BLISK: Boolean circuit Logic Integrated into the Single Key

v.1

Rarimo

December 2025

Abstract

This paper introduces **BLISK**, a framework that compiles any monotone Boolean authorization policy into a single signature verification key, enabling only the authorized signer subset to produce the standard constant-size aggregated signatures. BLISK combines (1) n-of-n multisignatures to realize conjunctions, (2) key agreement protocols to realize disjunctions, and (3) verifiable group operations based on the 0-ART framework [1]. BLISK avoids distributed key generation (allowing users to reuse their long-term keys), supports publicly verifiable policy compilation, and enables non-interactive key rotation.

1 Introduction

The secure control of cryptographic keys under shared authorization policies is a foundational challenge in distributed systems, especially in digital asset custody. In many real-world settings, control over an operation or process must be shared among multiple parties according to a policy that is neither purely centralized nor a simple threshold. A typical example of such a policy is "two out of three executives must approve, provided the compliance officer participates," instead of pure multisignature.

Today, most deployed systems *"address"* this problem using threshold signature schemes, which allow any k-of-n parties to produce a signature verifiable under a single public key jointly. While threshold signatures are elegant and efficient, they suffer from several fundamental limitations.

First, they support only a single, fixed access structure of the form k-of-n. Let's conduct a thought experiment and try to implement the following policy using only the k-of-n signature construction:

To consider the transaction valid, it:

1. Must be signed by Alice (A) or Bob (B) (1-of-2), and
2. Must be signed by Carol (C) or Dave (D) (one more 1-of-2)

The experiment yields the following observation: the authorization policy $(A \vee B) \wedge (C \vee D)$ cannot be realized by any standard or weighted threshold signature scheme.

Corollary 1.0.1. *Threshold schemes express only cardinality-based access structures and cannot enforce structured constraints across disjoint signer groups. Thus, threshold constructions are insufficient to express asymmetric or complex hierarchical policies.*

Second, threshold schemes require a distributed key generation (DKG) procedure – an interactive protocol that must be executed by all participants and repeated whenever the participant set or threshold changes. Additionally, DKG produces fresh key material, preventing users from reusing existing wallets or long-term public keys. These limitations significantly complicate deployment in custody, governance, and regulated environments.

An alternative approach is to enforce complex authorization logic at the verification layer, for example, by embedding scripts or multiple verification keys on-chain [Gno18; Bit09]. But obviously, this approach leads to another set of disadvantages and limitations:

[1] Basically the BLISK is the idea that follows the idea of 0-ART, extending the verifiable key agreement protocol for custom boolean conditions

- **Increased verification costs and witness sizes.** Modern multisignature schemes enable efficient (constant-size) aggregation of public keys and signatures. With on-chain policy verification, we revert to verifying a batch of single signatures.

- **Policy structure leakage.** The script

  $$\langle \mathsf{AlicePK} \rangle\ \mathsf{OP\_CHECKSIGVERIFY}\ 1\ \langle \mathsf{BobPK} \rangle\ \langle \mathsf{CarolPK} \rangle\ 2\ \mathsf{OP\_CHECKMULTISIG}$$

  says clearly what's happening and who is responsible for operating the specific UTXO. The same problem applies not only to Bitcoin; if the smart account implements policy management, that policy is visible to anyone.

Our approach. In this work, we take a different perspective. Rather than modifying the signing or verification algorithms to support richer policies, we asked ourselves, *"Can an expressive authorization policy be compiled into a single public key, such that any authorized subset of parties can produce a standard aggregated signature for it?"*.

That's how we came to **BLISK**, a cryptographic construction that compiles Boolean authorization policies over existing public keys into a single verification key. The resulting key can be combined with a multisignature scheme (Schnorr-based or BLS), without modifying the signature verification logic (and introducing new attack vectors).

At a high level, our construction relies on two observations:

1. *Multisignature n-of-n schemes* naturally implement logical AND ($\wedge$): all participating signers must contribute for a signature to be valid.

2. *Key agreement protocols* allow a group of participants to derive a shared secret such that any one member can compute it, effectively implementing logical OR ($\vee$).

By combining these two primitives, we can construct a policy circuit in which each internal node represents either an AND or an OR gate, and each input wire corresponds to an existing user's public key. Each node of the circuit is deterministically mapped to a public key, and the root key serves as the sole verification key for signatures generated under the specified policy. After the policy circuit is created and resolved, the signing process follows a standard multisignature protocol among the parties assigned to the policy.

Contributions. We summarize our contributions as follows:

1. We introduce policy circuits, a model for compiling Boolean policies into cryptographic key material.

2. We show how policy circuits can be combined with multisignature protocols to receive expressive authorization policies with efficient key aggregation.

3. We describe a mechanism for verifiable key rotation and membership changes that require no global circuit redeployment and are non-interactive.

4. We provide a reference implementation of the compiler for policy circuits based on BLISK.

5. We prove the security of $\Sigma_{\mathsf{BLISK}}^{\mathsf{MuSig2+0\text{-}ART}}$ instance of **BLISK** protocol, which extends the standard signature (MuSig2) and key agreement scheme (0-ART) with policy circuit.

This paper is organized as follows. Section 2 provides an overview of related protocols and approaches. Section 3 consists of preliminaries and formal definitions of underlying primitives. We provide details of BLISK authorization circuits and BLISK multisignature protocol in Section 4. Section 5 includes discussions around the BLISK. Section 6 provides details of the reference implementation of the BLISK compiler. We provide an exact example of a BLISK signature instance, configured with MuSig2, 0-ART, and a specific policy, in Appendix A to demonstrate the appropriate security model.

2 Related work

n-of-n Multisignatures. Efficient n-of-n multisignature schemes, such as Schnorr-based constructions (e.g., MuSig2 [NRS20]) and pairing-based schemes (e.g., BLS [BLS01]), allow a fixed set of parties to jointly produce a compact signature verifiable under a single aggregated public key. These schemes are typically analyzed under strong notions of unforgeability and rely on well-studied hardness assumptions, while offering minimal verification overhead and strong compatibility with existing systems. Conceptually, n-of-n multisignatures realize a logical conjunction: a valid signature can be produced if and only if all designated participants contribute.

Our work does not modify or weaken these schemes. Instead, we reuse n-of-n multisignatures as a primitive, preserving their original security guarantees, and extend their applicability by embedding them into a higher-level construction that supports general Boolean authorization policies. In particular, we combine conjunctions realized by multisignatures with disjunctions realized through key agreement, enabling expressive access structures while retaining single-key verification and standard unforgeability guarantees.

Efficient threshold signatures. Threshold signature schemes allow a set of n parties to jointly produce a signature such that any subset of size at least k can sign, while smaller subsets cannot. A defining property of threshold signatures is that all valid signatures are verified under a single public key, making them attractive for blockchains and other cost-sensitive verification environments.

There are many approaches to efficient threshold signature schemes: ECDSA-based [GGN16; GG19; Hai+18], Schnorr-based [Gen+99; Gen+07], BLS [Bol02; DR23], etc. However, a fundamental limitation shared by all classical threshold signature schemes is that they support only a single policy of the form k-of-n. This restriction precludes enforcing mandatory participation by a specific party, although partial adaptations are possible.

One such adaptation is provided by weighted threshold schemes [24], which allow the secret to be distributed among parties with different weights. As a result, a given participant may account for a majority of the total weight, thereby making their participation necessary to obtain a valid signature. However, this construction only allows assigning numerous shares to a single participant and doesn't resolve the significant limitation of threshold signatures.

Our work differs from threshold signatures in two key aspects: it does not require DKG, and supports general monotone boolean authorization policies while preserving single-key verification.

Linear Secret Sharing Schemes. Linear secret sharing schemes (LSSS) [Bei25] are a relatively old primitive in which the secret is an element of a finite field, and the shares are a linear combination of the secret and random elements from the field. These schemes often realize access structures represented by a monotone formula, such as CNF or DNF. Of particular interest to our work are monotone formulas, which are highly effective for defining diverse policies (basically, any policy can be described by a logical operation).

However, LSSS-based approaches are poorly suited to efficient aggregation of signatures. First, they typically require distributing multiple shares per participant, increasing storage and communication costs. Second, signing and verification costs scale with the size of the access structure, rather than remaining constant. Finally, dynamic updates to the access structure generally require redistributing shares or rerunning key generation procedures.

In contrast, we aim to support expressive monotone policies while producing constant-size signatures verifiable under a single public key, without redistributing secret shares.

Multi-Party Computation. MPC protocols [Lin20] are a powerful primitive that enables distributed computation among different but related parties. Many different techniques and schemes have been developed for constructing MPC protocols [EKR18]. Formally, MPC allows the construction of protocols for computing any shared function.

However, MPC fundamentally differs from policy-based approaches, which restrict participation to specific user subsets. Moreover, the protocols are often complex and require substantial interaction, coordination, and computational resources.

Ring Signatures and OR Constructions In [kAN20], Monero proposes a multisignature scheme in which combinations of various coalitions form a structure that closely resembles our approach. The construction begins with trivial threshold access structures, such as 1-of-n and $(n-1)$-of-n. The latter is implemented using Diffie-Hellman shared secrets: each user computes $(n-1)$ shared private keys with every other participant, yielding a total of $n \cdot (n-1)$ keys.

By extending this design with aggregation techniques, they derive a threshold group aggregation algorithm for multisignatures. Furthermore, they briefly discuss the possibility of embedding and extending this family of algorithms to support policy specifications. However, this direction is not explored further in their work. We try to systematize this idea by combining disjunction realized through group key agreement with conjunction realized through multisignature aggregation.

Tree-Based Group Key Agreement Tree-based continuous group key agreement (CGKA) protocols, such as ART (asynchronous ratchet trees) [Alw+20] and TreeKEM [Bar+19], are designed to maintain a shared group secret under dynamic membership. A key property of these protocols is that any group member can derive the same group key, while non-members cannot.

We leverage this property in a novel way: instead of using CGKA solely for secure messaging, we use it to implement logical OR gates in authorization policies. To the best of our knowledge, this is the first work to use CGKA as a building block for compiling authorization logic into signing keys.

Another work, so-called 0-ART [Hru+25], is dedicated to verifiable operations (update, delete, etc) in ARTs. By adopting this approach, we can make all authorization circuits' updates verifiable, which is particularly important for trustless signer environments.

3 Preliminaries

Denote $\mathcal{Q}$ as a quorum of signers with $|\mathcal{Q}| = n$. $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ a hash function modeled as a random oracle. We can label the hash function instance with the tag $\mathcal{H}_{\mathsf{tag}}$ to facilitate clear domain separation.

Definition 3.1. *We define an efficiently aggregatable multi-party signature scheme as the set of algorithms*

$$\Sigma = \{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyAgg}, \mathsf{SigGen}, \mathsf{SigAgg}, \mathsf{SigVerify}\},$$

where:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}_\Sigma$ *initializes public parameters*
- $\mathsf{KeyGen}(\mathsf{pp}_\Sigma) \to (\mathsf{sk}, \mathsf{pk})$ *generates a signing key pair*
- $\mathsf{KeyAgg}(L) \to \mathsf{pk}_\wedge$ *aggregates public keys* $L = \{\mathsf{pk}_i \ : i \in \mathcal{Q}\}$ *into a single value* $\mathsf{pk}_\wedge$
- $\mathsf{SigGen}(\mathsf{sk}_i, m) \to \sigma_i$ *generates a signature share of the user* $i \in \mathcal{Q}$
- $\mathsf{SigAgg}(\{\sigma_i : i \in \mathcal{Q}\}) \to \sigma$ *aggregates signature shares to the single signature* σ *of the message* m *under the key* $\mathsf{pk}_\wedge$*. Usually, this operation is considered as a part of* $\mathsf{SigGen}(\cdot)$*, but we separate it to make a further construction clearer*
- $\mathsf{SigVerify}(\mathsf{pk}_\wedge, m, \sigma) \to \{0,1\}$ *verifies signatures and responds with* 1 *iff the signature is valid*

$\mathsf{SigGen}(\cdot)$*operation can consist of one or several rounds. We assume* Σ *satisfies* ***EUF-CMA security*** *under its standard assumptions.*

Definition 3.2. *To realize logical disjunction, we require a key-agreement primitive. We define it as a set of algorithms:*

$$\mathcal{K} = (\mathsf{Setup}, \mathsf{Init}, \mathsf{Derive}, \mathsf{Resolve}, \mathsf{Update}),$$

where:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}_\mathcal{K}$ *initializes public parameters*
- $\mathsf{Init}(\mathsf{pp}_\mathcal{K}, L) \to (\mathsf{pk}_\perp, \mathsf{aux})$ *initializes an OR node over the given public keys, producing a public auxiliary data* aux. $\mathsf{pk}_\vee$ *is defined only after the* $\mathsf{Resolve}(\cdot)$ *operation.*

- $\mathsf{Derive}(\mathsf{pp}_{\mathcal{K}}, \mathsf{sk}, \mathsf{aux}) \to \mathsf{sk}_{\vee}$ *allows an authorized participant to derive the secret key of the node*
- $\mathsf{Resolve}(\pi_{\mathsf{res}}, \mathsf{sk}, (\mathsf{pk}_{\perp}, \mathsf{aux})) \to (\mathsf{pk}_{\vee}, \mathsf{aux})$ *allows to define the public key corresponding to the shared secret of node participants. Initially, it calls* $\mathsf{Derive}(\cdot)$ *to receive a node secret key and then performs a corresponding public key generation* $\mathsf{pk}_{\vee}$*. Proof* π_{res} *is required to make sure the user is authorized to resolve the node.* $\mathsf{Resolve}(\cdot)$ *might be a part of* $\mathsf{Init}(\cdot)$ *procedure if the initiator controls one of the keys in the node*
- $\mathsf{Update}(\mathsf{pp}_{\mathcal{K}}, \mathsf{sk}, \mathsf{sk}', \mathsf{aux}) \to (\mathsf{pk}'_{\vee}, \mathsf{aux}')$ *allows to refresh the key material*

We assume $\mathcal{K}$ satisfies the security under ***PRF-ODH*** *(ART) or* ***KEM IND-CCA*** *(MLS) assumptions.*

Definition 3.3. *We define a non-interactive argument of knowledge protocol as a set of algorithms*

$$\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$$

where:

- $\mathsf{Setup}(1^{\lambda}, \mathcal{R}) \to (\mathsf{vp}_{\Pi}, \mathsf{pp}_{\Pi})$ *is the setup algorithm that takes the security parameter $\lambda \in \mathbb{N}$ and relation $\mathcal{R}$ and outputs verification and proving parameters* $\mathsf{vp}_{\Pi}, \mathsf{pp}_{\Pi}$*, respectively*
- $\mathsf{Prove}(\mathsf{pp}_{\Pi}, x, w) \to \pi$ *is the proving algorithm takes prover parameters* pp_{Π}*, public statement x, and a witness w, and outputs the proof π*
- $\mathsf{Verify}(\mathsf{vp}_{\Pi}, x, \pi) \to \{0,1\}$ *is the verification procedure that requires verifier parameters* vp*, public statement x, and proof π, and outputs 1 iff π is a valid proof*

Requirements for Π are to be ***complete***, ***knowledge sound*** *and* ***zero-knowledge*** *(ideally).*

3.1 Propositional logic

According to [Sip12], let us define the propositional logic. Boolean logic is a mathematical system built around two values, True and False, called Boolean values, which are often represented by the values 1 and 0. A Boolean function f can be defined as a formula over Boolean variables:

$$f(u_1, \dots u_n) : \{0,1\}^n \to \{0,1\}$$

A Boolean formula over the variables $u_1, \dots, u_n$ consists of these variables combined using the logical operators AND ($\wedge$), NOT ($\neg$), and OR ($\vee$). Such a formula is said to be in *CNF* (*Conjunctive Normal Form*) if it is a conjunction ($\wedge$) of clauses, where each clause is a disjunction ($\vee$) of variables or their negations. More generally, a CNF formula has the form:

$$f_{\mathsf{CNF}}(u_1, ..., u_n) := \bigwedge_i \left(\bigvee_j v_{ij} \right),$$

where each v_{ij} is either a variable $u_{k \in [n]}$ or its negation $\neg u_k$.

Since every function can be represented by a formula that uses only the connectives $\neg, \vee, \wedge$, it follows that each such formula can be transformed into an equivalent CNF form using the following laws:

- **Associative laws.** $(x \vee y) \vee z = x \vee (y \vee z), \quad (x \wedge y) \wedge z = x \wedge (y \wedge z)$
- **Distributive laws.** $(x \vee y) \wedge z = (x \wedge z) \vee (y \wedge z), \quad (x \wedge y) \vee z = (x \vee z) \wedge (y \vee z)$
- **Absorption laws.** $x \wedge (x \vee y) = x, \quad x \vee (x \wedge y) = x$

4 Protocol

In this section, we describe our main contribution – a *multi-party signature protocol* based on policy circuits. The high-level idea of the protocol is to apply an arbitrary logic set by some Boolean circuit for an efficiently aggregatable multi-party signature protocol (such as *BLS*[BLS01], *MuSig2*[NRS20], *MuSig-DN*[Nic+20], *MuSig-L*[BTT22]), achieving efficiently "one public key verifiable" signatures.

Let's first observe that all *n-of-n* signature protocols model a conjunction of logical propositions: *1st party has signed* **and** *2nd party has signed* **and** ... **and** *n-th party has signed*. Hence, we could efficiently emulate an AND gate by applying some multi-party signature protocol. We could also naively emulate an OR gate by applying *a tree-based continuous group key agreement protocol (CGKA)* based on either *DHE* (such as *ART*) or *KEM* (e.g. *MLS*), as every leaf in a CGKA tree could efficiently compute a root key; therefore, it could be used during signature generation to represent the condition "one of the group has signed".

By combining these gates in the policy tree and verifiable operations, we obtain a complete construction that enables "program public keys" with arbitrary authorization logic.

4.1 BLISK Authorization Circuits

This section formalizes the authorization mechanism used in BLISK. We begin by defining the class of Boolean circuits used to express authorization policies. We then show how such circuits are compiled into cryptographic material using the primitives defined in Section 3, yielding what we call cryptographic authorization circuits. The compilation preserves the semantics of the original Boolean circuit, in that satisfying assignments correspond exactly to sets of participants capable of producing valid signatures.

Let $\mathcal{U}$ be a finite set of participants. An authorization policy is represented as a monotone Boolean circuit over variables included in $\mathcal{U}$.

Definition 4.1 (Monotone Boolean Circuit)**.** *A monotone Boolean circuit is a tuple*

$$\mathcal{C} = (\mathcal{V}, \mathcal{E}, r, \ell, \alpha), \quad \textit{where}$$

- $(\mathcal{V}, \mathcal{E})$ *is finite directed acyclic graph with nodes* $v \in \mathcal{V}$ *and edges* $\epsilon \in \mathcal{E}$
- r *is a circuit root (output node)*
- $\ell : \mathcal{V} \to \{\mathsf{in}, \wedge, \vee\}$ *the function that labels each node with a gate type (*in *is an input wire,* $\wedge$ *and* $\vee$ *are AND and OR gates, respectively)*
- $\alpha : \{v \in \mathcal{V} : \ell(v) = \mathsf{in}\} \to \mathcal{U}$ *the function that labels each input node* v *with a distinct participant*

The ***monotone*** *means the circuit contains no negation gates.*

We require that every policy circuit must be presented in *conjunctive normal form* (CNF) so that the root node contains the only $\wedge$ gate (see Figure 1). Note that this is true for any proposition from $\{\wedge, \vee\}$ gates, as it could be transformed into CNF with the distributive law.

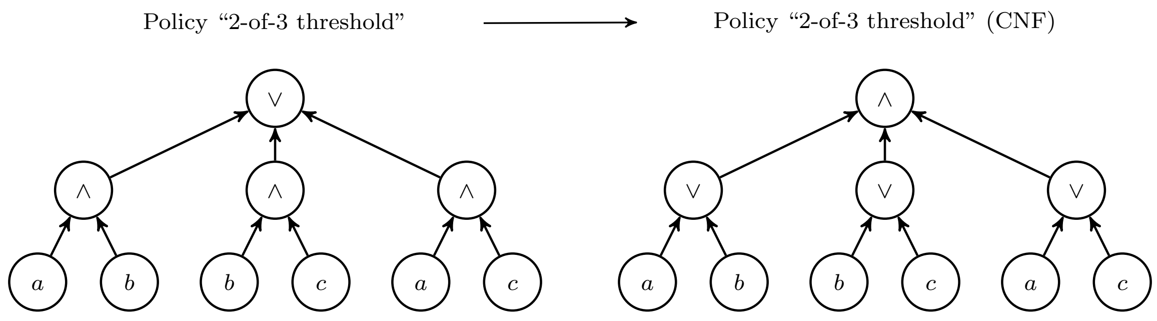

Figure 1: An illustration of transforming a general Boolean formula for a policy "2-of-3" threshold into CNF form.

Definition 4.2 (Circuit Evaluation)**.** *Given a subset* $\mathcal{Q} \subseteq \mathcal{U}$*, the evaluation function* $\mathsf{Eval} : (\mathcal{C}, \mathcal{Q}) \to \{\mathsf{True}, \mathsf{False}\}$ *on the circuit* $\mathcal{C}$ *on* $\mathcal{Q}$ *is defined as follows:*

1. *an input node labeled by* u *evaluates to* True *iff* $u \in \mathcal{Q}$
2. *a conjunction node evaluates to* True *iff all of its children evaluate to* True
3. *a disjunction node evaluates to* True *iff at least one of its children evaluates to* True

We say that $\mathcal{Q}$ *satisfies the circuit* $\mathcal{C}$ *if* $\mathsf{Eval}(\mathcal{C}, \mathcal{Q}) = \mathsf{True}$*.*

This class of circuits captures monotone access structures and is expressive enough to represent threshold, hierarchical, and general Boolean authorization policies.

4.1.1 Compiling the Circuit to the Key

We now describe how a monotone Boolean circuit is compiled into cryptographic material. The compilation is parameterized by the primitives introduced in Section 3 – a multisignature scheme with an effective aggregation Σ, a key-agreement primitive $\mathcal{K}$, and a non-interactive argument of a knowledge system Π.

Each participant $u \in \mathcal{U}$ holds a long-term signature key pair $(\mathsf{sk}_u, \mathsf{pk}_u) \leftarrow \Sigma.\mathsf{KeyGen}$.

Definition 4.3 (Authorization Circuit Compilation)**.** *Let* $\mathcal{C} = (\mathcal{V}, \mathcal{E}, r, \ell, \alpha)$ *be a monotone Boolean circuit and* $\mathcal{Q}$ *be a signer subset, such as* $\mathsf{Eval}(\mathcal{C}, \mathcal{Q}) = \mathsf{True}$*. We define the function* $\mathsf{Compile} : (\mathsf{pps}, \mathcal{C}, \mathcal{Q}) \to \mathcal{C}_{\mathsf{BLISK}} = \{\mathsf{pk}_v, \mathsf{aux}_v\}^{|\mathcal{V}|}$ *which matches each* v *from the Boolean circuit to the tuple* $(\mathsf{pk}_v, \mathsf{aux}_v)$*, where* pk_v *is a public key assigned for node* v *and* aux_v *is an auxiliary data. Note that the compilation depends on the gate type of* v*.*

Auxiliary data for the node v consists of the tuple $\mathsf{aux}_v = (Ch(v), \ell(v))$, where $Ch(v)$ is the set of children nodes for the node v and $\ell(v)$ duplicates the gate type from a Boolean circuit.

Gate-Level Compilation. We define the compilation process and a compiled state for each node type:

Input nodes. If v is an input node ($\ell(v) = \mathsf{in}$) labeled by participant u, then:

- $\mathsf{pk}_v \leftarrow \mathsf{pk}_u$
- $\mathsf{aux}_v \leftarrow (\perp, \mathsf{in})$
- $\mathsf{sk}_v \leftarrow \mathsf{sk}_u$

Thus, controlling an input node is equivalent to possessing the participant's signing key.

Conjunction gates. If v is a conjunction gate with children set $Ch(v) \subset \mathcal{V}$, then:

- $\mathsf{pk}_v \leftarrow \Sigma.\mathsf{KeyAgg}(Ch(v))$
- $\mathsf{aux}_v \leftarrow (Ch(v), \wedge)$
- sk_v exists only if all corresponding $\mathsf{sk}_{i \in Ch(v)}$ are available, reflecting the semantics of $\wedge$

Disjunction gates. If v is a disjunction gate with children $Ch(v) \subset \mathcal{V}$, the compilation uses $\mathcal{K}$ to realize logical OR. Note that all OR gates are fan-2(any fan-n OR gate could be considered a composition of fan-2 gates by associativity) and must be resolved interactively by one of their children during the initial phase, analogous to how the root key of an asynchronous ratchet tree(ART) is computed.

- $(\mathsf{pk}_\perp, \mathsf{aux}_v) \leftarrow \mathcal{K}.\mathsf{Init}(\mathsf{pp}_\mathcal{K}, Ch(v))$:

  pk_v isn't defined yet

  $\mathsf{aux}_v \leftarrow (Ch(v), \vee)$
- $\mathsf{sk}_v \leftarrow \mathcal{K}.\mathsf{Derive}(\mathsf{pp}_\mathcal{K}, \mathsf{sk}_{i \in Ch(v)}, \mathsf{aux}_v)$
- $(\mathsf{pk}_v, \mathsf{aux}_v) \leftarrow \mathcal{K}.\mathsf{Resolve}(\pi_{\mathsf{res}_\mathsf{i}}, \mathsf{sk}_i, (\mathsf{pk}_\perp, \mathsf{aux}_v))$, where $i \in Ch(v)$

Proof-Carrying Compilation. BLISK requires that the compilation of authorization circuits be publicly verifiable. In particular, disjunction gates introduce nontrivial structure that must be validated. For each disjunction gate $\vee$, we define an authorization relation $\mathcal{R}$:

$$\mathcal{R}_\vee = \left\{ \begin{array}{r|r} (\mathsf{pk}_v, \mathsf{aux}_v); & \pi_{\mathsf{res_i}} \leftarrow \mathsf{PoK}(\mathsf{pk} \in Ch(v)) \\ (\mathsf{sk}, \mathsf{pk}) & \mathsf{sk}_v \leftarrow \mathcal{K}.\mathsf{Derive}(\mathsf{pp}_\mathcal{K}, \mathsf{sk}, \mathsf{aux}_v) \\ & (\mathsf{pk}_v, \mathsf{aux}_v) \leftarrow \mathcal{K}.\mathsf{Resolve}(\pi_{\mathsf{res_i}}, \mathsf{sk}, (\mathsf{pk}_\perp, \mathsf{aux}_v)) \end{array} \right\}$$

We don't include the $\mathsf{Init}(\cdot)$ operation in the relation, because it naturally is performed by the group coordinator (and every participant can check the authorization circuit publicly).

So we make a setup for the proof system in advance: $(\mathsf{vp}_\Pi, \mathsf{pp}_\Pi) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}_\vee)$. The witness of the proof $\pi_{\vee_v} \leftarrow \Pi.\mathsf{Prove}(\mathsf{pp}_\Pi, (\mathsf{pk}_v, \mathsf{aux}_v), \mathsf{sk}_{v \in Ch(v)})$ attests that $(\mathsf{pk}_v, \mathsf{aux}_v)$ was produced by a valid execution of $\mathcal{K}.\mathsf{Resolve}(\cdot)$ and a secret key sk_v is derivable from at least one child secret key using $\mathcal{K}.\mathsf{Derive}(\cdot)$. Proof is verified by the instance $\Pi.\mathsf{Verify}(\cdot)$. A cryptographic authorization circuit is valid if all required proofs verify.

Definition 4.4 (Cryptographic Satisfiability of Circuit)**.** *A set of participants $\mathcal{Q} \subseteq \mathcal{U}$ cryptographically satisfies the compiled circuit if, for every node $v \in \mathcal{V}$ that evaluates* $\mathsf{Eval}(\mathcal{C}, \mathcal{Q}) = \mathsf{True}$*, the participants in $\mathcal{Q}$ can jointly obtain the corresponding secret key* sk_v *(obtain can mean not only the direct access to the key, but an ability to generate the signature shard with together with all required shares can satisfy the common public key).*

In particular, cryptographic satisfiability of the root node r implies possession of sk_r. Moreover, if $\mathcal{Q}$ satisfies $\mathcal{C}$, then $\mathcal{Q}$ can produce a valid signature under the single verification key pk_r associated with the root node, using the signature scheme Σ. So, this approach establishes a correspondence between Boolean authorization logic and cryptographic realizability: the policy's Boolean satisfiability is equivalent to cryptographic key control.

Sequential Compilation of Authorization circuits In addition to structural correctness, BLISK requires that authorization circuits be resolved sequentially by participants, ensuring that every compiled node reflects explicit approval by the participants responsible for that node.

A sequential compilation order is a total order $\prec$ on $\mathcal{V}$ such that:

1. $\prec$ is a topological order of $(\mathcal{V}, \mathcal{E})$, i.e., for every edge $(x \to v) \in \mathcal{E}$, we have $x \prec v$
2. all input nodes precede all internal nodes

For each node $v \in \mathcal{V}$, define the set of responsible participants $Resp(v) \subseteq \mathcal{Q}$ as follows:

- If $\ell(v) = \mathsf{in}$ and $\alpha(v) = u$, then $Resp(v) = \{u\}$.
- If $\ell(v) = \wedge$, then
$$Resp(v) = \bigcup_{x \in \mathbf{v}'} Resp(x).$$
- If $\ell(v) = \vee$, then $Resp(v) = Resp(x)$ for some child $x \in Ch(v)$.

For disjunction gates, the choice of x is existential: it suffices that at least one child determines the responsibility set.

Definition 4.5 (Sequential Compilation/Resolution Protocol)**.** *Let $\prec$ be a sequential compilation order. The* sequential compilation protocol *proceeds in rounds indexed by nodes $v \in \mathcal{V}$ in increasing order under $\prec$. In round v, the following steps are executed:*

1. Availability. *The compiled public states* $(\mathsf{pk}_x, \mathsf{aux}_x)$ *of all children $x \in Ch(v)$ are available.*

2. Resolution. *The participants in $Resp(v)$ jointly execute the compilation algorithm corresponding to $\ell(v)$, producing* $(\mathsf{pk}_x, \mathsf{aux}_x)$*.*

3. Proof generation. *The participants in $Resp(v)$ generate a proof* π_{res} *attesting to the correct resolution of node v with respect to its children.*

4. Publication. *The tuple* $(\mathsf{pk}_v, \mathsf{aux}_v, \pi_v)$ *is published and becomes immutable input for all subsequent rounds.*

4.2 BLISK Multisignature

Definition 4.6. *We define a BLISK signature as a set of algorithms:*

$$\Sigma_{\mathsf{BLISK}} = \{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{PolicyCircuitInit}, \mathsf{AcceptPolicy}, \mathsf{KeyAgg}, \mathsf{SigGen}, \mathsf{SigAgg}, \mathsf{SigVerify}, \mathsf{KeyUpdate}\}$$

Parameters setup. $\mathsf{Setup}^{\Sigma,\mathcal{K},\Pi}_{\mathsf{BLISK}}(1^\lambda) := (\Sigma.\mathsf{Setup}(1^\lambda), \mathcal{K}.\mathsf{Setup}(1^\lambda), \Pi.\mathsf{Setup}(1^\lambda)) \to (\mathsf{pps}, \mathsf{pp}_\Sigma, \mathsf{pp}_\mathcal{K}, \mathsf{pp}_\Pi)$. *On input 1^λ, the setup algorithm initializes the global parameters* pps *for the system, signature primitive Σ, the key-agreement primitive $\mathcal{K}$, and the zero-knowledge proof system Π by calling the corresponding protocols' setup instances. Since we describe the general framework, we don't define what these parameters can be. But you can find an example of parameters for the specific construction described in Appendix A.*

Key generation. $\mathsf{KeyGen} : (1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$. *Each signer in $\mathcal{U}$ generates a random secret value* sk *and derives a corresponding public key* pk.

Policy Circuit Initialization. $\mathsf{PolicyCircuitInit} : (\mathcal{Q}, \mathcal{C}) \to \mathcal{C}_{\mathsf{BLISK}}$. *Given a set of participants' public keys $\mathcal{Q} = \{\mathsf{pk}_u : u \in \mathcal{U}\}$ and a monotone Boolean authorization circuit $\mathcal{C} = (\mathcal{V}, \mathcal{E}, r, \ell, \alpha)$, the initialization algorithm compiles a policy circuit $\mathcal{C}_{\mathsf{BLISK}} = \{(\mathsf{pk}_{v/\perp}, \mathsf{aux}_v)\}_{v\in\mathcal{V}} \leftarrow \mathsf{Compile}(\mathsf{pps}, \mathcal{C}, \mathcal{Q})$ according to compilation rules. The output $\mathcal{C}_{\mathsf{BLISK}}$ is an inactive policy circuit: the root public key pk_r as well as keys for $\vee$ gates are not determined. Appropriate keys may be derived only later through the sequential resolution and acceptance procedures.*

Policy acceptance. $\mathsf{AcceptPolicy} : (\mathsf{sk}_u, \mathcal{C}_{\mathsf{BLISK}}) \to (\mathsf{branch}_u, \pi_u)$. *Given a participant secret key sk_u and a public policy circuit $\mathcal{C}_{\mathsf{BLISK}} = \{(\mathsf{pk}_{v/\perp}, \mathsf{aux}_v)\}_{v\in\mathcal{V}}$, the policy acceptance algorithm allows participant u to activate those nodes of the policy circuit that correspond to input nodes labeled by u and their induced parent nodes. An algorithm output consists of all derived secret material and proofs required to justify participant u's contribution to the policy tree. All proofs are publicly verifiable using* $\Pi.\mathsf{Verify}$. *Policy acceptance is a sequential process as described in Section 4.1. When all required proofs are provided, we know that the circuit $\mathcal{C}_{\mathsf{BLISK}}$ is active, so pk_r is defined.*

Key Aggregation. $\mathsf{KeyAgg}_{\mathsf{BLISK}} : (\mathcal{C}_{\mathsf{BLISK}}) \to \mathsf{pk}_{\mathsf{agg}}$. *Let $\mathcal{C}_{\mathsf{BLISK}}$ be an active public policy circuit. Intuitively, each $v \in V^{\vee}_{\max}$ represents a maximal authorized branch contributing signing authority under the policy. The aggregated public key is then computed as*

$$\mathsf{pk}_{\mathsf{agg}} \leftarrow \Sigma.\mathsf{KeyAgg}(\{\mathsf{pk}_v : v \in \mathcal{V}^{\vee}_{\max}\}).$$

Only public keys corresponding to nodes in $\mathcal{V}^{\vee}_{\max}$ are included in the aggregation. No public keys from internal nodes below these disjunction gates or from unsatisfied branches contribute to $\mathsf{pk}_{\mathsf{agg}}$.

Signature generation. $\mathsf{SigGen} : (\mathsf{sk}_i, \mathsf{pk}_{\mathsf{agg}}, m) \to (\sigma_i)$. *The signature generation algorithm is literally* $\Sigma.\mathsf{SigGen}$ *run on the clauses keys $\mathcal{V}^{\vee}_{max}$.*

Signature aggregation. $\mathsf{SigAgg}(\{\sigma_i\}) \to \sigma$ *aggregates signature shares to the single signature σ of the message m under the key* pk_r

Signature verification. $\mathsf{SigVerify} : (\mathsf{pk}_r, m, \sigma) \to \{0, 1\}$. $\mathsf{SigVerify}(\cdot)$ *is a regular instance of* $\Sigma.\mathsf{Verify}$ *which responds with 1 in the case of the correctness of the signature σ up to the public key* $\mathsf{pk}_{\mathsf{agg}}$ *and a message m, and 0 otherwise.*

Key update. $\mathsf{KeyUpdate} : (\mathsf{sk}_u, \mathsf{sk}'_u, \mathcal{C}_{\mathsf{BLISK}}) \to (\mathcal{C}'_{\mathsf{BLISK}}, \pi_{\mathsf{upd}})$. *Given a participant's old secret key sk_u, a refreshed secret key sk'_u, and a public policy circuit $\mathcal{C}_{\mathsf{BLISK}}$, the key update algorithm allows participant u to update all policy circuit nodes whose secret keys depend on sk_u. The output $\mathcal{C}'_{\mathsf{BLISK}}$ consists of the updated public states $(\mathsf{pk}'_v, \mathsf{aux}'_v)$ and corresponding proofs π_{upd} for all updated nodes.*

5 Discussion and applications

BLISK's major move is to compile authorization logic into a single verification key without changing the underlying signature verification algorithm. That seemingly small design choice has significant consequences for real applications: you can retain the "verify one key, one signature" interface while shifting expressiveness and policy management to an off-chain, proof-carrying compilation workflow.

Expressiveness VS "One-Key" Verification. BLISK targets the gap between what policies look like in practice and what classic threshold signatures can express. The model is a monotone Boolean circuit over participants (AND/OR only), which is broad enough to encode many real policies (role-based approvals, mandatory participants, hierarchical sign-offs, "either-of-these plus all-of-those," etc.).

At the same time, a BLISK's real-world behavior is often dominated by the AND/OR protocol properties. For example, we show below how 1-round signature protocols can significantly reduce the practical usage of BLISK. At the same time, several features (membership dynamics, coordinator assumptions, update costs, liveness) must be considered in the process of primitives' selection.

CNF Matters. A strong structural constraint is that policies must be represented in CNF with a single $\wedge$ at the root. That's a workable normalization, but it has an important practical implication: CNF conversion can blow up policy size in the worst case. Even if many realistic policies remain manageable, a "policy compiler" that automatically rewrites arbitrary formulas into CNF needs:

- Human-friendly policy language (like S-expressions, but with additional tools like policy size estimates, warnings, etc)
- Some additional tools that allow the reduction and optimization of the initial written policy, together with the formal verification that the resulting policy corresponds to the initial one.

Proof-Carrying Compilation. A proof system choice (soundness, setup assumptions, verification cost) becomes a key engineering and trust decision. In the paper, we abstract this as Π, but deployment and corresponding costs and assumptions will depend on the selected system.

Additional Policy Logic. The next line of research can focus on improving BLISK under conditions that monotone Boolean circuits can't express. For example, combining BLISK with adaptor signatures and DL contracts, HTLC contracts, etc., can expand the range of real-world use cases for BLISK.

Bitcoin Integration. BLISK protocols integrates naturally with Bitcoin Taproot(BIP 341) [WPS21] structure

$$Q = P + [\mathcal{H}(P, m)]G, \quad \text{where}$$

- **Key Path**: The aggregated root key of the policy circuit becomes the internal key P.
  - It allows spending coins with a single signature and one public key on-chain if the policy is satisfied.
  - It hides the complexity of the policy and the specific signers involved.
- **Script Path**: Can remain as a backup or for alternative spending conditions (with hash locks which aren't supported by BLISK yet) via the Merkle tree.

Ethereum Integration. We also see several options to apply BLISK to the Ethereum infrastructure:

- **BLS Instantiation** ($\Sigma_{\mathsf{BLISK}}^{\mathsf{BLS+ART}}$):
  - Unlike Schnorr (MuSig2), BLS signatures are non-interactive (1 round).
  - It allows asynchronous signing: parties sign their resolutions independently without coordinating rounds.
  - Crucial for distributed setups (e.g., DAOs, DVT) where liveness is a challenge.
- **Gas Efficiency via Aggregation.** Using BLISK, we can resolve the policy off-chain. The EVM only verifies a single aggregated signature against a single root public key (now, on-chain multisigs (e.g., Gnosis Safe) execute complex logic inside the EVM, consuming gas for every signer and check). Thus, BLISK reduces the calldata size and execution cost to O(1), regardless of policy complexity.

- **Account Abstraction (ERC-4337).** The aggregated root key of a Policy Tree can serve as the validation authority for an ERC-4337 [But+21] compliant smart account. The validation phase of the user operation would consist of a single signature check. This structure preserves privacy by keeping the internal governance rules (the specific "AND/OR" gates and signer identities) hidden from the public ledger, revealing only that a valid authorization was produced.

6 Implementation

We have implemented our policy tree protocol instantiated with *MuSig2* as a core multi-signature protocol and ART as a tree-based CGKA construction in *Rust*[2]. In that implementation we presented *the policy compiler* for policies presented as *S-expressions*. Our compiler supports automatic conversion of an input policy to minimal *CNF* form thus it permits any policy written using AND, OR gates, for example a 3-of-4 threshold policy with the signers set $\mathcal{S} = \{A, B, C, D\}$ could be written as:

```
(policy threshold_3_of_4_circuit
  (or
      (and A B C)
      (and A B D)
      (and A C D)
      (and B C D)))
```

References

[Alw+20] Joël Alwen et al. *Asynchronous Ratcheting Trees for Secure Group Messaging.* Introduces ART, a tree-based ratcheting construction for asynchronous group communication. 2020. URL: https://eprint.iacr.org/2020/1281.

[Bar+19] Richard Barnes et al. *Tree-Based Group Key Agreement.* Early formalization of TreeKEM used in MLS. 2019. URL: https://eprint.iacr.org/2019/1088.

[Bei25] Amos Beimel. *Secret-Sharing Schemes for General Access Structures: An Introduction.* Cryptology ePrint Archive, Paper 2025/518. 2025. URL: https://eprint.iacr.org/2025/518.

[Bit09] Bitcoin Core Developers. *OP_CHECKMULTISIG — Bitcoin Script Opcode.* Bitcoin scripting opcode documentation. 2009. URL: https://developer.bitcoin.org/reference/transactions.html#op-checkmultisig.

[Bol02] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *Public Key Cryptography — PKC 2003.* Ed. by Yvo G. Desmedt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 31–46.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *Advances in Cryptology — ASIACRYPT 2001.* Ed. by Colin Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 514–532. ISBN: 978-3-540-45682-7.

[BTT22] Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. "MuSig-L: Lattice-Based Multi-signature with Single-Round Online Phase". In: *Advances in Cryptology – CRYPTO 2022.* Ed. by Yevgeniy Dodis and Thomas Shrimpton. Cham: Springer Nature Switzerland, 2022, pp. 276–305. ISBN: 978-3-031-15979-4.

[Bün+18] Benedikt Bünz et al. *Bulletproofs: Short Proofs for Confidential Transactions and More.* Non-interactive zero-knowledge proofs without trusted setup. 2018. URL: https://eprint.iacr.org/2017/1066.

[But+21] Vitalik Buterin et al. *ERC-4337: Account Abstraction Using Alt Mempool.* https://eips.ethereum.org/EIPS/eip-4337. Ethereum Improvement Proposal. Accessed: 2025-12-26. 2021.

[2] https://github.com/zero-art-rs/art-of-signature

[DR23] Sourav Das and Ling Ren. *Adaptively Secure BLS Threshold Signatures from DDH and co-CDH.* Cryptology ePrint Archive, Paper 2023/1553. 2023. URL: `https://eprint.iacr.org/2023/1553`.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. "A Pragmatic Introduction to Secure Multi-Party Computation". In: *Foundations and Trends® in Privacy and Security* 2.2-3 (2018), pp. 70–246. ISSN: 2474-1558. DOI: `10.1561/3300000019`. URL: `http://dx.doi.org/10.1561/3300000019`.

[GG19] Rosario Gennaro and Steven Goldfeder. *Fast Multiparty Threshold ECDSA with Fast Trustless Setup.* Cryptology ePrint Archive, Paper 2019/114. 2019. URL: `https://eprint.iacr.org/2019/114`.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. "Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security". In: *Applied Cryptography and Network Security.* Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Cham: Springer International Publishing, 2016, pp. 156–174. ISBN: 978-3-319-39555-5.

[Gen+99] Rosario Gennaro et al. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Journal of Cryptology* 20 (1999), pp. 51–83. URL: `https://api.semanticscholar.org/CorpusID:3331212`.

[Gen+07] Rosario Gennaro et al. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Journal of Cryptology* 20.1 (2007), pp. 51–83. DOI: `10.1007/s00145-006-0347-3`. URL: `https://doi.org/10.1007/s00145-006-0347-3`.

[Gno18] Gnosis Ltd. *Gnosis Safe: A Secure Multisignature Wallet.* Now known as Safe. 2018. URL: `https://gnosis-safe.io`.

[Hai+18] Iftach Haitner et al. *Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody.* Cryptology ePrint Archive, Paper 2018/987. 2018. DOI: `10.1145/3243734.3243788`. URL: `https://eprint.iacr.org/2018/987`.

[Hru+25] Yevhen Hrubiian et al. *0-ART. Asynchronous and Verifiable Group Management for Decentralized Applications.* Cryptology ePrint Archive, Paper 2025/1874. 2025. URL: `https://eprint.iacr.org/2025/1874`.

[kAN20] koe, Kurt M. Alonso, and Sarang Noether. *Zero to Monero: A technical guide to a private digital currency; for beginners, amateurs, and experts.* Second. Public Domain, 2020. URL: `https://www.getmonero.org/library/Zero-to-Monero-2-0-0.pdf`.

[Lin20] Yehuda Lindell. *Secure Multiparty Computation (MPC).* Cryptology ePrint Archive, Paper 2020/300. 2020. DOI: `10.1145/3387108`. URL: `https://eprint.iacr.org/2020/300`.

[NRS20] Jonas Nick, Tim Ruffing, and Yannick Seurin. *MuSig2: Simple Two-Round Schnorr Multi-Signatures.* Cryptology ePrint Archive, Paper 2020/1261. 2020. DOI: `10.1007/978-3-030-84242-0_8`. URL: `https://eprint.iacr.org/2020/1261`.

[Nic+20] Jonas Nick et al. *MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces.* Cryptology ePrint Archive, Paper 2020/1057. 2020. DOI: `10.1145/3372297.3417236`. URL: `https://eprint.iacr.org/2020/1057`.

[24] *sips/sips/sip-025/sip-025-iterating-towards-weighted-schnorr-threshold-signatures.md at main · stacksgov/sips — github.com.* `https://github.com/stacksgov/sips/blob/main/sips/sip-025/sip-025-iterating-towards-weighted-schnorr-threshold-signatures.md?utm_source=chatgpt.com`. [Accessed 08-12-2025]. 2024.

[Sip12] M. Sipser. *Introduction to the Theory of Computation.* Cengage Learning, 2012. ISBN: 9781133187790. URL: `https://books.google.com.ua/books?id=P3f6CAAAQBAJ`.

[WPS21] Pieter Wuille, Andrew Poelstra, and Yannick Seurin. *BIP-341: Taproot: SegWit Version 1 Spending Rules.* Bitcoin Improvement Proposal. 2021. URL: `https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki`.

A Example of BLISK signature in MuSig2 and 0-ART settings

A.1 Security Assumptions of Underlying Primitives

Multisignature Scheme. The MuSig2 protocol is assumed to be existentially unforgeable under chosen-message attacks (EUF-CMA) in the ROM, under the algebraic one-more discrete logarithm (AOMDL) assumption.

Informally, the AOMDL assumption states that no efficient adversary, even with access to a discrete logarithm oracle, can compute more discrete logarithms than the number of oracle queries it makes. This assumption underpins MuSig2's security against rogue-key and related attacks and ensures that aggregated signatures cannot be forged without knowledge of all required secret-key material.

Key Agreement Protocol. The security of 0-ART relies on the pseudorandom function oracle Diffie–Hellman (PRF-ODH) assumption in the random oracle model. In this setting, the shared secret is derived as $\mathsf{sk}_\vee = \mathcal{H}_{\mathsf{art}}([\mathsf{sk}_i]\mathsf{pk}_j)$, where $\mathcal{H}_{\mathsf{art}}$ is modeled as a random oracle.Any authorized participant can derive the shared secret, whereas an unauthorized participant can learn nothing about it.

PRF-ODH ensures that the derived secrets are computationally indistinguishable from the random values for any adversary that doesn't control at least one key from the group.

Proof System. This case is based on the usage of Bulletproofs as a non-interactive proof system. Bulletproofs satisfy completeness (unconditionally), computational knowledge soundness (under the assumption that the discrete logarithm problem is hard in the underlying group), and computational zero-knowledge for arithmetic relations over a finite field (under the DL assumption and the ROM). Note, Bulletproofs don't require a trusted setup and produce logarithmic-size proofs.

A.2 Example

Let's consider the exact policy: **"Alice is required AND (Bob OR Carol)"**.

We set $\Sigma_{\mathsf{BLISK}}^{\mathsf{MuSig2}[v=4],\mathsf{0\text{-}ART}[\mathsf{BP}]}$:

$$\Sigma \leftarrow \mathsf{MuSig2}[v=4][\text{NRS20}]$$
$$\mathcal{K} \leftarrow \mathsf{0\text{-}ART}[\text{Hru+25}] : \Pi \leftarrow \mathsf{Bulletproofs}[\text{Bün+18}]$$
$$\mathbb{G} \leftarrow \mathsf{secp256k1}$$
$$\mathcal{H}_{\mathsf{sig}}, \mathcal{H}_{\mathsf{art}} - \text{Schnorr challenge and ART key derivation hashes}$$

1. Setup. We run a setup for the BLISK signature instance by

$$\mathsf{pps} = \{\mathbb{G}, q, G, \mathcal{H}_{\mathsf{sig}}, \mathcal{H}_{\mathsf{art}}\} \leftarrow \Sigma_{\mathsf{BLISK}}.\mathsf{Setup}(1^\lambda)$$

2. Key Generation. Every user $x \in A, B, C$ generates a cryptographic key pair $(\mathsf{sk}_x, \mathsf{pk}_x) \leftarrow \Sigma_{\mathsf{BLISK}}.\mathsf{KeyGen}(1^\lambda)$ as:

$$\mathsf{sk}_{x\in A,B,C} \xleftarrow{\$} \mathbb{F}_q, \mathsf{pk}_x = [\mathsf{sk}_x]G$$

3. Policy circuit initialization. The coordinator creates an inactive circuit for the policy. We have the following circuit (already in CNF):

$$A \wedge (B \vee C)$$

Circuit structure:

- Input nodes:

$$v_{x\in A,B,C} \leftarrow \mathsf{pk}_x$$

- Internal nodes:
  - $v_\vee$: OR gate over v_B and v_C
  - $v_r = v_\wedge$: AND gate over v_A and $v_\vee$

Or graphically:

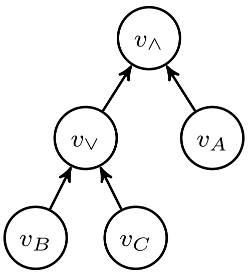

Figure 2: A tree-based schematic representation of the policy "Alice and one of Bob or Carol must sign".

For each node v the coordinator initializes a public state $\mathcal{C}_{\mathsf{BLISK}} = (\mathsf{pk}_v, \mathsf{aux}_v)$

$$\begin{gathered}\mathsf{pk}_{v_{x\in A,B,C}} \leftarrow \mathsf{pk}_x, \mathsf{aux}_v = \perp \\ (\mathsf{pk}_\perp, \mathsf{aux}_{v_\vee}) = (\mathsf{pk}_\perp, \{\mathsf{pk}_B, \mathsf{pk}_C\}) \leftarrow \text{0-}\mathsf{ART.Init}(\{\mathbb{G}, q, G, \mathcal{H}_{\mathsf{art}}\}, \{\mathsf{pk}_B, \mathsf{pk}_C\}) \\ \mathsf{pk}_\wedge \leftarrow \perp, \mathsf{aux}_\wedge \leftarrow \perp\end{gathered}$$

At this stage $\mathsf{pk}_{v_\vee}$ and $\mathsf{pk}_\wedge$ are undefined.

4. Sequential Acceptance and the Circuit Compilation. We fix a sequential order:

$$v_A \prec v_B \prec v_C \prec v_\vee \prec v_\wedge$$

Assume Bob is the party activating the $v_\vee$ gate (Carol's branch is symmetric). In this case each party provides $(\mathsf{branch}_u, \pi_u) \leftarrow \mathsf{AcceptPolicy}(\mathsf{sk}_u, \mathcal{C}_{\mathsf{BLISK}})$:

$$\begin{aligned} A :& (v_A, \sigma_{A,\mathsf{acc}}) \leftarrow \mathsf{PoK}(\mathsf{sk}_A, c := \mathcal{H}(\mathcal{C}_{\mathsf{BLISK}})) \\ B :& (v_B, v_\vee, \sigma_{B,\mathsf{acc}}, \pi_{B,\mathsf{acc}}) : \\ & \quad \sigma_{B,\mathsf{acc}} \leftarrow \mathsf{PoK}(\mathsf{sk}_B, c := \mathcal{H}(\mathcal{C}_{\mathsf{BLISK}})) \\ & \quad \pi_{\mathsf{acc}} \leftarrow \Pi.\mathsf{Prove}(\mathsf{pp}_\Pi, \mathcal{R}_{v_\vee}, \mathsf{sk}_B) \\ C :& (v_C, \sigma_{C,\mathsf{acc}}) \leftarrow \mathsf{PoK}(\mathsf{sk}_C, c := \mathcal{H}(\mathcal{C}_{\mathsf{BLISK}})) \end{aligned}$$

Note that the relation $\mathcal{R}_{v_\vee}$ in this case is:

$$\mathcal{R}_{v_\vee} = \left\{ \begin{array}{r|r} (v_B, v_C), (\mathsf{pk}_{v_\vee}, \mathsf{aux}_{v_\vee}) & \mathsf{aux}_{v_\vee} = (v_B, v_C) \\ & \mathsf{sk}_{v_\vee} = \mathcal{H}_{\mathsf{art}}([\mathsf{sk}_B]v_C) \\ \mathsf{sk}_B & \mathsf{pk}_{v_\vee} = [\mathsf{sk}_{v_\vee}]G \end{array} \right\}$$

We can count the circuit as compiled if proofs for all participants are provided.

5. Key aggregation After every user accepted the policy, the coordinator makes a key aggregation as $\mathsf{pk}_{\mathsf{agg}} \leftarrow \Sigma.\mathsf{KeyAgg}(\{\mathsf{pk}_v : v \in \mathcal{V}^\vee_{\max}\})$. Since we have only two children of the final $\wedge$ gate and following MuSig2 key aggregation procedure, the key is derived as:

$$\begin{gathered} a_{i\in L} = \mathcal{H}_{\mathsf{agg}}(L, \mathsf{pk}_i), \quad \text{where } L = \{\mathsf{pk}_A, \mathsf{pk}_\vee\} \\ \mathsf{pk}_{\mathsf{agg}} := \sum_{i\in L} [a_i]\mathsf{pk}_i = [a_A]\mathsf{pk}_A + [a_\vee]\mathsf{pk}_\vee \end{gathered}$$

6. Signature generation. Let Alice and Bob cooperate to create a signature which satisfies the policy $\mathsf{C}_{\mathsf{BLISK}}$. Having a 2-round MuSig2 protocol with $v = 4$, the signature generation process follows:
Round 1:

```
for $l = 1 \dots 4$ do
    $r_l^{(x \in A,B)} \xleftarrow{\$} \mathbb{F}_p, \quad R_l^{(x)} = [r_l^{(x)}]G$
$out^{(x)} = (R_1^{(x)}, \dots, R_4^{(x)})$
```

Round 2:

```
for $l = 1 \dots 4$ do
    $R_l = \sum_{x \in A,B} R_l^{(x)}$
$b = \mathcal{H}_{\mathsf{non}}(\mathsf{pk}_{\mathsf{agg}}, (R_1, \dots, R_4), m)$
$R = \sum_{l=1}^{4} [b^{l-1}] R_l$
$c = \mathcal{H}_{\mathsf{sig}}(\mathsf{pk}_{\mathsf{agg}}, R, m)$
$s_A = (\sum_{l=1}^{4} b^{l-1} \cdot r_l^{(A)}) + c \cdot a_A \cdot \mathsf{sk}_A$
$s_B = (\sum_{l=1}^{4} b^{l-1} \cdot r_l^{(B)}) + c \cdot a_\vee \cdot \mathcal{H}_{\mathsf{art}}([\mathsf{sk}_B]\mathsf{pk}_C)$
$s = s_A + s_B$
$\sigma = (R, s)$
```

We can see here that Alice is required here, since the knowledge of sk_A is needed to produce the signature. But at the same time, the second signature share requires the knowledge of $\mathcal{H}_{\mathsf{art}}([\mathsf{sk}_B]\mathsf{pk}_C) = \mathcal{H}_{\mathsf{art}}([\mathsf{sk}_C]\mathsf{pk}_B)$, which is accessible by Bob and Carol.

7. Signature Verification. The signature verification doesn't differ from the single verification. By running $\Sigma.\mathsf{SigVerify}(\mathsf{pk}_r, m, \sigma)$, the verifier sets the challenge

$$c = \mathcal{H}_{\mathsf{sig}}(\mathsf{tag}, \mathsf{pk}_r, R, m)$$

And checks if

$$[s]G \stackrel{?}{=} R + [c]\mathsf{pk}_r$$

8. Key Update. Let Bob rotate their secret key and make a verifiable update of the circuit $(\mathcal{C}'_{\mathsf{BLISK}}, \pi_{\mathsf{upd}}) \leftarrow \mathsf{KeyUpdate}(\mathsf{sk}_B, \mathsf{sk}'_B, \mathcal{C}_{\mathsf{BLISK}})$. First of all, Bob runs $(\mathsf{sk}'_B, \mathsf{pk}'_B) \leftarrow \Sigma_{\mathsf{BLISK}}.\mathsf{KeyGen}(1^\lambda)$ to receive new long-term key pair.

To update Bob's key, we need to update only the nodes in the circuit that depend on it – v_B, $v_\vee$, and v_r. To do it, Bob generates:

$$\sigma_{B,\mathsf{upd}} \leftarrow \mathsf{PoK}(\mathsf{sk}_B, c := \mathcal{H}(\mathsf{pk}'_B || \mathsf{C}'_{\mathsf{BLISK}}))$$
$$\pi_{\mathsf{upd}} \leftarrow \Pi.\mathsf{Prove}(\mathsf{pp}_\Pi, \mathcal{R}_{\mathsf{upd}}, \mathsf{sk}'_B)$$

The relation $\mathcal{R}_{\mathsf{upd}}$ in this case is:

$$\mathcal{R}_{\mathsf{upd}} = \left\{ \begin{array}{r|r} & v_B = [\mathsf{sk}_B]G \\ v'_B, (v_B, v_C), (\mathsf{pk}'_{v_\vee}, \mathsf{aux}'_{v_\vee}) & \mathsf{aux}'_{v_\vee} = (v'_B, v_C) \\ \mathsf{sk}_B, \mathsf{sk}'_B & \mathsf{sk}'_{v_\vee} = \mathcal{H}_{\mathsf{art}}([\mathsf{sk}'_B]v_C) \\ & \mathsf{pk}'_{v_\vee} = [\mathsf{sk}'_{v_\vee}]G \end{array} \right\}$$

If $\Sigma.\mathsf{SigVerify}(\mathsf{pk}'_B, m, \sigma_{B,\mathsf{upd}}) = 1$ and $\Pi.\mathsf{Verify}(\mathsf{vp}, \mathcal{R}_{\mathsf{upd}}, \pi_{\mathsf{upd}}) = 1$, the circuit updates with $v_B \leftarrow \mathsf{pk}'_B$, $v_\vee \leftarrow (\mathsf{pk}'_{v_\vee}, \mathsf{aux}'_{v_\vee})$ and $\mathsf{pk}_r = \mathsf{pk}_A + \mathsf{pk}'_{v_\vee}$. We see that Alice's key is still required to authorize the signature and Carol can recalculate the secret as $\mathsf{sk}_\vee \leftarrow \mathsf{Derive}(\mathsf{pp}_\mathcal{K}, \mathsf{sk}_C, \mathsf{aux}'_{v_\vee})$.