the challenger.
2. $\mathcal{A}$ is given the first $t-1$ secret key shares $sk_1, \dots, sk_{t-1}$, all public key shares $pk_1, \dots, pk_n$ and the aggregate public key pk.

   It also has access to the blind signature oracle $\mathcal{O}_{\text{bsig}}$, which generates all blind signature shares σ'_i from all honest signers $i \in \{t, \dots, n\}$ for *one* blind message m'. The oracle invocations are counted as q_s. See section 2.6 for why this oracle is appropriate in our setting.
3. $\mathcal{A}$ returns s distinct message-signature pairs and wins if all of them are valid for public key pk and $s > q_s$.

Definition 9 ((t,n)-OMF-CMA)**.** *A threshold signature scheme $\mathcal{TBS}$ is called (t,n)-one more forgery under chosen message attack ((t,n)-OMF-CMA) secure if there exists no polynomial time adversary $\mathcal{A}$ that can win (t,n)-$\texttt{OMF-CMA}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$ (fig. 2.7) with more than negligible probability.*

Robustness is defined similarly to $\mathcal{TS}$, but needs to take into account blinding and unblinding. The game (t,n)-$\texttt{robustness}_{\mathcal{TBS}}^{\mathcal{A}}(pk, pk_1, \dots, pk_n, sk_1, \dots, sk_n)$ (fig. 2.8) used to define $(t,n)-$robustness works as follows:

```
$(t,n)$-robustness$^{\mathcal{A}}_{\mathcal{TBS}}(pk, pk_1, \ldots, pk_n, sk_1, \ldots, sk_n)$
$m := \mathcal{A}^{\mathcal{O}_{\text{bsig}}}_{\text{msg}}(pk, pk_1, \ldots, pk_n, sk_1, \ldots, sk_{t-1})$
$(m', \beta) := \mathcal{TBS}.\texttt{Blind}(m)$
$\Sigma'_{\mathcal{A}} := \mathcal{A}^{\mathcal{O}_{\text{bsig}}}_{\text{sig}}(m')$
if $|\Sigma'_{\mathcal{A}}| > t-1 \vee \exists\ \sigma_i \in \Sigma_{\mathcal{A}} : i \in \{t, \ldots, n\}$ then
  return $0$
endif
$\Sigma'_C := \{\mathcal{TBS}.\texttt{Sign}(sk_i, m') \mid i \in \{t, \ldots, n\}\}$
$\Sigma^{\text{val}}_{\mathcal{A}} := \{\sigma'_i \mid \sigma'_i \in \Sigma'_{\mathcal{A}} : \ \mathcal{TBS}.\texttt{ShareVer}(pk_i, m', \sigma'_i)\}$
$\sigma' := \mathcal{TBS}.\texttt{Combine}(\Sigma^{\text{val}}_{\mathcal{A}} \cup \Sigma'_C)$
$\sigma := \mathcal{TBS}.\texttt{Unblind}(\sigma', \beta, pk)$
return $\mathcal{TBS}.\texttt{Verify}(pk, m, \sigma) = 0$
```

Figure 2.8.: Game (t,n)-robustness$^{\mathcal{A}}_{\mathcal{TBS}}$ used to define (t,n)-robustness for $\mathcal{TBS}$ schemes. Adversary $\mathcal{A} = (\mathcal{A}_{\text{msg}}, \mathcal{A}_{\text{sig}})$ consists of two sub-algorithms: $\mathcal{A}_{\text{msg}}$ which chooses a message to sign and $\mathcal{A}_{\text{sig}}$ which may contribute up to t blind signature shares.

1. The adversary $\mathcal{A}$ is given the public key pk, the public key shares $pk_1, \ldots, pk_n$ and the private key shares $sk_1, \ldots, sk_{t-1}$ and outputs a message m to be signed. $\mathcal{A}$ also has access to a blind signing oracle $\mathcal{O}_{\text{bsig}}$ as defined in fig. 2.7.
2. The challenger blinds m to m'.
3. $\mathcal{A}$ is then given m' and produces up to $t-1$ blind signature shares $\Sigma'_{\mathcal{A}} \subseteq \{\sigma'_i \mid i \in \{1, \ldots, t-1\}\}$.
4. The challenger calculates the blind signature shares of the honest signers $\Sigma'_C = \{\texttt{Sign}(sk_i, m') \mid i \in \{t, \ldots, n\}\}$ and filters $\mathcal{A}$'s shares into the set of valid shares $\Sigma^{\text{val}}_{\mathcal{A}} := \{\sigma'_i \mid \sigma'_i \in \Sigma'_{\mathcal{A}} : \ \texttt{ShareVer}(pk_i, m', \sigma'_i)\}$. These are then combined to one blind signature $\sigma' = \texttt{Combine}(\Sigma^{\text{val}}_{\mathcal{A}} \cup \Sigma'_C)$ which is then unblinded to receive σ.
5. The game returns whether $\texttt{Verify}(pk, m, \sigma) = 0$. $\mathcal{A}$ wins if the verification fails and returns 0.

Definition 10 ((t,n)-robustness $\mathcal{TBS}$)**.** *We call a scheme $\mathcal{TBS}$ (t,n)-robust if the probability $P[(t,n)$-robustness$^{\mathcal{A}}_{\mathcal{TBS}}(pk, pk_1, \ldots, pk_n, sk_1, \ldots, sk_n) = 1]$ is negligible for all polynomial time adversaries $\mathcal{A}$ and all key sets $(sk_1, \ldots, sk_n, pk_1, \ldots, pk_n, pk) \in [\mathcal{TBS}.\texttt{Keygen}(1^\lambda, t, n)]$.*

A threshold blind signature scheme is information-theoretically blind if an attacker, no matter how computationally strong, cannot correlate any signature-message pair with the signing session through which it was crated.

We use game $\texttt{Blindness}^{\mathcal{TBS}}_{\mathcal{A}}(1^\lambda)$ (fig. 2.9) to define blindness, informally it proceeds as follows:

1. Adversary $\mathcal{A}$ chooses two messages $m_0, m_1 \in \{0,1\}^*$ and an aggregate public key *pk* with which it will blind sign later.
2. The challenger, taking the position of a user requesting a signature, blinds both messages to receive m'_0 and m'_1. It then draws a uniformly random bit $b \leftarrow\!\!\$ \{0,1\}$, which will determine the order in which the blind signing will happen.
3. The adversary is given both blinded messages in the random order defined by b and returns the two corresponding blind signatures $(\sigma'_b, \sigma'_{1-b}) = \mathcal{A}_{\text{bsig}}(m'_b, m'_{1-b})$. After unblinding both received signatures the challenger has to verify these. $\mathcal{A}$ can only win if both signatures were correct.
4. $\mathcal{A}$ is now given both unblinded signatures in a known order and is tasked with guessing in which order these were signed by itself. $\mathcal{A}$ wins if it guesses correctly.

Note how $\mathcal{A}$ could trivially win if the challenger would not verify the signatures. $\mathcal{A}$ could generate a valid signature in the first signing round and an invalid one in the second. Being handed both signatures in order would allow to distinguish both signing rounds.

Definition 11 (Perfect Blindness)**.** *We call a threshold blind signature scheme perfectly blind if there exists no adversary $\mathcal{A}$ (not even a computationally unlimited one) that can win the game* $\texttt{Blindness}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$ *(fig. 2.9) with a probability higher than* $1/2$.

2.8. Protocol Instantiation

So far we described the signature schemes and security games in terms of algorithms. To actually implement such a scheme it is important to consider the communication of algorithm in- and outputs between the participants. In this section we describe an exemplary protocol.

We assume that all signers $1, \dots, n$ are connected by reliable and authenticated point-to-point connections. The communication via these is written as

`send`(r, m): reliably send message m to recipient r.

`receive`$()$: receive a messages from a peer sent during the last round.

Furthermore we assume a reliable broadcast protocol as defined by Bracha [18]. It guarantees the delivery of the same message to all honest recipients. We furthermore assume the messages to be authenticated. The reliable broadcast can be used to prevent equivocation as required by the discussed schemes. The unforgeability games implicitly make the assumption that every honest signer sees the same message due to the signing oracle's design as explained in section 2.6.

`broadcast`(m): reliably broadcast the same message m to participants $1, \dots, n$.

`receive_bc`$()$: receive a broadcast message from a peer sent during last round.

```
$\texttt{Blindness}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$
$(pk, m_0, m_1) := \mathcal{A}_{\text{msg}}(1^\lambda)$
$(m'_0, \beta_0) = \mathcal{TBS}.\texttt{Blind}(m_0)$
$(m'_1, \beta_1) = \mathcal{TBS}.\texttt{Blind}(m_1)$
// Run blind signing protocol in random order
$b \leftarrow\$ \{0,1\}$
$(\sigma'_b, \sigma'_{1-b}) := \mathcal{A}_{\text{bsig}}(m'_b, m'_{1-b})$
$\sigma_0 := \mathcal{TBS}.\texttt{Unblind}(\sigma'_0, \beta_0, pk)$
$\sigma_1 := \mathcal{TBS}.\texttt{Unblind}(\sigma'_1, \beta_1, pk)$
$v_0 := \mathcal{TBS}.\texttt{Verify}(pk, m_0, \sigma_0)$
$v_1 := \mathcal{TBS}.\texttt{Verify}(pk, m_1, \sigma_1)$
// Guess order in which the messages were signed
return $v_0 \land v_1 \land (b = \mathcal{A}_{\text{guess}}(\sigma_0, \sigma_1))$
```

Figure 2.9.: Game $\texttt{Blindness}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$ used to define blindness. Adversary $\mathcal{A} = (\mathcal{A}_{\text{msg}}, \mathcal{A}_{\text{bsig}}, \mathcal{A}_{\text{guess}})$ consists of three sub-algorithms: $\mathcal{A}_{\text{msg}}$ which generates the adversary's keys, returning public key pk and chooses two messages $m_0, m_1 \in \{0,1\}^*$ that will later be blind signed, $\mathcal{A}_{\text{bsig}}$ which is given two blinded messages and produces two blind signatures for the messages (in the same order) and $\mathcal{A}_{\text{guess}}$ which has to guess in which order the messages were signed after participating in signing. Note how $\mathcal{A}$ controls all signers and the combiner and is even allowed to generate all keys.

User $\mathcal{U}$ | **Signer $i \in 1, \ldots, n$**

```
User U:
$(m', \beta) := \mathcal{TBS}.\texttt{Blind}(m)$
```

$$\xrightarrow[\text{(reliable broadcast)}]{m'}$$

```
Signer i:
$\sigma_i' := \mathcal{TBS}.\texttt{Sign}(x_i, m')$
$\texttt{broadcast}(\sigma_i)$
    $\vdots$
$\mathit{Faults} := \{\}$
$\Sigma' := \{\}$
while true do
  $\sigma_i' := \texttt{receive\_bc}()$
  if $\texttt{ShareVer}(P_i, m', \sigma_i')$ then
    $\Sigma' := \Sigma' \cup \{\sigma_i'\}$
  else
    $\mathit{Faults} := \mathit{Faults} \cup \{i\}$
  endif
  if $|\Sigma'| = t$ then
    break
  endif
done
$\sigma' := \mathcal{TBS}.\texttt{Combine}(\Sigma', P)$

return $\mathit{Faults}$
```

$$\xleftarrow[\text{(from any up to t signers)}]{\sigma'}$$

```
User U:
// Try to find a honest signer/combiner
while true do
  $\sigma' := \texttt{receive}()$
  $\sigma := \mathcal{TBS}.\texttt{Unblind}(\sigma', \beta, P)$
  if $\mathcal{TBS}.\texttt{Verify}(P, m, \sigma)$ then
    return $\sigma$
  endif
done
```

Figure 2.10.: An example instantiation of our threshold blind signature protocol where user $\mathcal{U}$ acquires a blind signature on message m. It assumes that $\mathcal{TBS}$-$\mathcal{BLS}$.`Keygen` was run previously and the key material was distributed to the participants. The broadcasting of m' to all signers by $\mathcal{U}$ could be emulated by $\mathcal{U}$ sending m' to t signers and them broadcasting m' internally, which is more practical if there already exists a BFT consensus between signers, but it does not include $\mathcal{U}$.

In fig. 2.10 we describe an exemplary protocol instantiation of a $\mathcal{TBS}$ scheme using the reliable broadcast primitive introduced above. A user $\mathcal{U}$ acquires a blind signature from the signers $1, \dots, n$. The protocol for a threshold signature scheme $\mathcal{TS}$ would work equivalently, just without the blinding/unblinding.

We choose this particular variant where the signers aggregate the signature shares since it allows detection of uncooperative signers by the honest signers, something that would otherwise be hard to prove or disprove in case of user complaints. Each signer receives a set *Faults* at the end of a signing session containing any signers that submitted faulty shares. This allows to detect misbehavior and also reduces the number of signers $\mathcal{U}$ has to communicate with. In the worst case $\mathcal{U}$ needs to query t signers to receive the valid signature for the requested message.

Note that `ShareVer` and `Combine` could also be executed by $\mathcal{U}$, which would be more efficient and preferable in some settings.

For the robustness property of the schemes to be useful in practice, the reliable broadcast protocol needs to terminate.

3. Boldyreva's $\mathcal{TS}$ Scheme

Boldyreva proposes a $\mathcal{BLS}$, and thus pairing based, threshold signature scheme [3] $\mathcal{TS}\text{-}\mathcal{BLS} = (\texttt{Keygen}, \texttt{Sign}, \texttt{ShareVer}, \texttt{Combine}, \texttt{Verify})$ (fig. 3.1). It uses Shamir secret sharing $(x_1, \ldots, x_n) \xrightarrow{t} x$ of the private key $x \in \mathbb{Z}_q$ to allow every participant i to create signature shares σ_i which can then be combined to a signature σ using Lagrange polynomial interpolation. While their paper uses `GJKR_DKG` for key generation we will use a trusted dealer to describe the protocol and present the alternative DKG in appendix C.

From a participant i's perspective Boldyreva's scheme works as follows:

$(x_1, \ldots, x_n, P_1, \ldots, P_n, P) \leftarrow \texttt{Keygen}(1^\lambda, t, n)$: A secret sharing $(x_1, \ldots, x_n) \xrightarrow{t} x$ is drawn uniformly at random by a trusted dealer such that $x \in \mathbb{Z}_q^*$. The public key shares are defined as $P_i = x_i \cdot G_2$ for $i \in \{1, \ldots, n\}$ and the aggregate public key as $P = x \cdot G_2$.

$\sigma_i \leftarrow \texttt{Sign}(x_i, m)$: Every participant i computes a signature share $\sigma_i = x_i \cdot H_1(m)$ and reliably sends it to the combiner.

We assume that the participants agree on *one* message to sign per round externally, for example through a reliable broadcast protocol. Equivocation would lead to a forgeability attack described in appendix A.

$\{0, 1\} \leftarrow \texttt{ShareVer}(P_i, m, \sigma_i)$: The combiner filters the received signature shares for validity. Every valid signature share is a valid $\mathcal{BLS}$ signature with the secret key share, and is thus verified against the corresponding public key share P_i as follows: $\mathcal{BLS}.\texttt{Verify}(P_i, m, \sigma_i)$.

```
$\mathcal{TS}\text{-}\mathcal{BLS}.\texttt{Keygen}(1^\lambda, t, n)$
$(x_1, \ldots, x_n \xrightarrow{t} x) \leftarrow\!\$\, \mathbb{Z}_q[i]$
$P := x \cdot G_2$
$P_i := x_i \cdot G_2 \mid i \in \{1, \ldots, n\}$
return $(x_1, \ldots, x_n, P_1, \ldots, P_n, P)$

$\mathcal{TS}\text{-}\mathcal{BLS}.\texttt{Sign}(x_i, m)$
$\sigma_i = x_i \cdot H_1(m)$
return $\sigma_i$

$\mathcal{TS}\text{-}\mathcal{BLS}.\texttt{ShareVer}(P_i, m, \sigma_i)$
return $\mathcal{BLS}.\texttt{Verify}(P_i, m, \sigma_i)$

$\mathcal{TS}\text{-}\mathcal{BLS}.\texttt{Combine}(\Sigma)$
$\sigma := \texttt{Interpolate}(\{(i, \sigma_i) \mid \sigma_i \in \texttt{take}(t, \Sigma)\}, 0)$
return $\sigma$

$\mathcal{TS}\text{-}\mathcal{BLS}.\texttt{Verify}(P, m, \sigma)$
return $\mathcal{BLS}.\texttt{Verify}(P, m, \sigma)$
```

Figure 3.1.: Algorithms defining scheme $\mathcal{TS}\text{-}\mathcal{BLS}$.

$\sigma \leftarrow$ `Combine`(Σ): The combiner receives a set Σ of least t valid signature shares and interpolates the aggregate signature $\sigma = \texttt{Interpolate}(\{(i, \sigma_i) \mid \sigma_i \in \Sigma_t\}, 0)$ from it using a subset $\Sigma_t \subseteq \Sigma$ of size t.

$\{0,1\} \leftarrow$ `Verify`(P, m, σ): The verification function is the same as for $\mathcal{BLS}$ signatures with the aggregate public key P used as the public key.

Note that we assume the public key and public key shares to be valid, i.e. all are elements of $\mathcal{G}_2$ and $P = \texttt{Interpolate}(\{(1, P_1), \ldots, (n, P_n)\}, 0)$, which we expect to be ensured externally (e.g. on key exchange). The group check for $\sigma, \sigma_i \in \mathcal{G}_1$ already happens in $\mathcal{BLS}$ and is not necessary here.

We claim that $\mathcal{TS}$-$\mathcal{BLS}$ is a secure threshold signature scheme by arguing that it is (t, n)-EUF-CMA secure and for $t < n/2$ also (t, n)-robust.

Unforgeability We prove that the scheme $\mathcal{TS}$-$\mathcal{BLS}$ is unforgeable as defined in definition 7, meaning that no polynomial time adversary controlling up to $t - 1$ signers can produce a valid signature with more than negligible probability.

Lemma 2. *Assuming $\mathcal{BLS}$ is EUF-CMA secure, then $\mathcal{TS}$-$\mathcal{BLS}$ is (t, n)-EUF-CMA secure.*

Proof of lemma 2. We prove the unforgeability of $\mathcal{TS}$-$\mathcal{BLS}$ by showing that if a polynomial time adversary $\mathcal{A}$ existed that can win $(t,n)\text{-}\texttt{EUF-CMA}_{\mathcal{A}}^{\mathcal{TS}\text{-}\mathcal{BLS}}(1^\lambda)$ with non-negligible probability, EUF-CMA of $\mathcal{BLS}$ signatures would be broken.

For that we construct an algorithm $\mathcal{B}$ which can invoke $\mathcal{A}$ and wins the game $\texttt{EUF-CMA}_{\mathcal{B}}^{\mathcal{BLS}}(1^\lambda)$ if $\mathcal{A}$ wins $(t,n)\text{-}\texttt{EUF-CMA}_{\mathcal{A}}^{\mathcal{TS}\text{-}\mathcal{BLS}}(1^\lambda)$. $\mathcal{B}$ has access to a signature oracle $\mathcal{O}_{\text{sig}}$ which will $\mathcal{BLS}$-sign any message with the private key corresponding to public key P, which is given to $\mathcal{B}$ as an input.

To win $\texttt{EUF-CMA}_{\mathcal{B}}^{\mathcal{BLS}}(1^\lambda)$ for a given public key P, $\mathcal{B}(1^\lambda, P)$ proceeds as follows:

1. Choose random private key shares $x_1, \ldots, x_{t-1} \leftarrow\$ \mathbb{Z}_q$ that will be given to $\mathcal{A}$ later.
2. Calculate the public key shares $P_1, \ldots, P_n$ to give to $\mathcal{A}$ such that they are consistent with the previously generated private key shares and P.

   Since the discrete logarithm of P is unknown to $\mathcal{B}$ so are the private key shares $x_t, \ldots, x_n$. Thus instead of simply multiplying these with G_2, $\mathcal{B}$ must interpolate $P_t, \ldots, P_n$ from the known shares $P_1, \ldots, P_{t-1}$ and P.

   For that the adversary's public key shares $P_i = x_i \cdot \mathcal{G}_2$ for $i \in \{1, \ldots, t-1\}$ can be easily calculated from the private key shares. To define the public key polynomial, t shares are needed though. We recount that the shared secret, i.e. in this case the aggregate public key P, is the evaluation of the secret sharing's polynomial at place $\hat{x} = 0$ (see section 2.5). Thus $\mathcal{B}$ uses the t points $S = \{(0, P'), (1, P_1), \ldots, (t-1, P_{t-1})\}$ that define the public key polynomial to interpolate the remaining public key shares $P_i = \texttt{Interpolate}(S, i)$ for $i \in \{t, \ldots, n\}$.

   The private key shares $x_1, \ldots, x_{t-1}$, the public key shares $P_1, \ldots, P_n$ and P are then given to $\mathcal{A}$.

3. For every signing request for message m by the adversary $\mathcal{A}$, $\mathcal{B}$ does the following:
   - Query the signature oracle from $\texttt{EUF-CMA}_{\mathcal{B}}^{\mathcal{BLS}}$ for a signature $\sigma := \mathcal{O}_{sig}(m)$.
   - Calculate $\mathcal{A}$'s $t-1$ signature shares $\Sigma_{\mathcal{A}} = \{x_i \cdot H_1(m) \mid i \in \{1, \ldots, t-1\}\}$.
   - Define the signature polynomial using the t points $S = \{(0, \sigma)\} \cup \{(i, \sigma_i) \mid \sigma_i \in \Sigma_{\mathcal{A}}\}$.
   - Interpolate the signature shares $\sigma_i = \texttt{Interpolate}(S, i)$ for $i \in \{t, \ldots, n\}$ and give them to $\mathcal{A}$.
4. If $\mathcal{A}$ outputs a new message-signature pair (m, σ) for which no signing session was started this means that $\mathcal{B}$ also made no request to $\mathcal{O}_{sig}$ for that message. Since the verification algorithm for threshold $\mathcal{BLS}$ signatures is the same as for $\mathcal{BLS}$ signatures we can thus return (m, σ) as an existential forgery.

Let q'_s be the number of signing queries made to $\mathcal{O}_{sig}$ by $\mathcal{B}$ and q_s the started signing sessions by $\mathcal{A}$ as stated previously. The run time of $\mathcal{B}$ is $t_{\mathcal{B}} = t_{\mathcal{A}} + \mathcal{O}(q_s)$. The number of signing queries q'_s to the BLS signing oracle is equal to the number of started signing sessions q_s. The probability of $\mathcal{B}$ succeeding is equal to the probability of $\mathcal{A}$ succeeding as every valid output of $\mathcal{A}$ leads to a valid output of $\mathcal{B}$ and there is no condition under which $\mathcal{B}$ has to abort.

Since $\mathcal{BLS}$ is assumed to be EUF-CMA secure, such an adversary $\mathcal{B}$ cannot exist. $\mathcal{TS}$-$\mathcal{BLS}$ has to be (t, n)-EUF-CMA secure. □

We note that the above proof is subtly different from the one presented by Boldyreva [3] as we found a minor mistake that we describe in appendix B.

Robustness We prove that the scheme $\mathcal{TS}$-$\mathcal{BLS}$ is robust as defined in definition 8, meaning that as long as less than t signers are malicious a valid signature will be produced.

Lemma 3. *Assuming $\mathcal{BLS}$ is a correct signature scheme, then $\mathcal{TS}$-$\mathcal{BLS}$ is (t, n)-robust for $t < n/2$.*

Proof of lemma 3. The adversary $\mathcal{A}$ has control over the message m and $t-1$ signature shares $\sigma_1, \ldots, \sigma_{t-1}$. We argue that neither can be used to make the signing protocol fail or produce an invalid signature.

The message $m \in \{0,1\}^*$ is being hashed, there is no input to the hash function that could make it fail. It can also only output valid elements in $\mathcal{G}_1$.

Signature shares are valid $\mathcal{BLS}$ signatures themselves with the respective key share. `ShareVer` is used to filter out invalid signature shares by verifying them using $\mathcal{BLS}$'s verification algorithm $\mathcal{BLS}.\texttt{Verify}(P_i, m, \sigma_i)$. Any valid signature share will lie on the same signature polynomial of degree $t-1$, so there is no way for $\mathcal{A}$ to influence the secret sharing using valid signature shares and `Combine` will succeed and output a valid signature as long as it is supplied with at least t valid signature shares. Lastly, even if $\mathcal{A}$ contributes no valid signature shares, σ can still be interpolated by `Combine` as long as $t < n/2$, which we assumed.

This shows that even with up to $t-1$ adversarial signers the protocol still produces valid signatures and is thus robust. $\square$

4. A $\mathcal{BLS}$ Threshold Blind Signature Scheme

Boldyreva [3] also proposes a blind signature scheme based on $\mathcal{BLS}$ and proves it OMF-CMA secure. It is a very elegant scheme whose signatures are indistinguishable from $\mathcal{BLS}$ signatures. We use this idea in combination with their threshold scheme to build a threshold blind signature scheme $\mathcal{TBS}\text{-}\mathcal{BLS} = (\texttt{Keygen}, \texttt{Blind}, \texttt{Sign}, \texttt{Combine}, \texttt{Blind}, \texttt{Verify})$ (fig. 4.1). We denote the requester of the blind signature as user $\mathcal{U}$ and the signers by their index i.

$(x_1, \dots, x_n, P_1, \dots, P_n, P) \leftarrow \mathcal{TBS}\text{-}\mathcal{BLS}.\texttt{Keygen}$: A trusted dealer draws a secret sharing $(x_1, \dots, x_n) \xrightarrow{t} x$ uniformly at random such that $x \in \mathbb{Z}_q^*$. The secret shares $x_1, \dots, x_n$ are the signers' private key shares. From these the public key shares $P_i = x_i \cdot G_2$ for $i \in \{1, \dots, n\}$ are calculated. The aggregate public key is $P' = x \cdot \mathbb{Z}_q$

For unblinding we also need $\hat{P} \in \mathcal{G}_1$ as the isomorphic element to P' such that $\hat{P} = \varphi(P')$. It is calculated as $\hat{P} = x \cdot G_1$.

The algorithm returns $x_1, \dots, x_n, P_1, \dots, P_n$ and $P = (P', \hat{P})$.

$(m', \beta) \leftarrow \texttt{Blind}(m)$: The user $\mathcal{U}$ draws a random blinding key $\beta \leftarrow\!\$\, \mathbb{Z}_q$ and calculates the blinded message $m' = H_1(m) + \beta \cdot G_1$.

$\sigma_i \leftarrow \texttt{Sign}(x_i, m')$: Every signer i generates a signature share $\sigma_i = x_i \cdot m'$. σ_i is then reliably sent to the combiner, which may be $\mathcal{U}$ or any subset of at least t signers.

The blinded message m' is an untrusted input. The signer needs to check that it is an element of $\mathcal{G}_1$ and returns $\perp$ otherwise.

$\{0, 1\} \leftarrow \texttt{ShareVer}(P_i, m', \sigma_i')$: The combiner filters the received signature shares for validity using $\texttt{ShareVer}$. Every valid signature share is an element of $\mathcal{G}_1$ and forms a coDH tuple with the blind message and the public key share, which $\texttt{ShareVer}$ checks by calculating $\sigma_i \in \mathcal{G}_1 \wedge e(m', P_i) = e(\sigma_i', G_2)$. The algorithm also returns 0 if the blinded message is invalid because $m' \notin \mathcal{G}_1$.

$\sigma' \leftarrow \texttt{Combine}(\Sigma')$: The valid shares are used by the combiner to interpolate the blind signature $\sigma' = \texttt{Interpolate}(\{(i, \sigma_i') \mid \sigma_i' \in \Sigma_t'\}, 0)$ from a subset $\Sigma_t' \subseteq \Sigma'$ of size t. The blind signature is then given to $\mathcal{U}$.

$\sigma \leftarrow \texttt{Unblind}(\sigma', \beta, P)$: $\mathcal{U}$ unblinds the signature σ' using the inverse of its blinding key β and the blinding public key $\hat{P}$ to receive the final signature $\sigma = \sigma' + (-\beta) \cdot \hat{P}$.

```
$\mathcal{TBS}$-$\mathcal{BLS}$.Keygen($1^\lambda, t, n$)
$(x_1, \dots, x_n \xrightarrow{t} x) \leftarrow\$\, \mathbb{Z}_q[i]$
$P_i := x_i \cdot G_2 \mid i \in \{1, \dots, n\}$
$P' := x \cdot G_2$
$\hat{P} := x \cdot G_1$
$P := (P', \hat{P})$
return $(x_1, \dots, x_n, P_1, \dots, P_n, P)$
```

```
$\mathcal{TBS}$-$\mathcal{BLS}$.ShareVer($P_i, m', \sigma'_i$)
return $m' \in \mathcal{G}_1 \wedge \sigma'_i \in \mathcal{G}_1$
    $\wedge\, e(m', P_i) = e(\sigma'_i, G_2)$
```

```
$\mathcal{TBS}$-$\mathcal{BLS}$.Combine($\Sigma'$)
$\sigma' :=$ Interpolate$(\{(i, \sigma'_i) \mid \sigma'_i \in$ take$(t, \Sigma')\}, 0)$
return $\sigma'$
```

```
$\mathcal{TBS}$-$\mathcal{BLS}$.Blind($m$)
$\beta \leftarrow\$\, \mathbb{Z}_q$
$m' := H(m) + \beta \cdot G_1$
return $(m', \beta)$
```

```
$\mathcal{TBS}$-$\mathcal{BLS}$.Unblind($\sigma', \beta, P$)
if $\sigma' \notin \mathcal{G}_1$ then
  return $\bot$
endif
$(P', \hat{P}) := P$
$\sigma := \sigma' - \beta \cdot \hat{P}$
return $\sigma$
```

```
$\mathcal{TBS}$-$\mathcal{BLS}$.Sign($x_i, m'$)
if $m' \notin \mathcal{G}_1$ then
  return $\bot$
endif
$\sigma'_i = x_i \cdot m'$
return $\sigma'_i$
```

```
$\mathcal{TBS}$-$\mathcal{BLS}$.Verify($P, m, \sigma$)
$(P', \hat{P}) := P$
return $\mathcal{BLS}$.Verify($P', m, \sigma$)
```

Figure 4.1.: Algorithms defining scheme $\mathcal{TBS}$-$\mathcal{BLS}$.

The blind signature σ' is an untrusted input. The user $\mathcal{U}$ needs to check that $\sigma' \in \mathcal{G}_1$ and return $\bot$ otherwise.

$\{0, 1\} \leftarrow$ `Verify`$(1^\lambda, P, m, \sigma)$**:** The verification algorithm is exactly the same as for $\mathcal{BLS}$ using the aggregated public key P'.

Note that we assume the public key to be valid, i.e. $P_1, \dots, P_n, P' \in \mathcal{G}_2$, $\hat{P} \in \mathcal{G}_1$, $P' =$ `Interpolate`$(\{(1, P_1), \dots, (n, P_n)\}, 0)$ and $\hat{P} = \varphi(P')$, which we expect to be ensured externally (e.g. on key exchange).

We claim that $\mathcal{TBS}$-$\mathcal{BLS}$ is a secure threshold blind signature scheme by arguing that it is (t, n)-OMF-CMA secure, for $t < n/2$ also (t, n)-robust and unconditionally blind.

Unforgeability We prove that the scheme $\mathcal{TBS}$-$\mathcal{BLS}$ is unforgeable as defined in definition 9, meaning that no polynomial time adversary controlling up to $t - 1$ signers can produce more valid signatures than it starts signing sessions with more than negligible probability.

Lemma 4. *Let $(\mathcal{G}_1, \mathcal{G}_2)$ be a group pair with an isomorphism $\varphi : \mathcal{G}_2 \to \mathcal{G}_1$ and a pairing e for which the OMcoCDH problem is assumed to be hard, then $\mathcal{TBS}$-$\mathcal{BLS}$ is (t, n)-OMF-CMA secure.*

Proof of lemma 4. We show that if there exists a polynomial time adversary $\mathcal{A}$, that can win (t,n)-$\texttt{OMF-CMA}_{\mathcal{A}}^{\mathcal{TBS}\text{-}\mathcal{BLS}}(1^\lambda)$ we can construct an adversary $\mathcal{B}$ that can win $\texttt{OMcoCDH}_{\mathcal{B}}^{\varphi}(1^\lambda)$.

In game $\texttt{OMcoCDH}_{\mathcal{B}}^{\varphi}(1^\lambda)$ $\mathcal{B}$ is called with argument X, the constant group element with which each outputted pair of group elements has to form a valid Diffie-Hellman tuple. $\mathcal{B}$ also has access to a target oracle $\mathcal{O}_\mathrm{T}$, a Diffie-Hellman oracle $\mathcal{O}_\mathrm{DH}$ and the isomorphism oracle φ, which are used to construct the blind signature oracle $\mathcal{O}_\mathrm{bsig}$ (invocations counted as q_s) and the random oracle $\mathcal{O}_\mathrm{H}$ given to $\mathcal{A}$.

$\mathcal{B}$ is implemented as follows:

1. Set the target aggregate public key to $P' = X$. Choose random private key shares $x_1, \dots, x_{t-1} \leftarrow\!\$\, \mathbb{Z}_q$ to give to $\mathcal{A}$.

2. Calculate the public key shares making up the public key P to give to $\mathcal{A}$ such that they are consistent with the previously generated private key shares and P'.

   Since the discrete logarithm of P' is unknown to $\mathcal{B}$ so are the private key shares $x_t, \dots, x_n$. Thus instead of simply multiplying these with G_2, $\mathcal{B}$ must interpolate these from the other public key shares and P'.

   For that the adversary's public key shares $P_i = x_i \cdot \mathcal{G}_2$ for $i \in \{1, \dots, t-1\}$ can be easily calculated from the private key shares. To define the public key polynomial t shares are needed though. We recount that the shared secret, i.e. in this case the aggregate public key P', is the evaluation of the secret sharing's polynomial at place $\hat{x} = 0$ (see section 2.5). Thus $\mathcal{B}$ uses the t points $S = \{(0, P'), (1, P_1), \dots, (t-1, P_{t-1})\}$ that define the public key polynomial to interpolate the remaining public key shares $P_i = \texttt{Interpolate}(S, i)$ for $i \in \{t, \dots, n\}$.

   The blinding public key $\hat{P} = \varphi(X)$ on $\mathcal{G}_1$, which is part of the public key, is computed using the isomorphism oracle φ. The public key is set to $P = (P', \hat{P})$.

3. Call $\mathcal{A}_\mathrm{sig}^{\mathcal{O}_\mathrm{sig}, \mathcal{O}_\mathrm{H}}(P, P_1, \dots, P_n, x_1, \dots, x_{t-1})$, simulating the two oracles as described below.

4. If $\mathcal{A}$ wins, it outputs $s = q_s + 1$ valid message-signature pairs (m_i, σ_i). We know that:

   - for σ_i to be valid, $(\mathcal{O}_\mathrm{H}(m_i), X, \sigma_i)$ has to be a valid DH-tuple,
   - every group element returned by $\mathcal{O}_\mathrm{H}$ was returned by $\mathcal{O}_\mathrm{T}$,
   - $q_\mathrm{DH} = q_s$ due to the construction of $\mathcal{O}_\mathrm{bsig}(m')$ and
   - all message-signature pairs outputted by $\mathcal{A}$ being distinct is equivalent to all messages being distinct since $\mathcal{BLS}$ signatures are unique (lemma 1).

   That means that every message-signature pair (m_i, σ_i) can be transformed into an element $(\mathcal{O}_\mathrm{H}(m_i), \sigma) = (Y, x \cdot Y) = (Y, Z)$ of $\mathcal{B}$'s answer for game $\texttt{OMcoCDH}_{\mathcal{B}}^{\varphi}(1^\lambda)$.

   There is an edge case where two calls $\mathcal{O}_\mathrm{H}(m)$ produce the same output for different values of m. If this is the case for any two messages returned by $\mathcal{A}$, $\mathcal{B}$ aborts and raises the event $\texttt{collision}$.

If no collision happens all (Y, Z) tuples are distinct due to all messages and transitively the hashes thereof being distinct. $\mathcal{B}$ outputs $|DH| = s = q_s + 1 > q_{\mathrm{DH}}$ valid and distinct tuples, winning $\mathtt{OMcoCDH}^{\varphi}_{\mathcal{B}}(1^\lambda)$.

The oracles given to $\mathcal{A}$ are implemented as follows:

$\Sigma_G \leftarrow \mathcal{O}_{\mathrm{bsig}}(m')$: The oracle has access to the adversary's private key shares $x_1, \ldots, x_{t-1}$, which allows to calculate the $t-1$ blind signature shares $\sigma'_i = \mathcal{TBS}\text{-}\mathcal{BLS}.\mathtt{Sign}(x_i, m')$ for $i \in \{1, \ldots, t-1\}$ of $\mathcal{A}$.

Furthermore it requests the aggregated blind signature $\sigma' = \mathcal{O}_{\mathrm{DH}}(m')$ from the coDH oracle. σ' is the value of the secret sharing's polynomial at place $\hat{x} = 0$ as discussed in section 2.5.

These t points $S = \{(0, \sigma'), (1, \sigma'_1), \ldots, (t-1, \sigma'_{t-1})\}$ are used to interpolate the remaining blind signature shares $\sigma'_i = \mathtt{Interpolate}(S, i)$ for $i \in \{t, \ldots, n\}$.

The honest participants' blind signature shares $\sigma'_t, \ldots, \sigma'_n$ are then returned to $\mathcal{A}$.

$H \leftarrow \mathcal{O}_{\mathrm{H}}(m)$: The oracle keeps a table T of previously queried messages. If T contains an entry for m, return the associated group element H. Otherwise query $\mathcal{O}_{\mathrm{T}}()$, which returns a random $H \in \mathcal{G}_1$. Save (m, H) to T and return H.

There is one case in which $\mathcal{B}$ does not win although $\mathcal{A}$ won: when there is a collision in $\mathcal{O}_{\mathrm{H}}$, which is equivalent to $\mathcal{O}_{\mathrm{T}}$ randomly choosing the same group element twice. The probability of this happening $P[\mathtt{collision}] = P[\exists (m_1, H_1), (m_2, H_2) \in T : m_1 \neq m_2 \wedge H_1 = H_2]$ shrinks with increasing group size $|\mathcal{G}|$, which in turn grows exponentially with a growing security parameter λ. This means $P[\mathtt{collision}]$ is negligible and $\mathcal{B}$ wins with a negligibly lower probability than $\mathcal{A}$. Note that $P[\mathtt{collision}] \approx 1 - e^{-n^2/(2d)}$ since it is equivalent to the birthday problem.

The run time of $\mathcal{B}$ can be quantified as $t_{\mathcal{B}} = t_{\mathcal{A}} + \mathcal{O}(q_s) + \mathcal{O}(q_h)$, with q_h being the invocations of $\mathcal{O}_{\mathrm{H}}$. Both oracles run for polynomial time.

Since we assume OMcoCDH to be hard for the pairing we use to construct the scheme and $\mathcal{B}$ would break that assumption, this means that $\mathcal{TBS}\text{-}\mathcal{BLS}$ is (t, n)-OMF-CMA secure. □

Robustness We prove that the scheme $\mathcal{TBS}\text{-}\mathcal{BLS}$ is robust as defined in definition 10, meaning that as long as less than t signers are malicious a valid signature will be produced.

Lemma 5. *Let $(\mathcal{G}_1, \mathcal{G}_2)$ be a group pair with a pairing e and let $t < n/2$, then $\mathcal{TBS}\text{-}\mathcal{BLS}$ is (t, n)-robust.*

Proof of lemma 5. We argue that no adversary $\mathcal{A}$ can win the robustness game in fig. 2.8.

First the challenger gives the adversary $t-1$ keys and lets it choose a message m to be blind signed. Due to m being hashed to $\mathcal{G}_1$ on blinding and the algorithm `Blind` being infallible there is nothing the adversary can do to make this step fail.

Next, the adversary is allowed to contribute up to $t-1$ signatures. The challenger ensures that only signatures belonging to one of the adversary-controlled keys are contributed. Every blind signature share is also checked for validity using `ShareVer`. Every valid share is guaranteed to lie on the same blind signature polynomial and any t blind signature shares may be used to interpolate the aggregate blind signature σ'. The set of $\mathcal{A}$'s signatures that were valid is called $\Sigma_{\mathcal{A}}^{\text{val}}$.

The challenger now calculates the set Σ'_C of the remaining $n - t + 1$ blind signature shares using its own keys.

The union of both distinct sets is used to interpolate the blind signature σ'. Note that even without any valid contributions from $\mathcal{A}$ the challenger has $|\Sigma'_C| = n - t + 1$ of its own shares to interpolate σ', which for $t < n/2$ is sufficient. Even if $\mathcal{A}$ contributed valid shares, these lie on the same polynomial as the challenger's ones and will thus lead to the same valid σ'.

Finally any valid σ' can always be unblinded and will result in a valid signature σ for m. Thus `Verify` will return 1 and $\mathcal{A}$ will loose, making the signature scheme $\mathcal{TBS}$-$\mathcal{BLS}$ robust. □

Blindness We prove that the scheme $\mathcal{TBS}$-$\mathcal{BLS}$ is unconditionally blind as defined in definition 11.

Lemma 6. *$\mathcal{TBS}$-$\mathcal{BLS}$ is unconditionally blind.*

Proof of lemma 6. We claim that there exists no adversary that is able to distinguish two signing sessions when being handed the results of these. This means that nothing can be learned about (m, σ) from observing (m', σ') and in reverse. This is formalized in game $\texttt{Blindness}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$.

To prove that no adversary $\mathcal{A}$ can win $\texttt{Blindness}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$ we use a sequence of games shown in fig. 4.2 between which $\mathcal{A}$'s probability of winning (i.e. the game returning 1) does not change.

$\mathsf{Game}_1(\lambda)$: This game is equivalent to $\texttt{Blindness}_{\mathcal{A}}^{\mathcal{TBS}}(1^\lambda)$ with the algorithms expanded. Note though that the check for public key validity required by $\mathcal{TBS}$-$\mathcal{BLS}$.`Verify` only happes once (line 13) instead of twice since both checks would have the same result.

$\mathsf{Game}_2(\lambda)$: In the first hop we replace the unblinding step in line 9 and 10. Since our scheme produces $\mathcal{BLS}$ signatures and these are unique (lemma 1) the challenger can also just calculate them directly as $\sigma_i = \log_{G_2}(P') \cdot H_1(m_i)$ for $i \in \{0, 1\}$ instead of relying on $\mathcal{A}$'s blind signatures[1]. This could be detected by the adversary by outputting a wrong blind signature, but this would not change its probability of winning since wrong signatures make $\mathcal{A}$ lose the game.

Since the challenger now does not have the unblinded version of $\mathcal{A}$'s blind signature shares anymore, the validation of σ'_0 and σ'_1 in lines 11 and 12 is also adapted. The uniqueness of $\mathcal{BLS}$ signatures translates to uniqueness of the blind signatures. Thus

[1]Note that we in the realm of information theoretical security and thus able to calculate the discrete logarithm of $\mathcal{A}$'s public key.

$\mathsf{Game}_1(\lambda)$

```
$((P', \hat{P}), m_0, m_1) := \mathcal{A}_1(1^\lambda)$

// Blind messages
$\beta_0, \beta_1 \leftarrow\$ \mathbb{Z}_q$
$m'_0 := H_1(m_0) + \beta_0 \cdot \mathcal{G}_1$
$m'_1 := H_1(m_1) + \beta_1 \cdot \mathcal{G}_1$

// Run blind signing protocol
$b \leftarrow\$ \{0, 1\}$
$(\sigma'_b, \sigma'_{1-b}) := \mathcal{A}_2(m'_b, m'_{1-b})$

$\sigma_0 := \sigma'_0 - \beta_0 \cdot \hat{P}$
$\sigma_1 := \sigma'_1 - \beta_1 \cdot \hat{P}$
$v_0 := \mathcal{BLS}.\mathtt{Verify}(P', m_0, \sigma_0)$
$v_1 := \mathcal{BLS}.\mathtt{Verify}(P', m_1, \sigma_1)$
$v_2 := (e(\hat{P}, G_2) = e(G_1, P'))$

// Guess message sign order
return $v_0 \wedge v_1 \wedge v_2 \wedge (b = \mathcal{A}_3(\sigma_0, \sigma_1))$
```

$\mathsf{Game}_2(\lambda)$

```
$((P', \hat{P}), m_0, m_1) := \mathcal{A}_1(1^\lambda)$

// Blind messages
$\beta_0, \beta_1 \leftarrow\$ \mathbb{Z}_q$
$m'_0 := H_1(m_0) + \beta_0 \cdot \mathcal{G}_1$
$m'_1 := H_1(m_1) + \beta_1 \cdot \mathcal{G}_1$

// Run blind signing protocol
$b \leftarrow\$ \{0, 1\}$
$(\sigma'_b, \sigma'_{1-b}) := \mathcal{A}_2(m'_b, m'_{1-b})$

$\sigma_0 := \log_{G_2}(P') \cdot H_1(m_0)$
$\sigma_1 := \log_{G_2}(P') \cdot H_1(m_1)$
$v_0 := (\sigma'_0 = \log_{G_2}(P') \cdot m'_0)$
$v_1 := (\sigma'_1 = \log_{G_2}(P') \cdot m'_1)$
$v_2 := (e(\hat{P}, G_2) = e(G_1, P'))$

// Guess message sign order
return $v_0 \wedge v_1 \wedge v_2 \wedge (b = \mathcal{A}_3(\sigma_0, \sigma_1))$
```

$\mathsf{Game}_3(\lambda)$

```
$((P', \hat{P}), m_0, m_1) := \mathcal{A}_1(1^\lambda)$

// Draw random "blind message"
$m'_0 \leftarrow\$ \mathcal{G}_1$
$m'_1 \leftarrow\$ \mathcal{G}_1$

// Run blind signing protocol
$b \leftarrow\$ \{0, 1\}$
$(\sigma'_b, \sigma'_{1-b}) := \mathcal{A}_2(m'_b, m'_{1-b})$

$\sigma_0 := \log_{G_2}(P') \cdot H_1(m_0)$
$\sigma_1 := \log_{G_2}(P') \cdot H_1(m_1)$
$v_0 := (\sigma'_0 = \log_{G_2}(P') \cdot m'_0)$
$v_1 := (\sigma'_1 = \log_{G_2}(P') \cdot m'_1)$
$v_2 := (e(\hat{P}, G_2) = e(G_1, P'))$

// Guess message sign order
return $v_0 \wedge v_1 \wedge v_2 \wedge (b = \mathcal{A}_3(\sigma_0, \sigma_1))$
```

$\mathsf{Game}_4(\lambda)$

```
$((P', \hat{P}), m_0, m_1) := \mathcal{A}_1(1^\lambda)$

// Draw random "blind message"
$m'_0 \leftarrow\$ \mathcal{G}_1$
$m'_1 \leftarrow\$ \mathcal{G}_1$

// Run blind signing protocol
$b \leftarrow\$ \{0, 1\}$
$(\sigma'_0, \sigma'_1) := \mathcal{A}_2(m'_0, m'_1)$

$\sigma_0 := \log_{G_2}(P') \cdot H_1(m_0)$
$\sigma_1 := \log_{G_2}(P') \cdot H_1(m_1)$
$v_0 := (\sigma'_0 = \log_{G_2}(P') \cdot m'_0)$
$v_1 := (\sigma'_1 = \log_{G_2}(P') \cdot m'_1)$
$v_2 := (e(\hat{P}, G_2) = e(G_1, P'))$

// Guess message sign order
return $v_0 \wedge v_1 \wedge v_2 \wedge (b = \mathcal{A}_3(\sigma_0, \sigma_1))$
```

Figure 4.2.: Game hop showing that $\mathcal{TBS}$-$\mathcal{BLS}$ is perfectly blind. $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ is a stateful adversary where $\mathcal{A}_1$ chooses a public key and two messages m_0, m_1, $\mathcal{A}_2$ produces two blind signatures for two supplied blinded messages and $\mathcal{A}_3$ tries to guess in which order messages were supplied to $\mathcal{A}_2$. Game_1 is equivalent to $\mathtt{Blindness}_{\mathcal{A}}^{\mathcal{TBS}\text{-}\mathcal{BLS}}$.

the expected blind signature is calculated and compared to the blind signature returned by $\mathcal{A}$.

$\mathsf{Game}_3(\lambda)$: Now that the blinding keys β_0 and β_1 are only used for blinding and not for unblinding anymore it is easy to see that $\beta_i \cdot G_1$ with a uniformly drawn $\beta_i \leftarrow\$ \mathbb{Z}_q$ has the same uniform probability distribution as uniformly drawing an element from $\mathcal{G}_1$. Since $\mathcal{G}_1$ is a cyclic group we also know that the sum of the message element and a uniformly random group element is indistinguishable from a random group element [19]. Thus we replace the blinding of the actual message with the drawing of a random group element in lines 4 and 5 without $\mathcal{A}$'s advantage changing.

$\mathsf{Game}_4(\lambda)$: Finally we notice that the two random elements m'_0 and m'_1 can be given to $\mathcal{A}$ in any order since they were drawn randomly in the first place and thus $\mathcal{A}$'s advantage does not change.

It is easy to see that b is not used in any calculations in Game_4 anymore and $\mathcal{A}$ can thus not learn anything about it. This leaves $\mathcal{A}$ with a chance of $1/2$ at most[2] to win the game by guessing. Thus our scheme is unconditionally blind. $\square$

[2] $\mathcal{A}$ could also lower its chance by outputting wrong blind signatures.

5. Future Work

We introduced a new combination of existing $\mathcal{BLS}$ based signature schemes that together form a threshold blind signature scheme. We proved said scheme unforgeable, robust and blind under the OMcoCDH assumption in the random oracle model.

The scheme can be used to construct a federated version of chaumian e-cash. We believe that a federated e-cash protocol could be both a scaling solution and privacy improvement as a second layer on Bitcoin. As such we will work on constructing such a protocol, with our $\mathcal{TBS}$-$\mathcal{BLS}$ scheme being core part, next.

Bibliography

[1] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*, pp. 199–203, Springer, 1983.

[2] M. Kucharczyk, "Blind signatures in electronic voting systems," in *International Conference on Computer Networks*, pp. 349–358, Springer, 2010.

[3] A. 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 Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings* (Y. Desmedt, ed.), vol. 2567 of *Lecture Notes in Computer Science*, pp. 31–46, Springer, 2003.

[4] W.-S. Juang and C.-L. Lei, "Blind threshold signatures based on discrete logarithm," in *Annual Asian Computing Science Conference*, pp. 172–181, Springer, 1996.

[5] J. Kim, K. Kim, and C. Lee, "An efficient and provably secure threshold blind signature," in *International Conference on Information Security and Cryptology*, pp. 318–327, Springer, 2001.

[6] Z. Cao, H. Zhu, and R. Lu, "Provably secure robust threshold partial blind signature," *Science in China Series F: Information Sciences*, vol. 49, no. 5, pp. 604–615, 2006.

[7] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova, "On the (in)security of ros," in *Advances in Cryptology – EUROCRYPT 2021* (A. Canteaut and F.-X. Standaert, eds.), (Cham), pp. 33–53, Springer International Publishing, 2021.

[8] D. L. Vo, F. Zhang, and K. Kim, "A new threshold blind signature scheme from pairings," 2003.

[9] V. Kuchta and M. Manulis, "Rerandomizable threshold blind signatures," in *International Conference on Trusted Systems*, pp. 70–89, Springer, 2014.

[10] M. Bellare, C. Namprempre, and G. Neven, "Unrestricted aggregate signatures," in *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings* (L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, eds.), vol. 4596 of *Lecture Notes in Computer Science*, pp. 411–422, Springer, 2007.

[11] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008. Applications of Algebra to Cryptography.

[12] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” in *Advances in Cryptology — ASIACRYPT 2001* (C. Boyd, ed.), (Berlin, Heidelberg), pp. 514–532, Springer Berlin Heidelberg, 2001.

[13] S. Jarecki, A. Kiayias, and H. Krawczyk, “Round-optimal password-protected secret sharing and T-PAKE in the password-only model,” in *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II* (P. Sarkar and T. Iwata, eds.), vol. 8874 of *Lecture Notes in Computer Science*, pp. 233–253, Springer, 2014.

[14] H. Shacham, *New Paradigms in Signature Schemes.* PhD thesis, Stanford University, Dec. 2005.

[15] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, p. 612–613, Nov. 1979.

[16] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999* (M. I. Seltzer and P. J. Leach, eds.), pp. 173–186, USENIX Association, 1999.

[17] V. Kuchta and M. Manulis, “Rerandomizable threshold blind signatures,” in *Trusted Systems - 6th International Conference, INTRUST 2014, Beijing, China, December 16-17, 2014, Revised Selected Papers* (M. Yung, L. Zhu, and Y. Yang, eds.), vol. 9473 of *Lecture Notes in Computer Science*, pp. 70–89, Springer, 2014.

[18] G. Bracha, “Asynchronous byzantine agreement protocols,” *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[19] C. E. Shannon, “Communication theory of secrecy systems,” *Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, 1949.

[20] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, “Secure distributed key generation for discrete-log based cryptosystems,” *Journal of Cryptology*, vol. 20, pp. 51–83, May 2006.

[21] T. P. Pedersen, “Non-interactive and information-theoretic secure verifiable secret sharing,” in *Advances in Cryptology — CRYPTO ’91* (J. Feigenbaum, ed.), (Berlin, Heidelberg), pp. 129–140, Springer Berlin Heidelberg, 1992.

A. Equivocation Attack on Threshold Schemes

For every threshold scheme that is based on signature share generation by its participants of which any t can be combined to a valid signature, and does not enforce that every participant signs the same message through some external mechanism (as we assume to be the case in our security model in section 2.6), there exists the following standard forgery attack:

We assume $t = 2$ and $n = 3$, meaning any two participants' signature shares can be combined to a valid signature. The adversary, taking the role of the user requesting a signature (e.g. in some e-cash scheme) is allowed to initiate two signing sessions:

Session 1: peer 1 is given m_1 to sign, peer 2 is given m_1 to sign and peer 3 is given m_2 to sign. The shares from peer 1 and 2 are sufficient to build a valid signature.

Session 2: peer 1 is given m_2 to sign, peer 2 is given m_3 to sign and peer 3 is given m_3 to sign. Peer 3's share from the previous round and peer 1's share from this round create one valid signature. Furthermore the shares from peer 2 and 3 create another one.

We have now created three signatures in only two authorized signing sessions, which breaks unforgeability.

B. Flaw in Boldyreva's Proof

We argue that Boldyreva's proof in [3, Appendix A] makes an error in assuming that the DKG simulator [20, Figure 3] can return $n-t-1$ valid key shares for honest participants. Instead we claim that no consistent or useful key shares can be returned for simulated (i.e. "honest") participants.

B.1. Simplified Protocol

First we describe the honest path of the protocol as it can already be used to illustrate the flaw. Adding corner cases for dishonest participants would needlessly complicate things for this argument. Please note that, to keep consistent with the rest of this paper, we use additive group notation instead of multiplicative as in [20].

The scheme uses two secret sharing techniques to accomplish different goals as shown in fig. B.1:

Shamir Secret Sharing: As a result of the protocol we want each participant i to be in possession of a secret key share x_i, that is part of a (t, n)-Shamir secret sharing $(x_1, \dots, x_n) \xrightarrow{(t,n)} x$ and an aggregated public key $y = x \cdot G$.

Sum: But no participant should learn x or be able to control its value, that is why every participant i contributes their own secret sharing $(s_{i,1}, \dots, s_{i,n}) \xrightarrow{(t,n)} z_i$ (note that every such secret sharing is just a function f_i of degree t and that the key shares are evaluations $s_{i,j} = f_i(j)$ thereof). The final secret sharing is defined as the sum of all participant's secret sharings, such that $x = \sum_i z_i$ and respectively $x_j = \sum_i s_{i,j}$. This way no party learns the whole secret x.

After this **first protocol part ("Generating x")** every participant j should be in possession of $s_{i,j}$ for all $i \in \{1, \dots, n\}$ from which they can generate their secret key share $x_j = \sum_i s_{i,j}$.

In the **second part of the protocol ("Extracting y")** $y = x \cdot G$ is shared in a verifiable manner. For this the secret sharing of z_i every participant i created previously is reused and represent as a polynomial $f_i(x) = a_{i,0} + a_{i,1} \cdot x + \dots + a_{i,t-1} \cdot x^{t-1}$. Every participant i calculates t commitments $A_{i,k} = a_{i,k} \cdot G$ for $k \in \{0, \dots, t-1\}$. These are broadcast and each peer j can verify a peer i's commitment as follows:

$$s_{i,j} \cdot G = \sum_{k=0}^{t-1} A_{i,k} \cdot j^k \tag{B.1}$$

i \ $f_i(j)$	$f_i(0)$		$f_i(1)$	$f_i(2)$	$\dots$	$f_i(n)$
1	z_1	$\xleftarrow{(t,n)}$	$s_{1,1}$	$s_{1,2}$	$\dots$	$s_{1,n}$
2	z_2	$\xleftarrow{(t,n)}$	$s_{2,1}$	$s_{2,2}$	$\dots$	$s_{2,n}$
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	z_n	$\xleftarrow{(t,n)}$	$s_{n,1}$	$s_{n,2}$	$\dots$	$s_{n,n}$
$\sum$ (sum of above secret contributions)	x	$\xleftarrow{(t,n)}$	x_1	x_2	$\dots$	x_n

Figure B.1.: Visualization of the two dimensions of the DKG's secret sharing.

and by taking the discrete logarithm on both sides

$$s_{i,j} = \sum_{k=0}^{t-1} a_{i,k} \cdot j^k \tag{B.2}$$

$$s_{i,j} = f_i(j) \tag{B.3}$$

Since $x = \sum_i z_i$ and $A_{i,0} = a_{i,0} \cdot G = z_i \cdot G$, y can be easily extracted from these commitments by summing up the $A_{i,0}$ values: $y = \sum_i A_{i,0}$.

B.2. Simulator

The goal of the simulator is to convince an adversary $\mathcal{A}$ that controls the first[1] at most t participants that they are participating in a honest run of the protocol, while setting y to a predetermined value y' of which the discrete logarithm x' is not known. To do so the simulator runs participants $t, \dots, n$ while $\mathcal{A}$ runs participants $1, \dots, t-1$.

The first part of the protocol happens just as described above (assuming $\mathcal{A}$ behaves honestly). But for the second part the simulator needs to cheat: it wants to convince $\mathcal{A}$ that the aggregate public key is y', while in reality it is some other value $y \neq y'$ with a probability negligibly less than 1. Note that due to the lack of knowledge of $x' = \mathrm{dlog}_G\, y'$ the simulator could not influence the protocol in a way that would result in $y = y'$.

So instead the simulator manipulates one of the "honest" participant's (e.g. participant n's) commitments $(A_{n,0}, \dots, A_{n,k})$. Firstly $A'_{n,0}$ is set such that $y' = \sum_i A_{i,0}$:

$$A'_{n,0} = y' - \sum_{i=1}^{n-1} A_{i,0} \tag{B.4}$$

To be able to still satisfy eq. (B.1) $(A_{n,1}, \dots, A_{n,k})$ have to be also manipulated ($\lambda_{k,j}$ being the Lagrange coefficient for point j with the unbound variable k that can thus

[1]Gennaro et al. note that this can be assumed without loss of generality instead of any subset of at most t participants being controlled by $\mathcal{A}$.

later be set to $k \in \{1, \ldots, t-1\}$):

$$A'_{n,k} = \lambda_{k,0} \cdot A'_{n,0} + \sum_{j=1}^{t-1} s_{n,j} \cdot \lambda_{k,j} \cdot G \tag{B.5}$$

This means the simulator interpolates the t (participant, public key share)-points: $t-1$ adversary controlled evaluations of f_n that still need to fit to the commitments plus $A'_{n,0}$ to define the public key y' to commit to. The resulting function could be written as:

$$F^*_n(x) = A'_{n,0} + A'_{n,1} \cdot x + \cdots + A'_{n,t-1} \cdot x^{t-1} \tag{B.6}$$

The unknown (due to not knowing x') polynomial f^*_n defining the secret sharing belonging to this commitment polynomial F^*_n would be the following, with $a'_{n,k} = \operatorname{dlog}_G A'_{n,k}$:

$$f^*_n(x) = a'_{n,0} + a'_{n,1} \cdot x + \cdots + a'_{n,t-1} \cdot x^{t-1} \tag{B.7}$$

We note that $y \neq y' \Rightarrow A_{n,0} \neq A'_{n,0} \Rightarrow f_n \neq f^*_n$. To be more precise, we know that due to using $s_{n,j} \cdot G$ for $j \in \{1, \ldots, t\}$ to interpolate F^*_n, $\forall j \in \{1, \ldots, t-1\} : \; f_n(j) = f^*_n(j)$ and $\forall j \in \{t, n\} : \; f_n(j) \neq f^*_n(j)$. This means that the commitments $(A'_{n,0}, \ldots, A'_{n,k})$ do not fit to the previously shared secrets $(x_t, \ldots, x_n)$ anymore. These are now defined as $x'_j = \sum_{i=1}^{n-1} s_{i,j} + f^*_n(j)$ for $j \in \{t, \ldots, n\}$. But since the simulator does not know $f^*_n(j)$ for $j \in \{t+1, \ldots, n\}$ it lost its key shares by sharing a wrong public key. Any attempt at using the old shares $(x_t, \ldots, x_n)$ could be detected by $\mathcal{A}$ and make the simulation fail.

B.3. Proof Flaw and Fix

The proof mistakes only needing to manipulate one honest participant's commitments for only one honest participant's key being invalidated. But this ignores that the other "honest participants" are complicit by not complaining about the wrong commitments. We have shown that instead all simulator-controlled keys are invalidated and cannot be used to create e.g. signature shares that still lie on *the same* polynomial anymore.

This can be easily fixed. Instead of calculating σ_j for $j \in \{t, \ldots, n-1\}$ using the private key shares from the DKG simulation and only interpolating σ_n, each σ_j for $j \in \{t, \ldots, n\}$ has to be calculated such that together with the signature shares that $\mathcal{A}$ can generate, it would interpolate to the same signature σ provided by $\mathcal{O}_{\text{sig}}$. This can be done by interpolating the appropriate signature polynomial and evaluating it at the places $j \in \{t, \ldots, n\}$:

$$\sigma_j = \lambda_{j,0} \cdot \sigma + \sum_{i=1}^{t-1} \lambda_{j,i} \cdot x_i \cdot H_1(m) \tag{B.8}$$

The above equation shows the interpolation of the signature polynomial using $\mathcal{A}$'s signature shares and the aggregated signature. j is an unbound variable and to be set to $j \in \{t, \ldots, n\}$.

C. Secure Distributed Key Generation

The naive approach to generate a secret sharing is using a trusted dealer, that in the case of Shamir's scheme would randomly choose f. But clearly this is not suitable for settings where no such trusted third party exists. Distributed Key Generation (DKG) protocols allow to generate a secret sharings of a value unknown to any single of the n participant as long as sufficiently many of these are honest.

Gennaro et al. [20] describe a DKG scheme based on Shamir secret sharing. It differentiates itself from previous schemes in that it prevents attackers from biasing the generated secret as it proves is possible with Pedersen's scheme [21]. A high level overview is given in appendix B.1. Figure C.1 describes the protocol under the assumption that G and H are two generators of group $\mathcal{G}$ with order q for which the discrete logarithm problem is hard. Note that `ReconstructSecret` is left out for brevity, it reconstructs the secret of a compromised party in public. See [20] for details.

It is easy to see that the DKG can be used for Boldyreva's $\mathcal{TS}$-$\mathcal{BLS}$ scheme by setting $\mathcal{G} = \mathcal{G}_2$. For our $\mathcal{TBS}$-$\mathcal{BLS}$ scheme we need to make an addition to `PedersenVSS` since we also need to share $\hat{P}$, the isomorphism to the public key on group $\mathcal{G}_1$. This is easily done by requiring every peer j to also broadcast the isomorphism of $A_{j,0}$, we call it $\hat{A}_{j,0} = a_j, 0 \cdot G_1$. This allows to calculate $\hat{P} = \sum_{j \in QUAL} \hat{A}_{j,0}$.

```
GJKR_DKG $(1^\lambda, t, n)$
$f \leftarrow\$ \mathbb{Z}_q[X]$ such that $f(x) = a_0 + a_1 \cdot x + \cdots + a_t \cdot x^t$
$f' \leftarrow\$ \mathbb{Z}_q[X]$ such that $f'(x) = b_0 + b_1 \cdot x + \cdots + b_t \cdot x^t$
CommitToSecret$(f, f')$
SendContribution$(f, f')$
// Receive commitments
$(C, Faults_1) :=$ receive_bc()
ReceiveAndVerifyContributions$(f, f', C)$
$Faults_2 :=$ ProcessComplaints$(f, f', C)$
// Build honest participant list and calculate $x_i$
$QUAL := \{1, \ldots, n\} \setminus (Faults_1 \cup Faults_2)$
$x_i := \sum_{j \in QUAL} s_{j,i}$
// Run Pedersen VSS to extract $xG$
$(P, \mathcal{P}) :=$ PedersenVSS$(f)$
return $(x_i, P, \mathcal{P})$
```

```
ReceiveAndVerifyContributions$(f, f', C)$
$S :=$ receive()
for $j \in \{1, \ldots, n\}$ do
  if $(s_{j,i}, s'_{j,i}) \notin S \vee$
     $s_{j,i}G + s'_{j,i}H \neq \sum_{k \in \{0,\ldots,t-1\}} i^k \cdot C_{j,k}$ then
     broadcast(complaint against $j$: $(i, j)$)
  fi
done
```

```
CommitToSecret$(f, f')$
$C_i := (a_0G + b_0H, \ldots, a_{t-1}G + b_{t-1}H)$
broadcast$(C_i)$
```

```
SendContribution$(f, f')$
for $j \in \{1, \ldots, n\}$ do
  $(s_{i,j}, s'_{i,j}) := (f(j), f'(j))$
  send$(j, (s_{i,j}, s'_{i,j}))$
done
```

```
PedersenVSS$(f)$
$A_i := (a_{i,0}G, \ldots, a_{i,t}G)$
broadcast$(A_i)$
$A :=$ receive_bc()
for $A_j \in A$ do
  if $s_{j,i} \cdot G \neq \sum_{k=0}^{t-1} i^k \cdot A_{j,k}$ then
    broadcast(complaint $s_{j,i}$)
  fi
done $Complaints :=$ receive_bc()
for $s_{j,i} \in Complaints$ do
  $s_{j,0} :=$ ReconstructSecret$(s_{j,i})$
  $A_{j,0} := s_{j,0} \cdot G$
done
$P := \sum_{j \in QUAL} A_{j,0}$
$\mathcal{P} := \{A_{j,0} \mid j \in QUAL\}$
return $(P, \mathcal{P})$
```

```
ProcessComplaints$(f, f', C)$
$(Complaints, PFaults) :=$ receive_bc()
$CFaults := \{j \in \{1, \ldots, n\} \mid t - 1 < |\{(k, j') \in Complaints \mid j' = j\}|\}$
for $(j, i') \in Complaints$ if $i' = i$ do
  $(s_{i,j}, s'_{i,j}) := (f(j), f'(j))$
  broadcast$((s_{i,j}, s'_{i,j}))$
done
// Receive complaint replies
$CR =$ receive_bc()
$RFaults := \{(s_{j,j'}, s'_{j,j'}) \in CR \mid s_{j,j'}G + s'_{j,j'}H \neq \sum_{k \in \{0,\ldots,t-1\}} j'^k \cdot C_{j,k}\}$
return $RFaults \cup CFaults \cup PFaults$
```

Figure C.1.: Gennaro et al.'s [20] DKG protocol.