POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Attacking Schnorr based protocols with ROS: DahLIAS and Cross-Input Signature Aggregation

Relatore
Prof. Antonio José Di Scala

Candidato
Gianluca Cappiello

Anno Accademico 2024-2025

It isn't obvious that the world
had to work this way.
But somehow the universe
smiles on encryption.

[Julian Assange, Cypherpunks: Freedom and the Future of the Internet (2012)]

Contents

Introduction

Bitcoin, as the first and most widely adopted blockchain system, relies on strong cryptographic foundations to guarantee security, integrity, and decentralization. Among these, *digital signatures* play a central role: they provide spending authorization, ensure transaction authenticity, and underpin the trustless nature of the protocol. From its inception, Bitcoin employed the Elliptic Curve Digital Signature Algorithm (ECDSA), but the *Taproot* upgrade introduced a more advanced scheme: *Schnorr signatures*.

Schnorr signatures, originally introduced by Claus P. Schnorr, are celebrated for their mathematical elegance and efficiency. Compared to ECDSA, they yield shorter signatures, faster verification, and a simpler security proof. More importantly, their distinctive *linearity* property allows multiple signatures and public keys to be aggregated into a single compact signature. This aggregation significantly enhances both scalability and privacy in Bitcoin: complex multi-party transactions can be indistinguishable from simple single-signer transactions on-chain.

However, this very property also reveals a fundamental vulnerability. The so-called *ROS attack*, first discussed by Schnorr and later formalized by David Wagner, exploits linearity to break the security of naive multi-signature constructions. Understanding this attack is therefore crucial: without proper safeguards, signature aggregation protocols would be rendered insecure and unusable in practice. This challenge has motivated the design of robust multi-signature schemes such as *MuSig* and its improved variant *MuSig2*, which neutralize the ROS attack while preserving efficiency.

The research frontier has recently advanced even further with the introduction of *Cross-Input Signature Aggregation (CISA)*, a long-sought improvement for Bitcoin. While traditional signature aggregation is limited to inputs within the same script, CISA enables the aggregation of signatures across multiple transaction inputs, or even across multiple transactions in a block. This not only reduces blockchain footprint and verification costs, but also strengthens privacy by making collaborative protocols like CoinJoin more efficient and indistinguishable from regular transactions.

A milestone in this direction is the **DahLIAS** (Discrete Logarithm-Based Interactive Aggregate Signatures) protocol. DahLIAS represents the first concrete, provably secure, and Bitcoin-compatible construction enabling CISA. It combines the strengths of Schnorr signatures with interactive preprocessing and innovative aggregation techniques, producing constant-size signatures and offering faster verification compared to existing approaches. As such, DahLIAS is widely regarded as a promising candidate for future integration into Bitcoin, potentially reshaping its scalability and privacy landscape.

This thesis investigates these developments in depth. Starting from the theoretical underpinnings of Bitcoin's cryptography, it explores the transition from ECDSA to Schnorr, the challenges posed by the ROS attack, the design of secure aggregation protocols such as MuSig2, and finally the emergence of CISA and DahLIAS. The goal is to provide a comprehensive analysis that connects fundamental principles, real-world challenges, and cutting-edge cryptographic solutions in the context of Bitcoin ecosystem.

Structure

1. **Chapter 1** introduces the Bitcoin protocol, starting with the previous technologies on which it is based and then analyzing the main features that make it secure and efficient. It explores the fundamental components, including hash functions and digital signatures, and delves into the advanced cryptographic protocols that have been integrated to enhance privacy, efficiency, and scalability. It also reviews enhancements such as Taproot that extend Bitcoin's functionality.

2. **Chapter 2** presents Schnorr-based protocols, with a focus on multi-signature schemes like MuSig and MuSig2. These protocols exploit key and signature aggregation to enable efficient multi-party transactions in Bitcoin.

3. **Chapter 3** analyzes the ROS attack in detail, illustrating how it compromises naive Schnorr-based protocols, effectively compromising their security. Many attack variants have been proposed over the years, forcing researchers to develop schemes that are resistant to this type of attack.

4. **Chapter 4** explores Cross-Input Signature Aggregation (CISA) as a proposed improvement to Bitcoin, culminating in the study of DahLIAS, which allows the aggregation of signatures associated with different inputs within the transaction or block. The protocol is examined as the first practical and secure scheme to realize CISA, offering constant-size signatures, efficient verification, and strong privacy guarantees, allowing a single signature to verify public keys associated with different messages and thus positioning itself as the main candidate for possible adoption.

Chapter 1

The Bitcoin protocol

Bitcoin is a *peer-to-peer electronic cash system* introduced in 2008 by Satoshi Nakamoto in the whitepaper [Nak08]. The goal was to enable electronic payments between two parties without the need for a trusted intermediary, thereby eliminating the risk of censorship and double spending. However, the emergence of Bitcoin did not occur in isolation. Rather, it was the culmination of several decades of research in cryptography, distributed systems, and digital money. Understanding the intellectual and technical roots of Bitcoin helps to contextualize both its design choices and its disruptive nature.

1.1 Early visions of digital cash

Before Bitcoin, several attempts were made to design digital cash systems. The earliest conceptual foundations trace back to the 1980s, when David Chaum introduced the idea of *e-cash* [Cha83]. Chaum's protocols leveraged *blind signatures*, a cryptographic primitive that allows a bank to sign messages (in this case digital coins) without learning their content. In this construction, a user first "blinds" a digital coin by randomizing it with a blinding factor, obtains the bank's signature on the blinded value, and then "unblinds" it to recover a valid signature on the original coin. This mechanism allowed strong user privacy, but Chaum's design remained centralized: a trusted party (the bank) was required to issue and redeem tokens, and the system could not prevent double-spending without relying on this authority.

In 1997, Adam Back proposed *Hashcash* [Bac02], introducing the concept of *Proof-of-Work* (PoW). The idea was that users must solve computationally costly puzzles in order to gain access to a resource, thereby deterring abuse such as email spam. Although not conceived as a currency, PoW later became essential to prevent double spending and secure decentralized consensus mechanisms.

The *cypherpunk* movement of the 1990s extended these ideas, envisioning digital currencies independent of state or institutional control. Wei Dai's *b-money* (1998) [Dai98] outlined a system where participants would broadcast and verify transactions using cryptographic commitments, describing a collectively maintained ledger that anticipated blockchain-like structures. Nick Szabo's *Bit Gold* (2005) [Sza05] introduced the idea of linking proof-of-work puzzles into a chain, ensuring scarcity and chronological ordering without central authority: this was essentially a time-stamped sequence of costly-to-generate values, making each unit of currency provably scarce. Hal Finney's *Reusable Proof-of-Work* (RPoW, 2004) [Fin04] extended this notion by allowing PoW tokens to be transferred and reused, though it still relied on trusted hardware, limiting full decentralization.

In late 2008, under the pseudonym *Satoshi Nakamoto*, an unknown author released the whitepaper *Bitcoin: A Peer-to-Peer Electronic Cash System* [Nak08], introducing the first protocol to achieve a decentralized, censorship-resistant, and trustless digital currency. Bitcoin's novelty lies

in combining established cryptographic primitives (i.e. hash functions, digital signatures, and Merkle trees) with PoW-based consensus to eliminate the need for trusted intermediaries.
Satoshi's breakthrough was the synthesis of prior ideas into a working system that solved the "double-spending problem" without trusted intermediaries. Transactions were digitally signed and broadcast to a peer-to-peer network. Blocks of transactions were chained using SHA-256 hashes, while proof-of-work regulated the creation of new blocks and aligned incentives among participants. The "longest chain rule" ensured consensus: the valid chain is the one with the most accumulated proof-of-work, making it computationally infeasible for attackers to rewrite history without controlling the majority of the network's hash power.

The Bitcoin network was launched in January 2009, marking the beginning of a new era in digital finance and decentralized trust systems.

1.2 Cryptographic Primitives

At its core, Bitcoin's security and functionality rely on fundamental cryptographic primitives, such as *hash functions*, *public-key cryptography* and *digital signatures*. These components work in concert to ensure the integrity of the transaction ledger, the authenticity of transactions, and the ownership of bitcoin.

1.2.1 Hash Functions

Bitcoin extensively uses *cryptographic hash functions*, which are mathematical algorithms mapping an arbitrary-length input to fixed-size output, called *digest* or *hash*. These functions are deterministic (the same input will always yield the same output) and are designed to be computationally one-way, meaning it is infeasible to recover the input from its hash.
From a cryptographic standpoint, a secure hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ must satisfy:

- **Preimage resistance**: given y, it is infeasible to find any x such that $\mathsf{H}(x) = y$.
- **Second-preimage resistance**: given x, it is infeasible to find $x' \neq x$ such that $\mathsf{H}(x') = \mathsf{H}(x)$.
- **Collision resistance**: it is infeasible to find any pair (x_1, x_2) with $x_1 \neq x_2$ such that $\mathsf{H}(x_1) = \mathsf{H}(x_2)$.
- **Avalanche effect**: a single-bit change in the input should change roughly half the output bits in an unpredictable way.

The two primary hash functions employed in Bitcoin are:

- **SHA-256** (Secure Hash Algorithm, 256-bit output): A member of the SHA-2 family standardized by NIST, designed for high security and efficiency.
- **RIPEMD-160** (RACE Integrity Primitives Evaluation Message Digest, 160-bit output): Developed in the academic community, used in Bitcoin to shorten public key hashes while maintaining strong collision resistance.

These hash functions are central to both consensus and security mechanisms:

1. **Proof-of-Work**: SHA-256 is applied twice (`HASH256`) to block headers: miners search for a nonce that makes the hash output numerically less than the network's current target.
2. **Transaction integrity**: transaction data is hashed to create transaction IDs and to commit Merkle roots into block headers, ensuring immutability of past data.

3. **Address generation**: public keys are first hashed with SHA-256, then with RIPEMD-160, producing a compact 160-bit identifier. This reduces storage requirements, shortens addresses, and hides the full public key until spend time, limiting attack exposure.

In Bitcoin, SHA-256 is used extensively:

- Block and transaction identifiers are computed as **double SHA-256** hashes.
- Mining requires computing SHA-256 on block headers to satisfy PoW.
- Addresses and scripts use hashing for compactness and integrity.

1.2.2 Public-Key Cryptography and the secp256k1 Elliptic Curve

Public key cryptography (or asymmetric cryptography) allows the self-custody in Bitcoin: in such system, each participant possesses a *key pair*: a private key sk and a public key pk. The private key is a secret integer used to generate signatures to authorize transactions and prove ownership of funds, while the public key is deterministically derived from the private key and can be shared with the network without compromising security.

In particular, the Bitcoin protocol uses *elliptic curve cryptography* (ECC), which is based on the arithmetic of points on elliptic curves defined over finite fields.
ECC offers high security with shorter key lengths compared to other schemes (like RSA or DSA), thanks to the absence of known sub-exponential algorithms for solving the elliptic curve discrete logarithm problem (ECDLP) over general prime fields.
The specific elliptic curve used in Bitcoin is $\mathsf{secp256k1}$, defined by the equation:

$$EC: \quad y^2 \equiv x^3 + 7 \pmod{p} \tag{1.1}$$

where

$$p = 2^{256} - 2^{32} - 977$$

is a 256-bit prime number, thus defining the finite field $\mathbb{F}_p = \mathbb{Z}_p$.
The parameters $(a = 0, b = 7)$ and the base point G are chosen to enable efficient implementations, including scalar multiplication techniques using endomorphisms.
The generator point $G \in EC(\mathbb{Z}_p)$ has prime order

$$q = 115792089237316195423570985008687907852837564279074904382605163141518161494337$$

and cofactor $h = 1$, which implies that the cyclic subgroup $\mathbb{G} = \langle G \rangle$ coincides with the entire group of points on the curve $EC(\mathbb{Z}_p)$.

A user's private key is a uniformly random integer $\mathsf{sk} \in \mathbb{Z}_q^* = \mathbb{F}_q \setminus \{0\}$. The corresponding public key is computed via *scalar multiplication* on the curve:

$$\mathsf{pk} = \mathsf{sk} \cdot G$$

This operation is efficient in the forward direction (given sk and G, computes pk), but is believed to be computationally infeasible to recover the secret key sk from the public one pk due to the hardness of the ECDLP:

Definition 1.2.1 (Elliptic Curve Discrete Logarithm Problem)**.** *Given G and X with $X = x \cdot G$ for some unknown $x \in \mathbb{Z}_q^*$, determine x.*

The best known classical algorithms for the ECDLP on curves of prime order are Pollard's Rho and Baby-step Giant-step, both having complexity $O(\sqrt{q})$. For secp256k1, $\sqrt{q} \approx 2^{128}$, a value far beyond the reach of current and foreseeable classical computing power. This corresponds to an estimated *classical security level* of about 128 bits.

From a cryptographic perspective, this ensures that Bitcoin addresses derived from the public key $\mathsf{pk} = X \in EC(\mathbb{Z}_p)$ are secure against key recovery attacks, provided correct implementation and secure key generation.
The security is not absolute: a large-scale fault-tolerant quantum computer running Shor's algorithm could solve the ECDLP in polynomial time. While such technology remains purely theoretical today, the Bitcoin community is already exploring quantum-resistant approaches. These include signature schemes based on hash functions and address formats designed to minimize or delay public key exposure. In practice, Bitcoin is only susceptible to a quantum attack if the public key is revealed on-chain before the transaction is confirmed.

1.2.3 Digital Signatures

In order to spend bitcoin, a user must prove the ownership of the private key associated with the funds: this is accomplished through a **digital signature**.

The process of spending funds links addresses, public keys, and digital signatures in a single cryptographic workflow. Suppose Alice wishes to pay Bob. Bob's wallet generates a private-public key pair $(\mathsf{sk}, \mathsf{pk}) = (x, X) \in \mathbb{Z}_q^* \times \mathbb{G}$ on the secp256k1 curve and computes the HASH160 of the public key, producing an address. Alice uses this address in a transaction that locks funds to Bob's key hash. When Bob later spends these funds, he must provide both the original public key pk and a *digital signature* by signing the transaction data with the right private key sk. The signature is then included in the transaction and broadcast to the network. Nodes on the network can then use the public key to verify the signature without needing to know the private key.
Formally a *digital signature scheme* is defined as follows.

Definition 1.2.2 (**Digital Signature**). *A* digital signature *is a triple* $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *of probabilistic polynomial-time (PPT) algorithms such that:*

- *the* ***key-generation*** *algorithm* KGen *takes as input* 1^λ*, where* λ *is the security parameter, and outputs a public/private key-pair* $(\mathsf{pk}, \mathsf{sk})$*.*
- *the* ***signing*** *algorithm* Sign *takes as input a secret key* sk *and a message* $m \in M$ *(the* message space*), and outputs a* ***signature*** $\sigma \in S$ *(the* signature space*).*
  This operation is denoted as $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$*.*
- *the* ***verification*** *algorithm* Vf*, which is* deterministic*, takes as input a public key* pk*, a message* m *and a signature* σ*, and outputs a bit* $b \in \{0,1\}$*.*
  This operation is denoted as $b \leftarrow \mathsf{Vf}_{\mathsf{pk}}(m, \sigma)$*.*

Moreover, if $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$ *then*

$$\mathsf{Vf}_{\mathsf{pk}}(m, \mathsf{Sign}_{\mathsf{sk}}(m)) = 1 \quad \forall m \in M$$

except with negligible probability over the randomness of KGen*.*

Definition 1.2.3 (**The forging experiment**). *Let* $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *be a digital signature. For an adversary* $\mathcal{A}$*, the* ***signature forging experiment*** *is defined in Figure 1.1.*

The theorem formalises the notion of security: a digital signature is secure if every PPT adversary cannot generate a valid signature for a message m, even if $\mathcal{A}$ can choose the message and can obtain valid signatures for arbitrary messages $m' \neq m$.

$\mathsf{SigForge}^{\mathcal{A}}_{\mathsf{KGen},\mathsf{Sign},\mathsf{Vf}}(\lambda)$

1 : generate a key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
2 : give $\mathcal{A}$ the public key pk and access to the oracle $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$
3 : the adversary $\mathcal{A}$ outputs a message-signature pair (m, σ)
4 : **return** 1 if $\mathsf{Vf}_{\mathsf{pk}}(m, \sigma) = 1$ and $\mathcal{A}$ never queried m to the oracle, otherwise **return** 0

Figure 1.1. Signature Forging Experiment

Theorem 1.2.1 (**Secure Digital Signature**). *A digital signature* Π *is* ***existentially unforgeable under an adaptive chosen-message attack****, or* ***secure****, if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{SigForge}^{\mathcal{A}}_{\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

where the probability is over the randomness of $\mathcal{A}$ *and of the experiment* $\mathsf{SigForge}^{\mathcal{A}}_{\Pi}(\lambda)$.

Actually, it wasn't necessary for Satoshi Nakamoto to know the details of how digital signature systems work to be able to create Bitcoin. All they needed to know was that it does work, and that they could use it as the mechanism for sending and receiving money in the system they were building. In fact, the first version of Bitcoin actually used the *OpenSSL* library to provide the functionality for creating and verifying digital signatures, so it's not something they coded by hand themselves.

1.2.4 ECDSA: the first digital signature

The first digital signature algorithm employed in the Bitcoin protocol is the *Elliptic Curve Digital Signature Algorithm (ECDSA)*, instantiated over the secp256k1 elliptic curve. Conceptually, ECDSA is the elliptic curve analogue of the classical Digital Signature Algorithm (DSA), providing equivalent security with significantly shorter key lengths.

In Bitcoin, ECDSA binds the transaction authorization process to elliptic curve arithmetic: the private key sk is never revealed, yet its possession is cryptographically proven through the ability to generate a signature (R_x, s) satisfying the verification equation. Combined with address hashing, this mechanism ensures that ownership of funds can be demonstrated without exposing long-term secrets prior to the moment of spending.

Operationally, when a transaction is created, the wallet computes a cryptographic digest of the serialized transaction data. In Bitcoin, this digest is obtained via a double application of SHA-256 (i.e. HASH256), denoted as

$$h = \mathsf{H}(m) = \mathrm{SHA256}(\mathrm{SHA256}(m)),$$

where m is the transaction message. The signature is then generated over h using sk, and the resulting pair (R_x, s) is included in the transaction. Any network node can subsequently verify the signature using the public key pk, thereby confirming that the transaction was authorized by the legitimate key holder.

The protocol is formally described as follows. Let $\mathbb{G}$ be the cyclic subgroup of the elliptic curve of prime order q, generated by a base point G. Recall that a key pair is defined as $(\mathsf{sk}, \mathsf{pk}) = (x, xG)$, where $x \in \mathbb{Z}_q^*$ is the private key and $\mathsf{pk} = X \in \mathbb{G}$ is the corresponding public key.
The algorithm is shown in figure (1.2).

$\mathsf{KGen}()$	$\mathsf{Sign}_{\mathsf{sk}}(m)$	$\mathsf{Vf}_{\mathsf{pk}}(m,\sigma)$
$x \leftarrow\!\$\ \mathbb{Z}_q^*, \quad X = xG$	$x = \mathsf{sk}, \quad k \leftarrow\!\$\ \mathbb{Z}_q^*$	$(r,s) = \sigma$
$\mathsf{sk} = x; \quad \mathsf{pk} = X$	$R = kG = (x_R, y_R) \bmod q$	**assert** $(r,s) \neq (0,0)$
return $(\mathsf{sk}, \mathsf{pk})$	**assert** $r = x_R \neq 0$	$X = \mathsf{pk}, \quad h = \mathsf{H}(m)$
	$h = \mathsf{H}(m)$	$w = s^{-1} \bmod q$
	$s = k^{-1}(h + xr) \bmod q$	$(u_1, u_2) = (hw, rw) \bmod q$
	assert $s \neq 0$	$P = u_1 G + u_2 X = (x_P, y_P) \bmod q$
	return $\sigma := (r,s)$	**if** $x_P = r$ **then return** 1
		else return 0

Figure 1.2. Elliptic Curve Digital Signature Algorithm (ECDSA)

The use of `assert` enforces these non-zero conditions, preventing degenerate signatures that would be invalid or risk leaking information about the secret key x. If an assertion fails, the signing procedure restarts with a fresh nonce k.
The security of the scheme relies on the intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP) in $\mathbb{G}$ and on the unpredictability of the nonce k. Any leakage or reuse of k allows immediate recovery of the private key x.

Correctness. From $s = k^{-1}(h + rx)$, we have $ks = h + rx$.
Multiplying by G yields $s(kG) = hG + r(xG)$, i.e., $sR = hG + rX$.
Multiplying by $w = s^{-1}$ gives $R = u_1 G + u_2 X = P$, hence $x_P = x_R = r$.

Why only x-component? ECDSA uses only the x-coordinate to keep signatures compact: on `secp256k1` each coordinate is 32 bytes, so including y_R would increase the signature size from 64 bytes to 96 bytes without adding any information needed for verification.
The `secp256k1` curve (and Koblitz curves in general) has vertical symmetry: if (x, y) is a point on the curve, so is $(x, -y \bmod q)$. Thus, knowing only x_R yields two possible points:

$$R = (x_R, y_R) \quad \text{and} \quad -R = (x_R, q - y_R)$$

Both share the same x but have opposite signs in y. In ECDSA verification, the equation:

$$r \stackrel{?}{=} x_{(u_1 G + u_2 X)} \bmod q$$

depends only on x, so storing y_R in the signature is unnecessary.
This choice reduces storage requirements and improves efficiency.

Consequences: s Malleability and the "Low-s" Rule This symmetry introduces a side effect: if (r, s) is a valid signature, so is $(r, q - s)$. This happens because flipping the sign of y_R is equivalent to inverting the nonce k, which yields a complementary s.
In Bitcoin, this is a form of *malleability*: someone can take a valid signature and produce a different but still valid one. To prevent this, Bitcoin enforces the *"low-s rule"*:

$$s \leq \frac{q}{2}.$$

If s is greater, it is replaced with $q - s$ and the signature is recomputed, producing a canonical form.

The update was introduced in 2015 thanks to Bitcoin Improvement Proposal (BIP62/BIP66), i.e. an open-design document for introducing new standards and features to the protocol.

Schnorr signatures (adopted in Bitcoin with Taproot) also uses the "x-only" property, but with a different approach: it employs *x-only public keys* and fixes the y sign to a canonical value, eliminating the double-representation issue entirely. With ECDSA, the low-s rule was necessary as a corrective measure.

1.3 Protocol Design

Block Structure

A block consists of:

- **Header**: version, previous block hash, Merkle root, timestamp, difficulty bits, nonce.
- **Body**: list of transactions.

The block header is hashed twice with SHA-256:

$$\mathsf{H} = \mathrm{SHA256}(\mathrm{SHA256}(\mathrm{header})).$$

Each block header commits to:

$$\mathrm{header} = (\mathrm{version},\ \mathsf{H}(\mathrm{prev\_header}),\ \mathrm{MerkleRoot},\ \mathrm{time},\ \mathrm{nBits},\ \mathrm{nonce}),$$

and must satisfy the PoW condition $\mathsf{H}(\mathrm{header}) < T$. Because each header includes $\mathsf{H}(\mathrm{prev\_header})$, the chain is tamper-evident: altering a past block would require redoing the PoW for that block and all of its successors. Nodes consider the valid chain with the greatest cumulative work to be the ledger state.

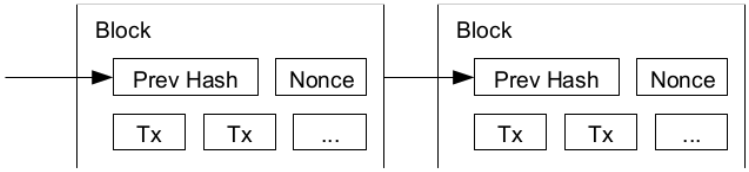

Figure 1.3. Previous header Hash [Nak08]

Transactions and UTXO Model

A bitcoin transaction is just a bunch of data that unlocks and locks up batches of bitcoins. To be more precise, a transaction:

- selects existing batches of bitcoins (*inputs*) and unlocks them;
- creates new batches of bitcoins (*outputs*) and puts new locks on them.

A single transaction can thus have multiple inputs and outputs and can be seen as a set $\mathcal{T} = \{(in_1, \ldots, in_n), (out_1, \ldots, out_m)\}$.
So a transaction can be seen as part of a chain of outputs; one transaction creates an output, and then a future transaction selects that output (as an input) and unlocks it to create new outputs.

In particular, Bitcoin follows the **Unspent Transaction Output (UTXO)** model. A transaction consumes previous UTXOs as inputs and creates new outputs. Each node has the updated *UTXO set* and can easily verify if each input in_i, $i = 1, \ldots, n$, is part of it. This means checking that the user who created the transaction does not try to spend bitcoins that have already been included in another transaction as input and therefore spent.

Scripts. Scripts are used to lock and unlock UTXOs:

- **Locking script (scriptPubKey)**: specifies spending conditions (e.g., requiring a signature for a public key).
- **Unlocking script (scriptSig)**: provides the data to satisfy the locking script.

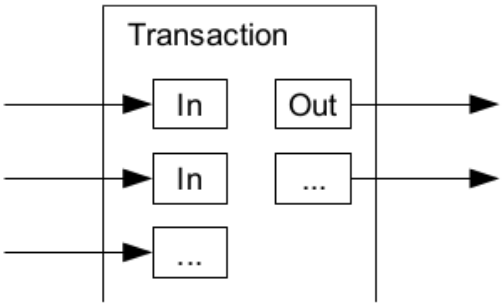

Figure 1.4. Transaction model [Nak08]

Merkle Trees

Transactions in a block are arranged into a binary **Merkle tree**. Each non-leaf node is computed as:

$$h_{parent} = \mathsf{H}(h_{left} \| h_{right}).$$

The Merkle root is then included in the block header.

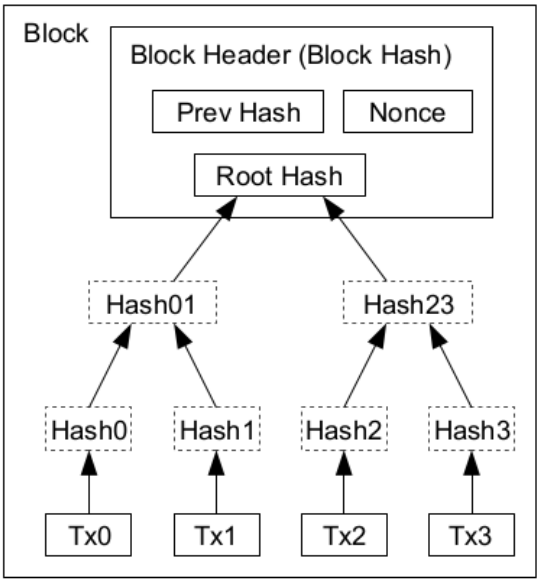

Figure 1.5. Transactions Hashed in a Merkle Tree [Nak08]

Merkle trees enable:

- **Efficient proofs**: Merkle paths allow verification of single transactions.
- **Lightweight clients**: Simplified Payment Verification (SPV) nodes need only block headers and proofs.

1.3.1 Bitcoin Address

A Bitcoin *address* is a human-readable encoding of information that allows the network to identify a destination for transferring value. Technically, an address represents data used to construct the locking script (`scriptPubKey`) that will control the spending conditions for the associated funds.

From a cryptographic standpoint, addresses are derived from elliptic curve public keys through successive applications of hash functions, not only to compress and format the data for practical use, but also to hide the underlying key until it must be revealed.
This *key-hiding* property is very important: when a user receives funds, the public key is not directly exposed on the *timechain*. Instead, only its hash, e.g. the output of a HASH160 operation (SHA-256 followed by RIPEMD-160), is made public. The actual public key is disclosed only when spending from that address, thereby narrowing the time window in which an adversary could attempt a cryptanalytic or quantum-based attack.

Esample: Pay-to-PubKey-Hash (P2PKH) A common example is the generation of a Pay-to-PubKey-Hash (P2PKH) address:

1. starts with a public key $\mathsf{pk} = X \in EC(\mathbb{Z}_p)$;
2. hash the pk obtaining a 160-bit digest $h = \text{RIPEMD160}(\text{SHA256}(X))$;
3. prepends a network version byte (e.g., 0 for mainnet);
4. appends the checksum derived from a double SHA-256 of the data (for typographical error detection);
5. everything is encoded using *Base58Check* to yield a human-readable string.

Over the years, Bitcoin has evolved beyond P2PKH:

- **P2SH (Pay-to-Script-Hash)**: starting with "3", supports multisig and complex scripts.
- **SegWit Bech32 (P2WPKH, P2WSH)**: starting with "bc1", improves efficiency and error detection.
- **Taproot (P2TR)**: Bech32 addresses leveraging Schnorr signatures and merkleized script trees.

In the context of quantum resistance, only address types that expose the raw public key (e.g. legacy P2PK) are immediately vulnerable. Modern formats keep keys hidden until spend time, mitigating exposure.

1.3.2 Proof-of-Work (PoW)

Consensus in Bitcoin is based on the **Proof-of-Work** mechanism, which makes certain actions *computationally expensive but easy to verify*. Each node (actually, the *miner*) competes to find a value (*nonce*) such that, once included in the block header, produces a hash below a difficulty threshold defined by the protocol.

The idea of *proof-of-work* was first formalized by Dwork and Naor in 1993 [DN93], and later popularized by Adam Back's *Hashcash* [Bac02]. Originally proposed as a defense against email spam and denial-of-service attacks, PoW became central to Bitcoin's consensus mechanism, ensuring that rewriting history or flooding the network carries a significant economic cost.

In Bitcoin, miners compete in the Proof-of-Work (PoW) process to add the next block to the timechain. The first miner to solve the cryptographic puzzle earns the right to append its block and is rewarded with newly minted bitcoins (the block subsidy) plus the transaction fees contained in that block. This economic incentive aligns the interest of miners with the honest operation of the protocol.

Beyond rewards, PoW also secures the network. It prevents *Sybil attacks* because creating many pseudonyms confers no advantage without controlling proportional computational power. It guarantees *immutability*, since altering a past block would require redoing the PoW for that block and all subsequent ones, overtaking the cumulative work of the honest chain. Finally, it serves as a *decentralized leader election* mechanism: the probability of mining the next block is proportional to a miner's share of the total hash rate.

To ensure stability, Bitcoin automatically adjusts the difficulty of PoW every 2016 blocks so that, on average, a new block is produced roughly every 10 minutes. This combination of economic incentives and cryptographic security allows Bitcoin to operate without any central authority, relying solely on open competition and mathematical guarantees.

Technical details. In a Hashcash-style puzzle, PoW requires finding a nonce such that

$$\mathsf{H}(x \,\|\, \text{nonce}) < T,$$

where H is a cryptographic hash function (in Bitcoin, double SHA-256) and T is the current *difficulty target*.
The success probability per independent hash trial is $p = T/2^{256}$, so the expected number of trials is $1/p$. With aggregate network hash rate $\mathcal{H}$ (hash/s), block arrivals follow a Poisson process with rate $\lambda \approx \mathcal{H} \cdot p$, yielding an expected inter-block time of $1/\lambda$. Bitcoin adjusts T every 2016 blocks to maintain ≈ 10 minutes per block by multiplying the old target by the ratio of actual to expected elapsed time, clamping changes to a factor of 4 for each re-target.

Reusable proof-of-work (RPoW). Hal Finney's *Reusable Proofs of Work* (RPoW) [Fin04] extended Hashcash by making PoW tokens transferable. Instead of being single-use, tokens could be submitted to an RPoW server, which—inside a tamper-resistant module—verified their validity and one-time use, then issued a fresh signed token. Security relied on trusted hardware and public audit logs, but the design was still not fully decentralized, as it assumed trust in a single server or a federation. RPoW demonstrated that PoW could underpin transferable digital value, anticipating Bitcoin's model. While RPoW showed *how to reuse work*, Bitcoin removed the central chokepoint by distributing validation across all miners and nodes: uniqueness and ordering are enforced collectively by the most-work chain, enabling decentralization at Internet scale.

1.3.3 Satoshi's solution to consensus and double spending

A central challenge for digital money is the *double-spending problem*: ensuring that the same coin cannot be spent twice without relying on a trusted authority. Satoshi's whitepaper [Nak08] models ownership as a *chain of digital signatures*, where each transaction authorizes the next owner:

$$\mathrm{tx}_i : \ \sigma_i = \mathsf{Sign}_{\mathsf{sk}_i}\big(\, \mathsf{H}(\mathrm{tx}_{i-1}) \,\|\, \mathsf{pk}_{i+1} \,\big).$$

This authenticates transfers, but does not by itself prevent double spending. The breakthrough was to embed all transactions into a public, append-only *timechain* (commonly called blockchain), where blocks are linked by SHA-256 hashes and secured by Proof-of-Work (PoW).

Consensus without trust. Classical Byzantine fault-tolerant protocols achieve consensus with known participants but do not scale to open, permissionless networks. Bitcoin combined (i) a public PoW-secured timestamp server with (ii) economic incentives. Nodes accept as valid the chain with the greatest cumulative work—the so-called *longest chain rule*. This guarantees **eventual consistency**: all honest nodes converge on the same ledger, and rewriting history requires redoing the PoW of the block and all successors, i.e. outpacing the entire honest network.

Security and confirmations. If an attacker controls a fraction q of global hash power (honest miners have $p = 1 - q$), the probability that the attacker *ever* catches up from z blocks behind is the gambler's-ruin probability $(q/p)^z$ for $q < p$.
The whitepaper refines this with a Poisson model of block arrivals. Let $\lambda = z \cdot q/p$ be the expected number of attacker blocks mined while the honest network finds z blocks. The probability that an attacker catches up after z confirmations is

$$\Pr[\text{catch up}] \ = \ \sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!}\, \alpha_{z-k}, \quad \alpha_{z-k} \ = \ \begin{cases} \left(\frac{q}{p}\right)^{z-k}, & k < z, \\ 1, & k \geq z, \end{cases}$$

which decays rapidly with z when $q < p$. In practice, a small number (to date, 6) of confirmations suffices to make successful double spending economically implausible.

Putting it together. Bitcoin's synthesis can be summarized as:

1. **Ownership/authentication:** a chain of digital signatures (UTXO model) authorizes spends.
2. **Global ordering:** PoW-secured blocks linked in a hash chain; Merkle trees for efficient inclusion proofs.
3. **Open consensus:** the longest-chain rule acts as a permissionless leader-election and finality heuristic; difficulty adjustment keeps average block interval stable.
4. **Economic security:** miners are rewarded for extending the valid chain, and the cost of rewriting increases with depth, making double spending infeasible without majority hash power.

1.4 SegWit and the Block Size debate

Segregated Witness (*SegWit*) is a protocol consensus upgrade activated on the Bitcoin network in August 2017, designed to fix transaction malleability and improve block capacity without directly increasing the legacy 1 MB block size limit.
The core change is that the signature data (the "witness") is separated from the transaction's main data structure, allowing it to be stored in a new data field outside the traditional block size accounting and enabling more efficient transaction formats.

Historical Context Between 2015 and 2017, the Bitcoin community engaged in a prolonged and heated debate over scaling the network. One faction advocated for larger blocks (increasing the 1 MB limit) to allow more transactions per block, while the other one warned that larger blocks would harm decentralization by raising the cost of running a full node. This ideological and technical dispute became known as the *Block Size War*.

SegWit was proposed in BIP141 as a compromise: instead of changing the block size limit directly, it redefined block capacity using a new metric called *block weight*. A block can have a maximum weight of 4.000.000 units, where witness data is discounted by a factor of four compared to other transaction data.
This effectively increases the throughput while keeping the traditional 1 MB limit for non-witness data and simultaneously fixed malleability (a prerequisite for advanced features like the *Lightning Network*) without a consensus hard fork.

UASF However, activation of SegWit required widespread miner signaling (from BIP9). By mid-2017, miner adoption had stalled due to political and economic disagreements, with large mining pools withheld support. In response, the community proposed a *User Activated Soft Fork* (UASF), formalized in the BIP148. This grassroots initiative set a clear deadline: starting on August 1, 2017, nodes running BIP148 would reject any block not signaling SegWit readiness. The threat of a chain split and the growing number of UASF-supporting nodes pressured miners to adopt SegWit signaling before the deadline.

Under this pressure, miners began signaling for SegWit in late July 2017, and the upgrade locked in shortly before the UASF activation date, avoiding a disruptive fork. SegWit became active at block height 481.824 on August 24.
The resolution of the Block Size War left a lasting cultural and political imprint in Bitcoin, strengthening the precedent that consensus changes can be initiated and enforced by the broader user base, not solely by miners or developers.

Technical Overview In the pre-SegWit format, a Bitcoin transaction is serialized as:

$$[\text{version} \mid \text{inputs} \mid \text{outputs} \mid \text{locktime}]$$

where each input contains its *scriptSig* with unlocking data (typically signatures and public keys). This unlocking data was part of the transaction ID computation, meaning that modifying a signature would change the *txid*: the so-called *transaction malleability* problem.
SegWit modifies the structure:

$$[\text{marker} \mid \text{flag} \mid \text{inputs} \mid \text{outputs} \mid \text{witness} \mid \text{locktime}]$$

where the *witness* contains the unlocking data for each input, but is excluded from the *txid* calculation (it is included only in a separate *wtxid* hash). Because the txid no longer depends on signatures, altering them does not change the txid, eliminating malleability for SegWit spends.
Block capacity is measured using *block weight*:

$$\text{weight} = \text{base size} \times 3 + \text{total size}$$

where *base size* is the transaction size without witness data. Witness bytes are thus discounted by 75% for block size accounting, effectively increasing throughput.

1.5 The Taproot Soft Fork

The *Taproot* upgrade, activated in November 2021, represents one of the most significant milestones in Bitcoin's evolution since SegWit. Its goal is to improve efficiency, privacy, and flexibility of transactions by introducing new cryptographic primitives and spending structures.
At its core lies the adoption of *Schnorr signatures*, a digital signature scheme invented by Claus-Peter Schnorr. Although technically elegant and more efficient than ECDSA, Schnorr signatures were covered by patents until 2008. This explains why Satoshi Nakamoto, when designing Bitcoin in 2008–2009, opted for ECDSA instead: it was widely used, well supported in cryptographic libraries, and free of legal restrictions. Once the Schnorr patent expired, the Bitcoin community could explore its integration, a process that culminated in the Taproot soft fork.

Schnorr Signatures (BIP 340)

The history of Schnorr signatures on Bitcoin is worth noting. The algorithm was patented by its creator, Claus Schnorr, in 1990, and the patent did not expire until 2008. This legal encumbrance meant that Schnorr's scheme was not widely adopted by standards bodies, including the U.S. National Institute of Standards and Technology (NIST), and was unavailable for use in Bitcoin's initial design.
Schnorr signatures overcome several limitations of ECDSA and provide unique properties:

- *linearity*: this is the most significant feature of Schnorr signatures. It allows multiple signers to jointly produce a single signature, making multi-signature transactions indistinguishable from single-signer transactions.
- *provable security*: the security of Schnorr signatures can be formally proven under standard cryptographic assumptions.
- *non-malleability*: Schnorr signatures are not susceptible to the same forms of signature malleability that can affect ECDSA.

The mathematical construction of a Schnorr signature is as follows:

- Choose a random nonce k and compute $R = k \cdot G$.
- Generate a challenge $c = \mathsf{H}(R, \mathsf{pk}, m)$.
- Compute the signature scalar $s = k + c \cdot \mathsf{sk} \pmod{q}$.

The signature is the pair (R, s), which can be verified by checking:

$$s \cdot G = R + c \cdot P.$$

In reality, for practical applications, it is common to use only the x-coordinate for point R, since it uniquely identifies the point once it has been agreed to use only points with even y-coordinates.

BIP 341: Taproot and Pay-to-Taproot (P2TR)

Taproot builds upon the introduction of Schnorr signatures to create a new, more private and efficient way to structure Bitcoin transactions.
It introduces a new output structure called *Pay-to-Taproot (P2TR)*, which can be spent in two different ways:

- *Key Path Spending:* the default and most efficient method, requiring only a single Schnorr signature from the owner of the public key. This ensures that single-signature and complex multi-signature transactions appear identical on-chain, greatly improving privacy.
- *Script Path Spending:* an alternative mechanism for more complex spending conditions, defined in a *Tapscript*. These conditions are encoded in a Merkle tree (*Merkleized Abstract Syntax Tree, MAST*) and only the specific script branch used for spending is revealed on the timechain.

Tapscript (BIP 342)

Tapscript is an updated version of Bitcoin's scripting language that is used in conjunction with Taproot. It includes some changes to opcodes and signature verification to work with Schnorr signatures and the P2TR structure.
Key improvements include:

- replacement of `OP_CHECKSIG` and `OP_CHECKMULTISIG` with `OP_CHECKSIGADD`, which is more efficient for batch verification of Schnorr signatures;
- greater flexibility for future extensions without requiring disruptive consensus changes.

MuSig2: Enhancing Multi-Signature Transactions

Schnorr signatures also enable advanced multi-signature protocols. The most notable is *MuSig*, which allows multiple participants to collaboratively create a single aggregated public key and so a single signature. This is a significant improvement over traditional multi-signature schemes in Bitcoin, which require multiple public keys and signatures to be included in the transaction, consuming more block space and revealing the multi-signature nature of the transaction.

The latest iteration, *MuSig2*, is a two-round protocol that simplifies interaction while retaining strong security guarantees. It is more practical than earlier versions and provides an efficient foundation for multi-party signing.

Other Emerging Protocols

Taproot is only the beginning: Schnorr signatures open the door to a new generation of cryptographic protocols, which will be analyzed in more depth in the next chapters. Some of the most promising include:

- *FROST (Flexible Round-Optimized Schnorr Threshold signatures):* an efficient threshold signature scheme designed for distributed environments.
- *Adaptor Signatures:* "conditional" signatures that link the validity of a transaction to the revelation of a secret, enabling atomic swaps and advanced payment channels.
- *Cross-Input Signature Aggregation (CISA):* a proposed technique to aggregate Schnorr signatures across multiple inputs within the same transaction, reducing data overhead even further.

These protocols, made possible by the Taproot upgrade, mark the beginning of a cryptographic evolution for Bitcoin toward greater scalability, stronger privacy, and richer functionality.

Chapter 2

Schnorr-based protocols

This chapter provides a comprehensive technical treatment of *Schnorr digital signatures*, their core cryptographic properties, and the evolution to multi-signature schemes, which have become fundamental to Bitcoin and modern cryptographic protocols.
The exposition proceeds from some mathematical preliminaries to Schnorr scheme, then rogue-key attacks, formal multi-signature construction, and finally MuSig2.

2.1 Notation and Assumptions

Let $(\mathbb{G}, +)$ be a cyclic group of prime order q with generator G and let λ be the security parameter, i.e. $\lambda = \log_2 q$. Elements of $\mathbb{G}$ are thus written in additive form; for $x \in \mathbb{Z}_q$, the notation xG denotes scalar multiplication. In this group, the DLog problem is assumed to be intractable: given $X \in \mathbb{G}$, find the *discrete logarithm* $x \in \mathbb{Z}_q$ such that $X = x \cdot G$.

Definition 2.1.1 (**Discrete Logarithm Problem**). *For a group $\mathbb{G} = \langle G \rangle$ of prime order q, the advantage of an adversary $\mathcal{A}$ in solving the discrete logarithm problem is*

$$\mathsf{Adv}^{\mathsf{DLog}}_{\mathbb{G}}(\mathcal{A}) = \Pr[X = xG : X \leftarrow\!\$\, \mathbb{G}, x \leftarrow \mathcal{A}(X)]$$

where the probability is taken over the random choices of $\mathcal{A}$ and the random selection of X.
An adversary $\mathcal{A}$ is said to (τ, ϵ)-break DLog if it runs in time at most τ and $\mathsf{Adv}^{\mathsf{DLog}}_{\mathbb{G}}(\mathcal{A}) \geq \epsilon$.
Discrete log problem is (τ, ϵ)-hard if no such adversary exists.

A stronger assumption underlying Schnorr-type protocols is the *One-More Discrete Logarithm* problem (OMDL) [BNPS03].

Definition 2.1.2 (**n-One-More Discrete Logarithm Problem**). *Let $\mathbb{G} = \langle G \rangle$ be a cyclic group of prime order q, and let $\mathcal{O}$ be an oracle for DLog in $\mathbb{G}$. An adversary $\mathcal{A}$ is given oracle access and may query at most n elements $X_1, \ldots, X_n \in \mathbb{G}$, obtaining the discrete logarithm $x_i \in \mathbb{Z}_q$ with $X_i = x_i G$.*
The n-OMDL game is solved if $\mathcal{A}$ outputs an additional pair (X^*, x^*) with $X^* = x^* G$ and $X^* \notin \{X_1, \ldots, X_n\}$. The advantage of $\mathcal{A}$ in solving n-OMDL is defined as

$$\mathsf{Adv}^{\mathsf{OMDL}}_{\mathbb{G}}(\mathcal{A}) \;=\; \Pr[X^* = x^* G : x^* \leftarrow \mathcal{A}(X^*), X^* \leftarrow\!\$\, \mathbb{G}, X^* \notin \{X_1, \ldots, X_n\}].$$

The problem is (τ, ϵ)-hard if no adversary running in time at most τ achieves $\mathsf{Adv}^{\mathsf{OMDL}}_{\mathbb{G}}(\mathcal{A}) \geq \epsilon$.

In other words, the goal is to solve one additional instance of the discrete logarithm problem beyond those already solved by the oracle, leveraging the information obtained from previous queries.

2.1.1 Hash Functions and the Random Oracle Model

The security of the Schnorr signature scheme is tipically proven in the **Random Oracle Model (ROM)**, an idealized construct where the hash function H is treated as a random function that an entity can query to get random responses, but which is consistent for identical queries. Although it is an idealized model, the ROM is a standard tool for the initial analysis of many cryptographic schemes.

The Random Oracle Model (ROM). Let $\mathbb{G} = \langle G \rangle$ be a cyclic group of prime order q with generator G and let $\mathbb{Z}_q$ denote the finite field of integers modulo q. Let H be an ideal hash function with range $\mathbb{Z}_q$ modeled as a *random oracle*: an idealized function that assigns independent, uniformly random outputs to new queries and is consistent across repeated queries. Formally, H is a random function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$ chosen at random over all maps of that type with uniform probability distribution.

Tagged Hashes. When the same hash primitive is used in different logical roles, *domain separation* must be enforced to prevent cross-protocol collisions. Following BIP340, given a tag string tag, the tagged hash $\mathsf{H}_{\mathsf{tag}}$ is defined as

$$\mathsf{H}_{\mathsf{tag}}(m) = \mathsf{H}\big(\mathsf{H}(\mathsf{tag}) \parallel \mathsf{H}(\mathsf{tag}) \parallel m\big).$$

This construction ensures that for different tags tag, the corresponding $\mathsf{H}_{\mathsf{tag}}$ behave as independent random oracles in the ROM.

In Schnorr-based multi-signature schemes, the following tagged hashes are typically instantiated:

$$\begin{aligned} \mathsf{H}_{\mathsf{agg}} &: \text{for key aggregation coefficients,} \\ \mathsf{H}_{\mathsf{nonce}} &: \text{for nonce binding,} \\ \mathsf{H}_{\mathsf{sig}} &: \text{for signature challenge computation.} \end{aligned}$$

2.2 The Schnorr Protocol

The Schnorr signature scheme [Sch91] is known for its elegance and simplicity, and it was one of the first schemes whose security was formally proven assuming the difficulty of the discrete logarithm problem.

2.2.1 Schnorr Identification as a Σ-protocol

For a public key $X = xG$, the Schnorr identification scheme is a 3-move Σ-protocol with *honest-verifier zero-knowledge*, *special soundness*, and *knowledge soundness*.

Special Soundness. From two valid transcripts with the same R and distinct $c \neq c'$, responses s, s' reveal

$$x \;=\; \frac{s - s'}{c - c'} \pmod{q}.$$

This extraction underpins *knowledge soundness*.

HVZK Simulation. *Honest-verifier zero-knowledge* follows from the ability to simulate (R, c, s) by sampling $c, s \leftarrow\!\$\, \mathbb{Z}_q$ and computing $R = sG - cX$.

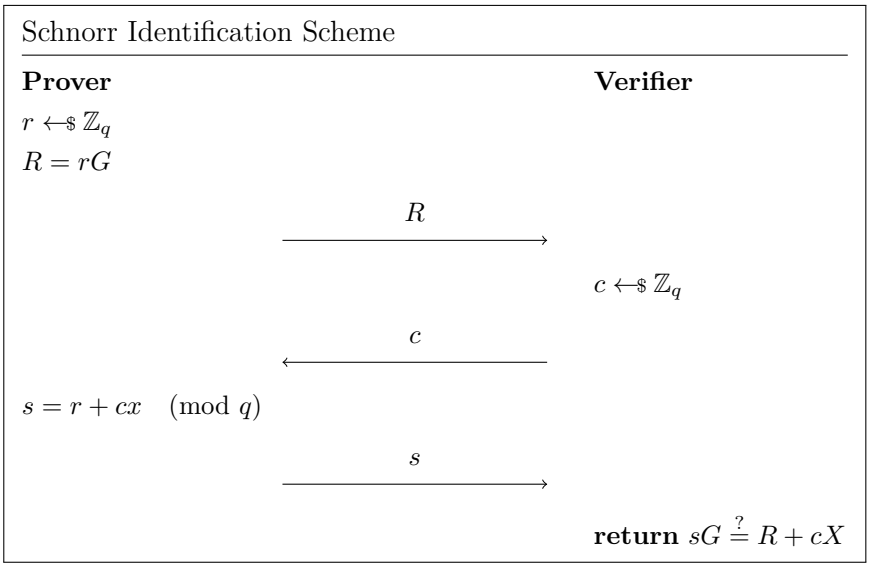

Figure 2.1. Schnorr Identification Scheme

2.2.2 Schnorr Signatures via Fiat–Shamir

The **Schnorr digital signature** is obtained by applying the *Fiat–Shamir transform* to the identification scheme with a hash function modeled as a random oracle $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$.
Let the secret key be $x \in \mathbb{Z}_q$ and the public key $X = xG$.

Key Generation (KGen) Each signer chooses a private key $x \leftarrow\!\!\$\ \mathbb{Z}_q$ randomly. The corresponding public key X is given by $X = x \cdot G$, where the operation $\cdot$ is scalar multiplication in the group $\mathbb{G}$.

Signing (Sign) To sign a message m, the signer performs the following steps:

1. choose a random nonce $r \leftarrow\!\!\$\ \mathbb{Z}_q$. The value r must be secret and used only once, in fact if a nonce is reused for two different messages, the private key can be recovered;
2. calculate the public nonce point $R = k \cdot G$;
3. calculate the challenge $c = \mathsf{H}(R \parallel X \parallel m)$, where $\parallel$ denotes concatenation;
4. calculate the signature scalar $s = r + c \cdot x$.

The final signature is the pair $\sigma := (R, s)$.

Verification (Vf) To verify a signature $\sigma = (R, s)$ on a message m with the public key X, the verifier recalculates the challenge $c' = \mathsf{H}(R \parallel X \parallel m)$ and checks if

$$s \cdot G \;=\; R + c' \cdot X.$$

The correctness of this equation is demonstrated by its linear structure.

Correctness. If $\sigma = (R, s)$ is computed honestly, then

$$s \cdot G = (r + c \cdot x) \cdot G = r \cdot G + c \cdot (x \cdot G) = R + c \cdot X.$$

If the signed message matches the verified message, the equation solves correctly, confirming the signature's validity.

$\mathsf{KGen}(1^\lambda)$	$\mathsf{Sign}_{\mathsf{sk}}(m)$	$\mathsf{Vf}_{\mathsf{pk}}(m,\sigma)$
$x \leftarrow\!\$\ \mathbb{Z}_q$; $\quad X = xG$	$r \leftarrow\!\$\ \mathbb{Z}_q$	$(R,s) = \sigma$
$\mathsf{sk} = x$; $\quad \mathsf{pk} = X$	$R = rG$	$c \leftarrow \mathsf{H}(X,R,m)$
return $(\mathsf{sk},\mathsf{pk})$	$c \leftarrow \mathsf{H}(X,R,m) \in \mathbb{Z}_q$	**return** $sG \stackrel{?}{=} R + cX$
	$s \leftarrow r + cx \pmod q$	
	return $\sigma := (R,s)$	

Figure 2.2. Schnorr Digital Signature Scheme

2.2.3 Security and Core Properties

EUF-CMA security. In the random-oracle model (ROM), Schnorr signatures are *existentially unforgeable under chosen-message attacks* (**EUF-CMA**), assuming the hardness of DLog. The reduction relies on programming the oracle H and applying special soundness on two forgeries with the same commitment R but distinct challenges, thereby extracting x. Tightness depends on the number of oracle queries.

Uniqueness and malleability. Given (R,s) and the verification equation, s is uniquely determined by (R,X,m) once H is fixed. However, if a curve encoding admits both R and $-R$ as distinct points, signatures (R,s) and $(-R,-s)$ satisfy different challenges $\mathsf{H}(R,\cdot)$ and $\mathsf{H}(-R,\cdot)$. Implementations such as BIP340 enforce a canonical encoding (e.g., *even-y* for R) to ensure uniqueness of valid encodings and eliminate trivial malleability at the representation level.

Nonce issues. Nonce r must be uniform and secret; deterministic generation with auxiliary randomness is recommended to avoid RNG failures and side channels (domain-separated hashing of (x,X,m,aux)). Nonce reuse across messages breaks security: if (R,s_1) and (R,s_2) are signatures on m_1,m_2, then

$$x \;=\; \frac{s_1 - s_2}{c_1 - c_2} \pmod q,$$

with $c_i = \mathsf{H}(R,X,m_i)$. Even partial nonce leakage or bias can enable key recovery.

Linearity. Schnorr enjoys linear relations:

$$(r_1 + r_2)G = R_1 + R_2, \qquad (s_1 + s_2)G = (R_1 + R_2) + c(X_1 + X_2),$$

which enables multi-/threshold signing with a single compact signature when the challenge c binds all relevant inputs. Linearity is a feature but also a source of pitfalls (rogue keys, nonce-cancelation), addressed below.

Batch verification. For signatures (R_i,s_i) on messages m_i under X_i, pick independent $\alpha_i \leftarrow\!\$\ \mathbb{Z}_q^*$ and check

$$\Big(\sum_i \alpha_i s_i\Big)G \stackrel{?}{=} \sum_i \alpha_i R_i \;+\; \sum_i \alpha_i\, \mathsf{H}(R_i,X_i,m_i)\, X_i.$$

Random coefficients prevent trivial linear-algebra attacks that can forge a passing batch without forging any individual signature.

Key tweaks. Public keys are tweakable linearly: for $t \in \mathbb{Z}_q$, define $X' = X + tG$ and $x' = x + t \pmod{q}$, preserving correctness since

$$sG \;=\; R + cX \iff sG \;=\; R + c(X' + (-t)G).$$

This property underlies Taproot key-path spending, script commitments, and adaptor signatures.

2.3 Multi-Signatures Schemes

Multi-signature schemes extend standard digital signatures by allowing n parties, each with a secret key $x_i \in \mathbb{Z}_q$ and public key $X_i = x_i G \in \mathbb{G}$, to jointly produce a single compact signature σ on a common message m.

Syntax A multi-signature scheme MS for n parties is defined as a tuple of PPT algorithms

$$\mathsf{MS} = (\mathsf{KGen}, \mathsf{AggKey}, \mathsf{Sign}, \mathsf{AggSig}, \mathsf{Vf})$$

with the following specifications:

- $\mathsf{KGen}(1^\lambda) \to (x_i, X_i)$: on input the security parameter $\lambda = \lfloor \log_2 q \rfloor$, each signer i outputs a secret key $x_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and the corresponding public key $X_i = x_i G$.
- $\mathsf{AggKey}(X_1, \ldots, X_n) \to X_{\mathsf{agg}}$: given public keys of all signers, the algorithm outputs an aggregate public key X_{agg} that will be shared by all the n party.
- $\mathsf{Sign}(x_i, L, m) \to \sigma_i$: each signer, using the secret key x_i, produces a *partial signature* σ_i for the message $m \in \{0,1\}^*$.
- $\mathsf{AggSig}(\sigma_1, \ldots, \sigma_n) \to \sigma$: the partial signatures are aggregated into a single signature σ.
- $\mathsf{Vf}_{X_{\mathsf{agg}}}(m, \sigma) \in \{0,1\}$: on input the aggregate key, the message, and the (multi-)signature, the verification algorithm accepts iff the signature is valid.

Correctness. A multi-signature scheme is **correct** if for all key pairs $(x_i, X_i) \in \mathbb{Z}_q \times \mathbb{G}$ generated honestly, and any message $m \in \{0,1\}^*$,

$$\mathsf{Vf}_{X_{\mathsf{agg}}}(m, \mathsf{AggSig}(\sigma_1, \ldots, \sigma_n)) = 1$$

whenever $\sigma_i \leftarrow \mathsf{Sign}_{x_i}(m)$ are valid partial signatures.

Additional Properties. Beyond correctness and EUF-CMA security, practical multi-signature schemes must satisfy the following properties:

- **Compactness:** the size of the final signature is the same as a Schnorr signature for a single user, regardless of n, i.e. $|\sigma| = O(1)$.
- **Rogue-Key Resistance:** naive key aggregation, where the aggregate key is defined by simple addition, is insecure, since it enables the classical rogue-key attack (see 2.3.2).
- **Round Efficiency:** the protocol must minimize communication rounds (MuSig2 achieves two rounds).
- **Transparency:** multi-signatures should be indistinguishable from ordinary signatures (ideal for applications like Bitcoin scriptless scripts).
- **Accountability:** every honest signer can verify that its contribution was included, ensuring no party can produce a valid signature without active participation of the required subset.

Nonce pitfalls. Beyond keys, nonces can also be abused (e.g., *nonce-cancelation* or *ROS attack*) if aggregated without binding. For example, if two signers publish nonce points R_1, R_2 such that $R_1 = -R_2$, the aggregate $R = R_1 + R_2$ vanishes, leading to undefined challenges $c = \mathsf{H}(R, \cdot)$. Modern schemes derive nonce coefficients from all nonces, the aggregate key, and the message, and use commit–reveal rounds to eliminate adaptivity.

Threshold Signatures. It is useful to contrast multi-signatures with *threshold signatures*. In multi-signatures, the set of participants is known, and all n must cooperate to sign. In threshold schemes, any subset of size t out of n can produce a valid signature. Security definitions are similar, but threshold schemes additionally require robustness against *signer dropout*.

2.3.1 Security Model

The standard security notion is **existential unforgeability under chosen-message attacks** (EUF-CMA), adapted to the multi-signer setting. An adversary $\mathcal{A}$ interacts with to two oracles:

- a *signing oracle* that on input a subset of signers $L \subseteq \{1, \ldots, n\}$ and a message m, the oracle runs the signing protocol and outputs a valid signature σ.
- a *corruption oracle*, where the adversary can corrupt some signers to obtain their secret keys x_i.

The unforgeability states that no PPT adversary $\mathcal{A}$ can produce σ on m under X_{agg} unless at least one corrupted party controls its secret key.

Definition 2.3.1 ($\mathsf{MS}-\mathsf{EUF}-\mathsf{CMA}$)**.** *A multi-signature scheme* MSig *is* $\mathsf{EUF}-\mathsf{CMA}$ *secure if for any PPT adversary* $\mathcal{A}$ *with access to the signing and corruption oracles, the probability that* $\mathcal{A}$ *outputs a forgery*

$$(m^*, \sigma^*, X^*_{\mathsf{agg}})$$

such that:

1. $\mathsf{Vf}_{X^*_{\mathsf{agg}}}(m^*, \sigma^*) = 1$,
2. *at least one uncorrupted signer* i *is included in* X^*_{agg},
3. $(m^*, X^*_{\mathsf{agg}})$ *was never queried to the signing oracle with the same set of participants,*

is negligible in the security parameter λ.

2.3.2 Rogue-Key Attack

A direct and intuitive extension of the Schnorr signature scheme to the multi-signature context is the *linear sum* of keys and nonces. The *rogue-key attack* exploits the intrinsic linearity of the aggregation protocol to allow an attacker to forge a joint signature without the cooperation of all participants. If public keys are aggregated naively as

$$X_{\mathsf{agg}} \;=\; \sum_{i=1}^{n} X_i \;=\; \Big(\sum_{i=1}^{n} x_i\Big)G,$$

then a malicious party can bias the resulting public key towards a value for which it knows the corresponding secret key (i.e. the discrete logarithm), thus compromising the security of the scheme enabling unilateral signing.

The Naive Construction. The naive multi-signature protocol construction unfolds as follows:

- $\mathsf{KGen}(1^\lambda) \to (x_i, X_i) \in \mathbb{Z}_q \times \mathbb{G}$
- $\mathsf{AggKey}(X_1, \ldots, X_n) \to X_{\mathsf{agg}} = \sum_{i=1}^{n} X_i$
- $\mathsf{Sign}_{x_i}(m) \to \sigma_i := (R_i, s_i)$, where the common challenge is $c = \mathsf{H}(R_{\mathsf{agg}}, X_{\mathsf{agg}}, m)$

- $\mathsf{AggSig}(\sigma_1, \ldots, \sigma_n) \to \sigma = (R_{\mathsf{agg}}, s_{\mathsf{agg}}) = (\sum_{i=1}^{n} R_i, \sum_{i=1}^{n} s_i)$
- $\mathsf{Vf}_{X_{\mathsf{agg}}}(m, \sigma) = 1 \iff s_{\mathsf{agg}} \cdot G = R_{\mathsf{agg}} + c \cdot X_{\mathsf{agg}}$

This seemingly harmless construction proves to be severely insecure because of a malicious interaction among participants.

Attack Description. Consider a scenario with $n = 2$ signers. The honest party has key pair (x_1, X_1) with $X_1 = x_1 G$. The adversary $\mathcal{A}$, instead of generating its key honestly, chooses $X_2 = X^* - X_1$, for some arbitrary target key $X^* \in \mathbb{G}$ of which the adversary knows the discrete logarithm x^*. When the public keys are naively aggregated as

$$X_{\mathsf{agg}} \;=\; X_1 + X_2 \;=\; X^*,$$

the resulting aggregate key is the one corresponding to the adversary's known secret x^*, allowing $\mathcal{A}$ to produce valid signatures under any desired X^* without knowledge of the honest secret key x_1.
The attack can be trivially generalized to the case with $n > 2$ users, where the malicious signer chooses $X_j = X^* - \sum_{i \neq j} X_i$.

$$X_j = X^* - \sum_{i \neq j} X_i$$

This attack shows that naive key aggregation leads to *complete forgery*: one party can unilaterally control the aggregate key and sign on behalf of the group without detection. Therefore, any secure multisignature scheme must incorporate a mechanism to prevent rogue-key attacks.

Knowledge of Secret Key (KOSK). The traditional defense against the rogue-key attack is to require a "Proof of Knowledge of Secret Key" (KOSK), where each signer must prove the knowledge of the private key corresponding to their public key before participating in the protocol, usually through a zero-knowledge proof. However, implementing KOSK is often complex and inapplicable in many contexts, where there is no trusted entity to provide such proof to. Therefore, the rogue-key attack is not a minor flaw but an intrinsic and catastrophic defect of any naive aggregation scheme, making the PPK model a fundamental design condition for any practical multi-signature protocol. Subsequent research has focused on solving this problem within the PPK model, without resorting to unrealistic assumptions about key knowledge.

Proof of Possession (POP). This approach is more pragmatic. A POP attests that a party has access to the private key associated with its public key, in general via a simple signature on the certificate request. Intuition suggests that POPs should be sufficient to stop rogue-key attacks, but initial analyses showed this was not the case for some schemes, which remained vulnerable despite the use of POPs. This observation underscores the need for formal security proofs rather than empirical or intuitive reliance. The evolution of these schemes required the use of modified POPs with separate hash functions, which provided the first formal guarantees that POP-based protocols could be used securely in practice.

Coefficient binding. The standard defense is to introduce *key-prefixing* or *coefficient binding*: instead of computing a simple sum of public keys, each public key is *weighted* by a coefficient that depends on all participants' public keys. Given a list of public keys $L = \{X_1, \ldots, X_n\}$, the aggregated key is

$$X_{\mathsf{agg}} \;=\; \sum_{i=1}^{n} a_i X_i, \qquad \text{where} \quad a_i \;=\; \mathsf{H}_{\mathsf{agg}}(L, X_i).$$

The aggregate private scalar (if all participants are honest and reveal no secrets) would be

$$x_{\mathsf{agg}} \;=\; \sum_{i=1}^{n} a_i x_i \mod q,$$

since $X_{\mathsf{agg}} = (\sum_i a_i x_i)G$. In this way, each coefficient a_i is linked to the entire set L of public keys. In particular, the coefficient a_j for the adversary's key depends on the same X_j chosen by the adversary, and so do the coefficients for the honest parties. Therefore, the adversary cannot algebraically cancel out the honest contributions by choosing X_j, since changing X_j modifies the coefficients in a cryptographically unpredictable way.

Security. Suppose a malicious party chooses X_j after seeing honest $\{X_i\}_{i\neq j}$. Without coefficients, naive $X_{\mathsf{agg}} = \sum_i X_i$ allows setting $X_j := X^* - \sum_{i\neq j} X_i$, yielding $X_{\mathsf{agg}} = X^*$ for a public key X^* whose discrete log is known to the adversary. With coefficients,

$$X_{\mathsf{agg}} \;=\; \sum_{i=1}^{n} a_i X_i = \Big(\sum_{i=1}^{n} a_i x_i\Big)G.$$

An adversary who wants the aggregate to equal $X^* = x^*G$ would need to choose x_j and thus X_j such that

$$\sum_{i=1}^{n} a_i x_i \equiv x^* \mod q. \tag{2.1}$$

However all a_i are functions of the full list L, so a_j is a function of X_j, which in turn is determined by x_j. Thus the equation to solve turns into a nonlinear equation in the unknown x_j. Given the hash used to produce the a_i's, solving this equation by algebraic manipulation is infeasible. In practice, the adversary would have to either invert the hash (computationally infeasible) or perform enormous brute force.

Summary. Without coefficients, aggregation is linear and an attacker who chooses a public key after seeing honest keys can set her key so that the aggregated key equals any target she knows the discrete log for. This is the rogue-key attack. Coefficient binding ties every participant's contribution to the entire key list L. Coefficients are computed by a cryptographic hash over L and each individual public key: changing one public key changes the coefficients unpredictably. As a result, the simple algebraic cancellation used in the rogue-key attack becomes a nonlinear problem that requires inverting or breaking the hash, that is computationally infeasible with secure hash functions and secp256k1-sized parameters. In a real implementation (for example MuSig2 or similar schemes), the coefficients are computed from the encoded EC points (not only scalars), and all encodings, ordering rules and hash domain separation must be carefully specified to avoid subtle attacks.

2.3.3 The Bellare-Neven Scheme

The work of Bellare and Neven [BN06] introduced one of the first rigorous treatments in the *plain public-key model*, where no trusted setup, certification authority, or PKI assumptions are made beyond each user publishing a public key. This stands in contrast to earlier schemes that either relied on a common reference string or stronger trust assumptions. Their scheme, $\mathsf{MS}-\mathsf{BN}$, demonstrated that it is possible to build a secure protocol that prevents *rogue key attacks* without requiring complex proofs of key knowledge, laying the groundwork for all future developments in this field.

The Plain Public-Key (PPK) Model. The protocol operates in the Plain Public-Key (PPK) model, where it's assumed that a signer only has a public key, without having to prove they possess the corresponding private key. This is a realistic and desirable model for decentralized systems, where there is no central authority for key registration or identity validation.

MS-BN. The core of Bellare and Neven's solution to the rogue-key attack lies in modifying how the challenge is calculated. Instead of using a single common challenge $c = \mathsf{H}(R, L, m)$, the $\mathsf{MS}-\mathsf{BN}$ protocol requires each signer i to calculate a *distinct* challenge that uniquely incorporates their own public key. One implementation of this principle is described in the scheme, where each signer calculates their partial signature on a challenge c_i that depends on their own key, i.e. $c_i = \mathsf{H}(X_i, R, L, m)$.

- KGen: generate random $x \leftarrow\!\!\$\ \mathbb{Z}_q$ and $X = xG$.
- $\mathsf{Sign}(x_i, L, m)$: each signer $i \in [n]$, performs these signing rounds:
  1. choose $r_i \leftarrow\!\!\$\ \mathbb{Z}_q$, compute $R_i = r_i G$ and send the *nonce commitment* $t_i := \mathsf{H}_{\mathsf{nonce}}(R_i)$ to other signers;
  2. receive t_j, $j \neq i$, and reveal the nonce point R_i;
  3. once received R_j, if $t_j = \mathsf{H}_{\mathsf{nonce}}(R_j)$, for all $j \neq i$, then abort the protocol; otherwise, compute $R = \sum_i R_i$, $c_i := \mathsf{H}(X_i, R, L, m)$ and send partial signature $s_i = r_i + c_i x_i \pmod q$;
  4. finally, compute $s = \sum_i s_i \bmod q$ and output the signature $\sigma = (R, s)$.
- $\mathsf{Vf}(L, m, \sigma)$: compute $c_i = \mathsf{H}(X_i, R, L, m)$ for all $i \in [n]$ and accept the signature if $sG = R + \sum_i c_i X_i$

This modification has a crucial impact: it prevents the attacker from pre-calculating a rogue key to nullify the contribution of an honest signer. Since each signer's coefficient is tied to their own public key, the attacker cannot manipulate their own key to influence another signer's coefficient. The simple and vulnerable linearity of the naive protocol is effectively broken.

Limitations. Despite its importance, the $\mathsf{MS}-\mathsf{BN}$ has a significant limitation: it does not support key and signature aggregation. For verification, a verifier must have access to *all* the individual public keys of the signers. Consequently, the size of the final signature and the computational complexity of the verification process $O(n)$ scale linearly with the number of signers.

The Bellare and Neven solution solved the security problem but introduced a new trade-off: inefficiency. By tying each partial signature to a distinct public key, they prevented the rogue-key attack vulnerability, but they also eliminated the possibility of benefiting from the linearity that allows for aggregation. This trade-off between security and efficiency created the need for a new wave of research, which led directly to the development of MuSig, with the goal of regaining the lost efficiency. The evolution of these protocols is a continuous quest to find the optimal point between security, compactness, and efficiency.

2.3.4 MuSig

While the $\mathsf{MS}-\mathsf{BN}$ scheme provided the first secure construction of multi-signatures in the plain public-key model, it remained primarily of theoretical interest due to its inefficiency in practice: it required multiple rounds of communication and was not optimized for deployment in systems like Bitcoin.

The **MuSig** protocol [MPSW19] was one of the first practical and provably secure solutions for the multi-signature problem with *key aggregation*, directly addressing the rogue-key attack in an aggregation context. Its security relies on a three-round interactive process, which makes it relatively cumbersome for real-world applications, especially in environments with network latency or high communication costs. This trade-off between security and practicality prompted researchers to seek more efficient solutions.

The scheme The MuSig protocol is a 3-rounds Schnorr-based multi-signature scheme, which can be seen as a variant of the $\mathsf{MS}-\mathsf{BN}$ allowing key aggregation in the plain public-key model. Since it is strictly based on the Bellare and Neven's work, they changed the way the challenges c_i are computed from $c_i = \mathsf{H}(L, X_i, R, m)$ to

$$c_i = \mathsf{H}_{\mathsf{agg}}(L, X_i) \cdot \mathsf{H}_{\mathsf{sig}}(X_{\mathsf{agg}}, R, m)$$

where X_{agg} is the so-called *aggregated public key* corresponding to the multiset of public keys $L = \{X_1, \ldots, X_n\}$, defined as

$$X_{\mathsf{agg}} \;=\; \sum_{i=1}^{n} a_i X_i$$

where $a_i := \mathsf{H}_{\mathsf{agg}}(L, X_i)$ only depends on the public keys of the signers. This way, the verification equation of a signature $\sigma = (R, s)$ on message m for public keys $L = \{X_1, \ldots, X_n\}$ becomes

$$sG \;=\; R + \sum_{i=1}^{n} a_i \cdot c \cdot X_i \;=\; R + c \cdot X_{\mathsf{agg}},$$

where $c := \mathsf{H}_{\mathsf{sig}}(X_{\mathsf{agg}}, R, m)$.
In other words, we have recovered the key aggregation property enjoyed by the naive scheme, albeit with respect to a more complex aggregated key $X_{\mathsf{agg}} = \sum_{i=1}^{n} a_i X_i$. Note that using the old $c = \mathsf{H}_{\mathsf{sig}}(L, R, m)$ yields a secure scheme as well, but does not allow key aggregation since verification is impossible knowing all the individual signer keys.

The Key Aggregation Function

MuSig solves the rogue-key attack while maintaining linearity. Instead of simply summing the public keys, the protocol aggregates them in a "weighted" manner. The aggregated public key X_{agg} is defined as a sum of individual keys, each multiplied by a random and unpredictable coefficient that depends on *all* the participants' keys.
Let $L = (X_1, \ldots, X_n)$ denote an ordered list of public keys. Each key is bound to a coefficient that incorporates both L and the individual key X_i:

$$a_i = \mathsf{H}_{\mathsf{agg}}(L, X_i),$$

where $\mathsf{H}_{\mathsf{agg}}$ is a cryptographic hash function modeled as a random oracle. The aggregate key is

$$X_{\mathsf{agg}} = \sum_{i=1}^{n} a_i X_i.$$

This ensure rogue-key resistance 2.3.2.

The signing protocol

The MuSig signing process is a 3-round interactive protocol.

Nonce Commitment. Each signer $i \in [n]$ samples a random nonce $r_i \in \mathbb{Z}_q$, computes the corresponding point $R_i = r_i G$, and broadcasts the commitment $t_i := \mathsf{H}(R_i)$. This prevents nonce manipulation attacks, but actually require an extra round.

Nonce Revelation. Each signer i receives commitments t_j, $j \neq i$, and sends back the nonce point R_i.

Partial Signatures. Once received the points R_j, $j \neq i$, he can check their validity by hashing and then comparing them with the commitments t_j. If there exists $j^* \in [n]$ such that $t_{j^*} \neq \mathsf{H}(R_{j^*})$ then abort the protocol; otherwise, compute the aggregate nonce

$$R = \sum_i R_i.$$

The challenge is then defined as

$$c = \mathsf{H}_{\mathsf{sig}}(X_{\mathsf{agg}}, R, m).$$

Finally, signer i computes and broadcasts its partial signature

$$s_i = r_i + c \cdot a_i x_i \pmod{q}.$$

The final signature is

$$\sigma = (R, s), \qquad s = \sum_{i=1}^{n} s_i \pmod{q}.$$

Verification. Verification is identical to the basic Schnorr:

$$sG \stackrel{?}{=} R + cX_{\mathsf{agg}}.$$

Security and Application

MuSig was designed with explicit attention to the requirements of real-world applications, particularly blockchain systems: compact signatures indistinguishable from Schnorr signatures, minimal rounds of communication, and tight rogue-key security. The scheme builds directly on the Bellare–Neven approach of coefficient binding, but adapts it into a practical protocol.
The advantages of key aggregation for blockchain ecosystems are numerous:

- **Data Reduction:** A single public key and a single signature, instead of many, significantly reduce the data needed for each transaction, improving network throughput and efficiency.
- **Improved Privacy:** Since the aggregated key is the only one visible on the blockchain, the identities of the individual signers are masked, enhancing participant privacy.
- **Efficient Verification:** The final signature can be verified by an external entity using only the aggregated key, in a manner identical to a normal Schnorr verification, making the process faster and more scalable.

MuSig also achieves strong $\mathsf{EUF} - \mathsf{CMA}$ security in the ROM, assuming hardness of DLog. The commitment phase thwarts *nonce manipulation attacks*, where a malicious signer could bias R to control the final challenge c.

However, MuSig requires *three rounds*: this overhead makes it less attractive for latency-sensitive protocols such as Bitcoin scriptless scripts or Lightning Network updates.

2.4 MuSig2: Two-Round Schnorr Multisignatures

The MuSig protocol, while innovative and secure, required three communication rounds, a requirement that could be impractical in environments with high latency or where communication is expensive. MuSig2 was introduced in [NRS21] as a further refinement, with the primary goal of reducing the communication rounds from three to two, making it significantly more practical and efficient for real-world use. This optimization was achieved while maintaining security even in the presence of concurrent signing sessions, a known vulnerability in other two-round schemes.

Technical details

MuSig2 is a multi-signature scheme specifically designed for practical deployment in Bitcoin (Taproot upgrade), providing compact 2-round protocol with *provable EUF-CMA security* under the secure OMDL assumption in the random-oracle model.

The scheme is a simple two-round variant of the MuSig scheme In particular, it removed the preliminary commitment phase, so that signers start right away by sending nonces. However, to obtain a scheme secure under concurrent sessions, each signer i sends a list of $\nu \geq 2$ nonces $R_{i,1}, \ldots, R_{i,\nu}$ (instead of a single nonce R_i), and effectively uses a linear combination $\hat{R}_i = \sum_{j=1}^{\nu} b^{j-1} R_{i,j}$ of these ν nonces, where b is a scalar derived using a hash function.

Key Generation (KGen). Each signer $i \in [n]$ generates a random secret key $x_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and returns the corresponding public key $X_i = x_i G$.

Key Aggregation (KeyAgg). Let $L = \{X_1, \ldots, X_n\}$ be a multiset of public keys. The key aggregation coefficient for L and a public key $X \in L$ is defined as $\mathsf{KeyAggCoef}(L, X) := \mathsf{H}_{\mathsf{agg}}(L, X)$. Then the aggregate key corresponding to L is $\widetilde{X} := \sum_{i=1}^{n} a_i X_i$, where $a_i := \mathsf{KeyAggCoef}(L, X_i)$.

First Signing Round. Each signer can perform the Sign step before the cosigners and the message to sign have been determined.
Sign: for each $j \in \{1, \ldots, \nu\}$, each signer generates random $r_{1,j} \leftarrow\!\!\$\ \mathbb{Z}_q$, computes the corresponding $R_{1,j} = r_{1,j} G$ and then sends the ν public nonce points $(R_{1,1}, \ldots, R_{1,\nu})$.
SignAgg: the aggregator receives $(R_{i,1}, \ldots, R_{i,\nu})$ from each signer $i \in [n]$, aggregates them by computing $R_j = \sum_{i=1}^{n} R_{i,j}$ for each $j \in [\nu]$ and outputs $(R_1, \ldots, R_\nu)$.

Second Signing Round. Sign′: the signer i uses the key aggregation algorithm to compute $\widetilde{X}$ and stores its own key aggregation coefficient $a_i = \mathsf{KeyAggCoef}(L, X_i)$. Upon reception of the aggregate first-round output $(R_1, \ldots, R_\nu)$, the signer computes $b := \mathsf{H}_{\mathsf{non}}(\widetilde{X}, (R_1, \ldots, R_\nu), m)$. Then it computes

$$R := \sum_{j=1}^{\nu} b^{j-1} R_j, \quad c := \mathsf{H}_{\mathsf{sig}}(\widetilde{X}, R, m), \quad s_i := c \cdot a_i x_i + \sum_{j=1}^{\nu} b^{j-1} r_{i,j}$$

and outputs s_i.
SignAgg′: the aggregator receives $(s_1, \ldots, s_n)$ and output the aggregated $s = \sum_i s_i \pmod q$.
Then, each signer compute the signature $\sigma := (R, s)$.

Verification (Vf). Given an aggregated public key $\widetilde{X}$, a message m and a signature $\sigma = (R, s)$, the verifier accepts the signature if $sG = R + c\widetilde{X}$.

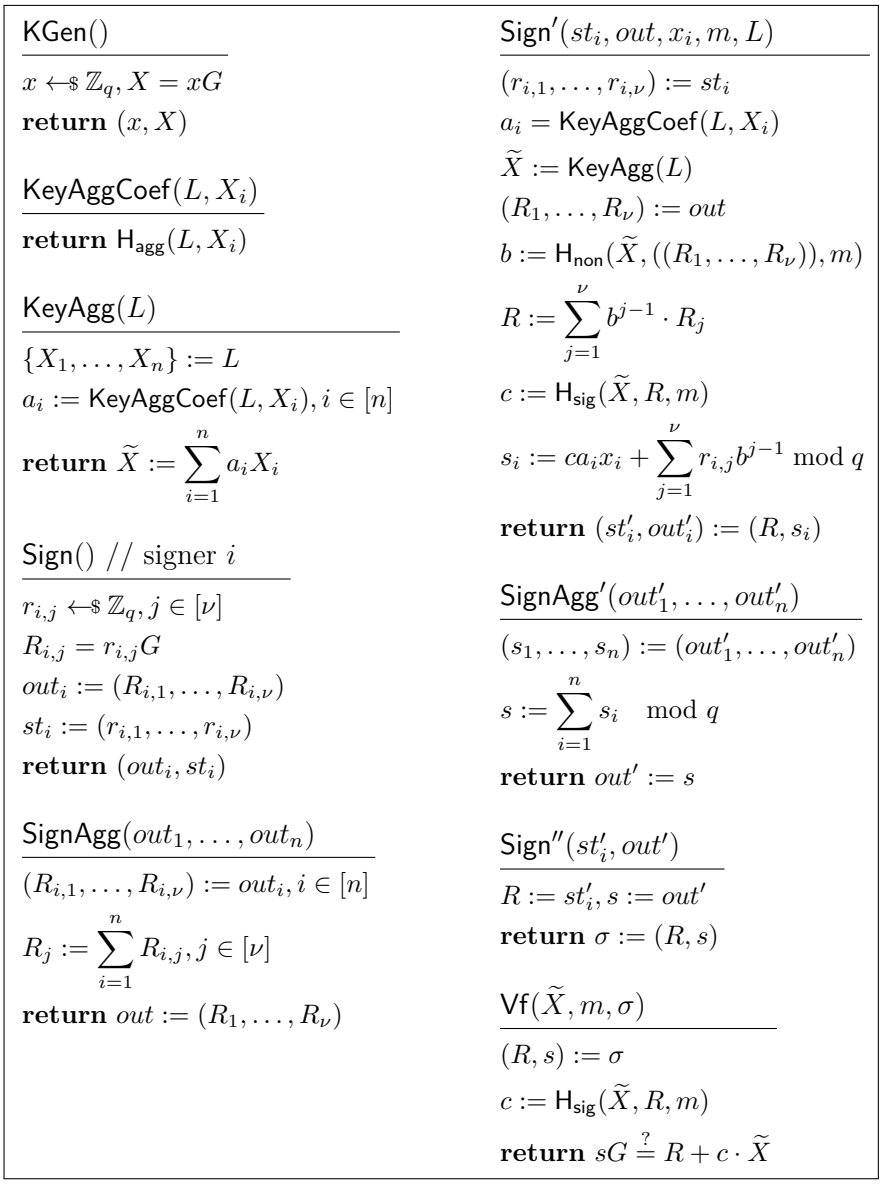

Figure 2.3. The MuSig2 protocol

Nonce Handling. Each signer i generates two secret nonces $r_{i,1}, r_{i,2} \leftarrow\$ \mathbb{Z}_q$ and the corresponding public nonce points $R_{i,1}, R_{i,2}$. Each signer sends these nonce points to the others. This step can be performed in advance, even before the message to be signed is available. Once all signers have received the public nonces from the others, the first communication round is complete.
Instead of a commitment scheme, security relies on requiring each signer to deterministically derive nonces from a secret seed and the message. This prevents adaptive nonce biasing and re-use vulnerabilities.
The aggregate nonce is computed as $R = R_1 + b \cdot R_2$, where $b = \mathsf{H}_{\mathsf{non}}(R_1, X_{agg}, m)$ ensures the binding of the second component to the message, preventing cancellation attacks.
The extra nonce and coefficient b eliminate attacks where a malicious signer adapts its nonce to bias R or to cancel honest nonces, while keeping the protocol two-round and non-interactive with respect to the message once commitments are fixed.

Security Analysis

MuSig2 is proven secure using only $\nu = 2$ nonces in the random oracle model (ROM) using a weaker assumption of the One-More Discrete Logarithm (OMDL), which strengthens the DLog problem to capture settings where an adversary has access to an oracle but must solve one more instance than queries made. This models the availability of multiple public keys and partial signatures in a multi-signer setting.

In the *algebraic* OMDL (AOMDL) variant, whenever $\mathcal{A}$ queries to the DLog oracle a group element X, it is required to include an algebraic representation $(\alpha, (\beta_i)_{i\in[c]})$ such that

$$X \;=\; \alpha G + \sum_{i=1}^{c} \beta_i X_i,$$

where c is the number of challenge group elements it has received thus far.
The scheme ensures robustness against rogue-keys, nonce cancellation, and adaptive biasing attacks, while maintaining only two rounds of interaction, a significant efficiency improvement over prior multi-round protocols:

- **Two-Round Efficiency:** MuSig2 eliminates the commitment phase, requiring only two rounds (nonce exchange, partial signatures). This reduction significantly improves deployability in interactive protocols.
- **Rogue-key resistance:** coefficient binding $a_i = \mathsf{H}(L, X_i)$ ties each contribution to the full set $\{X_j\}_{j\in[n]}$.
- **Robust nonce soundness:** the new two-component nonce structure with binding factor b prevents both replay and biasing attacks without requiring interactive commitments.
- **Tweak-compatibility:** linearity preserves correctness under Taproot tweaks $X' = X + tG$.

Implementation notes.

Nonces must be single-use per message and session; deterministic generation with auxiliary randomness is recommended, but *precomputation* of nonces must include a binding to future session identifiers to avoid cross-session reuse. Batch verification applies as in the simple Schnorr scheme with the aggregate key.

Deterministic Nonces A variant of MuSig2, known as MuSig-DN [NRSW20], focuses on the use of deterministic nonces to mitigate the risk of nonce reuse, a vulnerability that can lead to catastrophic private key loss.

Standardization and Adoption. MuSig2 is standardized in Taproot-enabled Bitcoin scripts, enabling efficient collaborative custody, payment channels, and multi-party scripts. It has been formally deployed in the proposal BIP327 and its implementation has been integrated into the "libsecp256k1" library. One of the biggest advantages of this adoption has been for off-chain protocols: two-round multisignatures reduce latency in payment networks such as the Lightning Network, where interactive signing is frequent and must remain lightweight, or Ark which allows the UTXO sharing using key aggregation.

The evolution from MuSig to MuSig2 is an example of cryptographic engineering. MuSig, while secure, had a three-round interaction that made it susceptible to network delays and failures. The reduction to two rounds in MuSig2 is not a simple incremental improvement but a critical optimization that made it robust and suitable for production environments. Its focus on issues like concurrent sessions, pre-processing, and standardization demonstrates a clear evolution from theoretical proof to practical readiness for the real world.

2.5 Further Constructions and Pitfalls

While MuSig2 represents a major step forward in practical multisignature design, the line of research did not stop there. Subsequent works focused on addressing two main directions: improving efficiency through signature aggregation and adaptor signatures, while maintaining security against ROS-type attacks.

The ROS Problem The ROS problem arises when an adversary manipulates concurrent signing sessions to bias the final challenge value or to reuse nonces across different contexts. Even with deterministic nonce generation, improper binding between nonces, messages, and participant keys may lead to leakage or forgery opportunities. Recent refinements of MuSig2 and its successors introduced stricter session separation and domain separation mechanisms, ensuring that nonces cannot be correlated across distinct executions. This line of defense was crucial in deploying multisignatures securely in adversarial environments such as the Bitcoin protocol.

Signature Aggregation Another major improvement concerns *signature aggregation*, which extends the concept of multisignatures to collections of independent signatures on distinct messages. Unlike multisignatures, where all participants sign the same message, aggregation schemes allow multiple signers to compress their signatures into a single short proof, significantly reducing storage and verification costs. This is particularly relevant for the Bitcoin system, where transaction throughput and block space are scarce resources.

Adaptor Signatures Adaptor signatures enable conditional transfer of knowledge using Schnorr's linearity. Given a point $T = tG$ (an *adaptor*) and a pre-signature (R', s') with $R = R' + T$, the final signature is

$$\sigma = (R,\, s' + t).$$

Anyone knowing σ and (R', s') learns $t = s - s'$; conversely, without t, (R, s) cannot be produced. This mechanism realizes atomic swaps and Lightning HTLCs without on-chain scripts, relying on DLog hardness and standard Schnorr verification.

Chapter 3

The ROS problem

A well-known problem in modern cryptography is the **ROS** problem (Random Inhomogeneities in a Overdetermined Solvable system of linear equations). First introduced by Schnorr [Sch01] in the study of *blind signature schemes*, a key component of anonymous digital cash, the ROS problem has since attracted considerable attention. The relevance of the ROS problem lies in its direct relationship with "one-more signature forgery" attacks, in which an adversary interacts ℓ times with a legitimate signer and manages to produce $\ell + 1$ valid signatures. Unlike traditional attacks that attempt to solve public key-related problems such as the discrete logarithm (DLog)), the ROS attack is generic in nature and does not depend on the signer's public key. This makes the ROS attack an intrinsic threat that cannot be countered by traditional countermeasures based on key indistinguishability. The problem is not simply solving a linear system, but rather finding a solvable subsystem among a multitude of instances, where the known terms (the "inhomogeneities") are generated by a random oracle.
The security of several interactive signature schemes, including threshold, multi-signature, and blind signature schemes, has been shown to be closely tied to the hardness of the ROS problem.

3.1 Formal description

Given a polynomial $\boldsymbol{\rho} = \rho_0 + \rho_1 x_1 + \ldots + \rho_\ell x_\ell \in \mathbb{Z}_q[x_1, \ldots, x_\ell]$ of total degree 1, let $\hat{\boldsymbol{\rho}} \in \mathbb{Z}_q^\ell$ be the vector having at the i-th position the coefficient of x_i, i.e. $\hat{\boldsymbol{\rho}} = (\rho_1, \ldots, \rho_\ell)$. Observe that the constant term is not included.
The ℓ–dimension ROS problem is stated as follows:

Definition 3.1.1 (**ROS problem**). *Given a prime number q and access to a random oracle $\mathsf{H}_{\mathsf{ROS}} : \mathbb{Z}_q^\ell \to \mathbb{Z}_q$, the ℓ-dimensional **ROS problem** asks to find $(\ell + 1)$ vectors $\hat{\boldsymbol{\rho}}_i \in \mathbb{Z}_q^\ell$ for $i \in [\ell + 1]$, and a vector $\mathbf{c} = (c_1, \ldots, c_\ell)$ such that*

$$\mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i) \;=\; \langle \hat{\boldsymbol{\rho}}_i, \mathbf{c} \rangle \quad \forall i \in [\ell + 1]. \tag{3.1}$$

In matrix form, the problem is

$$\begin{bmatrix} \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_1) \\ \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_2) \\ \vdots \\ \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_{\ell+1}) \end{bmatrix} = \begin{bmatrix} \rho_{1,1} & \ldots & \rho_{1,\ell} \\ \rho_{2,1} & \ldots & \rho_{2,\ell} \\ \vdots & \ddots & \vdots \\ \rho_{\ell+1,1} & \ldots & \rho_{\ell+1,\ell} \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_\ell \end{bmatrix} \tag{3.2}$$

It is always possible to find ℓ (out of $\ell + 1$) "partial solutions" to the ROS problem. Consider $i \in [\ell]$ and define the polynomials $\boldsymbol{\rho}_i(x) = x_i \in \mathbb{Z}_q[x_1, \ldots, x_\ell]$; observe that the elements $\hat{\boldsymbol{\rho}}_i$ are

the rows of the identity matrix of size ℓ, i.e.

$$\begin{bmatrix} \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_1) \\ \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_2) \\ \vdots \\ \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_{\ell+1}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ \rho_{\ell+1,1} & \dots & \dots & \rho_{\ell+1,\ell} \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_\ell \end{bmatrix} \tag{3.3}$$

Define $c_i := \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i)$, such that, for all $i \in [\ell]$, it holds that

$$\langle \hat{\boldsymbol{\rho}}_i, (c_1, \dots, c_\ell) \rangle \;=\; \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i).$$

In general, any list of ℓ polynomials of the form $\boldsymbol{\rho}_i = \rho_{i,i} x_i$ for $\rho_{i,i} \in \mathbb{Z}_q^*, i \in [\ell]$, is a valid partial solution, as long as $c_i := \rho_{i,i}^{-1} \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i)$.
The supposedly computationally hard problem is to find the last partial solution, that is, a non-trivial linear combination $\hat{\boldsymbol{\rho}}_{\ell+1}$ of these values c_i, that matches the hash image $\mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_{\ell+1})$.

Evolution of the attacks

One of the first cryptanalytic approaches to this problem was given by Wagner in [Wag02], introducing a solution with sub-exponential time complexity to the ROS problem. ROS solutions consist of finding a coefficient vector $\boldsymbol{c}$ that satisfies a system of linear equations. This can be reformulated as a k-sum problem over an additive group, where each element in each list is a term of the equation and the goal is to find a combination that sums to zero (or to a constant).
Fix $\hat{\boldsymbol{\rho}}_{\ell+1} = (1, \dots, 1)$ and build ℓ lists $L_1, \dots, L_\ell$ such that the i-th list is populated with polynomials of the form $\boldsymbol{\rho}_i = \rho_{i,i} x_i$ for random $\rho_{i,i} \in \mathbb{Z}_q^*$.
For every element in the list L_i, consider its respective coefficient $c_i = \rho_{i,i}^{-1} \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i)$.
Build an efficient algorithm that finds c_i 's satisfying

$$\langle \hat{\boldsymbol{\rho}}_{\ell+1}, \boldsymbol{c} \rangle \;=\; c_1 + \ldots + c_\ell \;=\; \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_{\ell+1}),$$

i.e. solving the *ℓ-list birthday problem* (see Appendix A). Wagner showed that the above problem can be solved in time $\mathcal{O}(\ell \cdot 2^{\lceil \log q \rceil / (1 + \lfloor \log \ell \rfloor)})$, improving the attack by using multiples of $(1, \dots, 1)$ as $\boldsymbol{\rho}_{\ell+1}$, which now reduces ROS to the *(ℓ+1)-list birthday problem* yielding a subexponential complexity, whenever $\ell > \log \lambda$ signatures are issued concurrently. At the time, it had major consequences to cryptanalysis, especially to schemes using the hardness of the ROS problem as part of their security assumptions.
However, the ROS problem itself allows for much more flexibility for the attacker: for example, the attacker can consider a subset of c_i 's (by setting some entries of $\boldsymbol{\rho}_{\ell+1}$ to zero), in which case we end up with a subset-sum problem that is, in general, *NP-hard*.
Wagner's solution shows that, although the problem had not been studied in general before, his algorithmic techniques can be applied with devastating effects to ROS and other cryptographic schemes, revealing an underlying vulnerability that was not previously considered.

Almost twenty years later, Benhemouda et al. [BLL+22] expanded Wagner's approach providing a polynomial solution to the ROS problem that runs in polynomial time for dimension $\ell > \log_2 q$ and a combination of this with Wagner's algorithm to make the attack effective even for smaller dimensions, thus proving the insecurity of many Schnorr-based blind signature schemes. Unlike Wagner's attack, which searches for a probabilistic solution, the *polynomial attack* constructs one. This method leverages the high number of available parallel interactions ($\ell \geq \lambda = \lceil \log q \rceil$, where λ is the bit length of the modulus q) to transform the ROS problem into a subset sum of powers of two, which is trivial to solve.
One of their main results is stated as follows:

Theorem 3.1.1 ([BLL+22]). *Let* $\mathsf{Pgen}(1^\lambda)$ *be a parameter generation algorithm that, given as input the security parameter* λ *in unary, outputs an odd prime of (bit) length* $\lambda = \lceil log_2 q \rceil$*. If* $\ell \geq \lambda$*, then there exists an adversary that runs in polynomial time and solves the ROS problem relative to* Pgen *with dimension* ℓ*.*

The idea of the proof is to build an attacker capable of constructing a polynomial whose coefficients depend on the information extracted from ℓ opened sessions. Then, it leverages the recursive nature of the ROS problem to obtain the target values $\boldsymbol{c}$ and $\boldsymbol{\rho}_{\ell+1}$ from (3.1), since the index i is in the interval $[\ell+1] = \{1, \ldots, \ell+1\}$. The crucial premise is that the number of dimensions ℓ is sufficient to accommodate the full binary decomposition of the target value.

The polynomial attack proposed in [BLL+22] was recently revisited in [JLS25]. This yields the first polynomial-time solution to the ROS problem for $\ell \gtrsim 0.725 \cdot \log_2 p$, together with an improved version of the generalized ROS attack: combined with Wagner's algorithm, the approach further reduces complexity for dimensions below $0.725 \cdot \log_2 p$. An implementation also demonstrates that the attack breaks the one-more unforgeability of blind Schnorr signatures over 256-bit elliptic curves in a few seconds with 192 concurrent sessions.

3.2 The polynomial attack

The goal is to construct an adversry $\mathcal{A}$ for a game $\mathsf{ROS}_{\mathsf{Pgen},\mathcal{A},\ell}(\lambda)$, where $\ell \geq \log q$, that outputs $(\hat{\boldsymbol{\rho}}_i)_{i\in[\ell+1]}$ and $\boldsymbol{c} = (c_1, \ldots, c_\ell)$ such that

$$\mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i) \;=\; \langle \hat{\boldsymbol{\rho}}_i, \boldsymbol{c} \rangle \qquad \text{for } i = 1, \ldots, \ell+1.$$

The game is formally described in Figure 3.1.

Game $\mathsf{ROS}_{\mathsf{Pgen},\mathcal{A},\ell}(\lambda)$
$q \leftarrow \mathsf{Pgen}(1^\lambda)$
$((\hat{\boldsymbol{\rho}}_i)_{i\in[\ell+1]}, \boldsymbol{c}) \leftarrow \mathcal{A}^{\mathsf{H}_{\mathsf{ROS}}}(q)$
return $\big(\forall i \neq j \in [\ell+1] : \quad \hat{\boldsymbol{\rho}}_i \neq \hat{\boldsymbol{\rho}}_j \quad \wedge \quad \langle \hat{\boldsymbol{\rho}}_i, \boldsymbol{c} \rangle = \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i)\big)$

Figure 3.1. ROS Game

Polynomial Construction. The adversary $\mathcal{A}$ can always define ℓ partial solutions by choosing two sets of ℓ polynomials,

$$\boldsymbol{\rho}_i^{(0)} \;:=\; x_i, \quad \boldsymbol{\rho}_i^{(1)} \;:=\; 2x_i, \qquad \text{for } i = 1, \ldots, \ell.$$

The coefficients $c_i^{(b)}$ are derived from the random oracle $\mathsf{H}_{\mathsf{ROS}}$ as

$$c_i^{(b)} \;:=\; 2^{-b}\, \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i^{(b)}) \qquad \text{for } b \in \{0,1\}.$$

If there exists $i^* \in [\ell]$ such that $c_{i^*}^{(0)} = c_{i^*}^{(1)}$, then $\mathcal{A}$ stops immediately and returns the ROS solution $(\hat{\boldsymbol{\rho}}_1^{(0)}, \ldots, \hat{\boldsymbol{\rho}}_\ell^{(0)}, \hat{\boldsymbol{\rho}}_{i^*}^{(1)})$ and $(c_1^{(0)}, \ldots, c_\ell^{(0)})$.
Otherwise, if $c_i^{(0)} \neq c_i^{(1)}$ for all $i \in \ell$, define the *linear* polynomial

$$\mathbf{f}_i(x_i) \;:=\; \frac{x_i - c_i^{(0)}}{c_i^{(1)} - c_i^{(0)}}, \tag{3.4}$$

such that $\mathbf{f}_i(c_i^{(b)}) = b$, for $b \in \{0,1\}$.

Decomposition. The adversary $\mathcal{A}$ constructs the multivariate polynomial of total degree 1

$$\boldsymbol{\rho}_{\ell+1}(x_1,\ldots,x_\ell) \;:=\; \sum_{i=1}^{\ell} 2^{i-1}\mathbf{f}_i(x_i) \;=\; \rho_{\ell+1,0} + \rho_{\ell+1,1}x_1 + \ldots + \rho_{\ell+1,\ell}x_\ell,$$

and defines

$$y \;:=\; \rho_{\ell+1,0} + \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_{\ell+1}) \;=\; \rho_{\ell+1,0} + \mathsf{H}_{\mathsf{ROS}}((\rho_{\ell+1,1},\ldots,\rho_{\ell+1,\ell})). \tag{3.5}$$

Since $\ell \geq \lambda$, i.e. $2^\ell > q$, it is possible to write y in binary form as

$$y \;=\; \sum_{i=1}^{\ell} 2^{i-1}b_i \mod q,$$

this implicitly defines the $b_i \in \{0,1\}$ coefficients.

Final Forgery. The adversary $\mathcal{A}$ outputs the ROS solution

$$(\hat{\boldsymbol{\rho}}_1^{(b_1)},\ldots,\hat{\boldsymbol{\rho}}_\ell^{(b_\ell)},\hat{\boldsymbol{\rho}}_{\ell+1}) \quad \text{and} \quad \boldsymbol{c} := (c_1^{(b_1)},\ldots,c_\ell^{(b_\ell)}). \tag{3.6}$$

In fact, for $i \in \ell$,

$$\langle \hat{\boldsymbol{\rho}}_i^{(b_i)}, \boldsymbol{c}\rangle \;=\; 2^{b_i}\cdot c_i^{(b_i)} \;=\; \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_i^{(b_i)}),$$

while for $i = \ell + 1$

$$\langle \hat{\boldsymbol{\rho}}_{\ell+1}, \boldsymbol{c}\rangle \;=\; \boldsymbol{\rho}_{\ell+1}(\boldsymbol{c}) - \rho_{\ell+1,0} \;=\; \sum_{i=1}^{\ell} 2^{i-1}\mathbf{f}_i(c_i^{b_i}) - \rho_{\ell+1,0} \;=\; \sum_{i=1}^{\ell} 2^{i-1}b_i - \rho_{\ell+1,0} \;=\; \mathsf{H}_{\mathsf{ROS}}(\hat{\boldsymbol{\rho}}_{\ell+1}).$$

3.3 Blind signatures protocol

For simplicity, as in [Sch01, BLL+22], it is better to explain the attack on the *Schnorr blind signatures*, i.e. schemes where a signer, who controls the secret signature key, can append his signature to a message without knowing its content, an operation fundamental to privacy.
For example, a *user* might want a *server* to sign a message that he does not want to reveal. By slightly altering the message (e.g. adding a random value) and then sending it to the signer, the signature the server returns on that altered message can be later used by the user to recover the original signature.

3.3.1 The blind scheme

As in the previous sections, assume the existence of a group generator algorithm $\mathsf{Pgen}(1^\lambda)$ that, given as input the security parameter in unary form outputs a cyclic group $(\mathbb{G}, +)$ of prime order q generated by G. Assume that the prime q is of length λ, so it holds $\lambda = \log_2 q$. In Schnorr blind signatures, a *signing* key is a scalar sampled uniformly at random $x \leftarrow\!\!\$\ \mathbb{Z}_q$, and its respective *verification* key $X = xG$ in the group $\mathbb{G}$.
A signature for a message $m \in \{0,1\}^*$ is a pair $(R, s) \in \mathbb{G} \times \mathbb{Z}_q$ such that $sG - cX = R$, where $c := \mathsf{H}(R, m)$, thus requiring the same verification process as the classic Schnorr scheme.
As described in [CP92], the signature protocol is called *blind* if it generates a signature (R, s) that is *statistically independent* of the interaction that provides the view of the signer. Later on, blind signatures cannot be identified and related to the signer interaction.
The protocol is shown in 3.2.

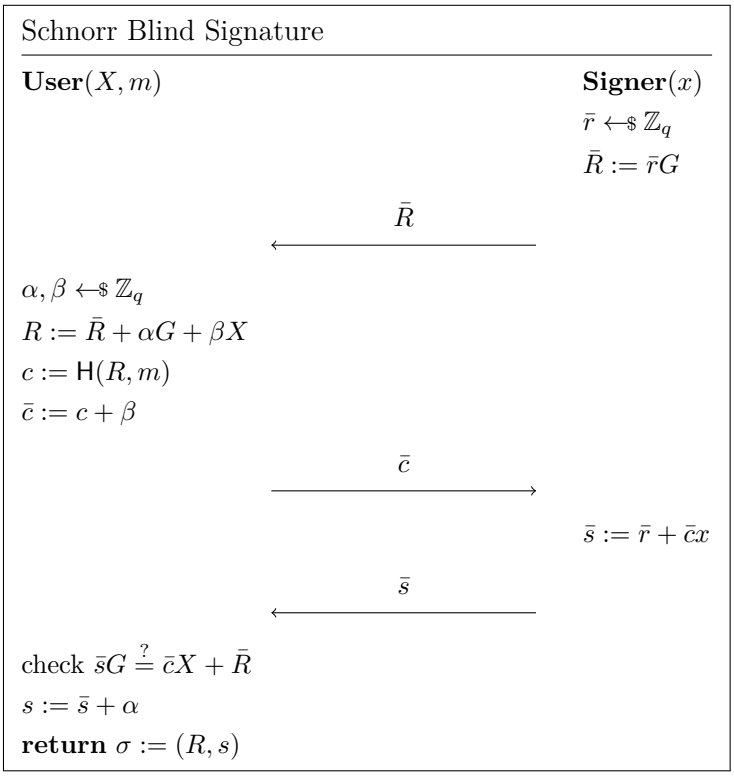

Figure 3.2. Schnorr Blind Signature signing protocol

In order to generate a blind signature (R, s) the user picks random numbers $\alpha, \beta \leftarrow\$ \mathbb{Z}_q$ and responses to the commitment $\bar{r}G$ by sending the challenge $\bar{c} = c + \beta$, where $c := \mathsf{H}((r + \alpha)G + \beta X, m) \in \mathbb{Z}_q$. After receiving $\bar{s} = \bar{r} + \bar{c}x \in \mathbb{Z}_q$ it is possible to compute $s = \bar{s} + \alpha$ and $c = \bar{c} - \beta$. Here α randomizes the response $\bar{s}$, while β hides the challenge c, making the final signature unlinkable to the original interaction. The generated signature (R, s) is uniformly distributed over all signatures on message m due to the random $\alpha, \beta \leftarrow\$ \mathbb{Z}_q$ and is produced for a unique pair (α, β) such that $\alpha = s - \bar{s}$ and $\beta = \bar{c} - c$.

Validity. Given the output of the interaction $(\bar{R}, \bar{c}, \bar{s}) = (\bar{R}, c + \beta, s - \alpha)$, it holds

$$sG - cX \;=\; (\bar{r} + \bar{c}x + \alpha)G - (\bar{c} - \beta)X \;=\; (\bar{r} + \alpha)G + \beta X \;=\; R.$$

Hence, $\mathsf{H}(sG - cX, m) = \bar{c} - \beta = c$ and thus (R, s) is a valid signature. Intuitively, the verification succeeds because the blinding factors α, β only shift the relation without altering its validity, thus "hiding" the original interaction.

3.3.2 Building the attack

Thanks to [BLL+22], it is possible to construct a probabilistic polynomial-time adversary $\mathcal{A}$ that is able to produce (with overwhelming probability) $\ell + 1$ signatures after opening $\ell \geq \lambda = \lceil \log_2 q \rceil$ parallel sessions. When applying the ROS attack to cryptographic schemes, it is mandatory to accept a negligible failure probability and use the additional flexibility in the random oracle input to use polynomials $\boldsymbol{\rho}_i = x_i$ (instead of either x_i or $2x_i$, for $i \in [\ell]$). This simplifies descriptions and makes attacks easier to read. The attack proceeds as follows.
The adversary $\mathcal{A}$ selects arbitrary messages $m_1, \ldots, m_{\ell+1} \in \{0,1\}^*$ for which it will output a signature. It opens ℓ sessions, obtaining the first message from the signer and receiving $\bar{\mathbf{R}} = (\bar{R}_1, \ldots, \bar{R}_l) \in \mathbb{G}^\ell$.
For $i \in [\ell]$, $\mathcal{A}$ samples uniformly at random three blinding factors $(\alpha_{i,0}, \alpha_{i,1}, \beta_i) \leftarrow\$ \mathbb{Z}_q^3$, and defines

$$R_{i,b} = \bar{R} + \alpha_{i,b}G + \beta_i X \quad \text{for } b \in \{0,1\}. \tag{3.7}$$

Let $c_i^b := \mathsf{H}(R_{i,b}, m_i)$ for $i \in [\ell]$ and $b \in \{0,1\}$. Assume $c_i^0 \neq c_i^1$ and fail otherwise (actually the failure probability is negligible).
Define the polynomial $\boldsymbol{\rho} \in \mathbb{Z}_q[x_1, \ldots, x_\ell]$:

$$\boldsymbol{\rho}(x_1, \ldots, x_\ell) \;:=\; \sum_{i=1}^{\ell} 2^{i-1} \cdot \frac{x_i - c_i^0}{c_i^1 - c_i^0} \;=\; \sum_{i=1}^{\ell} \rho_i x_i + \rho_0. \tag{3.8}$$

Observe that the above polynomial is such that, for any $(b_1, \ldots, b_\ell) \in \{0,1\}^\ell$ it holds

$$\boldsymbol{\rho}(c_1^{b_1}, \ldots, c_\ell^{b_\ell}) \;=\; \sum_i 2^{i-1} b_i. \tag{3.9}$$

Let $R_{\ell+1} := \langle \boldsymbol{\rho}, \bar{\mathbf{R}} \rangle = \sum_{i=1}^{\ell} \rho_i \bar{R}_i$ (the constant term ρ_0 is not included).
Define $c_{\ell+1} := \mathsf{H}(R_{\ell+1}, m_{\ell+1})$ and consider the binary decomposition

$$c_{\ell+1} - \sum_{i=1}^{\ell} \rho_i \beta_i + \rho_0 \;=\; \sum_{i=1}^{\ell} 2^{i-1} b_i. \tag{3.10}$$

Let $\bar{\mathbf{c}} = (c_1^{b_1} + \beta_1, \ldots, c_\ell^{b_\ell} + \beta_\ell)$ and complete the ℓ opened sessions by replying to the i-th session with $\bar{c}_i$, for $i \in [\ell]$.
The adversary thus obtains ℓ responses $\bar{\mathbf{s}} := (\bar{s}_1, \ldots, \bar{s}_\ell) \in \mathbb{Z}_q^\ell$, and defines $s_{\ell+1} := \langle \boldsymbol{\rho}, \bar{\mathbf{s}} \rangle =$

$\sum_{i=1}^{\ell} \rho_i \bar{s}_i$.
Finally, $\mathcal{A}$ proceeds unblinding the ℓ honest signatures by computing $\mathbf{s} := (\bar{s}_1 + \alpha_{1,b_1}, \ldots, \bar{s}_\ell + \alpha_{\ell,b_\ell})$ and outputs the $\ell + 1$ forgeries $(m_i, (R_i, s_i))_{i \in [\ell+1]}$, defined as:

$$(R_i, s_i) = \begin{cases} (\bar{R}_i + \alpha_{i,b_i} G + \beta_i X, \bar{s}_i + \alpha_{i,b_i}) & \text{for} \quad i = 1, \ldots, \ell, \\ \left(\sum_{i=1}^{\ell} \rho_i \bar{R}_i, \sum_{i=1}^{\ell} \rho_i \bar{s}_i\right) & \text{for} \quad i = \ell + 1. \end{cases} \tag{3.11}$$

Validity. By perfect correctness, the first ℓ signatures are valid, in fact for $i \in [\ell]$:

$$R_i = \bar{R}_i + \alpha_{i,b_i} G + \beta_i X = \bar{s}_i G - \bar{c}_i X + \alpha_{i,b_i} G + \beta_i X = s_i G - c_i^{b_i} X,$$

where $\bar{s}_i$ satisfies $\bar{s}_i G - \bar{c}_i X = \bar{R}_i$ and $c_i^{b_i} = \mathsf{H}(R_{i,b_i}, m_i) = \mathsf{H}(R_i, m_i)$ since $R_{i,b_i} = R_i$.
For the forgery case $(m_{\ell+1}, (R_{\ell+1}, s_{\ell+1}))$ it holds

$$R_{\ell+1} = \sum_{i=1}^{\ell} \rho_i \bar{R}_i = \sum_{i=1}^{\ell} \rho_i (\bar{s}_i G - \bar{c}_i X) = s_{\ell+1} G - c_{\ell+1} X,$$

where $c_{\ell+1} = \mathsf{H}(R_{\ell+1}, m_{\ell+1})$. The second equality comes from the fact that $\bar{s}_i G - \bar{c}_i X = \bar{R}_i$.
The last equality comes from:

$$\sum_{i=1}^{\ell} \rho_i \bar{c}_i \;=\; \sum_{i=1}^{\ell} \rho_i c_i^{b_i} + \sum_{i=1}^{\ell} \rho_i \beta_i \;=\; \boldsymbol{\rho}(c_1^{b_1}, \ldots, c_\ell^{b_\ell}) + \sum_{i=1}^{\ell} \rho_i \beta_i - \rho_0 \;=\; \sum_{i=1}^{\ell} 2^{i-1} b_i + \sum_{i=1}^{\ell} \rho_i \beta_i - \rho_0 = c_{\ell+1}.$$

3.4 Attack on multi-signatures

This section presents the ROS parallel attack on a multi-signature scheme that allows key aggregation [NRS21]. Suppose $n = 2$ for simplicity.

3.4.1 Building the attack

The attack requires an adversary $\mathcal{A}$ to open ℓ concurrent signing sessions, in which it plays the role of the signer with public key $X_2 = x_2 G$, and receives ℓ nonces $R_1^{(1)}, \ldots, R_1^{(\ell)}$ from the honest signer with public key $X_1 = x_1 G$. Let

$$\widetilde{X} \;=\; a_1 X_1 + a_2 X_2$$

be the corresponding aggregate public key. Given a forgery target message m^*, $\mathcal{A}$ computes $R^* = \sum_{i=1}^{\ell} R_1^{(i)}$ and uses Wagner's algorithm to find nonces $R_2^{(i)}$ to reply with such that

$$c^* \;:=\; \sum_{i=1}^{\ell} c^{(i)}, \tag{3.12}$$

where $c^{(i)} := \mathsf{H}(\widetilde{X}, R_1^{(i)}, R_2^{(i)}, m^{(i)})$ and $c^* := \mathsf{H}(\widetilde{X}, R^*, m^*)$.
Having received $R_2^{(i)}$, the honest signer will reply with partial signatures $s_1^{(i)} = r_1^{(i)} + c^{(i)} \cdot a_1 x_1$.
Let $r^* = \sum_{i=1}^{\ell} r_1^{(i)} = \log_G(R^*)$. The adversary is able to obtain

$$s_1^* \;=\; \sum_{i=1}^{\ell} s_1^{(i)} \;=\; \sum_{i=1}^{\ell} r_1^{(i)} + \left(\sum_{i=1}^{\ell} c^{(i)}\right) \cdot a_1 x_1 \;=\; r^* + c^* \cdot a_1 x_1$$

and then compute the value

$$s^* \;=\; s_1^* + c^* \cdot a_2 x_2 \;=\; r^* + c^* \cdot (a_1 x_1 + a_2 x_2),$$

thus leading to a valid forgery (R^*, s^*) for the message m^* with signature hash $c^* = \mathsf{H}_{\mathsf{sig}}(\widetilde{X}, R^*, m^*)$.

3.4.2 MuSig2 solution

The attack relies on the ability to control the signature hash by controlling the aggregate nonce $R_1^{(i)} + R_2^{(i)}$ in the first round of each of the concurrent signing sessions. Since all signers must know the aggregate nonce at the end of the first round, it seems hard to prevent the adversary from being able to control the aggregate nonce on the LHS without adding a preliminary commitment round. Our high-level idea to solve this problem and to foil the attacks is to accept that the adversary can control the LHS of the equation but prevent it from controlling the RHS instead.

Each signer $i \in [n]$ sends a list $\nu \geq 2$ nonces $R_{i,1}, \ldots, R_{i,\nu}$ and uses a random linear combinantion of those nonces

$$\hat{R}_i \;=\; \sum_{j=1}^{\nu} b^{j-1} R_{i,j},$$

where b is a scalar derived via a hash function as

$$b \;:=\; \mathsf{H}_{\mathsf{non}}(\widetilde{X}, (\sum_i R_{i,1}, \ldots, \sum_i R_{i,\nu}), m).$$

Since the values $R_{i,j}$ end up as input to a hash function, one may wonder why it's used the sum $\sum_{i=1}^{n} R_{i,j}$ instead of simply concatenating all nonces. This trivial solution will also tields a secure scheme, but the aggregate $\sum_{i=1}^{n} R_{i,j}$ anyway need to be computed when computing

$$R = \sum_{i=1}^{n} \hat{R}_i = \sum_{i=1}^{n} \sum_{j=1}^{\nu} b^{j-1} \hat{R}_{i,j} = \sum_{j=1}^{\nu} b^{j-1} \cdot \left(\sum_{i=1}^{n} R_{i,j} \right).$$

As a result, whenever the adversary $\mathcal{A}$ tries different values for R_2, the coefficient b changes, and so does the honest signer's effective nonce $\hat{R}_1 = \sum_{j=1}^{\nu} b^{j-1} R_{1,j}$. This ensures that the sum of the honest signer's effective nonces taken over all open sessions, i.e. $R^* = \sum_{k=1}^{\ell} \hat{R}_1^{(k)}$, is no longer a constant value.

Single Nonce. One might think of falling back on a single nonce ($\nu = 1$) but in fact relying just on the coefficient b such that $\hat{R}_1 = b \cdot R_1$. However, then the adversary can eliminate b by redifining $R^* = \sum_{k=1}^{\ell} R_1^{(k)}$, which is independent of all $b^{(k)}$, and considering the equation

$$\sum_{k=1}^{\ell} \frac{\mathsf{H}_{\mathsf{sig}}(\widetilde{X}, b^{(k)} \cdot (R_1^{(k)} + R_2^{(k)}), m^{(k)})}{b^{(k)}} = \mathsf{H}_{\mathsf{sig}}(\widetilde{X}, R^*, m^*)$$

instead of (3.12) to perform the attack.

3.5 Dimensional eROSion

One of the most recent results obtained in the field of ROS research is due to the work of Joux, Loss, and Santato [JLS25], which builds on existing cryptanalysis and presents several significant contributions:

- a polynomial-time attack that extends the range of dimensions for which a polynomial solution is known, reaching a new bound of $\ell \geq 0.725 \cdot \log_2 q$;
- an improved version of the generalized ROS attack, which combines Wagner's algorithm with the new polynomial attack, outperforming the complexity of the previous version for an additional range of cases with dimensions smaller than $0.725 \cdot \log_2 q$;
- the practical demonstration of the polynomial attack, which breaks the one-more unforgeability of Schnorr blind signatures over 256-bit elliptic curves in a few seconds, using 192 concurrent sessions.

3.5.1 The Multi-Base Decomposition Attack

The previous polynomial attack described in [BLL+22] used a binary decomposition to solve the ROS problem. An adversary $\mathcal{A}$ would select two sets of polynomials, each providing a valid partial solution for a vector $\boldsymbol{c}$. For each element i, $\mathcal{A}$ would construct a linear polynomial $\mathbf{f}_i$ that satisfied

$$\mathbf{f}_i(c_i^0) = 0 \quad \text{and} \quad \mathbf{f}_i(c_i^1) = 2^{i-1}.$$

This setup allowed any number up to q to be expressed as a linear combination of the c_i values, with coefficients determined by the binary digits of the number.
This new method starts from the intuition that using a larger decomposition base, such as ternary, can require fewer digits and therefore a smaller number of dimensions ℓ needed for the attack, since $\log_3 q < \log_2 q$.

A naive approach to replicate this property in a ternary decomposition would require finding a linear polynomial $\mathbf{f}_i$ that satisfies three conditions:

$$\mathbf{f}_i(c_i^0) = 0, \quad \mathbf{f}_i(c_i^1) = 3^{i-1} \quad \text{and} \quad \mathbf{f}_i(c_i^2) = 2 \cdot 3^{i-1}.$$

However, this approach fails because an exact solution would lead to a quadratic polynomial, violating the linearity requirement of the ROS problem.

To overcome this issue, the authors introduce an innovative technique that leverages lattice theory to find an approximate solution to the system of linear equations that defines the conditions for the polynomial. By solving a *closest vector problem* (CVP) in a two-dimensional lattice, the adversary can find an approximate constant μ_i for each $\mathbf{f}_i$. This approximation introduces a small error term, $\delta_{i,b}$, of magnitude approximately $q^{1/2}$.

The ternary decomposition attack then proceeds in two phases to address the problem of these errors. First, the higher-order digits of a number z are decomposed using the approximate ternary polynomials $\mathbf{f}_i(x_i)$, which introduces a cumulative error. The residual value, which is approximately of size $q^{1/2}$ and includes the accumulated errors, is then handled in a second, final, and exact phase, using the original binary decomposition attack. The combination of an approximate ternary decomposition for the higher digits and an exact binary decomposition for the remainder results in a solution that requires approximately $1/2 \log_3 q + 1/2 \log_2 q \approx 0.815 \cdot \log_2 q$ dimensions, a significant improvement over the previous bound of $\ell > \log_2 q$.

The Generalized Attack

The authors further extend this concept to support decompositions in any base $B > 2$. Using a larger base B allows for more dimensions to be saved to represent a number, but introduces a larger approximation error. The solution to this trade-off is a multi-stage process that balances these effects.

The process begins by using a larger base to approximate the higher-order digits, gradually reducing the base as the introduced error becomes comparable to the powers being approximated. This process continues until base 2 is reached, at which point the decomposition can be concluded with an exact final step. This sophisticated multi-base decomposition approach demonstrates a deep understanding of the causal relationship between base size, dimensionality, and error magnitude. A naive choice of an arbitrarily large base to minimize dimensions would lead to an unmanageable error. The multi-stage, decreasing-base strategy systematically manages these competing forces, ensuring that the accumulated error remains small enough for the final step.

The final result of this generalized technique is an attack that requires a number of dimensions approximately equal to $0.725 \cdot \log_2 q$. The paper notes that this bound is nearly reached even for relatively small values of B (e.g., for $B = 6$, $\ell \approx 0.746 \cdot \log_2 q$; for $B = 8$, $\ell \approx 0.737 \cdot \log_2 q$). This suggests that for practical purposes, the theoretical asymptotic bound is not far from the performance achievable with a small number of bases, indicating that a real attacker does not need to implement a complex high base algorithm to achieve most of the benefits.

3.5.2 Trade-offs and Fine-Tuning

The practical implementation reveals trade-offs not always evident in the theoretical proof. The proof relies on worst-case bounds to ensure asymptotic performance, but an attacker can leverage probabilistic effects and intelligent heuristics to achieve better performance than theoretically guaranteed.

- **Approximate CVP solutions**: The proof assumes exact solutions to the CVP problem. However, in practice, approximation algorithms like LLL and Babai can be used to reduce the runtime of the setup phase at the expense of a slight reduction in the decomposition success probability.
- **Digit-by-digit vs. base-by-base decomposition**: The paper compares two decomposition methods. The digit-by-digit decomposition can be more robust, as it handles approximation errors partially at each step, unlike the base-by-base method.
- **Reordering lattices**: A sophisticated technique involves estimating the "quality" of a lattice, which can be estimated from its Gram-Schmidt orthogonalization. Low-quality lattices (those that produce larger errors) can be reordered to higher positions where their impact on the final result is less critical, allowing for a more aggressive choice of dimensions.

The existence of a successful but less reliable aggressive attack shows that the work of the cryptanalyst does not end with the formal proof; the implementation phase offers crucial avenues for optimization and a deeper understanding of the practical limitations of the attack.

3.6 Conclusions

The evolution of attacks on the ROS problem has deeply influenced the security assumptions of Schnorr signatures and related protocols. Initially, security was believed to hold for any non-polylogarithmic number of parallel sessions. However, the polynomial-time attack of Joux et al. [JLS25] showed that security breaks already for $\ell \geq 0.725 \cdot \log q$, extending and surpassing the previous $\log q$ bound [BLL+22]. For a 256-bit elliptic curve, this reduces the threshold to about 192 parallel sessions—negligible compared to the millions of concurrent sessions routinely handled by modern servers.

The practical results confirm this vulnerability: the Joux et al. attack, requiring $\ell = 192$ sessions (instead of the classic polynomial attack that requires $\ell \geq 256$), can be executed in just a few seconds on standard hardware, thus breaking the one-more unforgeability of blind Schnorr signatures over 256-bit curves. These findings highlight that the "critical dimension" for security has been reached and even surpassed, transforming ROS-based weaknesses from a theoretical concern into a concrete and scalable threat.

This "dimensional race" has progressively shifted the security threshold from $\ell > \log q$ to $\ell \geq 0.725 \cdot \log q$, now within the range of practical large-scale attacks. Moreover, the combined approach of Joux et al., which integrates Wagner's subexponential method with the new polynomial technique, further improves the trade-off between complexity and dimension, offering significant efficiency gains compared to earlier approaches. A comparative summary of the main methodologies is shown in Table 3.1.

Attack	Type	Complexity	Dimension (ℓ)
Wagner	sub-exp	$O\left(\ell \cdot 2^{\log q/(1+\log \ell)}\right)$	$\ell = 2^w - 1$
BLL+21	poly	$O(\text{poly}(q))$	$\ell > \log_2 q$
BLL+21 (combined)	sub-exp	$O(2^w + L)$	$\ell \geq 2^w - 1 + \lceil \lambda - (w+1)L \rceil$
Joux et al.	poly	$O(\text{poly}(q))$	$\ell \geq 0.725 \log_2 q$

Table 3.1. Comparison of the main ROS attack methodologies.

Despite these advances, several open questions remain. Wagner's analysis [Wag02] relies on the conjecture that his algorithm succeeds with constant probability, but a formal proof is still lacking. Furthermore, the new polynomial-time attacks do not currently apply to variants of the ROS problem, such as mROS (Modified ROS) or WFROS (Weighted Fractional ROS), which could potentially offer greater resistance. Future work may therefore focus both on reinforcing cryptographic schemes with countermeasures, e.g. introducing non-linearities to prevent linear decompositions, and on testing the applicability of these attacks to more general ROS variants.

Chapter 4

Cross-Input Signature Aggregation with DahLIAS

In the world of distributed systems, verifying a large number of individual signatures represents a significant computational and storage cost. To mitigate this problem, multi-party signature schemes have been developed. The research document distinguishes between multi-signatures, threshold signatures, and aggregate signatures. *Multi-signatures*, such as MuSig2, are designed for a group of parties signing a *single* common message to produce a single compact signature. In contrast, **aggregate signatures** are designed for a different application: they allow multiple signers, each with their own key and *distinct* message, to produce a single, short signature that simultaneously attests to the validity of all the individual signatures. This functional distinction makes aggregate signatures particularly suitable for use cases like Bitcoin transactions, where multiple users sign for their own separate inputs.

Despite their potential for saving space and accelerating verification, the design of modern aggregate signature schemes based on the discrete logarithm (DLog) problem in pairing-free groups has received less attention. Existing solutions, such as BLS signatures, rely on pairing-based groups that are not compatible with the elliptic curve `secp256k1` used in Bitcoin. This highlights a long-standing open problem: the existence of an interactive aggregate signature scheme with constant-size signatures directly derived from pairing-free groups.

4.1 CISA: cross-input signature aggregation

The Bitcoin protocol, despite its robustness and decentralized nature, faces inherent challenges related to its on-chain transaction throughput and privacy model. The network's design, constrained by an average block creation time of 10 minutes and an original block size limit of 1 MB, imposes a maximum transaction processing capacity that is estimated to be between 3.3 and 7 transactions per second. This technical bottleneck has significant economic consequences. When network activity is high, the limited block space leads to increased competition, resulting in higher transaction fees and processing delays for users.

Simultaneously, the privacy of transactions on the Bitcoin blockchain remains a persistent concern. While Bitcoin operates on a pseudonymous basis, with transactions linked to addresses rather than personal identities, the public and transparent nature of the blockchain makes all spending histories globally visible. For individuals and businesses seeking to maintain financial confidentiality, this transparency can be problematic. The issue is exacerbated for complex transactions, particularly those involving multiple inputs, which can reveal more about a user's spending habits and wallet structure. Such patterns are readily identifiable by sophisticated chain analysis and can, in certain contexts, pose a genuine security risk to users, including human rights

activists operating in hostile regimes.

The economic and privacy challenges are intrinsically linked. Privacy-enhancing protocols, such as CoinJoin, often involve transactions with a larger data footprint due to their multiple inputs and corresponding signatures. This greater size results in higher fees, which can deter users from adopting these privacy measures. The act of paying a premium for a more complex, private transaction can itself function as a signal to observers that the user is attempting to conceal something, thereby creating a new vulnerability. This dynamic transforms a simple economic barrier into a potential social and political risk, further limiting the normalization of private transactions.

4.1.1 The CISA Proposal

Cross-Input Signature Aggregation (CISA) is a proposal to address these interconnected issues by fundamentally altering how signatures are handled within Bitcoin transactions.
CISA aims to reduce the data footprint of multi-input transactions by combining multiple signatures into a single, smaller signature. The core objective is to make transactions with numerous inputs, which are essential for collaborative privacy tools like CoinJoin, significantly cheaper and more efficient. By providing a modest reduction in transaction size, CISA could increase the total number of transactions that can be included in a block, thereby improving on-chain scalability and reducing fees for users. CISA is a complex proposal that requires a soft fork to change Bitcoin's consensus rules and is currently the subject of ongoing cryptographic research and developer discussion within the Bitcoin community.
The state of art is very well described in the Fabian Jahr's research [Jah25], supported by the Human Rights Foundation.

Key Aggregation vs. Cross-Input Aggregation

The Taproot upgrade introduced a form of key aggregation that allows multiple participants in a single transaction to aggregate their individual public keys into a *single* group key, which is then indistinguishable from a standard public key on the timechain. For this scheme to work, all participants must cooperate to produce a single signature that validates the aggregate key. The key detail is that this process is handled on the client side and does not require a change to the Bitcoin protocol's consensus rules.

CISA, by contrast, is a more ambitious proposal. It allows for the aggregation of signatures from multiple, distinct inputs within a single transaction, even if those inputs are controlled by different participants with different keys. This form of aggregation requires a fundamental change to Bitcoin's consensus rules, which necessitates a soft fork.

The deliberate decision by the Bitcoin developer community to activate Schnorr signatures and key aggregation with Taproot while explicitly postponing CISA reveals a core principle of Bitcoin development: a cautious, stepwise approach that prioritizes security and stability. The rationale for the exclusion of CISA was that its interaction with other protocol features, and that solutions to these interactions were "still in flux" at the time of Taproot's development. This exemplifies the community's commitment to thorough research and the resolution of all technical challenges before a consensus-level change is proposed and implemented.

4.1.2 A Technical Deep Dive

The fundamental principle of CISA is to condense the signature data of a transaction. Instead of a one-to-one relationship between each transaction input and its corresponding signature, CISA allows a single signature to validate multiple inputs. The process is a direct application of the linearity property of Schnorr signatures, which permits a single aggregate signature to be mathematically derived from a set of individual signatures. A node can then verify this single aggregate

signature in a single operation, a process that is far more efficient than validating each signature independently. This not only saves on-chain space but also reduces the computational load on nodes, improving the overall scalability of the network.

Technical Transaction Walkthrough: Alice's UTXOs To illustrate the mechanism, consider a user who wishes to spend two Unspent Transaction Outputs (UTXOs) that he controls. Both UTXOs are of the P2TR (Pay-to-Taproot) type. Under the current Bitcoin protocol, if he creates a transaction to spend both UTXOs via a keypath spend, she must include one 16-vbyte Schnorr signature for each input. The total signature data in this transaction would therefore be 32 vbytes, in addition to other transaction data such as the 36-vbyte outpoint for each UTXO.

With CISA, the process would be streamlined. Ideally, any network node, or even the user herself, could aggregate the signatures from her two UTXOs. Alice would then be able to produce a single 16-vbyte MuSig-style aggregate signature that corresponds to the single public keys. This single signature would cryptographically prove that Alice controls the private keys for both of the original public keys. The resulting transaction would still require the inclusion of other data, but its signature footprint would be reduced from 32 vbytes to 16 vbytes. While described as a "modest reduction" in overall transaction size, this optimization is particularly impactful for transactions with a high number of inputs, such as CoinJoins, where signature data can represent a substantial portion of the total transaction size.

4.1.3 Variants of Aggregation

The CISA proposal is not a singular concept but rather a family of potential implementations with different trade-offs. The two most prominent variants are full aggregation and half aggregation.

Full Aggregation Full aggregation is an *interactive* protocol where all participants must actively collaborate to produce a single, compact signature. The size of the resulting signature is *constant* (64 bytes), regardless of the number of inputs or signers involved. This method offers the maximum possible space savings on-chain, as multiple signatures are collapsed into a single, standard-sized signature. It also offers the greatest potential for validation efficiency by drastically reducing the number of cryptographic operations required to verify a block. However, the interactive nature of this approach presents a significant practical and engineering challenge, as all participants must be online and in communication during the signing process.

Half Aggregation Half aggregation presents a more pragmatic, *non-interactive* solution. In this approach, each signer produces a standard Schnorr signature, but only the s-values of these signatures are aggregated, while the R-values (nonce commitments) remain separate. The aggregation can be performed non-interactively by a third party, such as a broadcasting node or a miner, which is a major advantage over the full aggregation model. The resulting aggregate signature is not a single 64-byte signature but has a size of $32 \cdot n + 32$ bytes, where n is the number of inputs. While the space savings are less than with full aggregation, the simplicity and non-interactive nature of half aggregation make it a more viable candidate for immediate implementation. A draft BIP for half-aggregation of BIP 340 signatures has been written, and a formal specification has been developed using hacspec, a language designed for computer-aided formal proofs.

4.1.4 Consensus Challenges

Beyond the cryptographic security model, CISA faces significant implementation and consensus challenges. As a change to the fundamental rules of the Bitcoin protocol, CISA would require a

soft fork. Activating a soft fork necessitates overwhelming support from the network, typically requiring over 95% of miners to signal their readiness for the change.

One of the most complex engineering challenges relates to mempool management, particularly for *block-wide half-aggregation*. In a naive implementation, if a block with an aggregated signature is *reorged* out of the best chain, the individual transactions within that block are no longer recoverable, as they are not explicitly stored in the aggregated signature. This breaks the current mempool behavior, which assumes that transactions can be recovered from a reorged block and rebroadcast to the network. A solution would require new mechanisms, such as a *reorg-pool*, to temporarily retain transactions until a block is considered sufficiently secure.

Additionally, the interaction between CISA and other protocol features, such as adaptor signatures used in Layer 2 protocols like the Lightning Network, must be carefully considered. Naive implementation of block-wide half-aggregation could render adaptor signatures unusable, which would break certain Layer 2 functionality. The development community must resolve these complex interdependencies before a CISA soft fork can be proposed.

The development of CISA is a prime example of the decentralized, consensus-driven Bitcoin Improvement Proposal (BIP) process. Anyone can propose a BIP, and ideas are subject to years of discussion and debate on mailing lists and forums to achieve "rough consensus" before they can be formalized. The draft BIP for half-aggregation is a major topic of ongoing discussion among developers, who are actively exploring its trade-offs and open questions, while recent research on interactive schemes is about to take full aggregation a step forward.

4.1.5 Impact on the Bitcoin Ecosystem

On-Chain Scalability The most direct and tangible benefit of CISA would be an increase in Bitcoin's on-chain economic efficiency. By reducing the data required for transaction signatures, CISA would lower the cost of multi-input transactions. The half-aggregation proposal, for example, is estimated to offer a 20.6% reduction in transaction bytes for a historically average transaction. For transactions with a high number of inputs, such as Coinjoins or Payjoins, this would make the per-participant transaction fees moderately cheaper. This fee reduction directly translates to a more efficient use of block space, allowing more transactions to be included in each block and thus providing a modest increase in the network's on-chain throughput.

The implications of this efficiency gain extend beyond simple numbers. The small reduction in transaction size serves as a strategic economic catalyst. By making privacy-enhancing transactions less expensive, CISA lowers the barrier to entry for users who prioritize confidentiality. This reduction in cost incentivizes and normalizes the use of collaborative transactions like CoinJoin, which in turn strengthens the privacy of all network participants. A seemingly minor technical optimization thus has the potential to trigger a significant behavioral shift, fostering a more privacy-conscious ecosystem and reinforcing Bitcoin's core properties in the long term.

Privacy Contrary to what one might think, CISA by itself is not a magic bullet for privacy. In fact, CISA provides no direct on-chain anonymity improvement, it only seeks to compress signatures for efficiency. Any privacy gains would be indirect, for example, cheaper CoinJoins thanks to lower fees, which is outside the protocol change itself.

Focusing on the potential downsides, CISA could complicate privacy in several ways as well: by reducing the number of signatures visible in a transaction, CISA might initially confuse certain heuristics used by blockchain analysts. However, analysts will quickly adapt by developing new heuristics for aggregated signatures. In fact, in particular the *common-input heuristic* might become even stronger in the beginning of full-agg adoption: because full-agg requires coordination, a transaction that successfully used an aggregated signature is likely to have been crafted by a single entity controlling all inputs. Chain surveillance companies already assume all inputs are one owner unless there are obvious CoinJoin patterns; with CISA, an aggregated signature could

reinforce that assumption, as it suggests the inputs were signed in one go by cooperating keys. To prevent this it is paramount to have a robust signature scheme ready which allows collaborative transactions, such as PayJoin and CoinJoin, to adopt CISA quickly after deployment. Only when this is the case multi-user full-agg can spread quickly which can then prevent chain analysis from flagging full-agg multi-input transactions as definitely single-user.

The drive for CISA is not solely a technical endeavor. The importance of transaction privacy is particularly pronounced for individuals living under authoritarian regimes, where financial data can be used for surveillance and suppression. The Human Rights Foundation's involvement in promoting research on CISA in [Jah25] underscores the proposal's significance in the broader context of digital freedom and human rights. The technical development of CISA is thus interwoven with political and legal considerations, as the Bitcoin community works to preserve financial sovereignty and innovation in an environment of increasing regulatory scrutiny and legal threats to open-source developers. CISA is a microcosm of the ongoing struggle to build a more private and decentralized financial system.

4.1.6 Future Outlook

While CISA offers only a modest increase in on-chain transaction throughput, its indirect effects on user behavior and network dynamics could be profound. By reducing the economic penalty for privacy, CISA has the potential to normalize the use of collaborative transactions, leading to a more private and fungible Bitcoin network.

The successful implementation of CISA hinges on resolving several open technical questions, including the complex interaction with adaptor signatures and the development of robust mempool management solutions for block-wide aggregation. Ultimately, the activation of CISA will depend on achieving broad consensus within the developer community and obtaining the necessary support from miners and nodes to activate a soft fork.

In summary, CISA holds the potential to be one of the most impactful protocol upgrades since Taproot. Its ability to enhance user privacy, strengthen fungibility, and reduce transaction fees could fundamentally improve the user experience and solidify Bitcoin's core properties in the long term.

4.2 DahLIAS: a secure scheme for full-agg

In this context, the **DahLIAS** (Discrete Logarithm-Based Interactive Aggregate Signature) protocol, introduced in [NRS25], is positioned as the first solution to the full aggregation open problem. It is the first *interactive aggregate signature* (IAS) scheme with constant-size signatures that uses the discrete logarithm mathematics already present in Schnorr based protocols.
The scheme achieves the following key properties:

- **Constant-size signatures:** a `DahLIAS` signature has the same form as a standard Schnorr signature, a pair (R, s), regardless of the number of signers n.
- **Two-round protocol:** the interactive signing process requires only two communication rounds, the first of which can be pre-processed without the need to know the messages to be signed, making it efficient.
- **Efficient verification:** the verification time for a signature created by n signers is dominated by a single multi-scalar multiplication of size $n + 1$. This process is asymptotically twice as fast as batch verification of n individual Schnorr signatures.
- **Key tweaking compatibility:** The protocol is designed to be compatible with *key tweaking*.

4.2.1 Interactive Aggregate Signature

This part explores the relation between *interactive multi-signature* (IMS) schemes (where all signers sign the same message m) and IAS schemes. A two-round multi-signature scheme IMS consists of the following algorithms.

- The key generation algorithm `KeyGen` takes no input and returns a secret/public key pair (x, X).
- The interactive signature algorithm (`Sign`, `Coord`, `Sign'`, `Coord'`) consists of two algorithms `Sign` and `Sign'` run by signers and two algorithms `Coord` and `Coord'` run by the coordinator:
  - `Sign` takes no input and returns a first-round signer output out_i and a signer state st_i.
  - `Coord` takes a list of signer first-round outputs $(out_i)_{i\in[n]}$ and returns a session context ctx and a coordinator state st.
  - `Sign'` takes a secret key x_i, a signer state st_i, a list of all signers' public keys L, a message m to sign, and a session context ctx and returns a second-round signer output out'_i.
  - `Coord'` takes a coordinator state st and a list of second-round signer outputs $(out'_1, \ldots, out'_n)$ and returns a signature σ.
- The verification algorithm `Ver` takes a list of public keys $L = (X_i)_{i\in[n]}$, a message m, and a signature σ and returns true if the signature is valid and false otherwise.

Bellare and Neven informally suggested to turn an IMS scheme into an IAS by setting the message in the IMS scheme to the tuple of all public key/message pairs $((X_i, m_i)_{i\in[n]})$ of the signers of the IAS scheme [BN06]. Importantly, a safeguarding condition is needed in the second round signing algorithm, which checks that the input list L contains the signer's public key X_i *exactly once* together with the correct input message m_i. If the check is omitted then the resulting scheme is not even EUF-CMA-secure. This is necessary for the resulting IAS scheme to be secure even if the scheme is not unrestricted: it cannot produce a signature for a list L containing duplicate public keys.

The ROS attack here actually exploits the fact that, without this verification, the output does not depend on input message m_i. The adversary, on input the honest party's public key X, selects two distinct messages m_1 and m_2 and can perfectly emulate the behavior of the honest signer in a "phantom" session, with different message by simply copying the outputs out_1 and out'_1. Indeed, by copying the output out_1 of `Sign`, the adversary implicitly defines the state st_2 of the honest signer in the "phantom" session as the state st_1 in the real session. Since the output of algorithm `Sign'` does not depend on the input message, $out'_2 = out'_1$ is the correct answer oracle `Sign'` would return on input (m_2, ctx). Hence, σ is a valid signature for L. Since the adversary actually never queried `Sign'` for message m_2, this is a valid forgery breaking EUF-CMA security of IAS.

For this reason, the `DahLIAS` scheme implements a check in the second round of the signing phase.

4.2.2 The aggregation scheme

The `DahLIAS` signing protocol is a two-round interactive process involving n signers and an untrusted coordinator. An *interactive aggregate signature* (IAS) scheme allows n signers, each with their own key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and message m_i, to jointly produce a single short signature that proves m_i was signed under pk_i for every $i \in \{1, \ldots, n\}$.

Setup and KGen. Let $((\mathbb{G}, +), q, G)$ be the group description and H_{non} and H_{sig} hash functions. Each user $i \in \{1, \ldots, n\}$ has key pair $(x, xG) \in \mathbb{Z}_q \times \mathbb{G}$.

First Round

Signer. Each signer $i \in [n]$ generates two secret nonces, $r_{1,i}, r_{2,i}$, and computes the corresponding public nonces $R_{1,i} := r_{1,i}G$ and $R_{2,i} := r_{2,i}G$; then stores its state $st_i := (r_{1,i}, r_{2,i}, R_{2,i})$ and sends its first-round output $out_i := (R_{1,i}, R_{2,i})$ to the coordinator.

Coordinator. After collecting the triples (X_i, m_i, out_i) from all signers, the coordinator aggregates the public nonces, computing $R_1 := \sum_{i=1}^{n} R_{1,i}$ and $R_2 := \sum_{i=1}^{n} R_{2,i}$. It defines the session context ctx as the ordered list

$$ctx \;:=\; (R_1, R_2, ((X_j, m_j, R_{2,j}))_{j\in[n]}), \tag{4.1}$$

computes the aggregate challenge coefficient $b := \mathsf{H}_{non}(ctx)$ and keeps the actual aggregate nonce $R := R_1 + b \cdot R_2$. Finally sends the session context ctx to all signers.

Round 2

Validation check. Upon receiving the context $ctx = (R_1, R_2, ((X_j, m_j, R_{2,j}))_{j\in[n]})$, each signer performs a series of *critical validation checks* before proceeding: it verifies that its own second-round public nonce $R_{2,i}$ appears *exactly once* in the list of second-round nonces within ctx. Then verifies that the public key/message pair associated with its nonce $R_{2,i}$ in ctx matches its own public key X_i and message m_i. If the checks fail, the signer aborts the protocol and returns an error. Otherwise if the checks succeed, the signer extracts the list of key/message pairs $L := ((X_j, m_j)_{j\in[n]})$ from the session context. The signer computes the aggregated nonce $R = R_1 + bR_2$, the challenge $c_i := \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m_i)$ and its partial signature $s_i := r_{1,i} + br_{2,i} + c_i x_i$. Then sends its partial signature s_i to the coordinator.

Signature Aggregation. The coordinator receives all partial signatures $(s_1, \ldots, s_n)$ and returns the final aggregate signature $\sigma := (R, s)$, where $s := \sum_{i=1}^{n} s_i$.

$\mathsf{KGen}()$

$x \leftarrow\!\!\$\ \mathbb{Z}_q$
$X := xG$
return (x, X)

$\mathsf{Sign}()$ // signer i

$r_{1,i}, r_{2,i} \leftarrow\!\!\$\ \mathbb{Z}_q$
$R_{1,i} := r_{1,i}G,\ R_{2,i} := r_{2,i}G$
$out_i := (R_{1,i}, R_{2,i})$
$st_i := (r_{1,i}, r_{2,i}, R_{2,i})$
return out_i, st_i

$\mathsf{Coord}(((X_i, m_i, out_i))_{i\in[n]})$

$(R_{1,i}, R_{2,i}) := out_i, i \in [n]$
$R_1 = \sum_{i=1}^{n} R_{1,i}, R_2 = \sum_{i=1}^{n} R_{2,i}$
$ctx := (R_1, R_2, ((X_j, m_j, R_{2,j}))_{j\in[n]})$
$b := H_{\mathsf{non}}(ctx)$
$R := R_1 + b \cdot R_2$
$st := R$
return (ctx, st)

$\mathsf{Sign}(x_i, st_i, m_i, ctx)$

assert $\exists!\, u \in [n]$:
$R_{2,u} = r_{2,i}G \wedge (X_u, m_u) = (X_i, m_i)$
$r_i := r_{1,i} + br_{2,i},$
$b := H_{\mathsf{non}}(ctx)$
$R := R_1 + bR_2$
$c := \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m)$
$L := ((X_j, m_j))_{j\in[n]}$
$s_i := r_i + cx_i$
return $out'_i := s_i$

$\mathsf{Coord}(st, (out'_1, \ldots, out'_n))$

$R := st$
$(s_1, \ldots, s_n) := (out'_1, \ldots, out'_n)$
$s := \sum_{i=1}^{n} s_i$
return $\sigma := (R, s)$

$\mathsf{Vf}(L, \sigma)$

$((X_i, m_i))_{i\in[n]} := L$
assert $1_{\mathbb{G}} \notin \{X_i\}_{i\in[n]}$
$(R, s) := \sigma$
$c_i = \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m_i)$
return $sG \stackrel{?}{=} R + \sum_{i=1}^{n} c_i \cdot X_i$

Figure 4.1. The DahLIAS scheme

Correctness. Consider an execution of the protocol with n honest signers and an honest coordinator where each signer i has public key $X_i = x_iG$, message m_i, and nonces $(R_{1,i}, R_{2,i})$, the session context ctx, and partial signature s_i. Assume that no signer aborts. The signature returned by the coordinator is (R, s) where $R = \sum_{i=1}^{n} R_{1,i} + bR_{2,i}$ and $s = \sum_{i=1}^{n} s_i$. Each signer computed its partial signature as $s_i = r_{1,i} + br_{2,i} + c_ix_i$ where $c_i := \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m_i)$. Hence, one has

$$sG \;=\; \sum_{i=1}^{n} s_i \cdot G \;=\; \sum_{i=1}^{n} (r_{1,i} + br_{2,i} + c_ix_i) \cdot G \;=\; \sum_{i=1}^{n} R_{1,i} + bR_{2,i} + c_iX_i \;=\; R + \sum_{i=1}^{n} c_iX_i.$$

Since the coordinator is honest, no signer aborts unless some collision happens among $R_{2,i}$ values. The probability of this event is at most $n^2/2q \leq n^2/2^\lambda$. Hence, if at most N signers participate in any protocol execution, DahLIAS is $N^2/2^\lambda$-correct.

4.2.3 Security and Performance Analysis

DahLIAS is not merely a brilliant piece of engineering; it is built on robust cryptographic guarantees and formal security proofs. The protocol resists the rogue-key attack and, importantly, has been proven compatible with "key tweaking", a vulnerability that plagued other informal aggregate constructions derived from MuSig2. This highlights the difference between a provably secure construction and one that only works intuitively in a specific context.
From a performance standpoint, DahLIAS offers significant advantages:

- **Communication Rounds:** for signing, it requires two interactive rounds, similar to MuSig2. However, the first round can be completed before the messages to be signed are known, which improves flexibility and efficiency.
- **Verification Speed:** the verification of DahLIAS signatures is asymptotically twice as fast as schemes like partial Schnorr aggregation or the batch verification of individual signatures. Lower verification costs make it cheaper for full node operators to process transactions, which supports the decentralization of the network.
- **Signature Size:** the final signature is a constant 64 bytes in size. This drastically reduces the data overhead on complex transactions.

The research paper does not rely on ad-hoc analysis but introduces and applies rigorous formal security models.

- **co-EUF-CMA:** The *cosigners-aware* EUF-CMA ($\mathsf{co}-\mathsf{EUF}-\mathsf{CMA}$) model is a stronger security notion for interactive protocols. In this model, a signer can validate the entire list of co-signers, ensuring an "all-or-nothing" property. The adversary wins only if the aggregate signature is valid for a list L containing a pair (X, m) for which the full tuple (L, m) has never been queried to the signing oracle. `DahLIAS` natively achieves this stronger security notion.
- **EUF-CMA-TK:** The $\mathsf{EUF}-\mathsf{CMA}-\mathsf{TK}$, *unforgeability under chosen message attacks with key tweaking*, model extends the EUF-CMA model by allowing an adversary to request signatures on tweaked keys and to produce forgeries for tweaked keys. Unlike other schemes that fail in this model, `DahLIAS` is proven to be secure in $\mathsf{co}-\mathsf{EUF}-\mathsf{CMA}-\mathsf{TK}$, a feature that significantly distinguishes it.

Underlying Cryptographic Assumptions The security of `DahLIAS` is proven under the assumption of the *Algebraic One-More Discrete Logarithm* (AOMDL) in the random oracle model (ROM). AOMDL is a weaker variant of the well-studied OMDL assumption, which means that `DahLIAS` is more robust against potential attacks. Furthermore, the security proof of `DahLIAS` only requires the hash function H_{non} to be *collision-resistant*, a weaker and more common assumption than being a random oracle.

4.2.4 The ROS attack in DahLIAS schemes

This section presents the ROS attack applied on the `DahLIAS` schemes that allows signature aggregation [NRS25]. Actually, the attack can be applied to any variant of `DahLIAS` with the following properties:

1. Given key pairs $(x, xG) \in \mathbb{Z}_q \times \mathbb{G}$, a signature $(R, s) \in \mathbb{G} \times \mathbb{Z}_q$ is valid for $L = ((X_i, m_i))_{i\in[n]}$ if $sG = R + \sum_{i=1}^{n} X_i \cdot \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m_i)$.

2. The two-round protocol for computing the signature proceeds as follows:

   - first signing round: the i-th signer runs $(out_i, st_i) \leftarrow \mathsf{Sign}()$, sends output out_i to the coordinator, and keeps state st_i;
   - first coordinator round: on input $((X_i, m_i, out_i))_{i\in[n]}$, algorithm `Coord` simply sends $ctx = ((X_i, m_i, out_i))_{i\in[n]}$ to all signers;
   - second signing round: given the secret key x_i, the signer's state st_i, the message m_i, and the session context $ctx = ((\hat{X}_j, \hat{m}_j, \hat{out}_j))_{j\in[n]}$, the i-th signer computes its partial signature s_i as $s_i := r_i + c_i x_i$, where $c_i := \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m_i)$, $L := ((\hat{X}_j, \hat{m}_j))_{j\in[n]}$, and nonces
   $$r_i := f(x_i, m_i, st_i, ctx), \qquad R := \sum_{j=1}^{n} F(\hat{X}_j, \hat{m}_j, \hat{out}_j, ctx),$$
   where f and F are two functions such that
   $$F(X_i, m_i, out_i, ctx) = f(x_i, m_i, st_i, ctx) \cdot G \tag{4.2}$$
   One can think of $r_i = f(x_i, m_i, st_i, ctx)$ as the "effective" secret nonce of the i-th signer and of $F(X_i, m_i, out_i, ctx)$ as its "effective" public nonce.
   - second coordinator round: the final signature is (R, s) with $s = \sum_{i=1}^{n} s_i$.

3. Fix a signer key pair (x_i, X_i) and first-round output/state pair (out_i, st_i). Given x_i and out_i it is possible to find two distinct pairs of message and session context $(m_i^{(b)}, ctx^{(b)})$, $b \in \{0,1\}$, such that
   $$F(X_i, m_i^{(0)}, out_i, ctx^{(0)}) = F(X_i, m_i^{(1)}, out_i, ctx^{(1)}) \quad \text{and} \quad c_i^{(0)} \neq c_i^{(1)}, \tag{4.3}$$
   where $c_i^{(b)}$ is the challenge computed by the signer.

Hence, property (3) says that an adversary can close a signing session with a signer in two ways such that the signer computes the same effective secret/public nonce pair but two different challenges.

Note that Equation (4.2) ensures correctness of the IAS scheme: a signature computed following the protocol is valid since

$$\begin{aligned} sG &= \sum_{i=1}^{n} s_i G = \sum_{i=1}^{n} (r_i + c_i x_i) G = \sum_{i=1}^{n} f(x_i, m_i, st_i, ctx) G + \sum_{i=1}^{n} \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m_i) \cdot X_i = \\ &= \sum_{i=1}^{n} F(X_i, m_i, out_i, ctx) + \sum_{i=1}^{n} \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m_i) \cdot X_i = R + \sum_{i=1}^{n} X_i \cdot \mathsf{H}_{\mathsf{sig}}(L, R, X_i, m_i). \end{aligned}$$

Single nonce. These properties encompass the case of the *single-nonce* scheme where each signer simply draws $r_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and sets $st_i := r_i$ and $out_i := r_i G = R_i$ with functions f and F defined simply as

$$f(x_i, m_i, r_i, ctx) := r_i \quad \text{and} \quad F(X_i, m_i, R_i, ctx) := R_i.$$

Property (3) is satisfied. Assume that the signer has index $i = 1$: given $out_1 = R_1 = r_1 G$, fix an arbitrary public key X_2, arbitrary messages m_1 and m_2, and distinct value $R_2^{(0)} \neq R_2^{(1)}$, and for $b \in \{0,1\}$ let

$$ctx^{(b)} := ((X_1, m_1, R_1), (X_2, m_2, R_2^{(b)})).$$

Then $F(X_1, m_1, out_1, ctx^{(0)}) = F(X_1, m_1, out_1, ctx^{(1)}) = R_1$ and $c_1^{(b)} = \mathsf{H}_{\mathsf{sig}}(L, R^{(b)}, X_1, m_1)$ where $R^{(b)} = R_1 + R_2^{(b)}$, hence $c_1^{(0)} \neq c_1^{(1)}$ (except with negligible probability).

Naive two nonce. It also covers the case of the naive two-nonce scheme, where each signer draws $r_{i,1}, r_{i,2} \leftarrow\!\!\$\ \mathbb{Z}_q$ and sets $st_i := (r_{i,1}, r_{i,2})$ and $out_i = (R_{i,1}, R_{i,2}) := (r_{i,1}G, r_{i,2}G)$. Functions f and F are given by

$$f(x_i, m_i, (r_{i,1}, r_{i,2}), ctx) := r_{i,1} + b r_{i,2}, \quad F(X_i, m_i, (R_{i,1}, R_{i,2}), ctx) := R_{i,1} + b R_{i,2},$$

with $b := \mathsf{H}(ctx)$. Property (3) is satisfied as follows: given $out_1 = (R_{1,1}, R_{1,2})$, fix different messages $m_1^{(0)}$ and $m_1^{(1)}$ and let

$$ctx := ((X_1, m_1^{(0)}, out_1), (X_1, m_1^{(1)}, out_1)).$$

Then $(m_1^{(0)}, ctx)$ and $(m_1^{(1)}, ctx)$ are two distinct pairs such that

$$F(X_1, m_1^{(0)}, out_1, ctx) \;=\; F(X_1, m_1^{(1)}, out_1, ctx)$$

but the challenges are $\mathsf{H}_{\mathsf{sig}}(L, R, X_1, m_1^{(b)})$ which are different (except with negligible probability).

Building the attack

The attack requires an adversary $\mathcal{A}$ to open $\ell \geq \lambda$ concurrent signing sessions with an honest signer which has key pair $(x_1, X_1) \in \mathbb{Z}_q \times \mathbb{G}$.

1. Adversary requests for ℓ first round messages and the signer responds $out_{1,k}$ and keeps state $st_{1,k}$, $k \in [\ell]$.

2. For each session $k \in [\ell]$, the adversary find tho distinct pairs $((m_{1,k}^{(b)}, ctx_k^{(b)})$, $b \in \{0,1\}$, such that the effective nonce of the signer is

   $$R_{1,k}^{(b)} \;:=\; F(X_1, m_{1,k}^{(b)}, out_{1,k}, ctx_k^{(b)})$$

   and challenges $c_{1,k}^{(b)}$ satisfies $R_{1,k}^{(0)} = R_{1,k}^{(1)}$ and $c_{1,k}^{(0)} \neq c_{1,k}^{(1)}$.
   For convenience, let $R_{1,k} := R_{1,k}^{(0)} = R_{1,k}^{(1)}$.

3. The adversary $\mathcal{A}$ construct the multivariate polynomial

   $$\boldsymbol{\rho}(x_1, \ldots, x_\ell) \;:=\; \sum_{k=1}^{\ell} 2^{k-1} \cdot \mathbf{f}_k(x_k) \;=\; \sum_{k=1}^{\ell} 2^{k-1} \cdot \frac{x_k - c_{1,k}^{(0)}}{c_{1,k}^{(1)} - c_{1,k}^{(0)}}$$

   Recall that for each k and $b \in \{0,1\}$, $\mathbf{f}_k(c_{1,k}^{(b)}) = b$, and consider the coefficients ρ_i such that

   $$\boldsymbol{\rho}(x_1, \ldots, x_\ell) \;=\; \rho_0 \;+\; \sum_{k=1}^{\ell} \rho_k x_k.$$

4. Let m^* be the forgery message. The adversary sets

$$L' := ((X_1, m^*)), \qquad R^* := \sum_{k=1}^{\ell} \rho_k R_{1,k}, \qquad c^* := \mathsf{H}_{\mathsf{sig}}(L^*, R^*, X_1, m^*).$$

   Next, for each $k \in [\ell]$, let b_k be the k-th bit in the binary representation of $c^* + \rho_0$, so that $c^* + \rho_0 = \sum_{k=1}^{\ell} 2^{k-1} b_k$. For each k, set $c_{1,k} := c_{1,k}^{(b_k)}$. By construction of the polynomial $\boldsymbol{\rho}$, it holds

$$\sum_{k=1}^{\ell} \rho_k c_{1,k} = \boldsymbol{\rho}(c_{1,1}, \ldots, c_{1,\ell}) - \rho_0 = \sum_{k=1}^{\ell} 2^{k-1} b_k - \rho_0 = c^*.$$

5. The adversary then closes each session $k \in [\ell]$ with session context $ctx_k^{(b_k)}$ and obtains partial signatures s_k satisfying $s_k G = R_{1,k} + c_{1,k} X_1$. Then, $\mathcal{A}$ computes $s^* = \sum_{k=1}^{\ell} \rho_k s_k$ and returns L^* and $\sigma^* := (R^*, s^*)$.

The attack can be adapted to forge signatures for lists L^* also containing other public key/message pairs for which the adversary knows the corresponding secret key.

Validity. The adversary outputs a signature $\sigma^* = (R^*, s^*)$ valid for $L^* = ((X_1, m^*))$. For $k \in [\ell]$, let $r_{1,k}$ be the DLog of $R_{1,k}$, hence $s_{1,k} = r_{1,k} + c_{1,k} x_1$. Then

$$s^* G = \sum_{k=1}^{\ell} \rho_k s_k G = \sum_{k=1}^{\ell} \rho_k (r_{1,k} + c_{1,k} x_1) G = \sum_{k=1}^{\ell} \rho_k R_{1,k} + c^* X_1 = R^* + \mathsf{H}_{\mathsf{sig}}(L^*, R^*, x_1, m^*) X_1$$

Countermeasures

The `DahLIAS` protocol elegantly defends against this attack through the validation checks in its `Sign'` algorithm. The attack described above fails because the honest signer simply aborts the protocol and returns an error before producing any partial signature, in particular:

1. `assert #U = 1`: this check ensures that the honest signer's second-round nonce $R_{2,i}$ appears **exactly once** in the list provided by the coordinator. In the ROS attack, the nonce $R_{2,1}$ is duplicated. When the signer receives the request, this check fails immediately, causing the protocol to abort.

2. `assert` $(\hat{X}_u, \hat{m}_u) = (X_i, m_i)$: this check guarantees that the unique index 'u' that corresponds to the second-round nonce $R_{2,i}$ is associated with the correct public key/message pair. Even if an adversary could somehow bypass the first check, this second check ensures the signer is only signing for the message it expects.

These checks prevent the adversary from manipulating the session context to force the reuse of the effective nonce. For every signing session, the honest signer demands that its second-round nonce is uniquely identifiable within the session context and that it is correctly bound to its key and message. This syntactic constraint removes the ambiguity that the ROS attack exploits, making the protocol immune to this class of attacks.

4.3 Implementation and Future Directions

The Path to Implementation in Bitcoin

While DahLIAS provides a robust cryptographic solution to a long-standing problem, its adoption in Bitcoin is not immediate. The protocol is incompatible with Bitcoin's current consensus rules, and its integration would require a fundamental protocol change with a soft fork.

To initiate this process, the protocol's authors or community members would need to draft a Bitcoin Improvement Proposal (BIP). This document, which specifies the scheme in detail and considers its implications for implementation and consensus, requires community support to be adopted. The path from academic paper to real-world implementation is a collaborative, consensus-driven process that takes time and discussion.

Conclusions and Broader Implications

The detailed analysis presented in this treatise showcases the evolution of cryptographic thought, from the recognition of fundamental vulnerabilities (Rogue-key and ROS Attacks) to its resolution through increasingly efficient interactive protocols. The linearity of Schnorr signatures, while a source of potential vulnerability, is also the principle that has unlocked new solutions for scalability.

The emergence of MuSig and MuSig2 provided secure solutions for multi-signatures, but it was the DahLIAS protocol that took aggregation to the next level. DahLIAS solves a long-standing cryptographic challenge: full signature aggregation for multi-input transactions with constant size in pairing-free groups, solving an open problem and providing significant savings in space and verification speed. Its two-round protocol is meticulously designed to be efficient and resilient to attacks. The analysis of the ROS attack revealed its insidious nature and demonstrated how a naive design can be compromised by nonce. It is showed that the internal validation checks in the Sign' method of DahLIAS scheme are a simple yet effective defense that guarantees the security against this and other classes of forgery attacks. The protocol's robustness is further reinforced by its proven security in the $\mathsf{co}-\mathsf{EUF}-\mathsf{CMA}-\mathsf{TK}$ model, a fundamental guarantee for practical applications that utilize key tweaking.

By reducing the cost of complex transactions and making them indistinguishable from single-signature transactions, DahLIAS changes the economic incentive for users, encouraging them to adopt practices that enhance Bitcoin's privacy and fungibility. This example eloquently demonstrates how a mathematical abstraction, when rigorously applied, can have a tangible and meaningful impact on network behavior and value.

Bibliography

[Bac02] Adam Back. Hashcash - a denial of service counter-measure. http://www.hashcash.org/papers/hashcash.pdf, 2002.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International conference on the theory and application of cryptology and information security*, pages 435–464. Springer, 2018.

[BLL+22] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in) security of ros. *Journal of Cryptology*, 35(4):25, 2022.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399, 2006.

[BNPS03] Bellare, Namprempre, Pointcheval, and Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, 2003.

[CGKN21] Konstantinos Chalkias, François Garillot, Yashvanth Kondi, and Valeria Nikolaenko. Non-interactive half-aggregation of eddsa and variants of schnorr signatures. In *Cryptographers' Track at the RSA Conference*, pages 577–608. Springer, 2021.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203. Springer, 1983.

[CP92] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Annual international cryptology conference*, pages 89–105. Springer, 1992.

[CZ22] Yanbo Chen and Yunlei Zhao. Half-aggregation of schnorr signatures with tight reductions. In *European Symposium on Research in Computer Security*, pages 385–404. Springer, 2022.

[Dai98] Wei Dai. b-money. http://www.weidai.com/bmoney.txt, 1998.

[DEF+19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1084–1101. IEEE, 2019.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1993.

[Fin04] Hal Finney. Reusable proofs of work (rpow). https://nakamotoinstitute.org/finney/rpow/, 2004.

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 63–95. Springer, 2020.

[HS90] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990.

[Jah25] Fabian Jahr. Cross-input signature aggregation for bitcoin. Cisa research paper, Human Rights Foundation, March 2025. HRF CISA Fellowship.

[JLS25] Antoine Joux, Julian Loss, and Giacomo Santato. Dimensional erosion: Improving the ros attack with decomposition in higher bases. *IACR Cryptol. ePrint Arch.*, 2025:306, 2025.

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

[Nak08] Satoshi Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin.org/bitcoin.pdf-(: 17.07. 2019)*, 9:15, 2008.

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: Simple two-round schnorr multi-signatures. In *Annual International Cryptology Conference*, pages 189–221. Springer, 2021.

[NRS25] Jonas Nick, Tim Ruffing, and Yannick Seurin. Dahlias: Discrete logarithm-based interactive aggregate signatures. *Cryptology ePrint Archive*, 2025.

[NRSW20] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1717–1731, 2020.

[RCK+25] Bruno MF Ricardo, Lucas C Cardoso, Leonardo T Kimura, Paulo S Barreto, and Marcos A Simplicio Jr. Introducing two ros attack variants: breaking one-more unforgeability of bz blind signatures. *Cryptology ePrint Archive*, 2025.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[Sch01] Claus Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In *International Conference on Information and Communications Security*, pages 1–12. Springer, 2001.

[Sza05] Nick Szabo. Bit gold. https://bitcoinstan.io/prehistory/doc/1998_2.pdf, 2005.

[Wag02] David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

Appendix A

Wagner's birthday problem

Recall that the *classic birthday problem* states that given two lists, L_1 and L_2, of λ-bit values chosen uniformly and independently at random, find a pair of values, $x_1 \in L_1$ and $x_2 \in L_2$, such that $x_1 + x_2 = 0 \pmod{q}$. This problem is shown to be equivalent to finding a matching pair, $x_1 = x_2$.
The problem is famously well-understood, with a known optimal complexity of $\mathcal{O}\left(2^{\lambda/2}\right)$ time and space, assuming the lists are of a favorable size. This "square-root time" algorithm has numerous applications, most notably in finding collisions for cryptographic hash functions. For an λ-bit hash output, an adversary is expected to require roughly $2^{\lambda/2}$ operations to find a collision.

The k-list birthday problem

This foundational understanding of the birthday problem has led researchers to explore a generalized form, introduced in the Wagner's work [Wag02] as the **k-sum problem**. This generalization extends the classic problem from two lists to an arbitrary number k of lists.

Given k lists, $L_1, \ldots, L_k$, of elements drawn uniformly and independently at random from $\{0,1\}^\lambda$, the objective is to find a set of values, $x_1 \in L_1, \ldots, x_k \in L_k$, such that their XOR sum is zero: $x_1 + x_2 + \ldots + x_k = 0 \pmod{q}$. The paper notes that a solution to this problem is highly probable provided that the product of the list sizes satisfies $|L_1| \times \cdots \times |L_k| \gg 2^\lambda$, which is a natural extension of the birthday paradox. However, the true challenge lies not in the existence of a solution, but in explicitly and efficiently finding one.

The k-sum problem is a generalization of the classical one, stated as follows.

Definition A.0.1 (**Generalized Birthday Paradox**). *Given k lists $L_1, \ldots, L_k$ of elements drawn uniformly and independently at random from $\{0,1\}^\lambda$, find $x_1 \in L_1, \ldots, x_k \in L_k$ such that $x_1 + x_2 + \ldots + x_k \equiv 0 \pmod{q}$.*

The problem is most tractable when one can "freely" extend the size of the lists, a common scenario in cryptanalysis where the lists are populated by outputs from a random oracle, i.e. a hash function. The problem's inherent difficulty lies in its multi-dimensional nature, a feature that makes naive approaches prohibitively expensive.

Theoretic Lower Bounds

An information-theoretic lower bound for the k-sum problem can be established by considering the expected number of solutions. A solution to the k-sum problem exists with high probability if the product of the list sizes, $|L_1| \times \cdots \times |L_k|$, is on the order of 2^λ. Assuming equal list sizes, this implies that the size of each list must be approximately $2^{\lambda/k}$. Therefore, the computational

complexity of the problem cannot be less than the size of the lists themselves, leading to a weak lower bound of $\mathcal{O}\left(2^{\lambda/k}\right)$.

However, the most obvious algorithmic approaches for $k > 2$ do not come close to this theoretical lower bound. A straightforward method involves generating pairs of sums from subsets of the lists, which still results in a complexity of $\mathcal{O}\left(2^{\lambda/2}\right)$. This creates a substantial gap between the theoretical lower bound and the known algorithmic upper bound.

The discovery of a more efficient attack directly lowers the security benchmark, forcing designers to re-evaluate their parameters and assumptions.

The 4-List Algorithm

To provide a clear explanation, Wagner introduces the case $k = 4$, which can be easily generalized using the binary tree structure. Wagner uses XOR $\oplus$ properties to show the algorithm, but the approach can be easily extended by changing the *join* operator [Wag02]. The goal is thus to find values $x_1, \ldots, x_4$ from lists $L_1, \ldots, L_4$ such that $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$.

The core idea is to break the problem into two subproblems: finding values that sums to zero in their least significant bits, and then finding a match between these intermediate results. The process unfolds as follows:

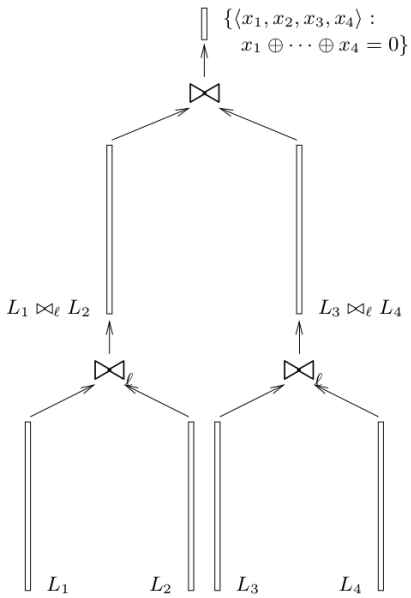

Figure A.1. Representation of the algorithm for the 4-sum problem. [Wag02]

List Extension: All four initial lists, $L_1, \ldots, L_4$, are extended to a size of approximately 2ℓ, where ℓ is a parameter to be determined later.

Intermediate List Generation: Two intermediate lists, L_{12} and L_{34}, are generated.

- L_{12} is created by taking pairs (x_1, x_2) from $L_1 \times L_2$ and computing $x_1 \oplus x_2$. Only those pairs are kept for which the least significant ℓ bits are zero, i.e., $low_\ell(x_1 \oplus x_2) = 0$. This is accomplished using a generalized join operator $\bowtie$.
- Similarly, L_{34} is generated from $L_3 \times L_4$ by finding pairs (x_3, x_4) where $low_\ell(x_3 \oplus x_4) = 0$.

Final Join and Solution: The algorithm then searches for a match between the values in L_{12} and L_{34}. A standard join operation is used to find a pair of values, one from each list, that are equal. Due to the properties of XOR, if $x_1 \oplus x_2 = x_3 \oplus x_4$, then it necessarily follows that $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$, which yields a solution to the 4-sum problem.

Complexity. For random values, the probability that $low_\ell(x_1 \oplus x_2) = 0$ is $1/2^\ell$. The expected size of the intermediate lists, L_{12} and L_{34}, is given by

$$\mathbb{E}[|L_{12}|] \;=\; |L_1| \times |L_2|/2^\ell \;=\; 2^{2\ell}/2^\ell \;=\; 2^\ell,$$

where the initial list sizes are $|L_1| = |L_2| = 2^\ell$. The final step is finding a match between L_{12} and L_{34}, which occurs with probability $2^\ell/2^\lambda$ and thus the expected number of elements in common

$$|L_{12}| \times |L_{34}|/2^{\lambda-\ell}$$

is at least 1 if $\ell \geq \lambda/3$. So setting $\ell = \lambda/3$, a solution to the problem is expected with non-trivial probability. By setting $\ell = \lambda/3$, the size of all lists and the number of operations are bounded by $2^{\lambda/3}$, which yields to a time and space complexity $\mathcal{O}\left(2^{\lambda/3}\right)$, a significant improvement over the naive $\mathcal{O}\left(2^{\lambda/2}\right)$ approach.
This tree-based approach can be generalized for larger values of k (specifically, powers of two), leading to an algorithm with a sub-exponential time complexity of $\mathcal{O}(k \cdot 2^{\lambda/(1+\lfloor \lg k \rfloor)})$ [Wag02].

Reduction to Subset-Sum

It is interesting to note that if the adversary were allowed to select an arbitrary subset of coefficients c_i (i.e., by setting an arbitrary subset of the components of $\boldsymbol{\rho}_{\ell+1}$ to zero), the decision task would be reduced to the classic problem of the *sum of subsets*. Specifically, identify each c_i with an integer a_i and consider a target value T corresponding to the required sum (or linear combination) that makes the attack succeed; asking whether there exists a choice of indices for which the sum of the selected c_i equals T is precisely an instance of subset-sum.
The subset-sum decision problem is well-known to be NP-complete (and NP-hard in the optimization variants), hence the unstructured version of the adversary's selection problem is computationally intractable in the worst case.

Recall the standard decision formulation:

Definition A.0.2 (Subset-Sum Problem)**.** *Given a finite set of integers $S = \{a_1, \ldots, a_n\}$ and a target integer T, decide whether there exists a subset $I \subseteq \{1, \ldots, n\}$ such that*

$$\sum_{i \in I} a_i = T.$$

Theorem A.0.1. *If the adversary is allowed to freely set entries of $\boldsymbol{\rho}_{\ell+1}$ to zero, the resulting decision problem reduces to Subset-Sum.*

Proof. Let (S, T) be an arbitrary instance of Subset-Sum, with $S = \{a_1, \ldots, a_n\}$. Construct an instance of the adversary's problem by setting $c_i = a_i$ for $1 \leq i \leq n$. The adversary succeeds if it can choose a subset of the c_i whose sum equals T. This choice corresponds exactly to defining indicator variables $\rho_i \in \{0,1\}$, where $\rho_i = 1$ if c_i is included and $\rho_i = 0$ otherwise. Thus, the condition

$$\sum_{i=1}^{n} \rho_i c_i = T$$

is satisfied if and only if the original Subset-Sum instance admits a solution. This mapping is computable in polynomial time, and hence establishes a reduction from Subset-Sum to the adversary's selection problem. □

Since Subset-Sum is NP-complete, it follows that the unrestricted version of the adversary's problem is NP-hard. This reduction clarifies that the hardness arises precisely from the ability to select arbitrary subsets of coefficients. By contrast, Wagner's formulation deliberately restricts this freedom: the adversary must pick *exactly one* element from each of k disjoint lists. The resulting k-sum problem is strictly more structured, and this structure is what enables Wagner's k-tree algorithm to achieve sub-exponential complexity. Thus, the NP-hardness of the general subset-selection task underscores the importance of Wagner's insight in reformulating the problem into a tractable variant.

Application to the ROS Attack

Wagner demonstrates how to formulate the ROS attack for $k = 4$ as an instance of the 4-sum problem over the additive group of $\mathbb{Z}_q$.

1. The attacker constructs a specific 4×4 matrix M, where its entries depend on four variables x_1, x_2, x_3, x_4.
2. The condition for this matrix to be singular (which is necessary for the attack to succeed) translates into the following equation:

$$\frac{\mathsf{H}(x_1,0,0)}{x_1} + \frac{\mathsf{H}(0,x_2,0)}{x_2} + \frac{\mathsf{H}(0,0,x_3)}{x_3} - \frac{\mathsf{H}(x_4,x_4,x_4)}{x_4} \equiv 0 \pmod{q}$$

   where F is a cryptographic hash function.
3. This is precisely an instance of the 4-sum problem, where the lists are constructed as follows:
   - L_1 contains values of the form $\mathsf{H}(x_1,0,0)/x_1$ for various choices of x_1.
   - L_2 contains values of the form $\mathsf{H}(0,x_2,0)/x_2$ for various choices of x_2.
   - L_3 contains values of the form $\mathsf{H}(0,0,x_3)/x_3$ for various choices of x_3.
   - L_4 contains values of the form $-\mathsf{H}(x_4,x_4,x_4)/x_4$ for various choices of x_4.

By applying the 4-list k-tree algorithm, Wagner reduces the complexity of the ROS attack from the expected $\mathcal{O}(q^{1/2})$ of a birthday attack to $\mathcal{O}(q^{1/3})$.