Blockchain Governance via Sharp Anonymous Multisignatures

Wonseok Choi
DGIST, Daegu, South Korea

Xiangyu Liu[1]
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Vassilis Zikas
Georgia Tech, Atlanta, GA, USA

Abstract

Electronic voting has occupied a large part of the cryptographic protocols literature. The recent reality of blockchains – in particular, their need for online governance mechanisms – has brought new parameters and requirements to the problem. We identify the key requirements of a blockchain governance mechanism, namely correctness (including eliminative double votes), voter anonymity, and traceability, and investigate mechanisms that can achieve them with minimal interaction and under assumptions that fit the blockchain setting.

First, we define a signature-like primitive, which we term *sharp anonymous multisignatures* (in short, $\sharp$AMS) that tightly meets the needs of blockchain governance. In a nutshell, $\sharp$AMSs allow any set of parties to generate a signature, e.g., on a proposal to be voted upon, which, if posted on the blockchain, hides the identities of the signers/voters but reveals their number. This can be seen as a (strict) generalization of threshold ring signatures (TRS).

We next turn to constructing such $\sharp$AMSs and using them in various governance scenarios – e.g., single vote vs. multiple votes per voter. In this direction, although the definition of TRS does not imply $\sharp$AMS, one can compile some existing TRS constructions into $\sharp$AMS. This raises the question: What is the TRS structure that allows such a compilation? To answer the above, we devise templates for TRSs. Our templates encapsulate and abstract the structure that allows for the above compilation – most of the TRS schemes that can be compiled into $\sharp$AMS are, in fact, instantiations of our template. This abstraction makes our template generic for instantiating TRSs and $\sharp$AMSs from different cryptographic assumptions (e.g., DDH, LWE, etc.). One of our templates is based on chameleon hashes, and we explore a framework of lossy chameleon hashes to understand their nature fully.

Finally, we turn to how $\sharp$AMS schemes can be used in our applications. We provide fast (in some cases non-interactive) $\sharp$AMS-based blockchain governance mechanisms for a wide spectrum of assumptions on the honesty (semi-honest vs malicious) and availability of voters and proposers.

2012 ACM Subject Classification Security and privacy → Digital signatures

Keywords and phrases Blockchain, E-voting, Threshold Ring Signatures, Threshold Cryptography

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.5

Related Version *Full Version*: `https://eprint.iacr.org/2023/1881`

Funding Wonseok Choi and Xiangyu Liu were supported in part by Sunday Group, Inc.
Vassilis Zikas: Vassilis Zikas was supported in part by NSF Award No. 2448339, AnalytiXIN, and Sunday Group, Inc.

Acknowledgements This work was done when authors were at Purdue University and Georgia Tech.

[1] Corresponding author.

1 Introduction

Since the emergence of Bitcoin [53] in 2009, the world of cryptocurrencies and blockchain platforms has witnessed a surge in popularity. One of the distinguishing features of these blockchain platforms is their decentralized nature, wherein decision-making authority is distributed among various actors within the ecosystem.

There are two basic governance mechanisms: off-chain governance, as seen in Bitcoin and Ethereum, and on-chain governance, exemplified by projects like Algorand [17], Tezos [32], and EOS. While off-chain governance allows core contributors to work more seamlessly, it contradicts the philosophy of decentralization. Conversely, on-chain governance faces technical challenges, and this area of research is still relatively new and in its early stages.

Regardless of the governance mechanism used, blockchain platforms face a common vulnerability: community divisions, often resulting in hard forks. A hard fork arises when stakeholders disagree on a critical change, leading to some sticking with the current chain while others adopt the new one. Alternatively, competing updates may further fragment the community. These divisions can erode cohesion, devalue the platform, and jeopardize security. Security concerns are paramount, as a reduced number of resources supporting a fork can make it susceptible to attacks like 51% attacks.

On-Chain Governance and Voting Systems for Improvement Proposals

On-chain governance formalizes decision-making by embedding governance rules directly into the blockchain protocol, offering a transparent and verifiable process for implementing upgrades. At the heart of this system lies the voting mechanism, which determines how stakeholders collectively approve or reject proposals that shape the network's evolution. The integrity and security of this voting process are therefore critical: any vulnerability can undermine trust, distort consensus, and jeopardize the decentralized nature of the ecosystem.

Generally, there are three periods for on-chain governance: the posting, voting, and announcement periods. The developers submit their improvement proposals to the blockchain during the posting period. Then, in the voting period, eligible voters participate in the voting protocol to vote for their preferred proposals. Finally, the voting result is announced in the announcement period, and the most voted proposal is elected.

There are several foundational requirements for a robust voting system.

Correctness. The accuracy of the voting result is perhaps the most important property. A key goal here is to prevent double-voting, wherein a voter casts more than one vote on the same or different proposals. This act of multiple-voting contradicts the standard single-vote setting, wherein each voter is restricted to voting only once, irrespective of the proposal chosen. We emphasize that the permissibility of multiple-voting is contingent upon the specific application. In this context, double-voting includes instances where a malicious voter attempts to cast more than one vote for a single proposal, and such duplications are promptly nullified.

(Unconditional) Anonymity of Voters. A crucial factor to consider in blockchain systems is their immutability. Once signatures are uploaded, they remain permanently etched in the system. This implies that over time, the authorship of a linkable or traceable ring signature [50, 27] could potentially be unveiled due to inadequacies in the underlying computational assumptions.[2] Therefore, for optimal applicability in blockchain governance, the assurance of unconditional anonymity (i.e., the anonymity does not rely on any hardness assumptions) is one of the imperative features of a voting system in blockchain.

[2] Even assuming unconditionally anonymous linkable ring signatures [48, 18], the resulted protocol still suffers from a quadratic signature size.

Traceability. If a malicious voter attempts to vote twice, we need to have an efficient mechanism to trace its identity. Note that this traceability property is a relaxation/fallback of the above strong correctness to allow for more practical constructions. Indeed, traceability is irrelevant if double-voting is infeasible.

Receipt-freeness. It should be impossible to generate a proof that a voter indeed cast an intended vote for others to see. This property has been widely studied [2, 3, 10, 23, 24, 35, 37, 39, 40, 45, 46, 47, 52, 54, 59].

Cryptographic Mechanisms for Blockchain Voting/Governance Systems

In pursuing our objective to design a voting system that ensures traceability and unconditional anonymity, we encounter inherent challenges when leveraging traditional cryptographic tools such as signature schemes or Multi-Party Computations (MPC).

Linkable Ring Signatures (LRS). Linkable ring signatures (LRS) [50] are a prevalent tool in constructing e-voting systems. In LRS, a user can sign a message on behalf of a group/ring while maintaining anonymity. Moreover, if a user signs twice for the same message and on behalf of the same group, these signatures are "linked", making it evident that they originate from a single actual signer. This inherent linkability property of LRS plays a vital role in preventing double-voting. However, it's important to note that the link algorithm in LRS does not disclose the specific identity of the signer when two signatures are linked. Consequently, LRS does not fulfill the traceability property, implying that a malicious voter engaging in double-voting may go unpunished. This lack of traceability can undermine the system's balance and fairness, necessitating additional measures to ensure accountability and uphold the integrity of the voting process.

Traceable Ring Signature. An alternative approach is to employ traceable ring signatures [27], where the link algorithm establishes the link between signatures and discloses the signer's identity. However, unconditional anonymity is compromised in traceable ring signatures, as proved in [18]. Striking a balance between traceability and unconditional anonymity in ring signatures appears challenging, presenting a fundamental trade-off within this cryptographic context.

Multi-Party Computation. Multi-party computation (MPC) [64, 31, 9, 15] appears to be the ultimate solution to the above voting problem. MPC allows n parties to compute any given function on their inputs securely so that no malicious party (or coalition) can learn the inputs of other parties (privacy), and no party can affect the output any more than choosing their own input. MPC can directly be used to realize our voting functionality by having each voter submit their votes and output the appropriate tally. MPC-privacy (which can be information-theoretic [9, 15, 57]) will ensure unconditional anonymity; MPC-correctness (for the appropriate function) can ensure our above voting correctness property. Traceability is a more elusive goal, but it can also be achieved by so-called identifiable MPC [38] (which ensures that upon aborting the identity of a cheater is revealed).[3]

Unfortunately, despite its very general functionality, MPC is also not the right solution to our problem: For starters, information-theoretic MPC needs an honest majority of the parties [31], an assumption which is unrealistic in our setting.[4] Even if one is willing to

[3] In our blockchain governance application, we need a property which is stronger than identifiability, namely *public verifiability*, which informally ensures that an abort provides a cheating certificate that can be verified even by a non-MPC party later on, e.g., [4].

[4] Although there are solutions which replace the honest majority of parties assumption with an assumption on the resource distribution, e.g., honest majority of hashing-power or stake [28], they come at a high cost in terms of blockchain utilization – multiple on-chain rounds – which renders them mainly of theoretical interest.

resort to security with (identifiable) abort and no fairness,[5] we still need to implement Byzantine broadcast – which requires in the worst case a polylogarithmic in the party-set size – or use again several on-chain rounds. Even given broadcast, turning an MPC protocol identifiable above into one with guaranteed output delivery (which is needed in our application) would require restarting the computation whenever it aborts (and potentially removing the identified cheater); this would again yield a larger number of rounds which is undesirable.

From Signature Schemes to an Interactive Structure

The above discussion highlights the inherent challenge of satisfying all requirements simultaneously. However, despite this seemingly conflicting nature, there is a way to address it. In this work, we provide a positive answer by proposing a structured approach embedded in the signature generation process. We carefully define an interactive structure – a simple protocol – within the signature generation. This structured approach is a key innovation that enables us to design a signature scheme that simultaneously conceals individual identities while revealing the number of signers involved. This dual property is a critical advancement for secure and reliable e-voting systems. We call this new signature protocol Sharp Anonymous Multisignatures ($\sharp$AMS). The term "sharp" is used in analogy with complexity theory, as in $\sharp P$, indicating that the signatures output the number of valid signers rather than a mere validity bit. Meanwhile, "anonymous" denotes that the signers' identities remain hidden. Equipped with these properties, $\sharp$AMS is a perfect fit for e-voting systems, especially in the context of blockchain governance. We discuss the details of our contributions below.

1.1 Our Contributions

$\sharp$ Anonymous MultiSignatures

Our first and major contribution is proposing and formalizing a new concept of signing *protocol*, dubbed $\sharp$AMS that tightly meets the needs of blockchain governance. The protocol allows any set of parties to collaborate jointly and outputs unconditionally anonymous signatures. To compare with threshold ring signatures (TRS), $\sharp$AMS does not need the threshold and always generates a valid signature regardless of the number of parties, and the verification algorithm reveals the number of parties. Regarding the number of parties as a threshold, which may vary every signing, this can be seen as a strict generalization of TRS.

Generic Compiler from TRS with A Flexible Threshold

Despite the above separation of TRS and $\sharp$AMS, it turns out that several instantiations of TRS actually possess a *flexible threshold* property, i.e., these TRSs can change the threshold depending on the actual number of signers.

To characterize the class of TRSs that admit such a lifting to $\sharp$AMSs, we provide a generic template for TRSs that (1) abstracts many such "liftable" schemes, and (2) admits a generic compiler to transform to $\sharp$AMS. Several existing TRS constructions can be seen as instantiations of our template, which implies that those TRS schemes are more versatile than previously known.

[5] This is already a discount in security that should be avoided.

♯AMS Constructions from Chameleon Hashes

Using the above, we provide concrete ♯AMS schemes from (lossy) chameleon hashes in a black-box model – which covers the post-quantum case. In a nutshell, we propose the following types of constructions:

C1. A basic construction with three communication rounds. This construction achieves unconditional anonymity.

C2. A fault-tolerant variant of C1 – for an arbitrary number of corruptions – without any overhead. This construction achieves unconditional anonymity and public verifiability.

Voting Systems from ♯AMS Constructions

Since anyone can verify the number of signers from a ♯AMS signature, we can immediately turn the constructions into voting systems. Furthermore, we develop a *conditioned key generation paradigm* to enable the single-vote setting.

V1. A basic system implemented by C2. This allows multiple voting. The system can tolerate malicious voters (those who claimed to vote but later quit). Including the posting period and the announcement period, voting can be completed within two on-chain blocks.

V2. A round-optimal system implemented by C1 in the multiple-vote setting by leveraging one-time key generation. This system satisfies the "vote-and-go" property.

V3. A variant of V2 for the single-vote setting, based on the conditioned key generation paradigm. V3 requires one additional on-chain round when there are malicious users. However, if the maximum number of proposals is known in advance, the voting process still requires only two on-chain blocks, as in V1 and V2.

Notably, our voting systems possess several desirable properties, which are discussed in depth in the full version [19]. We briefly introduce these properties here:

On-Chain. The vote will be uploaded on the blockchain and cannot be altered further. All our systems achieve this property.

Vote-and-go. Voters can leave immediately after casting their vote (by sending their cryptographic information on the vote) without any interaction. V2 and V3 achieve this property.

Vote-Count Concealment. Voters remain unaware of the current vote count (and therefore of the votes of others) until the results are announced. V2 and V3 achieve this property.

▶ Remark 1 (On receipt-freeness). We would like to mention that our protocols do not satisfy receipt-freeness properties due to their simple structure. Any voting scheme that requires voters to choose their own randomness, like ours, is vulnerable to a well-known generic receipt-freeness attack [37]. In such an attack, a malicious voter can generate randomness using a one-way function and later provide the input to the one-way function to prove the source of the randomness. To resolve this problem, one would need to rely on a much more complex system (e.g., [37] assumes homomorphic encryption, an honest assumption on the authorities, and many other assumptions). While this is an intriguing and interesting open problem, we are focusing on proposing a new signature protocol, ♯AMS, and thus leave this problem for future research.

A Framework of Lossy Chameleon Hashes

As a by-product, we explore a framework of lossy chameleon hashes, including its relationship with existing cryptographic primitives (e.g., lossy identification and lossy encryption) and more concrete constructions from various assumptions. See Fig. 1 for details.

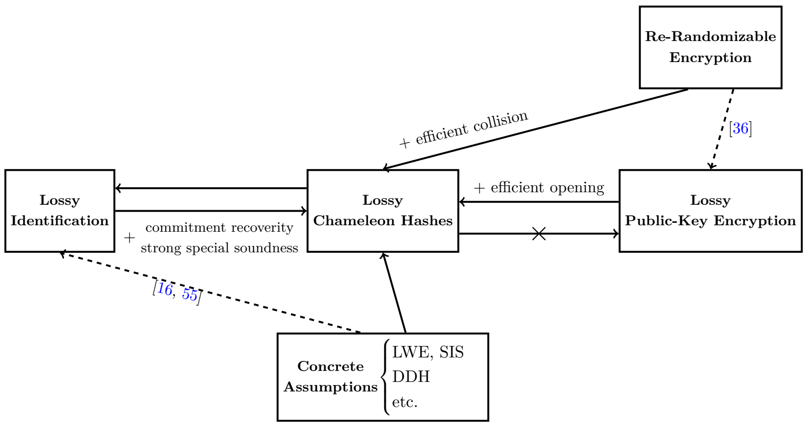

Figure 1 A framework for lossy chameleon hashes (solid arrows: shown in this work; dashed line: shown in previous works [16, 55, 36]).

1.2 Related Works

We refer to the full version [19] for more related works on multisignatures, threshold signatures, and ring signatures.

Threshold Ring Signatures. ♯AMS in this paper are highly related to threshold ring signatures (TRS) [11]. An (n, t)-TRS scheme allows t or more members to generate a ring signature together, and the actual signers remain anonymous. The verification algorithm in TRS will output either 0 or 1, indicating the validity w.r.t. the threshold t. Differently, by defining ♯AMS, we emphasize the property that the verification outputs exactly the credibility of the signature, i.e., how many users have participated in the generation. This holds even against malicious users who behave wantonly when signing. For example, a malicious user may quit at the middle or contribute senseless results in the signing process. Therefore, ♯AMS is strictly stronger than TRS.

There are many threshold ring signature schemes [49, 14, 63, 20] whose threshold t is changeable every signing. By adding t into the message to be signed, a TRS scheme with a flexible threshold can be tuned to a ♯AMS scheme.

Graded Signatures. Kiayias, Osmanoglu, and Tang [43] proposed the concept of graded signatures that allow a combiner/user to combine a set of different signatures w.r.t. the same message to a "consolidated" signature. Meanwhile, the consolidated signature leaks the actual number of signatures used to consolidate the graded signature and nothing else, which is the same as a ♯AMS scheme. However, graded signatures [43] have two phases in signing, which means that every participant has to sign by themselves and then pass their signature share to someone for consolidation. Due to such a definition, graded signatures cannot achieve unconditional anonymity. Therefore, ♯AMS provides more robust security.

Meanwhile, a trusted authority is required in graded signatures for generating global key pairs. The concrete instantiation in [43] relies on building blocks like structure-preserving signatures and Groth-Sahai proofs [33] and consequently cannot achieve post-quantum security.

Blockchain Governance and E-voting. Beck et al. [5], Pelt et al. [60], and Kiayias and Lazos [42] discussed the core properties for blockchain governance in multiple disciplines. Khan et al. [41], Venugopalan and Homoliak et al. [61], and Gersbach et al. [30] also focused on blockchain governance, especially based on decision-making processes and voting in terms of game-theoretical analyses.

Many papers in various fields have studied e-voting systems that combine blockchain and cryptographic primitives to realize distributed and decentralized voting systems. However, the cryptography community, which is the one that can technically contribute the most, has less interest in the topic. A few papers introduce e-voting systems derived from advanced cryptographic primitives such as linkable ring signatures. In terms of ring signatures, Lyu et al. [51] and Russo et al. [58] present systems that use blockchain, smart contracts, and ring signatures. Most of the other blockchains' e-voting systems rely not only on simple cryptographic primitives but also heavily on complex functionalities, e.g., smart contracts and zero-knowledge proofs. Note that our proposed solutions can simplify their approaches much since $\sharp$AMS provides not only privacy and authenticity but also signatures with the corresponding number of signers in the primitive level.

1.3 Technical Overview

This subsection briefly overviews the techniques and concepts used in this paper.

Formalization of $\sharp$AMS

We start from formalizing $\sharp$Anonymous Multisignature ($\sharp$AMS) and its security definitions. Suppose a group of t users, $\{U_i\}_{i \in G}$, wants to sign a message msg together, and the resulting signature σ reveals only the number of participants t and nothing else. We refer to t as the credibility of the signature.

The first issue is the anonymity property, which means that the identities of the t actual signers are hidden among the total of n users ($n \geq t$). Besides, as a (group) signature scheme, $\sharp$AMS should ensure unforgeability; namely, any adversary controlling fewer than t users cannot forge a valid signature on a new message that shows a credibility of t.

As we discussed above, if every signer U_i contributes their share using their own secret key and the signature is simply a concatenation of different shares (as in linkable/traceable ring signatures), it seems impossible to achieve both unconditional anonymity and traceability [18].[6] Therefore, we focus on interactive signing processes and introduce a moderator P in the signing protocol. To generate a $\sharp$AMS signature, each signer communicates only with P and not with other signers. It is P's responsibility to count the number of participants and finally produce a $\sharp$AMS signature. In this case, even a signer cannot de-anonymize a $\sharp$AMS signature (i.e., they remain unaware of the other participants). P can be a member of G or not, and is assumed to be honest. We believe this is a simple yet reasonable assumption; see more discussion in Section 3. Even if a malicious P leaks the quorum of signers later, their own identity would be revealed at the same time, and consequently, P would face punishment from the system.

[6] In the application of voting systems, if double voting is feasible, then traceability is necessary.

Generic Compiler from Threshold Ring Signatures

A $\sharp$AMS scheme directly implies a threshold ring signature (TRS) scheme since the verification algorithm Ver in $\sharp$AMS returns the real number of participants in the signing process, while Ver in TRS returns only a single bit. Therefore, the definition of $\sharp$AMS is stronger than that of TRS.

Nevertheless, we surprisingly notice that many TRS schemes (e.g., [49, 14, 13, 62, 34]) possess a *flexible threshold* property. Namely, the threshold t does not need to be fixed when generating the public/secret key pairs; it can be changed in every signing session. Motivated by this, we design a generic compiler that tunes any TRS scheme with a flexible threshold into a $\sharp$AMS scheme:

- A participant P among the signers is selected randomly as the moderator in $\sharp$AMS.
- P knows the quorum of signers, hence the number t. To generate a $\sharp$AMS signature on message msg, it starts the TRS signing protocol on the message $(\mathsf{msg}||t)$ and outputs the TRS signature $\tilde{\sigma}$ and t as the final $\sharp$AMS signature.
- In the verification of $\sharp$AMS, if $\tilde{\sigma}$ is valid in TRS, then the threshold t is returned.

Thanks to this compiler and the rich literature on TRS, we immediately obtain many $\sharp$AMS schemes from well-studied TRS schemes, based on the DL assumption [14], the RSA assumption [49], the SIS assumption [13], the code assumption [22], and others.

C1: Construction from (Lossy) Chameleon Hashes

A chameleon hash (CH) function [44] is a special hash function indexed by a hash key hk, which is associated with a trapdoor td. It takes as input two parts: a message m[7] and randomness r. On the one hand, given only the hash key, it is hard to find a collision. On the other hand, with the help of the trapdoor, finding collisions becomes easy.

Chameleon hashes can be converted into signature schemes via the well-known Fiat-Shamir paradigm [26], where hk and td serve as the public key and the signing key, respectively. To sign a message msg, the signer first randomly samples dummy values $\bar{m}$ and $\bar{r}$, and then uses its trapdoor to find a randomness r for m so that it collides with $(\bar{m}, \bar{r})$ on h, where $m = H(\mathsf{msg}, h)$ with $H(\cdot)$ a random oracle and h the chameleon hash value of $(\bar{m}, \bar{r})$.

Now, we extend this idea to the $\sharp$AMS setting. Suppose there are n users, and each user U_i has its own hash key hk_i and trapdoor td_i. We borrow the idea of the t-out-of-n zero-knowledge proof from [21] by Cramer, Damgård, and Schoenmakers. That is, given n random values m_i, the group of signers can find t random values r_i that create collisions. For the message msg, we design a $\sharp$AMS signature in the form of $\sigma = (t, m_1, \ldots, m_n, r_1, \ldots, r_n)$, where $m_1, \ldots, m_n$ are required to satisfy the following linear equations.

$$\begin{cases} a_{1,1}m_1 + a_{1,2}m_2 + \ldots + a_{1,n}m_n = u_1, \\ a_{2,1}m_1 + a_{2,2}m_2 + \ldots + a_{2,n}m_n = u_2, \\ \ldots \\ a_{t,1}m_1 + a_{t,2}m_2 + \ldots + a_{t,n}m_n = u_t. \end{cases} \tag{1}$$

Here, the coefficients $(a_{i,j})$ are public parameters, and $(u_1, \ldots, u_t)$ are the outputs of hash function $H(h_1, \ldots, h_n, \mathsf{msg}, t)$ (modeled as a random oracle), with $(h_1, \ldots, h_n)$ being the corresponding chameleon hash (CH) values. To satisfy Eq. (1), at least t trapdoors are necessary to determine the corresponding randomness r_i, thereby proving that at least t users have participated in the signing process.

[7] Note that the message m in chameleon hashes is different from the message msg to be signed in signature schemes. Here, we slightly abuse the terminology to keep consistent with previous works.

Instead of using the above linear equations, we can also set a polynomial g of degree at least $n-t$ and require that

$$g(0) = u, \quad g(i) = m_i \text{ for all } i \in [n], \text{ where } u = H(h_1, \ldots, h_n, \mathsf{msg}, t). \tag{2}$$

The security of the CH-based signature scheme, as well as our $\sharp$AMS scheme, is based on the collision-resistance property of the underlying chameleon hash scheme. However, the reduction in the security proof is not black-box since it relies on the forking lemma [7] to find a collision. Inspired by the ideas of lossy trapdoor functions [56] and lossy encryption [6, 36], we define lossy chameleon hashes (LCH) in this work to enable a black-box reduction in the security proof. An LCH scheme operates in two modes: the collision mode, which behaves as a standard CH, and the lossy mode, where a lossy hash key is used instead of a normal hash key, and the adversary (even if computationally unbounded) cannot find randomness for a random message that hashes to a previously chosen hash value.

We present an efficient instantiation of LCH from the learning with errors (LWE) assumption and the short integer solution (SIS) assumption. We also explore the relationship between LCH and other primitives, including lossy identification schemes [1], re-randomizable encryption, and lossy encryption [6, 36]; see Fig. 1 and more details in the full version [19].

C2: Fault-Tolerant Variant

For better application in blockchain governance, we consider a variant of the above construction in the faulty signer setting. A faulty signer may quit in the middle of the protocol or return malicious results to the moderator. To address this, we develop a fault-tolerant variant of our $\sharp$AMS scheme. This is possible because our (L)CH-based $\sharp$AMS scheme has "backward compatibility": namely, even if a voter quits in the middle, by exposing the identity of the faulty voter, the moderator can still output a signature with credibility $(t-1)$. More precisely, let F be the set of faulty signers. In the fault-tolerant $\sharp$AMS scheme, we tailor the signature into the form

$$\sigma' := \left(t, F, \{m_i, r_i\}_{i\in[n]\setminus F}, \{h_i\}_{i\in F}\right).$$

If $\{m_i\}_{i\in[n]}$ satisfy the linear equation (1) or (2), then the verification algorithm will output $(t-|F|)$ instead of t. Consequently, the voters in F will lose their anonymity as a cost of their malfeasance.

V1: Blockchain Governance via $\sharp$AMS Schemes

$\sharp$AMS directly implies a voting protocol since a $\sharp$AMS signature reveals the number of signers. The protocol consists of the following four periods using scheme C2:

1. The posting period, in which each developer publishes their improvement proposal on the blockchain.
2. The declaration period, in which each voter (a signer in $\sharp$AMS) signals their willingness to support a proposal by sending a hash value of a dummy message and randomness to the developer.
3. The signing period, in which the developer performs the signature algorithm based on the number of supporters. Specifically, developers compute all m_i values according to the received h_i and Eq. (1) or (2). They then return the m_i values to the supporters, who use their trapdoors to find collisions r_i and send them back to the developer. This helps the developer complete the generation of the $\sharp$AMS signature.

4. The announcement period, in which each developer uploads their $\sharp$AMS signature to the blockchain. The voting result is then included in the block and published across the network.

Our protocol enables multiple improvement proposals to compete for votes, and the proposal with the most votes is elected.

To generate a $\sharp$AMS signature, the developer needs three rounds of interaction with its supporters during the signing period. This opens the door to faulty attacks by malicious voters: for instance, a voter may send h_i and claim to participate, but then abort or send incorrect randomness r' after receiving m from the developer. Thanks to our fault-tolerant $\sharp$AMS scheme, such attacks do not compromise the entire voting system, since a (fault-tolerant) $\sharp$AMS signature always reveals the true number of (honest) participants.

V2: Round Optimization

Notice that in the protocol above, voters must wait for the message m_i from the developer after declaring their support. To achieve the "vote-and-go" property, we further optimize the protocol to a single round. Our idea is to use $\sharp$AMS in a one-time paradigm. That is, voters U_i, regardless of whether they intend to vote for a proposal, generate a new hash key and trapdoor pair $(hk_i^{(j)}, td_i^{(j)})$ for some proposal by developer P_j. Then, supporters of P_j secretly send their trapdoors to P_j using a standard public-key encryption scheme. With knowledge of all secret keys, P_j can generate a $\sharp$AMS signature without further interaction with the supporters. To prevent a malicious developer from generating one-time hash keys on their own, we additionally require every hash key to be accompanied by a signature proving the authority of its owner.

V3: Single-Vote Setting via the Conditioned Key Generation Paradigm

We further extend the above protocol to the single-vote setting, where each user in the voting system can vote for at most one proposal. Since there are multiple developers P_j, each with their own proposal, every voter U_i now needs to generate multiple key pairs $\{(hk_i^{(j)}, td_i^{(j)})\}$, where the superscript (j) indicates the key pair is for the j-th proposal.

The protocol as previously described does not work in the single-vote setting because a malicious voter could vote on different proposals using different trapdoors. We employ the same method used in constructing $\sharp$AMS to prevent multiple votes. Recall that to vote on a proposal $IP^{(j)}$ (i.e., participate in the signing process for that proposal), user U_i must possess the trapdoor $td_i^{(j)}$ corresponding to the hash key $hk_i^{(j)}$ specifically designed for $IP^{(j)}$. Suppose there are p proposals $(IP^{(j_1)}, ..., IP^{(j_p)})$, then U_i must generate p hash keys $(hk_i^{(j_1)}, ..., hk_i^{(j_p)})$. We apply a so-called *conditioned key generation paradigm*, which sets a restriction among the p hash keys so that the voter can know at most one trapdoor.

Let $\mathcal{HK}$ be the space of hash keys for (lossy) chameleon hashes, and let $\mathbf{B} = (b_{i,j}) \in \mathcal{HK}^{(p-1)\times p}$ be a matrix of full rank. Define $\mathbf{hk}_i^\top = (hk_i^{(j_1)}, ..., hk_i^{(j_p)})^\top$. We require $\mathbf{B} \cdot \mathbf{hk}_i = \hat{\mathbf{hk}}_i$, where $\hat{\mathbf{hk}}_i \in \mathcal{HK}^{p-1}$ is the output of some hash function $H(IP^{(j_1)}, ..., IP^{(j_p)}, i)$.

If $H(\cdot)$ behaves as a random oracle, then to satisfy the above equation, every user can know at most one trapdoor among the total p hash keys. Consequently, the single-voting property is achieved.

1.4 Roadmap

This paper is organized as follows. In Section 2, we introduce the definitions of (lossy) chameleon hashes. The definition and security properties of ♯AMS are formally described in Section 3. We provide a generic transformation for most threshold ring signature (TRS) schemes in Section 4. In Section 5, we propose an efficient construction of ♯AMS from (lossy) chameleon hashes ((L)CH). Section 6 presents three voting systems and their applications in blockchain governance. We refer to the full version [19] for more details, including basic notations, cryptographic primitives, more security notions of ♯AMS and formal proofs.

2 Preliminaries

▶ **Definition 2** (Chameleon Hashes [44]). *A chameleon hash (CH) scheme consists of the following three algorithms. Namely* $\mathsf{CH} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{TdColl})$.

- $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$*: The key generation algorithm takes as input the security parameter* 1^λ*, and outputs a hash key* hk *and a trapdoor* td*. W.l.o.g., we assume* hk *implicitly determines the message space* $\mathcal{M}$*, the randomness space* $\mathcal{R}$*, and the hash space* $\mathcal{H}$.
- $h \leftarrow \mathsf{Hash}(hk, m, r)$*: The hash algorithm takes as input* hk*, a message* $m \in \mathcal{M}$ *and a randomness* $r \in \mathcal{R}$*, and outputs the hash value* $h := \mathsf{Hash}(hk, m, r)$.
- $r' \leftarrow \mathsf{TdColl}(td, m, r, m')$*: The trapdoor collision algorithm takes as input the trapdoor* td*, a message-randomness pair* (m, r) *and another message* m'*, and outputs* r' *such that* $\mathsf{Hash}(hk, m, r) = \mathsf{Hash}(hk, m', r')$.

We introduce lossy chameleon hashes (LCH) as follows.

▶ **Definition 3** (Lossy Chameleon Hashes). *A lossy chameleon hash (LCH) scheme consists of four algorithms,* $\mathsf{LCH} = (\mathsf{Gen}, \mathsf{LGen}, \mathsf{Hash}, \mathsf{TdColl})$*, where* $\mathsf{Gen}, \mathsf{Hash}, \mathsf{TdColl}$ *are defined as in Def. 2, and* LGen *is defined as follows.*

- $hk \leftarrow \mathsf{LGen}(1^\lambda)$*: The lossy key generation algorithm takes as input the security parameter* 1^λ*, and outputs a lossy hash key* hk.

We explore a detailed framework of LCH in the full version [19].

3 Sharp Anonymous Multisignatures

In this section we formally define sharp anonymous multisignatures and their security notions. Let n be the total number of signers, $G \subseteq [n]$ be a group of signers, and $t = |G|$.

▶ **Definition 4** (Sharp Anonymous Multisignatures (♯AMS)). *A sharp anonymous multisignature (♯AMS) scheme* $\sharp\mathsf{AMS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *consists of the following three algorithms/protocols:*

- $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$*. The key generation algorithm* Gen *takes as input the security parameter* λ *and the number of signers* n*, and outputs a verification key* vk*, and signing keys* $(sk_1, ..., sk_n)$ *for different signers. W.l.o.g., we assume that* vk *is implicitly contained in every* sk_i.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, P, G \subseteq [n], \{sk_i\}_{i \in G})$*. The signing protocol takes place between a moderator* P *and a group of signers* $G \subseteq [n]$*, where* P *takes as input* vk *and the message* msg*, and each signer* $U_i \in G$ *takes* msg *and its own secret key* sk_i *as input. The moderator* P *can be a member of* $[n]$ *or not. Finally,* P *outputs a signature* σ.
If we focus solely on the algorithmic properties of the signing process, we will ignore the moderator P *by denoting it as* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, G \subseteq [n], \{sk_i\}_{i \in G})$.

- $t \leftarrow \mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. *The verification algorithm* Ver *takes as input the verification key* vk*, a message* msg *and a signature* σ*, and outputs a number* $t \in [n] \cup \{0\}$*, indicating the number of signers for this signature.*[8]
  We say a signature σ *(w.r.t. a message* msg*) is* t*-valid (resp., invalid), if* $\mathsf{Ver}(vk, \mathsf{msg}, \sigma) = t$ *(resp.,* $\mathsf{Ver}(vk, \mathsf{msg}, \sigma) = 0$*).*

Correctness. *For any* $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$*, any message* msg*, any group* $G \subseteq [n]$ *of honest signers, any honest moderator* P*, and* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, P, G, \{sk_i\}_{i\in G})$*, it holds that* $\mathsf{Ver}(vk, \mathsf{msg}, \sigma) = |G|$.

In the above definition, we assume all users' keys are generated via a centralized algorithm Gen, which is more generalized and covers the case where there is a trusted setup. In this work, we focus on the non-trusted setup case, and each user samples its own key pair and then publishes the public key. Moreover, we focus on cryptographic property of ♯AMS and do not consider the details of the signing protocol, e.g., adaptive corruptions during the running, protocol rounds, and just provide a proof-of-concept use of ♯AMS.

We require the unforgeability and anonymity for the security of ♯AMS.

- **Unforgeability**. The adversary that controls less than t participants cannot forge a signature that is t-valid.
- **Anonymity**. From a signature the adversary learns nothing about the quorum of the signers G that contributed to the signature, except the size $|G|$.

We formalize the security definitions via the following security experiments.

▶ **Definition 5** (Unforgeability of ♯AMS)**.** *Consider the following unforgeability experiment* $\mathsf{Exp}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *between the challenger* $\mathcal{C}$ *and the adversary* $\mathcal{A}$*.*

1. $\mathcal{A}$ *sets the maximum number of signers* n*.*
2. $\mathcal{C}$ *generates* $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$ *and passes* vk *to* $\mathcal{A}$*.*
3. $\mathcal{A}$ *has access to two oracles* $\mathcal{O}(\cdot,\cdot,\cdot)$ *and* $\mathcal{O}_{corr}(\cdot)$*. Here the signing oracle* $\mathcal{O}(\mathsf{msg}, P, G)$ *returns* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, P, G, \{sk_i\}_{i\in G})$ *(with* P *the moderator) and adds* (msg, σ) *into the set* $\mathcal{S}$*. The corruption oracle* $\mathcal{O}_{corr}(i)$ *returns* sk_i*.*
4. *Finally* $\mathcal{A}$ *outputs* $(\mathsf{msg}^*, \sigma^*)$*.*

Let $t^* \leftarrow \mathsf{Ver}(vk, \mathsf{msg}^*, \sigma^*)$*.* $\mathsf{Exp}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *outputs* 1 *if*

(1) $t^* > t'$*, where* t' *is the total number of queries to* $\mathcal{O}_{corr}(\cdot)$*; and*
(2) $\mathcal{A}$ *never asks* $\mathcal{O}(\mathsf{msg}^*, P, G)$ *such that* $|G| = t^*$*.*

Define by $\mathsf{Adv}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *the probability that* $\mathsf{Exp}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *outputs 1. We say that* ♯AMS *is unforgeable, if for all PPT adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *is negligible in* λ*.*

▶ Remark 6 (On the Formalization of Unforgeability). One might wonder why we require "$\mathcal{A}$ never asks $\mathcal{O}(\mathsf{msg}^*, P, G)$ s.t. $|G| = t^*$" at the end of the experiment. Intuitively, a more "reasonable" definition should allow $\mathcal{A}$ to win, if it asks $\mathcal{O}(\mathsf{msg}^*, P, G)$ with $|G| = t^*$, and later forges a t^*-valid signature σ^* from a different set $G' \neq G$. However, since the identities are hidden from the signature, the challenger cannot detect whether σ^* comes from a group G' that is different from G (i.e., whether $\mathcal{A}$ wins in a non-trivial way). Therefore, to prevent trivial attacks, in Def. 5, $\mathcal{A}$ is prohibited to query $\mathcal{O}(\mathsf{msg}^*, P, G)$ with $|G| = t^*$.

[8] Note that vk contains information of $[n]$ so we omit $[n]$ from the input of Ver, which will be explicitly denoted for TRS schemes because of consistency from previous works.

▶ **Definition 7** (Strong Unforgeability). *Consider the strong unforgeability experiment* $\mathsf{Exp}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$*, which is defined as* $\mathsf{Exp}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *in Def. 5, except that condition (2) is replaced with*
(2') $(\mathsf{msg}^*, \sigma^*) \notin \mathcal{S}$.

Define by $\mathsf{Adv}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *the probability that* $\mathsf{Exp}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *outputs 1. We say that* $\sharp\mathsf{AMS}$ *is strongly unforgeable, if for all PPT adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *is negligible in* λ*.*

In the full version [19] we also define the weak unforgeability and (strong/weak) unforgeability under static corruptions.

Now we formally define the strong anonymity of $\sharp\mathsf{AMS}$.

▶ **Definition 8** ((Unconditionally) Strong Anonymity of $\sharp\mathsf{AMS}$). *Consider the following strong anonymity experiment* $\mathsf{Exp}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *between* $\mathcal{C}$ *and* $\mathcal{A}$*.*

1. $\mathcal{A}$ *sets the maximum number of signers* n*.*
2. $\mathcal{C}$ *generates* $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$ *and passes* vk *to* $\mathcal{A}$*. Meanwhile,* $\mathcal{C}$ *randomly samples a bit* $b \xleftarrow{\$} \{0,1\}$*.*
3. $\mathcal{A}$ *has access to two oracles* $\mathcal{O}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *and* $\mathcal{O}_{corr}(\cdot)$*, where the signing oracle* $\mathcal{O}(\mathsf{msg}, P_0, G_0, P_1, G_1)$ *returns* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, P_b, G_b, \{sk_i\}_{i\in G_b})$ *if* $|G_0| = |G_1|$ *and* $\perp$ *otherwise, and the corruption oracle* $\mathcal{O}_{corr}(i)$ *returns* sk_i*.*
4. *Finally* $\mathcal{A}$ *outputs* b'*.*

$\mathsf{Exp}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *outputs 1 if* $b' = b$*. Let* $\mathsf{Adv}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \Rightarrow 1] - 1/2|$*. We say that* $\sharp\mathsf{AMS}$ *has strong anonymity (resp., unconditional and strong anonymity) if for all PPT (resp., computationally unbounded) adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *is negligible in* λ*.*

The aforementioned definition requires that anonymity holds even if all secret keys are leaked (that is why we formalize it as "strong" anonymity). In the above definition, if we restrict that $\mathcal{A}$ cannot ask $\mathcal{O}_{corr}(i)$ with $i \in (G_0 - G_1) \cup (G_1 - G_0)$ for all (G_0, G_1) involved in $\mathcal{O}(\cdot,\cdot,\cdot,\cdot,\cdot)$, then we define the anonymity where it might be easy to decide whether a signer i has participated in the generation of a signature with the knowledge of sk_i. This is somewhat similar to the so-called *culpability* property in [50]. In other words, one signer is able to claim the authorship of some (ring) signature by revealing its secret key (and some other private information, if necessary) to the public.

4 Generic Compiler of $\sharp$AMS from TRS with A Flexible Threshold

In this section, we provide a generic transformation for threshold ring signature (TRS) schemes with flexible threshold, which implies that such TRS schemes are more versatile than their original definitions. We refer to the full version [19] for the definition and security of TRS, and a detailed discussion for the relationship between TRS and $\sharp$AMS.

Generic compiler from TRS with flexible threshold to $\sharp$AMS. Here, we show the most generalized version of compilers without considering optimization. We will take a deeper look at specific cases in the next section. Let $\mathsf{TRS} = (\mathsf{Gen}, \mathsf{TSign}, \mathsf{Ver})$ be a TRS scheme with a flexible threshold. We design $\sharp$AMS scheme $\sharp$AMS as follows.

- $\mathsf{Gen}(1^\lambda, n)$. For $i = 1, ..., n$, invoke $(pk_i, sk_i) \leftarrow \mathsf{TRS.Gen}(1^\lambda)$. Return $vk := (pk_1, ..., pk_n)$ and $\{sk_i\}_{i\in[n]}$.

- $\mathsf{Sign}(\mathsf{msg}, P, G \subseteq [n], \{sk_i\}_{i\in G})$. Every signer U_i where $i \in G$ first sends a HELLO message to the moderator P so that P will know the number of signers $t := |G|$. Then, the t signers run the threshold signing protocol $\mathsf{TRS.TSign}(\mathsf{msg}||t, [n], vk, G, \{sk_i\}_{i\in G})$ with P works as a moderator. Let $\tilde{\sigma}$ be the output of $\mathsf{TRS.TSign}$. Finally, P outputs $\sigma := (t, \tilde{\sigma})$ as a $\sharp$AMS signature.[9]
- $\mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. Parse $\sigma = (t, \tilde{\sigma})$. If $\mathsf{TRS.Ver}(t, [n], vk, \mathsf{msg}, \tilde{\sigma}) = 1$ then output t. Otherwise, output 0.

The security of $\sharp$AMS constructed above inherits from the security of the underlying TRS scheme, and we omit the detailed proof here.

5 Constructions of $\sharp$AMS from Lossy Chameleon Hashes

In this section we propose efficient constructions of $\sharp$AMS from lossy chameleon hashes (LCH) in a black-box way.

5.1 Formalization of Constraint Functions

First we introduce the definition of constraint functions, an important tool in constructing the t-out-of-n proofs [21].

▶ **Definition 9** (Constraint Functions). *Let n, t be positive integers, $n \geq t \geq 1$, and $\mathcal{M}, \mathcal{U}$ be two finite sets. We call $\mathbf{F}_\theta : (\mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathcal{M}^n \times \mathcal{U}) \to \{0,1\}$ a series of constraint functions indexed by θ, if it has the following properties.*

- *$\mathbf{F}_\theta(n, t, m_1, ..., m_n, u)$ is efficiently evaluated.*
- *There exists an efficient and deterministic algorithm $f(\cdot)$, such that $f(n, t, m_1, ..., m_n)$ outputs the unique u (if it exists) satisfying $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1$.*
- *For any $G \subseteq [n]$ and $|G| = t$, there exists two efficient sample algorithms s_{fwd} and s_{back} that both output $(m_1, ..., m_n, u)$, and the two distributions are identical.*
  - *Forward sample algorithm $s_{fwd}(n, t, G)$ first samples $\{m_i\}_{i\in[n]}$, and then computes u according to $\mathbf{F}_\theta$.*
  - *Backward sample algorithm $s_{back}(n, t, G)$ first randomly chooses $\{m_i\}_{i\in[n]\setminus G}$ and u, and then computes $\{m_i\}_{i\in G}$ according to $\mathbf{F}_\theta$.*
- *Interdependency.*
  - *If $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = \mathbf{F}_\theta(n, t, m'_1, ..., m'_n, u) = 1$, then either $(m_1, ..., m_n) = (m'_1, ..., m')$, or there are at least t different $i \in [n]$ such that $m_i \neq m'_i$.*
  - *For randomly sampled u and u', if $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = \mathbf{F}_\theta(n, t, m'_1, ..., m'_n, u') = 1$, then with overwhelming probability there are at least t different $i \in [n]$ such that $m_i \neq m'_i$.*
- *Randomness. Conditioned on $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1$, if u distributes uniformly, then*
  - *either there exist at least t different $i \in [n]$ such that m_i distribute uniformly, or*
  - *for any $i \in [n]$, m_i distributes uniformly.*

We refer to two instantiations of $\mathbf{F}_\theta$ from linear equations and polynomial interpolation in the full version [19].

[9] If $\tilde{\sigma}$ itself already contains a threshold t then P just outputs $\sigma := \tilde{\sigma}$.

5.2 C1: Interactive ♯AMS

Now, we describe our generic construction of ♯AMS from (lossy) chameleon hashes ((L)CH) as follows. We note that the underlying building blocks (LCH or CH) affect only the way of security proofs but not the construction itself.

Construction. Let $\mathsf{LCH} = (\mathsf{Gen}, \mathsf{LGen}, \mathsf{Hash}, \mathsf{TdColl})$ be a lossy chameleon hash scheme with message space $\mathcal{M}$ a field. Let $\mathbf{F}_\theta : (\mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathcal{M}^n \times \mathcal{U}) \to \{0,1\}$ be a constraint function indexed by θ. $H(\cdot) : \{0,1\}^* \to \mathcal{U}$ is a hash function that is modeled as a random oracle.

- $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$. For $i \in [n]$, invoke $(hk_i, td_i) \leftarrow \mathsf{LCH.Gen}(1^\lambda)$. Return $vk := (hk_1, ..., hk_n)$ and $\{sk_i\}_{i\in[n]} := \{td_i\}_{i\in[n]}$.
- $\mathsf{Sign}(\mathsf{msg}, P, G, \{sk_i\}_{i\in G})$.
  1. For every signer $i \in G$, it samples $\bar{m}_i \xleftarrow{\$} \mathcal{M}, \bar{r}_i \xleftarrow{\$} \mathcal{R}$, and sends $h_i \leftarrow \mathsf{Hash}(hk_i, \bar{m}_i, \bar{r}_i)$ to the moderator P.
  2. The moderator counts $t := |G|$ from the received messages.
  3. For $i \in [n] \setminus G$, P samples $m_i \xleftarrow{\$} \mathcal{M}, r_i \xleftarrow{\$} \mathcal{R}$ and computes $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$.
  4. P invokes $u \leftarrow H(vk, h_1, ..., h_n, \mathsf{msg}||t)$.[10] Then it computes $\{m_i\}_{i\in G}$ according to the backward sample algorithm $s_{back}(\cdot)$ of $\mathbf{F}_\theta$ such that
  $$\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1.$$
  For $i \in G$, P sends m_i to signer i.
  5. For $i \in G$, signer i computes $r_i \leftarrow \mathsf{LCH.TdColl}(td_i, \bar{m}_i, \bar{r}_i, m_i)$ and sends r_i to P.
  6. Finally P outputs the signature $\sigma := (t, \{m_i\}_{i\in[n]}, \{r_i\}_{i\in[n]})$.
- $\mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. Parse $\sigma = (t, \{m_i\}_{i\in[n]}, \{r_i\}_{i\in[n]})$. Compute $h_i \leftarrow \mathsf{Hash}(hk_i,\ m_i, r_i)$ for all $i \in [n]$. Let $u \leftarrow H(vk, h_1, ..., h_n, \mathsf{msg}||t)$. Return t if $\mathbf{F}_\theta(n, t, m_1,\ ..., m_n, u) = 1$, and 0 otherwise.

Generality of the Construction. Our construction exhibits strong generality. In Section 4, we have shown a generic compiler from TRS with a flexible threshold to ♯AMS. In fact, many existing TRS constructions can be categorized within the above framework due to the equivalence between chameleon hashes and Sigma protocols (identification schemes) [8] and the result (of (L)CH) in this work. For example, by instantiating with $\mathbf{F}_\mathbf{A}$ (see the full version [19]) and the DL-based CH [44], we get the TRS scheme in [14], and by instantiating with $\mathbf{F}_p$ and the DL-based CH, we get the TRS scheme in [49]. Thanks to this generic construction, we immediately get more schemes of ♯AMS (also TRS) from lattices [29, 12], isogenies [25], etc.

▶ **Theorem 10.** *If* LCH *is strongly secure (i.e., it has κ-uniformity, γ-random trapdoor collision, strong collision resistance, indistinguishability, and ϵ-lossiness) and unique, and $\mathbf{F}_\theta$ is a constraint function, then* ♯AMS *constructed above has strong unforgeability and unconditionally strong anonymity under static corruptions. More precisely, for any PPT adversary $\mathcal{A}$, there exist PPT algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$, such that* $\max(Time(\mathcal{B}_1), Time(\mathcal{B}_2)) \approx Time(A)$, *and*

[10] By including both msg and the threshold t into the hash input, the non-malleability is achieved in our signature scheme.

$$\mathsf{Adv}^{s\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq n(\mathsf{Adv}^{s\text{-}cr}_{\mathsf{LCH},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}_2}(\lambda)) + n \cdot \epsilon + \frac{1}{|\mathcal{M}|} + \frac{Q_{sign} + Q_H}{2^{n\kappa}} + Q_{sign} n \cdot \gamma,$$

$$\mathsf{Adv}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq \frac{Q_{sign} n}{2} \cdot \gamma,$$

where Q_{sign} and Q_H are the numbers of signing queries (in the strong unforgeability experiment or the strong anonymity experiment) and hash queries, respectively.

We refer to the full version [19] for the proof.

Similarly, we get the following theorem for (regular) unforgeability.

▶ **Theorem 11.** *If* LCH *is secure (i.e., it has κ-uniformity, γ-random trapdoor collision, collision resistance, indistinguishability, and ϵ-lossiness) and $\mathbf{F}_\theta$ is a constraint function, then* ♯AMS *constructed above has unforgeability and unconditional strong anonymity under static corruptions. More precisely, for any PPT adversary $\mathcal{A}$, there exist PPT algorithm $\mathcal{B}$ such that $Time(\mathcal{B}) \approx Time(A)$, and*

$$\mathsf{Adv}^{unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq n\mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}}(\lambda) + n \cdot \epsilon + \frac{1}{|\mathcal{M}|} + \frac{Q_{sign} + Q_H}{2^{n\kappa}} + Q_{sign} n \cdot \gamma,$$

where Q_{sign} and Q_H are the numbers of signing queries and hash queries, respectively.

Security under Adaptive Corruptions and the Tightness of Unforgeability

We prove the security of unforgeability in the static corruption model. The proof can also be extended to the adaptive model (Def. 5 and 7), but it suffers from a large loss factor 2^n. We present the security bound under adaptive corruptions in the full version [19]. At a high level, to make use of the lossiness property of LCH, the challenger has to make sure that all hash keys of non-corrupted users are generated in the lossy mode. However, there is no corresponding trapdoor for a lossy hash key, which means that to deal with $\mathcal{A}$'s adaptive corruptions, the challenger has to decide the way of key generation very carefully so that it can offer the trapdoor of a user when a corruption happens on it. We also give another proof based on normal chameleon hashes in the full version [19].

5.3 C2: The Fault-Tolerating Variant

In this subsection, we consider faulty signers, i.e., signers who quit in the middle of signing or return faulty results to the moderator P. We propose a fault-tolerant variant of the (L)CH-based ♯AMS construction. In this variant, even if there exist faulty signers who behave maliciously, the moderator P can still output a ♯AMS that is t-valid, where t is the number of honest signers in the generation of that signature. Besides, the identities of those faulty signers are exposed, and anonymity holds only for the honest signers.

Faulty Attacks. We first introduce possible faulty attacks and then introduce a variant of (L)CH-based ♯AMS for tolerating faults while generating signatures. Note that our constructions need communication with a developer within a fixed period to ensure the number of total signers. This leads to the following faulty attacks. In the declaration period, the faulty node U_k faithfully follows the first two steps of the interaction. Then after receiving m_k from P, it crashes or sends a value instead of r_k.

Since P cannot compute r_k without td_k, it cannot produce a t-valid ♯AMS signature. Furthermore, since $\{m_j\}_{j\in G}$ are determined by $H(vk, h_1, ..., h_n, \mathsf{msg}||t)$ from $\{h_j\}_{j\in[n]}$ and $t = |G|$, P cannot generate a signature of $G \setminus \{k\}$ by simply discarding U_k.

Fault-Tolerant ♯AMS. We follow the same notion as the previous section. The Gen algorithm and the Sign protocol are also essentially the same as the previous one, but only the following treatment is included:

- In the signing protocol, if a signer U_i does not response for P's message m_i, or the signer U_i returns a wrong r_i such that $\mathsf{Hash}(hk_i, m_i, r_i) \neq h_i$, then P include i into the set F. Here h_i is the first message U_i sens to P (if U_i does not send h_i, then P would not include U_i into the signer group G).
  Finally, P outputs the fault-tolerated signature
  $$\sigma := \left(t, F, \{m_i, r_i\}_{i\in[n]\setminus F}, \{m_i, h_i\}_{i\in F}\right).$$
- $t \leftarrow \mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. Let $\sigma = \left(t, F, \{m_i, r_i\}_{i\in[n]\setminus F}, \{h_i\}_{i\in F}\right)$ and $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$ for $i \in [n]\setminus F$. Let $u \leftarrow H(vk, h_1, ..., h_n, \mathsf{msg}||t)$. Return $(t - |F|)$ if $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1$ and 0 otherwise.

We show that the fault-tolerant ♯AMS scheme has weak unforgeability (against malicious signers) and unconditional strong anonymity (for honest signers).

▶ **Theorem 12.** *If* LCH *is secure (i.e., it has κ-uniformity, γ-random trapdoor collision, strong collision resistance, indistinguishability, and ϵ-lossiness) and $\mathbf{F}_\theta$ is a constraint function, then the fault-tolerant ♯AMS scheme above has weak unforgeability and strong anonymity under static corruptions. More precisely, for any PPT adversary $\mathcal{A}$, there exist PPT algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$, such that* $\max(Time(\mathcal{B}_1), Time(\mathcal{B}_2)) \approx Time(A)$,

$$\mathsf{Adv}^{w\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq n(\mathsf{Adv}^{s\text{-}cr}_{\mathsf{LCH},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}_2}(\lambda)) + n\cdot\epsilon + \frac{1}{|\mathcal{M}|} + \frac{Q_{sign}+Q_H}{2^{n\kappa}} + Q_{sign}n\cdot\gamma,$$

$$\mathsf{Adv}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq \frac{Q_{sign}n}{2}\cdot\gamma$$

where Q_{sign} and Q_H are the numbers of signing queries (in the strong unforgeability experiment or the strong anonymity experiment) and hash queries, respectively.

6 Applications: Blockchain Governance and the Beyond

♯AMS can be massively applied in the scenario of blockchain and privacy-preserving, where authenticity and privacy are required simultaneously. The first and the most significant application of ♯AMS is blockchain governance, especially about ranking improvement proposals, which is one of the most uprising topics in the blockchain era. Our main goal is to implement a ranking – in other words, a voting – system on formal on-chain blockchain governance. (E-)voting systems need a signature scheme to prevent double-voting or any other possible exploitations related to confidentiality.

In this section, we give a generic treatment to achieve blockchain governance by developing a voting system via ♯AMS. Note that our applications are simplified to give a showcase of using our ♯AMS. Applying our scheme to more complex voting designs could be an interesting direction. However, our simple solutions have already achieved many attractive properties, e.g., lightweight and publicly verifiable – thus, there is no need for tallying authorities.

Note that in our schemes, anonymity does not hold for the moderator, meaning that the moderator (whether honest or malicious) knows the identities of all signers. However, unforgeability still holds even against a malicious moderator. Recall that in blockchain governance, a developer P who intends to propose an improvement proposal will start the ♯AMS protocol as the moderator. P's goal is to gather as many supporters as possible, and

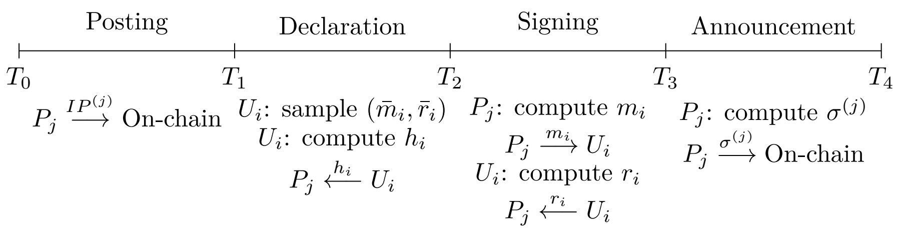
Posting
Declaration
Signing
Announcement
T_0
T_1
T_2
T_3
T_4
$P_j \xrightarrow{IP^{(j)}}$ On-chain
U_i: sample $(\bar{m}_i, \bar{r}_i)$
U_i: compute h_i
$P_j \xleftarrow{h_i} U_i$
P_j: compute m_i
$P_j \xrightarrow{m_i} U_i$
U_i: compute r_i
$P_j \xleftarrow{r_i} U_i$
P_j: compute $\sigma^{(j)}$
$P_j \xrightarrow{\sigma^{(j)}}$ On-chain

Figure 2 Timeline of V1. Here U_i votes to $IP^{(j)}$.

then generate a ♯AMS signature to demonstrate the credibility of the proposal and thereby win the RFP. A malicious moderator P can do nothing except reduce t – which trivially harms P's utility. For this reason, we assume the moderator is honest in the following analysis.

6.1 V1: Blockchain Governance via ♯AMS

As introduced in Section 3, the verification function of ♯AMS returns the number of the signers who participated in the signature generation, which can turn into a voting protocol. In our concrete scheme of ♯AMS, each participant generates their own keys and publishes them, which is a straightforward pre-processing step in voting via blockchain. The voting protocol enables multiple improvement proposals to compete for votes, and the one with the most votes is elected. Our on-chain governance voting protocol processes as follows:

1. A *voting session* implies the whole process of this protocol. For each RFP, the participants run a voting session to evaluate proposals. Each session has four time periods: *posting period*, *declaration period*, *signing period*, and *announcement period*. We denote $[n]$ as the index set of voters while U_i implies an i-th voter for $i \in [n]$. Similarly, for $p < n$, $[p]$ is the index set of developers who propose an improvement proposal while P_j stands for j-th developer and $IP^{(j)}$ stands for the proposal made by P_j. Note that $P_j = U_j$ for $j \in [p]$, which means P_j is also eligible to vote and will lead the quorum of $IP^{(j)}$. P_j serves a dual role as both the initiator of $IP^{(j)}$ and the representative of voters of that proposal, ensuring the concealment of their identities during signature generation.
2. In the posting period, for all $j \in [p]$, P_j submits $IP^{(j)}$. This can be done via the blockchain network using a smart contract.
3. In the declaration period, for each $i \in [n]$, U_i expresses their will to vote on $IP^{(j)}$ by generating a random string pair $(\bar{m}_i, \bar{r}_i)$ using their own randomness, which should be kept in secret, and sending $h_i = \mathsf{Hash}(hk_i, \bar{m}_i, \bar{r}_i)$ to P_j.
4. In the signing period, P_j sends m_i to U_i and U_i sends r_i to P_j, where m_i and r_i are defined in the description of ♯AMS in Section 5.
5. In the announcement period, P_j generates a signature $\sigma^{(j)}$ from (m_i, r_i) pairs for each voter and uploads it to the blockchain system. In the end, the most voted proposal is elected.

   See Fig. 2 for the pictorial explanation of our voting system.

6.2 V2: Round Optimization

Recall that in the voting scheme above, after publishing the improvement proposal, each participant has to execute three rounds of interaction before uploading the ♯AMS signature on the blockchain.

- Round 1 (in the declaration period): P_j receives h_i as a voting-claim from its supporter U_i.
- Round 2 (in the signing period): P_j computes m_i according to the signing algorithm of $\sharp$AMS and sends m_i to U_i.
- Round 3 (in the signing period): U_i returns r_i such that $\mathsf{Hash}(hk_i, m_i, r_i) = h_i$.

After Round 3, P_j can finish signing and upload the signature σ on the blockchain. Now we come up with the following question:

Can we reduce the round complexity? Namely, can we optimize the protocol such that the developer P_j can generate the $\sharp$AMS signature immediately after receiving the voting claims from its supporters?

The natural idea is to let the supporter, say U_i, send its secret key (the trapdoor of chameleon hash schemes) directly to the developer P_j. With the knowledge of all secret keys, P_j can now generate a $\sharp$AMS signature without interaction with its supporters. However, this will totally expose users' secret keys to the developers in a single vote event, which is not the case users want.

Our idea is to use $\sharp$AMS in a one-time paradigm, see Fig. 3. That is, for each proposal $IP^{(j)}$ proposed by the developer P_j, the supporter U_i generates a new hash key and trapdoor pair $(hk_i^{(j)}, td_i^{(j)})$, and then sends the key pair to P_j. For a user U_k who does not want to support $IP^{(j)}$, it also generates a key pair $(hk_k^{(j)}, td_k^{(j)})$, but then sends only the hash key to P_j. This hash key is used for P_j to add U_k into the anonymous group to hide the identities of the real voters. Moreover, we have the following two modifications.

1. To make sure a hash key $hk_i^{(j)}$ is generated from voter U_i but not developer P_j (otherwise it can always make an n-valid $\sharp$AMS signature), U_i will sign on $hk_i^{(j)}$ via a standard signature scheme to show its authority.
2. The message from U_i to P_j is encrypted using P_j's public key, so that no eavesdropper except P_j will know the corresponding trapdoor $td_i^{(j)}$. Meanwhile, the message, either $hk_i^{(j)}$ or $(hk_i^{(j)}, td_i^{(j)})$, is padded to the same length, preventing the side-leakage of anonymity.

Formally, the round-optimal protocol is described as follows. Let $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ be a (regular) signature scheme with unforgeability, and PKE be a public key encryption scheme with CPA security. At the beginning, we assume that every user U_i for $i \in [n]$ has its own key pairs $(\widetilde{vk}_i, \widetilde{sk}_i)$ of Sig and (pk_i, sk_i) of PKE.

1. In the posting period, developers $P_1, ..., P_p$ submit improvement proposals $IP^{(1)}, ..., IP^{(p)}$ to the blockchain network using a smart contract.
2. In the declaration & signing period, for all $i \in [n]$ and $j \in [p]$,
   a. U_i invokes $(hk_i^{(j)}, td_i^{(j)}) \leftarrow \mathsf{LCH.Gen}(1^\lambda)$ and $cert_i^{(j)} \leftarrow \mathsf{Sign}(\widetilde{sk}_i, hk_i^{(j)})$;
   b. If U_i wants to vote $IP^{(j)}$, U_i computes $ct_{i\to j} \leftarrow \mathsf{Enc}(pk_j, hk_i^{(j)} || cert_i^{(j)} || td_i^{(j)})$;
   c. If U_i does not want to vote $IP^{(j)}$, U_i computes $ct_{i\to j} \leftarrow \mathsf{Enc}(pk_j, hk_i^{(j)} || cert_i^{(j)} || \mathbf{0})$, where $\mathbf{0}$ is a zero-string of length $|td_i^{(j)}|$;
   d. U_i sends $ct_{i\to j}$ to P_j.
3. In the announcement period, for each $j \in [p]$, after decrypting $ct_{i\to j}$ for all $i \in [n]$, the developer P_j obtains a series of hash keys $\{hk_i^{(j)}\}_{i\in[n]}$ as well as their corresponding certificates $\{cert_i^{(j)}\}_{i\in[n]}$. Meanwhile, P_j also gets a group of trapdoors $\{td_i^{(j)}\}$ from users U_i who are willing to support $IP^{(j)}$. P_j can check the validity of certificates using long-term verification keys. If the verification fails with respect to user U_i, then P_j discards U_i from the anonymous group. Finally, P_j generates a $\sharp$AMS signature $\sigma^{(j)}$ for its proposal $IP^{(j)}$ and uploads the $\sharp$AMS signature, all hash keys, and certificates concerning the proposal $IP^{(j)}$ to the blockchain system.

Note that if a user U_i behaves in a Byzantine manner, for example, by failing to send $ct_{i \to j}$ to P_j, this misbehavior will be easily detected after the announcement period ends. In such cases, all developers can simply re-run the announcement period with the participant set updated to $[n] \setminus i$. This approach naturally generalizes to tolerate any number of faulty users, with the only cost being an additional announcement period for each round of fault recovery.

6.3 V3: Single-Vote Setting via the Conditioned Key Generation Paradigm

A voting system in the single-vote setting is desirable in many applications. For example, if two proposals conflict, an unruly user voting on both will undoubtedly disrupt the normal voting process and outcomes. However, in the protocols described above, users can vote on several proposals. More precisely, suppose there are two distinct proposals, $IP^{(j)}$ by P_j and $IP^{(k)}$ by P_k with $j \neq k \in [p]$. User U_i can generate two key pairs $(hk_i^{(j)}, td_i^{(j)})$ and $(hk_i^{(k)}, td_i^{(k)})$, then vote on both proposals by sending the corresponding key pairs to P_j and P_k, respectively. Due to the anonymity property of ♯AMS, the verifier cannot determine which users have voted multiple times.

To prevent double voting, we introduce a new paradigm, dubbed the *conditioned key generation paradigm*. Recall that to vote on a proposal $IP^{(j)}$ (i.e., to participate in the signing process for the proposal), user U_i must have the trapdoor $td_i^{(j)}$ corresponding to the hash key $hk_i^{(j)}$ specifically designed for $IP^{(j)}$. User U_i needs to generate p hash keys $(hk_i^{(1)}, ..., hk_i^{(p)})$ initially. Our idea is to enforce a restriction such that among all p hash keys, the user can know at most one trapdoor.

The conditioned key generation paradigm is applying t-out-of-n proof strategy during the subkey generation. Assume the hash key space $\mathcal{HK}$ is a field. Let $\mathbf{B} = (b_{i,j}) \in \mathcal{HK}^{(p-1)\times p}$ be a public matrix of full rank. Let $\mathbf{hk}_i^\top = (hk_i^{(1)}, ..., hk_i^{(p)})^\top$. Now we require

$$\mathbf{B} \cdot \mathbf{hk}_i = \hat{\mathbf{hk}}_i,$$

where $\hat{\mathbf{hk}}_i \in \mathcal{HK}^{p-1}$ is the output of some hash function $H(IP^{(j_1)}, ..., IP^{(j_p)}, i)$.

If the hash key generated via $\mathsf{Gen}(1^\lambda)$ is computationally indistinguishable from a uniform hash key in $\mathcal{HK}$, and $H(\cdot)$ is a random oracle, then to satisfy the aforementioned equation, each user has at most one trapdoor of the p hash keys. Single-voting property is achieved as a result. See Fig. 3 for the pictorial explanation of our round-optimal voting system in multiple and single-vote settings.

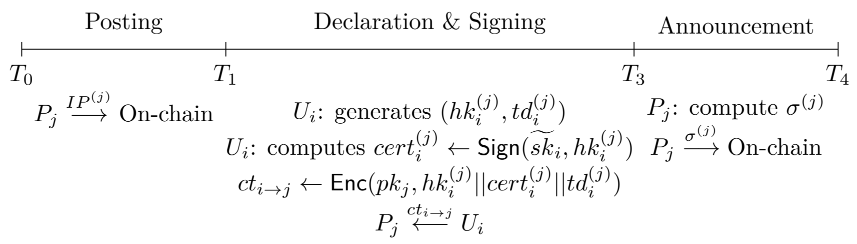
Posting
Declaration & Signing
Announcement
T_0
T_1
T_3
T_4
$P_j \xrightarrow{IP^{(j)}}$ On-chain
U_i: generates $(hk_i^{(j)}, td_i^{(j)})$
U_i: computes $cert_i^{(j)} \leftarrow \mathsf{Sign}(\widetilde{sk}_i, hk_i^{(j)})$
$ct_{i\to j} \leftarrow \mathsf{Enc}(pk_j, hk_i^{(j)} || cert_i^{(j)} || td_i^{(j)})$
$P_j \xleftarrow{ct_{i\to j}} U_i$
P_j: compute $\sigma^{(j)}$
$P_j \xrightarrow{\sigma^{(j)}}$ On-chain

Figure 3 Timeline of V2 and V3. Here U_i votes to $IP^{(j)}$. In V2, U_i also needs to compute and send $ct_{i\to k} = \mathsf{Enc}(pk_k, hk_i^{(k)} || cert_i^{(k)} || td_i^{(k)})$ or $ct_{i\to k} = \mathsf{Enc}(pk_k, hk_i^{(k)} || cert_i^{(k)} || \mathbf{0})$ to P_k for all $k \in [p]$. In V3, there should be a unique $j \in [p]$ for each $i \in [n]$ such that U_i uniquely upvotes to $IP^{(j)}$. For all $k \in [p]$ such that $k \neq j$, U_i should generate $hk_i^{(k)}$ by following the rule described in the context.

References

1 Michel Abdalla, Pierre-Alain Fouque, Vadim Lyubashevsky, and Mehdi Tibouchi. Tightly-secure signatures from lossy identification schemes. In *EUROCRYPT 2012*, volume 7237, pages 572–590. Springer, 2012. doi:10.1007/978-3-642-29011-4_34.

2 Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In *CRYPTO 2015*, pages 763–780. Springer, 2015. doi:10.1007/978-3-662-48000-7_37.

3 Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *CSF 2008*, pages 195–209. IEEE Computer Society, 2008. doi:10.1109/CSF.2008.26.

4 Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In *CRYPTO 2020*, volume 12171, pages 562–592. Springer, 2020. doi:10.1007/978-3-030-56880-1_20.

5 Roman Beck, Christoph Müller-Bloch, and John Leslie King. Governance in the blockchain economy: A framework and research agenda. *J. Assoc. Inf. Syst.*, 19(10):1, 2018. URL: https://aisel.aisnet.org/jais/vol19/iss10/1.

6 Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *EUROCRYPT 2009*, volume 5479, pages 1–35. Springer, 2009. doi:10.1007/978-3-642-01001-9_1.

7 Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS 2006*, pages 390–399. ACM, 2006. doi:10.1145/1180405.1180453.

8 Mihir Bellare and Todor Ristov. Hash functions from sigma protocols and improvements to VSH. In *ASIACRYPT 2008*, volume 5350, pages 125–142. Springer, 2008. doi:10.1007/978-3-540-89255-7_9.

9 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC 1988*, pages 1–10. ACM, 1988. doi:10.1145/62212.62213.

10 Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC 1994, Montréal, Québec, Canada*, pages 544–553. ACM, 1994. doi:10.1145/195058.195407.

11 Emmanuel Bresson, Jacques Stern, and Michael Szydlo. Threshold ring signatures and applications to ad-hoc groups. In *CRYPTO 2002*, volume 2442, pages 465–480. Springer, 2002. doi:10.1007/3-540-45708-9_30.

12 David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT 2010*, volume 6110, pages 523–552. Springer, 2010. doi:10.1007/978-3-642-13190-5_27.

13 Pierre-Louis Cayrel, Richard Lindner, Markus Rückert, and Rosemberg Silva. A lattice-based threshold ring signature scheme. In *LATINCRYPT 2010*, volume 6212, pages 255–272. Springer, 2010. doi:10.1007/978-3-642-14712-8_16.

14 Tony K. Chan, Karyin Fung, Joseph K. Liu, and Victor K. Wei. Blind spontaneous anonymous group signatures for ad hoc groups. In *ESAS 2004*, volume 3313, pages 82–94. Springer, 2004. doi:10.1007/978-3-540-30496-8_8.

15 David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC 1988*, pages 11–19. ACM, 1988. doi:10.1145/62212.62214.

16 David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *EUROCRYPT 1987*, volume 304, pages 127–141. Springer, 1987. doi:10.1007/3-540-39118-5_13.

17 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. doi:10.1016/j.tcs.2019.02.001.

18 Wonseok Choi, Xiangyu Liu, Lirong Xia, and Vassilis Zikas. K-linkable ring signatures and applications in generalized voting. *IACR Cryptol. ePrint Arch.*, page 243, 2025. URL: `https://eprint.iacr.org/2025/243`.

19 Wonseok Choi, Xiangyu Liu, and Vassilis Zikas. Blockchain governance via sharp anonymous multisignatures. *Cryptology ePrint Archive*, 2023.

20 Sherman S. M. Chow, Lucas Chi Kwong Hui, and Siu-Ming Yiu. Identity based threshold ring signature. In *ICISC 2004*, volume 3506, pages 218–232. Springer, 2004. `doi:10.1007/11496618_17`.

21 Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO 1994*, volume 839, pages 174–187. Springer, 1994. `doi:10.1007/3-540-48658-5_19`.

22 Léonard Dallot and Damien Vergnaud. Provably secure code-based threshold ring signatures. In *Cryptography and Coding 2009*, volume 5921, pages 222–235. Springer, 2009. `doi:10.1007/978-3-642-10868-6_13`.

23 Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *CSFW-19 2006*, pages 28–42. IEEE Computer Society, 2006. `doi:10.1109/CSFW.2006.8`.

24 Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000, pages 289–309. Springer, 2010. `doi:10.1007/978-3-642-12980-3_18`.

25 Ali El Kaafarani, Shuichi Katsumata, and Federico Pintore. Lossy csi-fish: Efficient signature scheme with tight reduction to decisional CSIDH-512. In *PKC 2020*, volume 12111, pages 157–186. Springer, 2020. `doi:10.1007/978-3-030-45388-6_6`.

26 Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO 1986*, volume 263, pages 186–194. Springer, 1986. `doi:10.1007/3-540-47721-7_12`.

27 Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *PKC 2007*, volume 4450, pages 181–200. Springer, 2007. `doi:10.1007/978-3-540-71677-8_13`.

28 Juan A. Garay, Aggelos Kiayias, Rafail M. Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In *EUROCRYPT 2020*, volume 12106, pages 129–158. Springer, 2020. `doi:10.1007/978-3-030-45724-2_5`.

29 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC 2008*, pages 197–206. ACM, 2008. `doi:10.1145/1374376.1374407`.

30 Hans Gersbach, Akaki Mamageishvili, and Manvir Schneider. Vote delegation and misbehavior. In *SAGT 2021*, volume 12885, page 411. Springer, 2021. URL: `https://link.springer.com/content/pdf/bbm%3A978-3-030-85947-3%2F1.pdf`.

31 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC 1987*, pages 218–229. ACM, 1987. `doi:10.1145/28395.28420`.

32 LM Goodman. Tezos – A self-amending crypto-ledger white paper. URL: `https://www.tezos.com/static/papers/whitepaper.pdf`, 4:1432–1465, 2014.

33 Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, pages 415–432. Springer, 2008. `doi:10.1007/978-3-540-78967-3_24`.

34 Abida Haque and Alessandra Scafuro. Threshold ring signatures: New definitions and post-quantum security. In *PKC 2020*, volume 12111, pages 423–452. Springer, 2020. `doi:10.1007/978-3-030-45388-6_15`.

35 James Heather and Steve A. Schneider. A formal framework for modelling coercion resistance and receipt freeness. In *FM 2012*, volume 7436 of *Lecture Notes in Computer Science*, pages 217–231. Springer, 2012. `doi:10.1007/978-3-642-32759-9_19`.

36 Brett Hemenway, Benoît Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In *ASIACRYPT 2011*, volume 7073, pages 70–88. Springer, 2011. doi:10.1007/978-3-642-25385-0_4.

37 Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 539–556. Springer, 2000. doi:10.1007/3-540-45539-6_38.

38 Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO 2014*, volume 8617, pages 369–386. Springer, 2014. doi:10.1007/978-3-662-44381-1_21.

39 Hugo L. Jonker and Erik P. de Vink. Formalising receipt-freeness. In *ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 476–488. Springer, 2006. doi:10.1007/11836810_34.

40 Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000, pages 37–63. Springer, 2010. doi:10.1007/978-3-642-12980-3_2.

41 Nida Khan, Tabrez Ahmad, Anass Patel, and Radu State. Blockchain governance: An overview and prediction of optimal strategies using nash equilibrium. *CoRR*, abs/2003.09241, 2020. arXiv:2003.09241.

42 Aggelos Kiayias and Philip Lazos. Sok: Blockchain governance. In *AFT 2022*, pages 61–73. ACM, 2022. doi:10.1145/3558535.3559794.

43 Aggelos Kiayias, Murat Osmanoglu, and Qiang Tang. Graded signatures. In *ISC 2015*, volume 9290, pages 61–80. Springer, 2015. doi:10.1007/978-3-319-23318-5_4.

44 Hugo Krawczyk and Tal Rabin. Chameleon hashing and signatures. *IACR Cryptol. ePrint Arch.*, page 10, 1998. URL: http://eprint.iacr.org/1998/010.

45 Ralf Küsters and Tomasz Truderung. An epistemic approach to coercion-resistance for electronic voting protocols. In *(SP 2009)*, pages 251–266. IEEE Computer Society, 2009. doi:10.1109/SP.2009.13.

46 Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, privacy, and coercion-resistance: New insights from a case study. In *SP 2011*, pages 538–553. IEEE Computer Society, 2011. doi:10.1109/SP.2011.21.

47 Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A game-based definition of coercion resistance and its applications. *Journal of Computer Security*, 20(6):709–764, 2012. doi:10.3233/JCS-2012-0444.

48 Joseph K Liu, Man Ho Au, Willy Susilo, and Jianying Zhou. Linkable ring signature with unconditional anonymity. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):157–165, 2013. doi:10.1109/TKDE.2013.17.

49 Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. A separable threshold ring signature scheme. In *ICISC 2003*, volume 2971, pages 12–26. Springer, 2003. doi:10.1007/978-3-540-24691-6_2.

50 Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In *ACISP 2004*, volume 3108, pages 325–335. Springer, 2004. doi:10.1007/978-3-540-27800-9_28.

51 Jiazhuo Lyu, Zoe Lin Jiang, Xuan Wang, Zhenhao Nong, Man Ho Au, and Junbin Fang. A secure decentralized trustless e-voting system based on smart contract. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 570–577. IEEE, 2019. doi:10.1109/TRUSTCOM/BIGDATASE.2019.00082.

52 Markus Michels and Patrick Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 125–132. Springer, 1996. doi:10.1007/BFB0034841.

53 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, May 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

54 Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols, 5th International Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 1997. doi:10.1007/BFB0028157.

55 Jiaxin Pan and Benedikt Wagner. Lattice-based signatures with tight adaptive corruptions and more. In *PKC 2022*, volume 13178, pages 347–378. Springer, 2022. doi:10.1007/978-3-030-97131-1_12.

56 Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC 2008*, pages 187–196. ACM, 2008. doi:10.1145/1374376.1374406.

57 Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC 1989*, pages 73–85. ACM, 1989. doi:10.1145/73007.73014.

58 Antonio Russo, Antonio Fernández Anta, María Isabel González Vasco, and Simon Pietro Romano. Chirotonia: A scalable and secure e-voting framework based on blockchains and linkable ring signatures. In *2021 IEEE International Conference on Blockchain, Blockchain 2021*, pages 417–424. IEEE, 2021. doi:10.1109/BLOCKCHAIN53845.2021.00065.

59 Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth. In *EUROCRYPT 1995*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 1995. doi:10.1007/3-540-49264-X_32.

60 Rowan van Pelt, Slinger Jansen, Djuri Baars, and Sietse Overbeek. Defining blockchain governance: A framework for analysis and comparison. *Inf. Syst. Manag.*, 38(1):21–41, 2021. doi:10.1080/10580530.2020.1720046.

61 Sarad Venugopalan and Ivan Homoliak. Always on voting: A framework for repetitive voting on the blockchain. *CoRR*, abs/2107.10571, 2021. arXiv:2107.10571.

62 Baodian Wei, Yusong Du, Huang Zhang, Fangguo Zhang, Haibo Tian, and Chong-zhi Gao. Identity based threshold ring signature from lattices. In *NSS 2014*, volume 8792, pages 233–245. Springer, 2014. doi:10.1007/978-3-319-11698-3_18.

63 Duncan S. Wong, Karyin Fung, Joseph K. Liu, and Victor K. Wei. On the rs-code construction of ring signature schemes and a threshold setting of RST. In *ICICS 2003*, volume 2836, pages 34–46. Springer, 2003. doi:10.1007/978-3-540-39927-8_4.

64 Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS 1982*, pages 160–164. IEEE Computer Society, 1982. doi:10.1109/SFCS.1982.38.