Blockchain Governance via Sharp Anonymous Multisignatures

Wonseok Choi, Xiangyu Liu, and Vassilis Zikas

Purdue University, West Lafayette, IN, USA
{wonseok, liu3894, vzikas}@purdue.edu

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 put 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, ♯AMS) that tightly meets the needs of blockchain governance. In a nutshell, ♯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 ♯AMSs and using them in various governance scenarios—e.g., single vs. multiple vote per voter. To this direction, we observe that although the definition of TRS does not imply ♯AMS, one can compile some of the existing TRS constructions into ♯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 ♯AMS are, in fact, instantiations of our template. This abstraction makes our template generic for instantiating TRSs and ♯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 fully understand its nature.

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

Keywords: blockchain, e-voting, threshold ring signature, threshold cryptography

1 Introduction

Blockchain and Blockchain Governance. Since the emergence of Bitcoin [48] 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.

In the aftermath of the hard forks experienced by Bitcoin and Ethereum [13], the notion of blockchain governance has arisen to establish communal consensus in the management and distribution of decentralized ledgers, as well as the evolution and growth of blockchain ecosystems. Blockchain governance encompasses a system wherein members of a blockchain network engage in decentralized decision-making and consensus-building. This process is pivotal for the advancement and prosperity of blockchains as it determines various aspects, including blockchain mining and rewards, consensus algorithms, cryptoeconomics, development direction, and more.

There are two basic governance mechanisms: off-chain governance, as seen in Bitcoin and Ethereum, and on-chain governance, exemplified by projects like Algorand [20], Tezos [34], 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 employed, they share a common vulnerability—the potential for community divisions, typically manifesting as hard forks. A hard fork occurs when blockchain community stakeholders disagree on a critical change, leading some to stick with the current chain while others embrace the new one. Alternatively, multiple competing updates may be proposed, further fragmenting the community. These divisions can erode community cohesion, reduce the platform's overall value, and jeopardize its security.

The security aspect is particularly concerning because a reduced number of resources supporting a fork can render it susceptible to attacks, such as 51% attacks, where malicious actors gain control over the majority of the network's computing power. Consequently, it is imperative for blockchain platforms to navigate governance challenges cautiously to uphold their stability and security while fostering a sense of unity within their communities.

On-Chain Governance and Voting Systems for Improvement Proposals. On-chain governance has arisen as a substitute for informal governance systems, aiming to tackle the issues of centralization linked to blockchain technology. This method incorporates all nodes within a blockchain network into the decision-making procedure. It is, however, important for the blockchain community to guarantee the decentralization of on-chain governance and prevent it from being unduly dominated by a limited faction of developers and miners who might unilaterally enforce alterations. This vigilance is motivated by the potential for conflicts and hard forks as blockchain networks evolve, potentially causing divisions within the blockchain community.

Central to on-chain governance, the voting mechanism stands out as a vital component. In a typical on-chain governance scenario, the process initiates with an improvement proposal, which outlines potential developments for the entire ecosystem and may involve substantial changes. Once such a proposal is put forward, on-chain governance enables individuals, i.e. stakeholders, to participate in voting to determine whether to endorse or reject the proposal.

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

Voting System Requirements. There are several foundational requirements for a robust voting system.

Correctness. The voting result's accuracy is perhaps the more 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 revolves around 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 [44, 28] could potentially be unveiled due to inadequacies in the underlying computational assumptions. Therefore, for optimal applicability in blockchain governance, the assurance of unconditional anonymity is one of the imperative features of a voting system in blockchain.

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.

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 Signature (LRS). Utilizing linkable ring signatures (LRS) is a prevalent tool in constructing e-voting systems. In linkable ring signatures

[44], a user can sign a message on behalf of a group/ring while maintaining anonymity, as their identity remains concealed within the signature. Moreover, if a user signs two ring signatures 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 [28], where the link algorithm establishes the link between signatures and discloses the signer's identity. However, unconditional anonymity is compromised in traceable ring signatures. Striking a balance between traceability and unconditional anonymity in ring signatures appears challenging, presenting a fundamental trade-off within this cryptographic context. Achieving both properties concurrently remains an open problem.

Multi-Party Computation. Multi-Party computation (MPC) [65, 33, 9, 17] appears to be the ultimate solution to the above voting problem. MPC allows n parties to compute any given function on their inputs in a secure manner, so that no malicious party (or coallition) can learn the inputs of other parties (privacy) and no party can affect the output any more than choosing their own input. As such 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, 17, 53]) 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 [37] (which ensures that upon abort the identity of a cheater is revealed)[1].

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 [33], an assumption which is unrealistic in our setting[2]. And even if one is willing to resort to security with (identifiable) abort and no fairness[3], we still need to implement byzantine broadcast—

[1] In fact, 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., [3].

[2] 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 [29], they come at a high cost in terms of blockchain utilization—multiple on-chain rounds—which renders them mainly of theoretical interest.

[3] This is already a discount in security which should anyway be avoided.

which requires in the worst case a polylogarithmc in the party-setsize number of parties—or use again several on-chain rounds. In fact, 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.

Beyond Signature Schemes to An Interactive Structure. The above discussion highlights the inherent challenge of satisfying all requirements simultaneously. However, despite the seemingly conflicting nature, there is a way to address this. Our paper provides a positive answer by proposing a structured approach during the signature generation process.

We can overcome the seemingly inherent limitations by carefully defining an interactive structure, a simple protocol, within the signature generation. This structured approach is a key innovation that allows us to design a signature scheme that meets the dual objectives of concealing individual identities while revealing the number of signers involved—a critical advancement for a secure and reliable e-voting system.

We call this new signature protocol Sharp Anonymous Multisignatures (♯AMS). Here, the term "sharp" is used consistently with complexity theory, as in $\sharp P$ to indicate that our signatures output the number of signers that validly signed, rather than a bit (valid or not). On the other hand, the term "anonymous", is used to designate that signers' identities are hidden. Equipped with these properties, ♯AMS is the perfect match for e-voting systems in particular in the context of blockchain governance. We discuss the details of our contributions below.

Post-Quantum Security. The security of many e-voting systems, as seen in works such as [44, 28, 60], is based on standard number theoretic assumptions like the (hardness of) computing discrete logarithms and RSA. However, these assumptions may become vulnerable when faced with quantum computers, given the efficiency of quantum algorithms like Shor's algorithm for factoring and solving discrete logarithm problems [56, 57], as well as Regev's recent work [54].

Post-quantum cryptography has emerged as a focus of extensive research and development over the past decades to mitigate this potential vulnerability. Among various post-quantum cryptographic schemes, lattice-based approaches have gained traction due to their attractive features, such as simplicity, efficiency, and worst-case hardness. This is underscored by the fact that in the final round of NIST's post-quantum standards [49], three out of four candidates are based on lattice problems.

1.1 Our Contributions

♯ *Anonymus MultiSignatures.* Our first and major contribution is proposing and formalizing a new concept of signing *protocol*, dubbed ♯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 (TRSs), ♯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 varies 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 ♯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 ♯AMSs, we provide a generic template for TRSs that, (1) abstracts many such "liftable" schemes, and (2) admits a generic compiler to transform to ♯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.

♯AMS Constructions from Chameleon Hashes. Using the above, we provide concrete ♯AMS schemes from (lossy) chameleon hashes in a black-box model. 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. This system can tolerate malicious voters (the voters who claimed to vote but quit later). Including the posting period and the announcement period, the 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 with the single-vote setting from the conditioned key generation paradigm. V3 requires one more on-chain round when there exist malicious users. However, If the maximum number of proposals is pre-known, the voting process still takes two on-chain blocks like V1 and V2.

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.

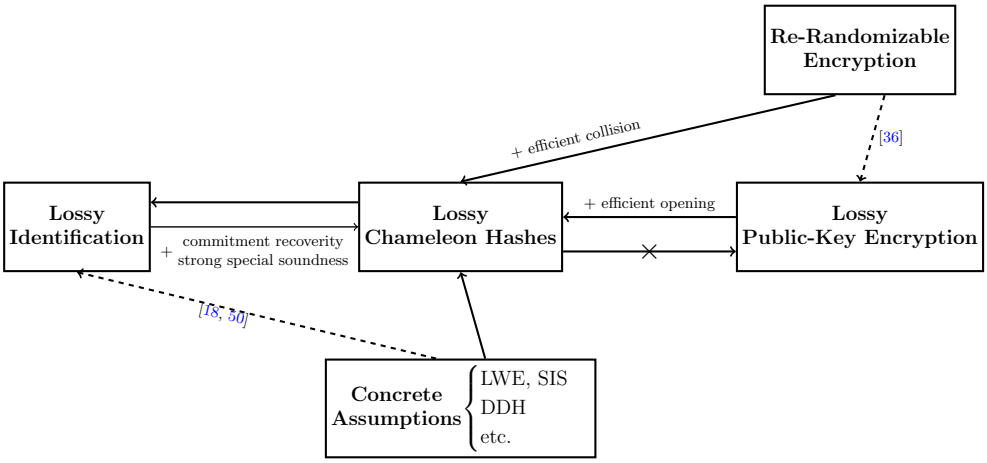

Fig. 1. A Framework for Lossy Chameleon Hashes (solid arrows: shown in this work; dashed line: shown in previous works).

1.2 Related Works

Beck *et al.* [4], Pelt *et al.* [52], and Kiayias and Lazos [39] discussed core properties for blockchain governance in multiple disciplines. Khan *et al.* [38], Venugopalan and Homoliak *et al.* [61], and Gersbach *et al.* [32] also focused on blockchain governance especially based on decision-making processes and voting in terms of game-theoretical analyses.

The concept of ♯AMS we proposed in this paper has close connection with existing cryptographic primitives like multisignature, ring signature, threshold signatures, etc. Here we review some related works on them.

Multisignatures. Multisignatures (MS) [11, 6] allow n different parties generate a single signature on a common message, and the size of the signature is short. Multisignatures are massively applied in the blockchain to reduce the storage of blockchain transcations. However, the privacy is not considered in multisignatures, and the identities of signers/authors are totally exposed in the signature.

Threshold signatures. A (t, n)-threshold signature scheme enables a group of n parties to sign on a message if more than t parties participate in the signing. Besides, the threshold t and the quorum of t participants are hidden from the signature. Most threshold signatures [11, 58, 59, 40] are based on the secret sharing scheme, and they all face a shortcoming that the threshold t needs to be fixed before the generation of public/secret keys.

Ring signatures. In ring signatures (RS) [55] (a.k.a. spontaneous anonymous group signatures [16]), a user can sign a message on behalf of a group without revealing its identity. But different from group signatures [19] where there is a group manager that is responsible for the key generation and traceability if

necessary, in ring signatures there is no group manager and anonymity holds unconditionally.

We would like to discuss more about threshold ring signatures (TRS) [12], which are highly related to ♯AMS in this paper. A (n, t)-TRS scheme allows t or more members 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. And 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 singing process. Therefore, ♯AMS is stronger that TRS.

There are many threshold ring signature schemes [43, 16, 63, 21] whose threshold t is changeable every signing. By adding t into the message to be signed, a TRS scheme with flexible threshold can be tuned to a ♯AMS scheme.

1.3 Technical Overview

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

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

The first issue is the anonymity property, which means that the identities of t actual signers are hidden in the total n $(n \geq t)$ users. Besides, as a (group) signature scheme, ♯AMS should have the unforgeability property, namely any adversary controlling less 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 its share using its own secret key, and the signature is just a concatenation of different shares (as in linkable/traceable ring signatures), then it seems impossible achieve both unconditional anonymity and traceability[4]. Therefore, we focus on interactive signing processes, and introduce a moderator P in the signing protocol. To generate a ♯AMS signature, every signer just communicates with P but not other signers, and it is P's responsibility to count the number of participants and finally outputs a ♯AMS signature. In this case, even the signer itself cannot de-anonymized a ♯AMS signature (i.e., it is unaware of other participants). P can be a member of G or not, and it is assumed to be honest. We believe it is a simple but reasonable assumption. Even a malicious P later leaks the quorum of signers, its own identity will be revealed at the same time. And consenquently, P will face a punishment from the system.

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

Generic Compiler from Threshold Ring Signatures. A ♯AMS scheme directly implies a threshold ring signature (TRS) scheme, since the verification algorithm Ver in ♯AMS returns the real number of participants in signing, while Ver in TRS returns only one bit. Therefore, the definition of ♯AMS is stronger than TRS.

Nevertheless, we surprisingly notice that, many TRS schemes (e.g., [43, 16, 15, 62, 35]) have a *flexible threshold* property. Namely, the threshold t does not need to be fixed when generating public/secret key pairs, and it is changeable every signing. Then we design a generic compiler that tunes any TRS scheme with flexible threshold to a ♯AMS scheme:

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

Thanks to the compiler and the full literature of TRS, we immediately obtain a lots of ♯AMS schemes, from the DL assumption [16], the RSA assumption [43], the SIS assumption [15], the code assumption [23], etc.

C1: Construction from (Lossy) Chameleon Hashes. Chameleon hash (CH) function [41] is a special hash function indexed by a hash key hk, which is associated with a trapdoor td. It has two parts of input, message m[5] and randomness r. On the one hand, given the hash key only, it is hard to find a collision. On the other hand, one can easily find collisions with the help of the trapdoor.

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

Now we extend it to the ♯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 t-out-of-n zero-knowledge proof from [22] by Cramer, Damgård, and Schoenmakers. That is, given n random m_i, the group of signers are able to find t randomness r_i that makes the collisions happen. For the message msg, we design a ♯AMS signature as in the form of $\sigma = (t, m_1, ..., m_n, r_1, ..., r_n)$, where $m_1, ..., m_n$ are require to satisfy the following linear equations.

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

[5] 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 notation to keep consistent with previous works.

Here coefficients $(a_{i,j})$ are public parameters, $(u_1, ..., u_t)$ are the random outputs of some hash function $H(h_1, ..., h_n, \mathsf{msg}, t)$, and $(h_1, ..., h_n)$ are the corresponding hash values. Now, to satisfy the equation (1), at least t trapdoors are necessary to find out the corresponding randomness r_i, which proves that at least t users have participated in the signing process.

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

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

The security of the CH-based signature scheme, as well as our ♯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 [6] to find a collision. Following the idea of lossy trapdoor functions [51] and lossy encryption [5, 36], in this work we define lossy chameleon hashes (LCH), to make a black-box reduction in the security proof. An LCH scheme can be worked in two modes, the collision mode, which is as a normal CH, and the lossy mode, where a lossy hash key is used instead of a normal hash key, and the adversary (even it is computationally unbounded) cannot find a randomness for a random message that hashes to a previous hash value of its own choice.

We present an efficient instantiation of LCH from the learning with error (LWE) assumption and the short integer solution (SIS) assumption. We also explore the relationship between LCH and some other primitives, including lossy identification schemes [1], re-randomizable encryption, and lossy encryption [5, 36], see Fig. 1.

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. We develop a fault-tolerant variant of our ♯AMS scheme. This is due to the fact that our (L)CH-based ♯AMS scheme has "backward compatibility". Namely, even one voter quits at the middle, by exposing the identity of that faulty voter, the developer can still outputs a signature with creditablity $(t-1)$. More precisely, let F be the set of faulty signers. In the fault-tolerand ♯AMS scheme, we tailor the signature in to the form of

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

If there exists a solution $\{m_i\}_{i \in F}$ for 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: Blochchain Governance via ♯AMS Schemes. ♯AMS directly implies a voting protocol since a ♯AMS signature leaks the number of the signers. The protocol consists of the following four periods using scheme C2.

1. The posting period, where each developer publishes its improvement proposal on the blockchain.
2. The declaration period, where each voter (the signer in ♯AMS) claims its willingness to the developer, by sending a hash value of dummy message and randomness to the developer.
3. The signing period, where the developer performs the signature algorithm based on the number of supporters. Specifically, developers calculates all m_i according to the received h_i and equation (1) or (2), and then returns m_i to the supporters. Then the supporters use their own trapdoors to find collisions r_i and return them to the developers, helping the developer finally finishing the generation of the ♯AMS signature.
4. The announcement period, where each developer uploads its ♯AMS signature on the blockchain. Then the voting result is uploaded on the block and published all over the network.

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

To generate a ♯AMS signature, the developer needs three rounds interaction with its supports in the signing period. This makes the faulty attack by malicious voters feasible. Namely, the voter first sends h_i and claims the participation of voting, but later after receiving the message m from the developer, it aborts or sends a faulty randomness r'. Thanks to our fault-tolerant ♯AMS scheme, this attack would not threaten the whole voting system, since a (fault-tolerant) ♯AMS signature always tells the real number of (honest) participants.

V2: Round Optimization. Notice that voters in the protocol above need to wait for the message m_i from the developers after declaring their supports. To achieve "vote-and-go" property, we further consider optimizing the protocol to one-round. Our idea is to use ♯AMS in a one-time paradigm. That is, we now let the voters U_i (no mater whether it wants to vote on the proposal), generates a new hash key and trapdoor pair $(hk_i^{(j)}, td_i^{(j)})$ for some proposal by the develop P_j. And the supports of P_j secretly send their trapdoors to P_j using a standard public-key encryption scheme. With the knowledge of all secret keys, P now can generate a ♯AMS signature without the interaction with its supporters. To prevent a malicious developer generating one-time hash keys by itself, we additionally require that every hash key is attached with a signature, showing the authority by its owners.

V3: Single-Vote Setting via the Conditioned Key Generation Paradigm. We further extend the above protocol into the single-vote setting, where each user in the voting system can vote on one proposal at most. Since there exist multiple developers P_j and each of them may have its own proposal, every voter U_i now needs to generate multiple key pairs $\{(hk_i^{(j)}, td_i^{(j)})\}$. The superscript (j) indicates that this key pair is used for the j-th proposal.

The protocol does not work in the single-vote setting, since a malicious voter might vote on different proposals using different trapdoors. We use the same

method in constructing ♯AMS to prevent multiple votes. Recall that to vote on some 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 $hk_i^{(j)}$ that is designed for $IP^{(j)}$ particularly. Suppose there are p proposals $(IP^{(j_1)}, ..., IP^{(j_p)})$, then U_i needs to generate p hash keys $(hk_i^{(j_1)}, ..., hk_i^{(j_p)})$ first. Our idea is to take a so-called *conditioned key generation paradigm*. Namely, we set a restriction among totally p hash keys so that the voter can know at most one trapdoor.

Let $\mathcal{HK}$ be the space of hash key for (lossy) chameleon hashes, and $\mathbf{B} = (b_{i,j}) \in \mathcal{HK}^{(p-1)\times p}$ be a matrix of full rank. Let $\mathbf{hk}_i^\top = (hk_i^{(j_1)}, ..., hk_i^{(j_p)})^\top$. 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 $H(\cdot)$ is a random oracle, then to satisfy the abovementioned equation, every user has at most one trapdoor of the total p hash keys. And the single-voting property is achieved as a result.

1.4 Roadmap

This paper is organised as follows. In Section 2, we present basic notations and introduce the definition 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). In Section 6, we propose three voting systems and show their application in blockchain governance.

2 Preliminaries

Let $\lambda \in \mathbb{N}$ denote the security parameter. For $\mu \in \mathbb{N}$, define $[\mu] := \{1, 2, ..., \mu\}$. Denote by $x := y$ the operation of assigning y to x. Denote by $x \xleftarrow{\$} \mathcal{S}$ the operation of sampling x uniformly at random from a set $\mathcal{S}$. For a distribution $\mathcal{D}$, denote by $x \leftarrow \mathcal{D}$ the operation of sampling x according to $\mathcal{D}$. For an algorithm $\mathcal{A}$, denote by $y \leftarrow \mathcal{A}(x; r)$, or simply $y \leftarrow \mathcal{A}(x)$, the operation of running $\mathcal{A}$ with input x and randomness r and assigning the output to y. For deterministic algorithms $\mathcal{A}$, we also write as $y := \mathcal{A}(x)$ or $y := \mathcal{A}(x; r)$. "PPT" is short for probabilistic polynomial-time. We refer Appendix A for more primitives.

Statistical distances and min-entropy. Let X and Y be two random variables (distributions) defined over $\mathcal{S}$. The min-entropy of X is defined as $\mathbf{H}_\infty(X) := -\log(\max_{s\in\mathcal{S}} \Pr[X = s])$. And the statistical distance between X and Y is defined as $\Delta_{X,Y} := 1/2 \sum_{s\in\mathcal{S}} |\Pr[X = s] - \Pr[Y = s]|$. If $\Delta_{X,Y} = \epsilon$, we also say X and Y are ϵ-close.

Now, we recall the definition of CH, and propose its lossy variant, i.e., lossy chameleon hashes (LCH).

2.1 Chameleon Hashes and Their Lossy Variants

Definition 1 (Chameleon Hashes [41]). *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* $\mathsf{Hash}(hk, m, r) = \mathsf{Hash}(hk, m', r')$*.*

Definition 2 (Security of CH). *A chameleon hash scheme* CH *is secure (resp., strongly secure) if it has uniformity, random trapdoor collision, and collision resistance (resp., strong collision resistance).*

κ-uniformity. *For any* $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$*, if* (m, r) *is distributed uniformly over* $\mathcal{M} \times \mathcal{R}$*, then* $\mathbf{H}_\infty(\mathsf{Hash}(hk, m, r)) \geq \kappa$*. Specifically, if* $\mathbf{H}_\infty(\mathsf{Hash}(hk, m, r)) = \log(|\mathcal{H}|)$*, we say that* CH *has perfect uniformity*[6]*.*

γ-random trapdoor collision (RTC). *For any* $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$ *and any* m, m'*, if* r *is uniformly distributed over* $\mathcal{R}$*, then* $r' \leftarrow \mathsf{TdColl}(td, m, r, m')$ *has a statistical distance* γ *to the uniform distribution over* $\mathcal{R}$*. If* $\gamma = 0$*, we say that* CH *has perfect RTC.*

Collision resistance (CR). *For any PPT adversary* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}^{cr}_{\mathsf{CH},\mathcal{A}}(\lambda) := \Pr\left[\begin{array}{c}(hk, td) \leftarrow \mathsf{Gen}(1^\lambda);\\ (m, r, m', r') \leftarrow \mathcal{A}(hk)\end{array} : \begin{array}{c}\mathsf{Hash}(hk, m, r) = \mathsf{Hash}(hk, m', r')\\ \wedge\ m \neq m'\end{array}\right].$$

is negligible in λ*.*

Strong collision resistance (S-CR). *For any PPT adversary* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}^{s\text{-}cr}_{\mathsf{CH},\mathcal{A}}(\lambda) := \Pr\left[\begin{array}{c}(hk, td) \leftarrow \mathsf{Gen}(1^\lambda);\\ (m, r, m', r') \leftarrow \mathcal{A}(hk)\end{array} : \begin{array}{c}\mathsf{Hash}(hk, m, r) = \mathsf{Hash}(hk, m', r')\\ \wedge\ (m, r) \neq (m', r')\end{array}\right].$$

is negligible in λ*.*

Remark 1 (Random trapdoor collision in lattice-based constructions). For lattice-based CH schemes (e.g., [14]), the randomness r is sampled according to a (non-uniform) distribution $\tilde{\mathcal{R}}$ over $\mathcal{R}$. In this case, we modify the definition to the case over non-uniform distributions. Namely, for any m, m', if r is sampled according to $\tilde{\mathcal{R}}$, then the output $r' \leftarrow \mathsf{TdColl}(td, m, r, m')$ enjoys a distribution which is γ-close to $\tilde{\mathcal{R}}$.

[6] We relax the definition of uniformity by taking m's randomness into the probability space. A stronger definition guarantees the min-entropy for any (hk, td) and any m. As we will see, the relaxed definition here is sufficient for the security proof in Section 5.

Now, 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 Definition 1, 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*.*

Definition 4 (Security of LCH). *A lossy chameleon hash* LCH *is secure (resp. strongly secure), if it has uniformity, random trapdoor collision (RTC), collision resistance (resp., strong collision resistance), indistinguishability, and lossiness. The first three properties are defined as in Definition 2 (note that uniformity and RTC also hold for all* $hk \leftarrow \mathsf{LGen}(1^\lambda)$*), and the last two are defined as follows.*

Indistinguishability. *For any PPT adversary* $\mathcal{A}$*, the distinguish advantage*

$$\mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{A}} := |\Pr[(hk, td) \leftarrow \mathsf{Gen}(1^\lambda) : \mathcal{A}(hk) = 1] - \Pr[hk \leftarrow \mathsf{LGen}(1^\lambda) : \mathcal{A}(hk) = 1]|$$

is negligible in λ*.*

ϵ-Lossiness. *For any (even information-theoretic) adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, it holds that*

$$\Pr\left[\begin{array}{c} hk \leftarrow \mathsf{LGen}(1^\lambda); (h, st) \leftarrow \mathcal{A}_1(hk); \\ m \xleftarrow{\$} \mathcal{M}; r \leftarrow \mathcal{A}_2(m, st) \end{array} : \mathsf{Hash}(hk, m, r) = h\right] \leq \epsilon.$$

Remark 2 (The difference between LCH and lossy identification scheme). Lossy chameleon hashes have a very close relationship with lossy identification schemes [1]. In Appendix C we formally prove that, under specific conditions, lossy chameleon hashes and lossy identification schemes are equivalent. This result derives from the equivalence between chameleon hashes and sigma protocols proved by Bellare and Ristov [8]. We also show a framework for constructing LCH from other primitives like re-randomizable encryption and lossy encryption, and show direct constructions from several well-known hardness assumptions (e.g., LWE & SIS, DDH), see Fig. 1.

For the proof of strong unforgeability in Section 5, we also need the uniqueness property of (L)CH defined as follows.

Definition 5 (Uniqueness of (L)CH). *We say* CH *or* LCH *is perfectly unique, if for every* hk *generated from* $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$ *and* $hk \leftarrow \mathsf{LGen}(1^\lambda)$*, every* m*, there do not exist two district* $r \neq r'$ *such that* $\mathsf{Hash}(hk, m, r) = \mathsf{Hash}(hk, m, r')$*.*

3 Definition and Security Notions of Sharp Anonymous Multisignatures

In this section we formally define sharp anonymous multisignatures and their security properties. Let n be the total number of signers, and $G \subseteq [n]$ be a group of signers that want to generate a ♯AMS signature together. Let $t = |G|$.

Definition 6 (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 *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.*
  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* $\mathsf{Ver}(vk, \mathsf{msg}, \sigma) = |G|$*.*

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.

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

Remark 3 (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 Definition 7, $\mathcal{A}$ is forbidden to query $\mathcal{O}(\mathsf{msg}^*, P, G)$ with $|G| = t^*$.

Definition 8 (Strong Unforgeability). *Consider the strong unforgeability experiment* $\mathsf{Exp}^{s\text{-}unforg}_{\sharp AMS,\mathcal{A}}(\lambda)$*, which is defined as* $\mathsf{Exp}^{unforg}_{\sharp AMS,\mathcal{A}}(\lambda)$ *in Definition 7, except that condition* (2) *is replaced with*

(2') $(\mathsf{msg}^*, \sigma^*) \notin \mathcal{S}$*.*

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

We also define the weak unforgeability and (strong/weak) unforgeability under static corruptions, see Appendix B.

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

Definition 9 (Strong Anonymity of $\sharp$AMS). *Consider the following strong anonymity experiment* $\mathsf{Exp}^{s\text{-}anony}_{\sharp AMS,\mathcal{A}}(\lambda)$ *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'*.*

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

The abovementioned 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 $i \in (G_0 - C_1) \cup (G_1 - G_0)$ 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 [44]. I.e., 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 ♯AMS from TRS with a Flexible Threshold

In this section, we provide a generic transformation for most threshold ring signature (TRS) schemes, dubbed TRS with flexible threshold, which implies that such TRS schemes are more versatile than their original definitions. First, we will see issues of TRS schemes from the perspective of communication. Then we give a generalized compiler of ♯AMS from TRS constructions that can represent most TRS schemes in the literature.

4.1 Issues of Threshold Ring Signatures

We first recall the definition of threshold ring signatures (TRS) from [12] and [43].

Definition 10 (TRS). *A threshold ring signature (TRS) scheme consists of three algorithms:* $\mathsf{TRS} = (\mathsf{Gen}, \mathsf{TSign}, \mathsf{Ver})$.

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda, t)$*: The key generation algorithm takes as input the security parameter* 1^λ *and a threshold* t*, and outputs a public key* pk *and a secret key* sk*.*
- $\sigma \leftarrow \mathsf{TSign}(\mathsf{msg}, R, \{pk_i\}_{i\in R}, T, \{sk_i\}_{i\in T})$*: The threshold signing algorithm/protocol takes as input a message* msg*, a group of users* R *and their public keys* $\{pk_i\}_{i\in R}$*, a group of real signers* T *and their the secret key* $\{sk_i\}_{i\in T}$*, and outputs a signature* σ*.*
- $1/0 \leftarrow \mathsf{Ver}(t, R, \{pk_i\}_{i\in R}, \mathsf{msg}, \sigma)$*: The verification algorithm* Ver *takes as input* t*, a group of users* R *and their public keys* $\{pk_i\}_{i\in R}$*, a message* msg *and a signature* σ*, and outputs a bit indicating the validity of* σ*.*

Correctness. *For any* $T \subseteq R$ *such that* $|T| \geq t$*, any* msg *and* $\sigma \leftarrow \mathsf{TSign}(\mathsf{msg}, R, \{pk_i\}_{i\in R}, T, \{sk_i\}_{i\in T})$*, it holds that* $\mathsf{Ver}(t, R, \{pk_i\}_{i\in R}, \mathsf{msg}, \sigma) = 1$.

In many previous works [12, 43, 16, 21, 23], the threshold signing process TSign is just regarded as an algorithm, but rather than an interactive protocol among at least t signers. Under such definitions, the behavior of TRS becomes ambiguous:

Does the signer know the threshold t? Does the anonymity hold among the signer group T, i.e., whether a signer in T knows its cooperators' identity?

To address this issue, when defining ♯AMS we introduce a moderator in the signing process. We stress that the moderator perceives the quorum of the signers (hence also t) during the signing process, and every signer only needs to communicate with the moderator. Moreover, even if the moderator gets corrupted later and is willing to reveal the quorum of real signers, it either cannot convince others about the disclosure in our construction C1, or will leak its identity as well and consequently be penalized from the system in our construction C2.

4.2 Relationship between TRS and ♯AMS

Our newly defined ♯AMS closely relates to TRS, though ♯AMS is a stronger primitive than TRS.

1. By the definition of TRS, a group of t signers can get together to sign a signature. And the threshold t might be fixed (e.g., [42, 64]) or not (e.g., [43, 16, 10]) when generating the public/secret key pairs. But in ♯AMS, any number of signers can cooperate and finally output a signature. That is, the creditability t is flexible in every signing.
2. The output of the verification algorithm in TRS is a single bit indicating whether a signature w.r.t. a message and a threshold is valid or not. While in ♯AMS, the output t is an integer indicating the *real* number of signers who have participated in the signing process, which offers more information than that in TRS.

Owing to the security concerns in ad-hoc networks, TRS has attracted increasingly widespread attention for the past decades. In fact, for most existing TRS schemes [12, 43, 16, 45, 15, 10, 62, 35], the threshold t is changeable every signing, though how to share information of t is ambiguous from the definition. We call this kind of scheme *TRS with flexible threshold*. Next, we show a generic compiler to tune any TRS scheme with flexible threshold to our ♯AMS scheme.

Generic compiler from TRS with flexible threshold to ♯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 ♯AMS scheme ♯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 ♯AMS signature[7].

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

– $\mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. 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 ♯AMS constructed above inherits from the security of the underlying TRS scheme, and we omit the detailed proof here due to the page limitation.

Remark 4 (Why sign the threshold t together with the message msg?). A secure signature should include all public information, e.g., a receiver's address, side-channel information, and the threshold t in our case, into the message to be signed. We show an unsafe counterexample of ♯AMS from linkable ring signatures (LRS) [44] here. As introduced in the introduction, LRS allows a user to sign a message on behalf of a ring, and its two signatures on the same message will be linked. A natural idea for constructing ♯AMS from LRS is to let every signer in the group contribute its own linkable ring signature, and the verification algorithm just returns the count of unlinked and valid signatures. However, if the creditability t is not included in the message to be signed, then unforgeability does not hold anymore. To see this, suppose the forger now corrupted a signer U_i and gets its secret key. Then after seeing a t-valid ♯AMS signature σ generated by a group excluding U_i, the forger can easily output a $(t+1)$-valid ♯AMS signature $(\sigma||\sigma_i)$, where σ_i is a linkable ring signature by U_i. Another example of insecurity is shown in [63].

5 Constructions of ♯AMS from Lossy Chameleon Hashes

In this section we propose efficient constructions of ♯AMS from lossy chameleon hashes (LCH) in a black-box model.

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 [22].

Definition 11 (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 randomly chooses $\{m_i\}_{i \in [n]}$, and then computes/samples $\{m_i\}_{i \in G}$ and 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 two instantiations of $\mathbf{F}_\theta$ from linear equations and polynomial interpolation in Appendix F.

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 just the way of security proofs, but not the construction itself.

Construction. Let $\mathsf{LCH} = (\mathsf{LCH.Gen}, \mathsf{LCH.LGen}, \mathsf{LCH.Hash}, \mathsf{LCH.TdColl})$ 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 $\{sk_i\}_{i\in[n]} := \{td_i\}_{i\in[n]}$.
– $\mathsf{Sign}(\mathsf{msg}, P, G, \{sk_i\}_{i\in G})$.
  - 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.
  - The moderator counts $t := |G|$ from the received messages.
  - For every user $i \in [n] \setminus G$, P samples $m_i \xleftarrow{\$} \mathcal{M}, r_i \xleftarrow{\$} \mathcal{R}$ and computes $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$.
  - P invokes $u \leftarrow H(vk, h_1, ..., h_n, \mathsf{msg}||t)$. Then it computes $\{m_i\}_{i\in G}$ according to the backward sample algorithm $s_{back}(\cdot)$ of $\mathbf{F}_\theta$ such that

$$\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1.$$

    Then for $i \in G$, P sends m_i to signer i.
  - For every $i \in G$, 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.
  - 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 above exhibits strong generality. In Section 4, we have already shown a generic compiler from TRS with 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) [7] and the result in this work (Appendix C). For example, by instantiating with $\mathbf{F}_\mathbf{A}$ and the DL-based CH [41], we get the TRS scheme in [16], and by instantiating with $\mathbf{F}_p$ and the DL-based CH, we get the TRS scheme in [43]. Thanks to this generic construction, we immediately get more schemes of ♯AMS (also TRS) from lattices [31, 14], isogeny [25], and etc.

Theorem 1. *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 $\sharp\mathsf{AMS}$ constructed above has strong unforgeability and unconditional 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*

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

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

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 Appendix D for the proof due to the page limitation.

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

Theorem 2. *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 $\sharp\mathsf{AMS}$ 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 (Definition 18 in Appendix B). The proof can also be extended to the adaptive model (Definition 7 and 8), but if suffers from a large loss factor 2^n. We present the security bound under adaptive corruptions in Appendix E. 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 (Definition 1) in Appendix G, which has a quadratic loss but relies on the forking lemma (Definition 1 in Appendix A). Note that the reduction is tight in the algebraic group model (AGM) [27], if we instantiate ♯AMS with DL-based chameleon hashes [41].

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 in the following. Note that our constructions need communication with a developer within a fixed period to ensure the number of total singers. This leads to faulty attacks in the presence of a faulty node that declares to join the signing process but fails to generate a partial signature. In other words, the signing process suffers from the following attack even if only one faulty node exists: In the declaration period, the faulty node U_k faithfully follows the first two steps of the interaction. And after receiving m_i from P, it aborts or sends a faulty r_k.

Since P cannot compute r_k without td_k, it cannot produce a signature. Furthermore, since it computes $\hat{m} \leftarrow H^t(vk, h_1, ..., h_n, \mathsf{msg}||t)$ from h_j and $t = |G|$, P_j 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

$$\sigma := \left(t, F, \{m_i, r_i\}_{i\in[n]\setminus F}, \{h_i\}_{i\in F}\right).$$

– $t \leftarrow \mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. Pause $\sigma := \left(t, F, \{m_i, r_i\}_{i\in[n]\setminus F}, \{h_i\}_{i\in F}\right)$. Compute $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$ for all $i \in [n]\setminus F$. Let $u \leftarrow H(vk, h_1, ..., h_n, \mathsf{msg}||t)$. Return $(t-|F|)$ if there exist a solution $\{m_i\}_{i\in F}$ such that $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1$, 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 3. *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* $\max(Time(\mathcal{B}_1), Time(\mathcal{B}_2)) \approx Time(A)$,

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

and

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

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 Appendix H for the proof sketch.

6 Applications: Blockchain Governance and the Beyond

♯AMS can be massively applied in the scenario of blockchain and privacy-preserving, where the authenticity and the 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. Our first agenda is to make

solutions as simple as possible. While it might be possible to construct voting systems via complex compositions of signatures, zero-knowledge proofs, homomorphic/functional encryptions, and/or other cryptography schemes, there should be simpler ways, as we will see in this section.

6.1 V1: Blockchain Governance via ♯AMS

From the above observation, instead of using a pre-existing scheme in a black-box manner, we first introduce the notion of ♯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. This protocol enables multiple improvement proposals to compete for votes, and the one with the most votes is elected. A voting session begins with the community's consensus. Note that there should be a clear guideline that explicitly denotes the criteria for voting eligibility and specifies the duration of the voting period. With those things, a request for proposals (RFP) shall be published. After that, 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.

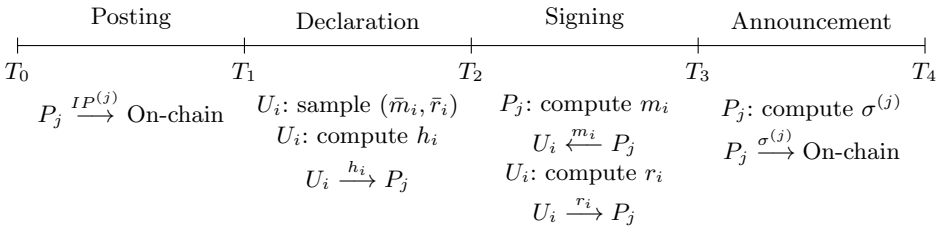

Fig. 2. Timeline of V1. Here U_i votes to $IP^{(j)}$.

- 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 ♯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 ♯AMS signature immediately after receiving the voting-claims from its supporters?

A 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 now can generate a ♯AMS signature without the interaction with its supporters. However, this will totally expose users' secret keys to the developers in one vote event, which is not the case users want.

Our idea is to use ♯AMS in a one-time paradigm. 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 that a hash key $hk_i^{(j)}$ is generated from voter U_i but not from developer P_j (otherwise it can always make an n-valid ♯AMS signature with n the total number of voters), U_i will sign a signature on $hk_i^{(j)}$ to show its authority. (See Appendix A for the syntax of signatures.)
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 their 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 $(hk_i^{(j)}, td_i^{(k)}) \leftarrow \mathsf{LCH.Gen}(1^\lambda)$ 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 ♯AMS signature $\sigma^{(j)}$ for its proposal $IP^{(j)}$ and uploads the ♯AMS signature, all hash keys, and certificates concerning the proposal $IP^{(j)}$ to the blockchain system.

Note that if U_i does any Byzantine behavior, e.g., U_i does not send $ct_{i\to j}$ to P_j, it will be easily detected after the end of the announcement period. If there is any such behavior, all developers can re-run the announcement period again, but with the participants set $[n] \setminus \{i\}$. It can be easily generalized to any number of faulty users, and the cost of this toleration will be one more announcement period.

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

A voting system in the single-vote setting is attractive 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 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 for $j \neq k \in [p]$. User U_i can generate two key pairs $(hk_i^{(j)}, td_i^{(j)})$ and $(hk_i^{(k)}, td_i^{(k)})$, and then vote on both the two proposals by sending two key pairs to P_j and P_k respectively. And due to the anonymity of ♯AMS, the verifier cannot know who in the group has actually voted twice.

We introduce a new paradigm, dubbed the *conditioned key generation paradigm*, to prevent double voting. 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 $hk_i^{(j)}$ that is designed for $IP^{(j)}$ particularly. U_i needs to generate p hash keys $(hk_i^{(1)}, ..., hk_i^{(p)})$ first. Our idea is to set a restriction 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 abovementioned equation, every user has at most one trapdoor of the p hash keys. And 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. See also Appendix J for further discussion.

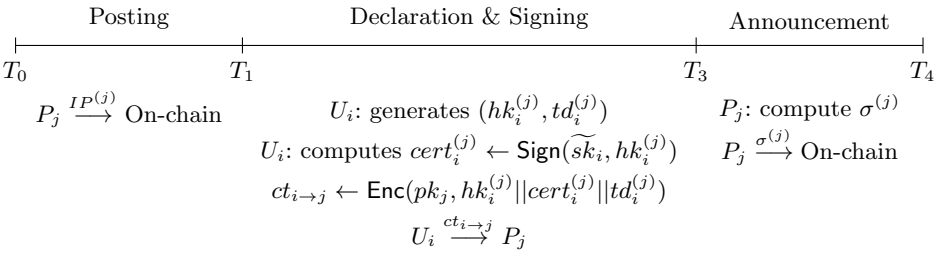

Fig. 3. Timeline of V2 and V3. Here U_i votes to $IP^{(j)}$. In V2, U_i also needs to compute and send $ct_{i\to k} = \mathsf{Enc}(pk_k, hk_i^{(k)}||cert_i^{(k)}||td_i^{(k)})$ or $ct_{i\to k} = \mathsf{Enc}(pk_k, hk_i^{(k)}||cert_i^{(k)}||\mathbf{0})$ to P_k for all $k \in [p]$. In V3, there should be 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] Abdalla, M., Fouque, P., Lyubashevsky, V., Tibouchi, M.: Tightly-secure signatures from lossy identification schemes. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology - EUROCRYPT 2012, Cambridge, UK, April 15-19, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7237, pp. 572–590. Springer (2012), https://doi.org/10.1007/978-3-642-29011-4_34

[2] Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996. pp. 99–108. ACM (1996), https://doi.org/10.1145/237814.237838

[3] Baum, C., Orsini, E., Scholl, P., Soria-Vazquez, E.: Efficient constant-round MPC with identifiable abort and public verifiability. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12171, pp. 562–592. Springer (2020), https://doi.org/10.1007/978-3-030-56880-1_20

[4] Beck, R., Müller-Bloch, C., King, J.L.: Governance in the blockchain economy: A framework and research agenda. J. Assoc. Inf. Syst. 19(10), 1 (2018), https://aisel.aisnet.org/jais/vol19/iss10/1

[5] Bellare, M., Hofheinz, D., Yilek, S.: Possibility and impossibility results for encryption and commitment secure under selective opening. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 1–35. Springer (2009), https://doi.org/10.1007/978-3-642-01001-9_1

[6] Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006. pp. 390–399. ACM (2006), https://doi.org/10.1145/1180405.1180453

[7] Bellare, M., Ristov, T.: Hash functions from sigma protocols and improvements to VSH. In: Pieprzyk, J. (ed.) Advances in Cryptology - ASIACRYPT 2008, Melbourne, Australia, December 7-11, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5350, pp. 125–142. Springer (2008), https://doi.org/10.1007/978-3-540-89255-7_9

[8] Bellare, M., Ristov, T.: A characterization of chameleon hash functions and new, efficient designs. J. Cryptol. 27(4), 799–823 (2014), https://doi.org/10.1007/s00145-013-9155-8

[9] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 1–10. ACM (1988), https://doi.org/10.1145/62212.62213

[10] Bettaieb, S., Schrek, J.: Improved lattice-based threshold ring signature scheme. In: Gaborit, P. (ed.) PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7932, pp. 34–51. Springer (2013), https://doi.org/10.1007/978-3-642-38616-9_3

[11] Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Desmedt, Y. (ed.) Public Key Cryptography - PKC 2003, Miami, FL, USA, January 6-8, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2567, pp. 31–46. Springer (2003), https://doi.org/10.1007/3-540-36288-6_3

[12] Bresson, E., Stern, J., Szydlo, M.: Threshold ring signatures and applications to ad-hoc groups. In: Yung, M. (ed.) Advances in Cryptology - CRYPTO 2002, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2442, pp. 465–480. Springer (2002), https://doi.org/10.1007/3-540-45708-9_30

[13] Buterin, V.: Ethereum: A next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)

[14] Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: Gilbert, H. (ed.) EUROCRYPT 2010, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6110, pp. 523–552. Springer (2010), https://doi.org/10.1007/978-3-642-13190-5_27

[15] Cayrel, P., Lindner, R., Rückert, M., Silva, R.: A lattice-based threshold ring signature scheme. In: Abdalla, M., Barreto, P.S.L.M. (eds.) Progress in Cryptology - LATINCRYPT 2010, Puebla, Mexico, August 8-11, 2010, Proceedings. Lecture Notes in Computer Science, vol. 6212, pp. 255–272. Springer (2010), https://doi.org/10.1007/978-3-642-14712-8_16

[16] Chan, T.K., Fung, K., Liu, J.K., Wei, V.K.: Blind spontaneous anonymous group signatures for ad hoc groups. In: Castelluccia, C., Hartenstein, H., Paar, C., Westhoff, D. (eds.) Security in Ad-hoc and Sensor Networks, First European Workshop, ESAS 2004, Heidelberg, Germany, August 6, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3313, pp. 82–94. Springer (2004), https://doi.org/10.1007/978-3-540-30496-8_8

[17] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 11–19. ACM (1988), https://doi.org/10.1145/62212.62214

[18] Chaum, D., Evertse, J., van de Graaf, J.: An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In: Chaum, D., Price, W.L. (eds.) Advances in Cryptology - EUROCRYPT '87, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings. Lecture Notes in Computer Science, vol. 304, pp. 127–141. Springer (1987), https://doi.org/10.1007/3-540-39118-5_13

[19] Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) Advances in Cryptology - EUROCRYPT '91, Brighton, UK, April 8-11, 1991, Proceedings. Lecture Notes in Computer Science, vol. 547, pp. 257–265. Springer (1991), https://doi.org/10.1007/3-540-46416-6_22

[20] Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theor. Comput. Sci. 777, 155–183 (2019), https://doi.org/10.1016/j.tcs.2019.02.001

[21] Chow, S.S.M., Hui, L.C.K., Yiu, S.: Identity based threshold ring signature. In: Park, C., Chee, S. (eds.) Information Security and Cryptology - ICISC 2004, Seoul, Korea, December 2-3, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3506, pp. 218–232. Springer (2004), https://doi.org/10.1007/11496618_17

[22] Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) Advances in Cryptology - CRYPTO '94, Santa Barbara, California, USA, August 21-25, 1994, Proceedings. Lecture Notes in Computer Science, vol. 839, pp. 174–187. Springer (1994), https://doi.org/10.1007/3-540-48658-5_19

[23] Dallot, L., Vergnaud, D.: Provably secure code-based threshold ring signatures. In: Parker, M.G. (ed.) Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5921, pp. 222–235. Springer (2009), https://doi.org/10.1007/978-3-642-10868-6_13

[24] Diemert, D., Gellert, K., Jager, T., Lyu, L.: More efficient digital signatures with tight multi-user security. In: Garay, J.A. (ed.) Public-Key Cryptography - PKC 2021, Virtual Event, May 10-13, 2021, Proceedings, Part II. Lecture Notes in

Computer Science, vol. 12711, pp. 1–31. Springer (2021), https://doi.org/10.1007/978-3-030-75248-4_1

[25] El Kaafarani, A., Katsumata, S., Pintore, F.: Lossy csi-fish: Efficient signature scheme with tight reduction to decisional CSIDH-512. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography - PKC 2020, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12111, pp. 157–186. Springer (2020), https://doi.org/10.1007/978-3-030-45388-6_6

[26] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986), https://doi.org/10.1007/3-540-47721-7_12

[27] Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 33–62. Springer (2018), https://doi.org/10.1007/978-3-319-96881-0_2

[28] Fujisaki, E., Suzuki, K.: Traceable ring signature. In: Okamoto, T., Wang, X. (eds.) Public Key Cryptography - PKC 2007, Beijing, China, April 16-20, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4450, pp. 181–200. Springer (2007), https://doi.org/10.1007/978-3-540-71677-8_13

[29] Garay, J.A., Kiayias, A., Ostrovsky, R.M., Panagiotakos, G., Zikas, V.: Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 129–158. Springer (2020), https://doi.org/10.1007/978-3-030-45724-2_5

[30] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. IACR Cryptol. ePrint Arch. p. 432 (2007), http://eprint.iacr.org/2007/432

[31] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.) Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008. pp. 197–206. ACM (2008), https://doi.org/10.1145/1374376.1374407

[32] Gersbach, H., Mamageishvili, A., Schneider, M.: Vote delegation and misbehavior. In: Caragiannis, I., Hansen, K.A. (eds.) SAGT 2021, Aarhus, Denmark, September 21-24, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12885, p. 411. Springer (2021), https://link.springer.com/content/pdf/bbm%3A978-3-030-85947-3%2F1.pdf

[33] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA. pp. 218–229. ACM (1987), https://doi.org/10.1145/28395.28420

[34] Goodman, L.: Tezos—a self-amending crypto-ledger white paper. URL: https://www. tezos. com/static/papers/white paper. pdf 4, 1432–1465 (2014)

[35] Haque, A., Scafuro, A.: Threshold ring signatures: New definitions and post-quantum security. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.)

Public-Key Cryptography - PKC 2020, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12111, pp. 423–452. Springer (2020), https://doi.org/10.1007/978-3-030-45388-6_15
[36] Hemenway, B., Libert, B., Ostrovsky, R., Vergnaud, D.: Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology - ASIACRYPT 2011, Seoul, South Korea, December 4-8, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7073, pp. 70–88. Springer (2011), https://doi.org/10.1007/978-3-642-25385-0_4
[37] Ishai, Y., Ostrovsky, R., Zikas, V.: Secure multi-party computation with identifiable abort. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8617, pp. 369–386. Springer (2014), https://doi.org/10.1007/978-3-662-44381-1_21
[38] Khan, N., Ahmad, T., Patel, A., State, R.: Blockchain governance: An overview and prediction of optimal strategies using nash equilibrium. CoRR abs/2003.09241 (2020), https://arxiv.org/abs/2003.09241
[39] Kiayias, A., Lazos, P.: Sok: Blockchain governance. In: Herlihy, M., Narula, N. (eds.) AFT 2022, Cambridge, MA, USA, September 19-21, 2022. pp. 61–73. ACM (2022), https://doi.org/10.1145/3558535.3559794
[40] Komlo, C., Goldberg, I.: FROST: flexible round-optimized schnorr threshold signatures. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) Selected Areas in Cryptography - SAC 2020, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12804, pp. 34–65. Springer (2020), https://doi.org/10.1007/978-3-030-81652-0_2
[41] Krawczyk, H., Rabin, T.: Chameleon hashing and signatures. IACR Cryptol. ePrint Arch. p. 10 (1998), http://eprint.iacr.org/1998/010
[42] Li, X., Zheng, D., Chen, K.: Efficient linkable ring signatures and threshold signatures from linear feedback shift register. In: Jin, H., Rana, O.F., Pan, Y., Prasanna, V.K. (eds.) ICA3PP 2007, Hangzhou, China, June 11-14, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4494, pp. 95–106. Springer (2007), https://doi.org/10.1007/978-3-540-72905-1_9
[43] Liu, J.K., Wei, V.K., Wong, D.S.: A separable threshold ring signature scheme. In: Lim, J.I., Lee, D.H. (eds.) Information Security and Cryptology - ICISC 2003, Seoul, Korea, November 27-28, 2003, Revised Papers. Lecture Notes in Computer Science, vol. 2971, pp. 12–26. Springer (2003), https://doi.org/10.1007/978-3-540-24691-6_2
[44] Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3108, pp. 325–335. Springer (2004), https://doi.org/10.1007/978-3-540-27800-9_28
[45] Melchor, C.A., Cayrel, P., Gaborit, P.: A new efficient threshold ring signature scheme based on coding theory. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5299, pp. 1–16. Springer (2008), https://doi.org/10.1007/978-3-540-88403-3_1
[46] Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology - EUROCRYPT

2012, Cambridge, UK, April 15-19, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7237, pp. 700–718. Springer (2012), https://doi.org/10.1007/978-3-642-29011-4_41
[47] Micciancio, D., Regev, O.: Worst-case to average-case reductions based on gaussian measures. In: 45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings. pp. 372–381. IEEE Computer Society (2004), https://doi.org/10.1109/FOCS.2004.72
[48] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (May 2009), http://www.bitcoin.org/bitcoin.pdf
[49] NIST: Nist to standardize encryption algorithms that can resist attack by quantum computers. https://www.nist.gov/news-events/news/2023/08/nist-standardize-encryption-algorithms-can-resist-attack-quantum-computers (2023)
[50] Pan, J., Wagner, B.: Lattice-based signatures with tight adaptive corruptions and more. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography - PKC 2022, Virtual Event, March 8-11, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13178, pp. 347–378. Springer (2022), https://doi.org/10.1007/978-3-030-97131-1_12
[51] Peikert, C., Waters, B.: Lossy trapdoor functions and their applications. In: Dwork, C. (ed.) Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008. pp. 187–196. ACM (2008), https://doi.org/10.1145/1374376.1374406
[52] van Pelt, R., Jansen, S., Baars, D., Overbeek, S.: Defining blockchain governance: A framework for analysis and comparison. Inf. Syst. Manag. 38(1), 21–41 (2021), https://doi.org/10.1080/10580530.2020.1720046
[53] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: Johnson, D.S. (ed.) Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA. pp. 73–85. ACM (1989), https://doi.org/10.1145/73007.73014
[54] Regev, O.: An efficient quantum factoring algorithm. CoRR abs/2308.06572 (2023), https://doi.org/10.48550/arXiv.2308.06572
[55] Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) Advances in Cryptology - ASIACRYPT 2001, Gold Coast, Australia, December 9-13, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2248, pp. 552–565. Springer (2001), https://doi.org/10.1007/3-540-45682-1_32
[56] Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994. pp. 124–134. IEEE Computer Society (1994), https://doi.org/10.1109/SFCS.1994.365700
[57] Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. 41(2), 303–332 (1999), https://doi.org/10.1137/S0036144598347011
[58] Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) Advances in Cryptology - EUROCRYPT 2000, Bruges, Belgium, May 14-18, 2000, Proceeding. Lecture Notes in Computer Science, vol. 1807, pp. 207–220. Springer (2000), https://doi.org/10.1007/3-540-45539-6_15
[59] Stinson, D.R., Strobl, R.: Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. In: Varadharajan, V., Mu, Y. (eds.) Information Security and Privacy, 6th Australasian Conference, ACISP

2001, Sydney, Australia, July 11-13, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2119, pp. 417–434. Springer (2001), https://doi.org/10.1007/3-540-47719-5_33

[60] Sun, S., Au, M.H., Liu, J.K., Yuen, T.H.: Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) Computer Security - ESORICS 2017, Oslo, Norway, September 11-15, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10493, pp. 456–474. Springer (2017), https://doi.org/10.1007/978-3-319-66399-9_25

[61] Venugopalan, S., Homoliak, I.: Always on voting: A framework for repetitive voting on the blockchain. CoRR abs/2107.10571 (2021), https://arxiv.org/abs/2107.10571

[62] Wei, B., Du, Y., Zhang, H., Zhang, F., Tian, H., Gao, C.: Identity based threshold ring signature from lattices. In: Au, M.H., Carminati, B., Kuo, C.J. (eds.) NSS 2014, Xi'an, China, October 15-17, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8792, pp. 233–245. Springer (2014), https://doi.org/10.1007/978-3-319-11698-3_18

[63] Wong, D.S., Fung, K., Liu, J.K., Wei, V.K.: On the rs-code construction of ring signature schemes and a threshold setting of RST. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003, Huhehaote, China, October 10-13, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2836, pp. 34–46. Springer (2003), https://doi.org/10.1007/978-3-540-39927-8_4

[64] Xu, F., Lv, X.: A new identity-based threshold ring signature scheme. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Anchorage, Alaska, USA, October 9-12, 2011. pp. 2646–2651. IEEE (2011), https://doi.org/10.1109/ICSMC.2011.6083996

[65] Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982. pp. 160–164. IEEE Computer Society (1982), https://doi.org/10.1109/SFCS.1982.38

A More Preliminaries

A.1 Public key encryption

Definition 12 (Public-Key Encryption). *A public key encryption (PKE) scheme consists of the following three algorithms. Namely,* $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$. *The key generation algorithm takes as input the security parameter* 1^λ*, and outputs an encryption public key* pk *and a decryption secret key* sk.
- $ct \leftarrow \mathsf{Enc}(pk, \mu)$. *The encryption algorithm takes as input* pk *and a message* μ*, and outputs a ciphertext* ct*. If the randomness* r *is specific, then we also denote it as* $ct \leftarrow \mathsf{Enc}(pk, \mu; r)$ *or* $ct := \mathsf{Enc}(pk, \mu; r)$.
- $\mu' \leftarrow \mathsf{Dec}(sk, ct)$. *The decryption algorithm takes as input* sk *and a ciphertext* ct*, and outputs the decryption result* μ'.

Correctness. *For every* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ *and every message* μ*, it holds that* $\mathsf{Dec}(sk, \mathsf{Enc}(pk, \mu)) = \mu$.

Definition 13 (CPA Security of PKE). *A public key encryption scheme* PKE *is secure, if for every PPT adversary* $\mathcal{A}$*, its advantage*

$$\begin{aligned}\mathsf{Adv}^{cpa}_{\mathsf{PKE},\mathcal{A}}(\lambda) := |\Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda) : \mathcal{A}^{\mathcal{O}_0(pk,\cdot,\cdot)} = 1] \\ - \Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda) : \mathcal{A}^{\mathcal{O}_1(pk,\cdot,\cdot)} = 1]|\end{aligned}$$

is negligible in λ*, where oracle* $\mathcal{O}_b(pk, \mu_0, \mu_1)$ *returns* $ct \leftarrow \mathsf{Enc}(pk, \mu_b)$.

Definition 14 (Min-Entropy of PKE). *A PKE scheme* PKE *has* κ*-min-entropy, if for every* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*, every* μ*, it holds that* $\mathbf{H}_\infty(\mathsf{Enc}(pk, \mu; r)) \geq \kappa$*, where* $r \xleftarrow{\$} \mathcal{R}$ *and* $\mathcal{R}$ *is the randomness space of* PKE.

A.2 Signatures

Definition 15 (Signatures). *A signature (SIG) scheme consists of the following three algorithms. Namely,* $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$.

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*: The key generation algorithm takes as input the security parameter* 1^λ*, and outputs a public key* pk *and a secret key* sk.
- $\sigma \leftarrow \mathsf{Sign}(sk, \mathsf{msg})$*: The signing algorithm takes as input* sk *and a message* msg*, and outputs a signature* σ.
- $1/0 \leftarrow \mathsf{Ver}(pk, \mathsf{msg}, \sigma)$*: The verification algorithm* Ver *takes as input* pk*, a message* msg *and a signature* σ*, and outputs a bit indicating the validity of* σ.

Correctness. *For any* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*, any* msg *and* $\sigma \leftarrow \mathsf{Sign}(sk, \mathsf{msg})$*, it holds that* $\mathsf{Ver}(pk, \mathsf{msg}, \sigma) = 1$.

Definition 16 (Unforgeability of Signatures). *Let* Sig *be a Sig scheme. Consider the following unforgeability experiment* $\mathsf{Exp}^{unforg}_{\mathsf{Sig},\mathcal{A}}(\lambda)$ *between the challenger* $\mathcal{C}$ *and the adversary* $\mathcal{A}$*.*

1. $\mathcal{C}$ *generates* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ *and passes* pk *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *has access to the signing oracle* $\mathcal{O}(\mathsf{msg})$ *that returns* $\sigma \leftarrow \mathsf{Sign}(sk, \mathsf{msg})$*.*
3. *Finally* $\mathcal{A}$ *outputs a forgery* $(\mathsf{msg}^*, \sigma^*)$*.*

$\mathsf{Exp}^{unforg}_{\mathsf{Sig},\mathcal{A}}(\lambda)$ *outputs 1 if* $\mathsf{Ver}(vk, \mathsf{msg}^*, \sigma^*) = 1$ *and* $\mathcal{A}$ *never asks* $\mathcal{O}(\mathsf{msg}^*)$*.*

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

A.3 Forking Lemma

In this subsection, we review the forking lemma proposed by Bellare and Never [6], which is an important tool for the security proof of the CH-based construction.

Lemma 1 (Forking Lemma [6]). *Fix an integer* Q *and a set* $\mathcal{M}$*. Let* $\mathcal{B}$ *be a randomized algorithm that on input* $x, u^{(1)}, ..., u^{(Q)}$ *returns a pair* (j, σ)*, where the first element is an integer in* $[Q]$ *and the second element is referred as a side output. Let* $\mathcal{X}$ *be a distribution. We define acc, the accepting probability of* $\mathcal{B}$ *in the experiment as follows.*

$$adv := \Pr\left[\begin{matrix} x \xleftarrow{\$} \mathcal{X}, u^{(1)}, ..., u^{(Q)} \xleftarrow{\$} \mathcal{M} \\ (j, \sigma) \leftarrow \mathcal{B}(x, u^{(1)}, ..., u^{(Q)}) \end{matrix} : 1 \le j \le Q\right].$$

The forking algorithm $\mathcal{F}_\mathcal{B}$ *associated with* $\mathcal{B}$ *is a randomized algorithm that takes input* x *and proceeds as follows.*

Set random coins ρ *for* $\mathcal{B}$
$u^{(1)}, ..., u^{(Q)} \xleftarrow{\$} \mathcal{M}$
$(j, \sigma) \leftarrow \mathcal{B}(x, u^{(1)}, ..., u^{(Q)}; \rho)$
If $j = 0$ *then return* $(0, \perp, \perp)$
$u^{(j)'}, ..., u^{(Q)'} \xleftarrow{\$} \mathcal{M}$
$(j', \sigma') \leftarrow \mathcal{B}(x, u^{(1)}, ..., u^{(j-1)}, u^{(j)'}, ..., u^{(Q)'}; \rho)$
If $(j = j' \wedge u^{(j)} \neq u^{(j)'})$ *then return* (j, σ, σ')
Otherwise return $(0, \perp, \perp)$

Define

$$frk := \Pr[x \leftarrow \mathcal{X}, (b, \cdot, \cdot) \leftarrow \mathcal{F}_\mathcal{B}(x) : b \neq 0].$$

Then

$$frk \ge acc \cdot (acc/Q - 1/|\mathcal{M}|).$$

B More Security Notions of ♯AMS

Definition 17 (Weak Unforgeability). *Let ♯AMS be an ♯AMS scheme. Consider the weak unforgeability experiment* $\mathsf{Exp}^{w\text{-}unforg}_{\sharp AMS,\mathcal{A}}(\lambda)$*, which is defined as* $\mathsf{Exp}^{unforg}_{\sharp AMS,\mathcal{A}}(\lambda)$ *in Definition 7, except that condition* (2) *is replaced with*

(2") $\mathcal{A}$ *never asks* $\mathcal{O}(\mathsf{msg}^*, P, G)$ *such that* $|G| \geq t^*$.

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

Remark 5 (On the meaning of weak unforgeability). Recall that the output t of the verification algorithm is a metric to measure how "reliable" a signature is, i.e., how many different signers agree on the message and participate in the signing process. Therefore, in many applications (e.g., the blockchain governance discussed in Section 6), the adversary's goal is to forge a t-valid signature with t as high as possible. To formalize this kind of security, we define the above-mentioned weak unforgeability by strengthening the restriction of the adversary.

Adaptive corruption endows a powerful attack capability for the adversary, which might be too strong to realize. The static corruption model is enough for many applications such as applications: V2 and V3. We formally define the (strong/weak) unforgeability under static corruptions.

Definition 18 ((Strong/Weak) Unforgeability under Static Corruptions). *Let ♯AMS be an ♯AMS scheme. Consider the following unforgeability experiment* $\mathsf{Exp}^{unforg\text{-}sta\text{-}corr}_{\sharp AMS,\mathcal{A}}(\lambda)$ *between the challenger* $\mathcal{C}$ *and the adversary* $\mathcal{A}$.

1. $\mathcal{A}$ *sets the maximum number of signers* n *and claims a group of signers* $G' \subseteq [n]$ *to be corrupted.*
2. $\mathcal{C}$ *generates* $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$ *and passes* $(vk, \{sk_i\}_{i\in G'})$ *to* $\mathcal{A}$.
3. $\mathcal{A}$ *has access to the signing oracle* $\mathcal{O}(\cdot,\cdot,\cdot)$*, which inputs* (msg, P, G) *and returns* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, G, \{sk_i\}_{i\in G})$ *and adds* (msg, σ) *into the set* $\mathcal{S}$.
4. *Finally* $\mathcal{A}$ *outputs* $(\mathsf{msg}^*, \sigma^*)$.

Let $t^* \leftarrow \mathsf{Ver}(vk, \mathsf{msg}^*, \sigma^*)$. $\mathsf{Exp}^{unforg\text{-}sta\text{-}corr}_{\sharp AMS,\mathcal{A}}(\lambda)$ *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 $\mathsf{Adv}^{unforg\text{-}sta\text{-}corr}_{\sharp AMS,\mathcal{A}}(\lambda)$ *the probability that* $\mathsf{Exp}^{unforg\text{-}sta\text{-}corr}_{\sharp AMS,\mathcal{A}}(\lambda)$ *outputs 1. We say that ♯AMS is unforgeable under static corruptions, if for all PPT adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{unforg\text{-}sta\text{-}corr}_{\sharp AMS,\mathcal{A}}(\lambda)$ *is negligible in* λ.

If condition (2) *is replaced with*

(2') $(\mathsf{msg}^*, \sigma^*) \notin \mathcal{S}$.

then we define the strong unforgeability of $\sharp\mathsf{AMS}$*, the corresponding experiment and advantage are denoted by* $\mathsf{Adv}^{s\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *and* $\mathsf{Adv}^{s\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$.

If condition (2) *is replaced with*

(2") $\mathcal{A}$ *never asks* $\mathcal{O}(\mathsf{msg}^*, P, G)$ *such that* $|G| \geq t^*$.

then we define the weak unforgeability of $\sharp\mathsf{AMS}$*, the corresponding experiment and advantage are denoted by* $\mathsf{Adv}^{w\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *and* $\mathsf{Adv}^{w\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$.

C A Framework for Lossy Chameleon Hashes

In this section we show a framework of lossy chameleon hashes defined in Definition 3, including the equivalence with lossy identification schemes, some generic constructions, and some concrete constructions from the DDH assumption and the LWE & SIS assumptions.

C.1 Equivalence with Lossy Identification Schemes

Bellare and Ristov [7, 8] proved that chameleon hashes and 3-move Sigma protocols are equivalent, where the hash value, the message, and the randomness of a chameleon hash corresponds to the commitment, the challenge, and the response of a Sigma protocol, respectively. In this subsection, we extend the equivalence to the lossy mode, by showing that lossy chameleon hashes and lossy identification schemes [1] are equivalent.

We first recall the definition and security properties of lossy identification schemes. By default, the lossy identification scheme we consider here is a 3-move protocol (as that in sigma protocols) which consists of a commitment, a challenge, and a response as its transcript.

Definition 19 (Lossy Identification [1]). *A lossy identification scheme consists of the following four algorithms. Namely,* $\mathsf{LID} = (\mathsf{Gen}, \mathsf{LGen}, \mathsf{Prove}, \mathsf{Ver})$.

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$. *The normal key generation algorithm takes as input the security parameter* λ *and outputs a verifier's public key* pk *and a prover's secret key* sk.
- $pk \leftarrow \mathsf{LGen}(1^\lambda)$. *The lossy key generation algorithm takes as input the security parameter* λ *and outputs a lossy public key* pk.
- Prove. *The prover algorithm that as input the current conversation transcripot and outputs the next message to be sent to the verifier.*
- Ver. *The (deterministic) verification algorithm* Ver *takes the transcript as input and output a bit, where 1 indicates acceptence and 0 otherwise.*

We assume by default, the commitment space, the challenge space $\mathcal{CH}$*, and the response space* $\mathcal{RP}$ *are defined by the public key* pk *(either generated via normal or lossy way). Moreover, for every* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*, there is a transcript*

oracle that works as follows.

$$\begin{array}{l}\mathcal{O}^{trans}_{pk,sk}():\\ (cmt,st)\leftarrow \mathsf{Prove}(sk)\\ ch\xleftarrow{\$}\mathcal{CH}\\ resp\leftarrow \mathsf{Prove}(sk,st,ch)\\ output\ (cmt,ch,resp)\end{array}$$

Definition 20. *Let LID be an identification scheme. We say LID is secure if it satisfies the follow properties.*

ρ-completeness for normal keys. *For every $(pk,sk)\leftarrow \mathsf{Gen}(1^\lambda)$, $\mathsf{Ver}(pk,cmt,ch,resp)=1$ holds with probability at least ρ, where $(cmt,ch,resp)\leftarrow \mathcal{O}^{trans}_{pk,sk}()$.*

Simulatability of transcripts (honest-verifier zero-knowledge). *For every $(pk,sk)\leftarrow \mathsf{Gen}(1^\lambda)$, there exists a PPT simulator Sim that takes only pk as input and outputs a simulated transcript $(cmt,ch,resp)\leftarrow \mathsf{Sim}^{trans}_{pk}()$, which distributes η-close to the output of $\mathcal{O}^{trans}_{pk,sk}()$.*

κ-min-entropy. *For any $(pk,sk)\leftarrow \mathsf{Gen}(1^\lambda)$, let $(cmt,st)\leftarrow \mathsf{Prove}(sk)$ and $[\cdot]_1$ be a function that maps (cmt,st) to cmt, then*

$$\mathbf{H}_\infty([\mathsf{Prove}(sk)]_1)\geq\kappa,$$

where the probability is taken over the randomness coin used in Prove.

Indistinguishability. *For any PPT adversary $\mathcal{A}$, the distinguish advantage*

$$\mathsf{Adv}^{ind}_{\mathsf{LID},\mathcal{A}}(\lambda):=|\Pr[(pk,sk)\leftarrow\mathsf{Gen}(\lambda):\mathcal{A}(pk)=1]-\Pr[pk\leftarrow\mathsf{LGen}:\mathcal{A}(pk)=1]|$$

is negligible in λ.

ϵ-lossiness. *For any (even all-powerful) adversary $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)$, it holds that*

$$\Pr\left[\begin{array}{c}pk\leftarrow\mathsf{LGen}(1^\lambda);(cmt,st)\leftarrow\mathcal{A}_1^{\mathsf{Sim}^{trans}_{pk}()}(pk);\\ ch\xleftarrow{\$}\mathcal{CH};r\leftarrow\mathcal{A}_2(ch,st)\end{array}:\mathsf{Ver}(pk,cmt,ch,resp)=1\right]\leq\epsilon,$$

where $\mathsf{Sim}^{trans}_{pk}()$ is the simulator described above.

Besides, we also need the following two properties for identification schemes when constructing lossy chameleon hashes from them.

Definition 21 (Commitment recoverability of LID). *LCH is commitment recoverable if there exists a determistic algorithm Com that takes as inputs $(pk,ch,resp)$ and outputs a recovered commitment cmt', and $\mathsf{Ver}(pk,cmt,ch,resp)=1$ if and only if $cmt=\mathsf{Com}(pk,ch,resp)$.*

Moreover, there exists a special simulator Sim, which outputs a η-close simulated transcrip by randomly sampling $(ch,resp)$ and then outputting $(\mathsf{Com}(pk,ch,resp),ch,resp)$, and $(ch,resp)$ can be serverd as the inner state st for $cmt=\mathsf{Com}(pk,ch,resp)$ in the Prove algorithm.

Definition 22 (Strong special soundness of LID [8]). *LCH has strong special soundness, if for any PPT adversary, its advantage*

$$\mathsf{Adv}^{sss}_{\mathsf{LID},\mathcal{A}}(\lambda) := \Pr\left[\begin{array}{c|c} & (ch, resp) \neq (ch', resp') \\ (pk, sk) \leftarrow \mathsf{Gen}(1^\lambda) & \wedge\ \mathsf{Ver}(pk, cmt, ch, resp) = 1 \\ (cmt, ch, resp, ch', resp') \leftarrow \mathcal{A}(pk) & \wedge\ \mathsf{Ver}(pk, cmt, ch', resp') = 1 \end{array}\right]$$

is negligible in λ.

From LCH to LID. Let $\mathsf{LCH} = (\mathsf{LCH.Gen}, \mathsf{LCH.LGen}, \mathsf{Hash}, \mathsf{TdColl})$ be a secure lossy identification scheme. We construct a lossy identification scheme LID from LCH as follows.

$\underline{\mathsf{Gen}(1^\lambda)}$:
$(hk, td) \leftarrow \mathsf{LCH.Gen}(1^\lambda)$
Output $(pk, sk) := (hk, td)$

$\underline{\mathsf{LGen}(1^\lambda)}$:
$hk \leftarrow \mathsf{LCH.LGen}(1^\lambda)$
Output $pk := hk$

$\underline{\text{Prove and Ver:}}$

Prover (hk, td)		Verifier hk
$\bar{m} \xleftarrow{\$} \mathcal{M}$; $\bar{r} \xleftarrow{\$} \mathcal{R}$		
$h := \mathsf{Hash}(hk, \bar{m}, \bar{r})$	$\xrightarrow{cmt:=h}$	
	$\xleftarrow{ch:=m}$	$m \xleftarrow{\$} \mathcal{M}$
$r \leftarrow \mathsf{TdColl}(td, \bar{m}, \bar{r}, m)$	$\xrightarrow{resp:=r}$	If $h = \mathsf{Hash}(hk, m, r)$: output 1
		Otherwise: output 0

Fig. 4. Construction of lossy identification schemes from lossy chameleon hashes.

Theorem 4. *Let LCH be a strongly secure lossy chameleon hash scheme (i.e., it has κ-uniformity, γ-random trapdoor collision, strong collision, indistinguishability anmd ϵ-lossiness), then LID constructed in Fig. 4 is a secure lossy identification scheme with commitment recoverability.*

Proof. We show that ID has completeness, simulatability of transcripts, min-entropy, indistinguishability, lossiness, and commitment recoverability.

1-completeness. This is directly implied by the correctness of $\mathsf{LCH.TdColl}$.

γ-simulatability of transcripts. We construct the PPT simulator Sim as follows.

$\mathcal{O}^{trans}_{hk,td}()$:	$\mathsf{Sim}_{pk}()$:
$\bar{m}, m \xleftarrow{\$} \mathcal{M}; \bar{r} \xleftarrow{\$} \mathcal{R}$	$m \xleftarrow{\$} \mathcal{M}; r \xleftarrow{\$} \mathcal{R}$
$h := \mathsf{Hash}(hk, m, r)$	$h := \mathsf{Hash}(hk, m, r)$
$r \leftarrow \mathsf{TdColl}(td, \bar{m}, \bar{r}, m)$	output (h, m, r)
output (h, m, r)	

Since h is totally determined by pk, m, and r, the only difference between the above two distribution is the generation of r. Due to the γ-random trapdoor collision property of LCH, we know that $\mathcal{O}^{trans}_{hk,td}()$ and $\mathsf{Sim}_{pk}()$ have a statistical distance γ.

κ-min-entropy. Since the commitment cmt for Prove algorithm is just the hash value for random $\bar{m}$ and $\bar{r}$, the κ-min-entropy directly follows from the κ-uniformity of LCH.

Indistinguishability. This is directly implied by the indistinguishability of LCH.

ϵ-lossiness. This is directly implied by the ϵ-lossiness of LCH.

Commitment recoverability. This is straightforward since the commitment is the hash value of $m = ch$ and $r = resp$.

From LID to LCH. Let $\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LGen}, \mathsf{Prove}, \mathsf{Ver})$ be a secure lossy identification scheme with commitment recoverability. We construct a lossy chameleon hash scheme LCH from LID as follows.

$\underline{\mathsf{Gen}(1^\lambda)}$: $(pk, sk) \leftarrow \mathsf{LID.Gen}(1^\lambda)$ Output $(hk, td) := (pk, sk)$ $\underline{\mathsf{LGen}(1^\lambda)}$: $pk \leftarrow \mathsf{LID.Gen}(1^\lambda)$ Output $hk := pk$	$\underline{\mathsf{Hash}(pk, ch, resp)}$: $cmt := \mathsf{Com}(pk, ch, resp)$ // Com is the determistic algorithm defined in // commitment recoverability Output $h := cmt$ $\underline{\mathsf{TdColl}(sk, ch, resp, ch')}$: $cmt := \mathsf{Com}(pk, ch, resp)$ $st := (ch, resp)$ $resp' \leftarrow \mathsf{Prove}(sk, st, ch')$

Fig. 5. Construction of lossy chameleon hashes schemes from lossy identification schemes.

Theorem 5. *Let LID be a secure lossy identification scheme (i.e., it has ρ-completeness, η-imulatability, κ-min-entropy, indistinguishability, ϵ-lossiness) with commitment recoverability and strong special soundness, then LCH constructed in Fig. 5 is a secure lossy chameleon hash scheme.*

Proof. We show that LCH constructed in Figure 5 has uniformity, random trapdoor collision, strong collision resistance, indistinguishability, and lossiness.

Correctness of trapdoor collision. This is guaranteed by the η-simulatability and ρ-completeness of LID. We notice that the correctness of LCH might be imperfect.

$(\kappa + \log \eta)$-uniformity. Due to the commitment recoverability and simulatability of LID, the simulated distribution $(cmt = \mathsf{Com}(pk, ch, resp), ch, resp)$ (for random ch and $resp$) is η-close to the normal protocol transcript distribution $(cmt', ch', resp')$. Moreover, we know cmt' has a min-entropy of κ. Therefore, $cmt := \mathsf{Com}(pk, ch, resp)$ has a min-entropy at least $(\kappa + \log \eta)$.

η-random trapdoor collision. According to the η-simulatability of LID, the simulated transcript $(cmt, ch, resp)$ (for random ch and $resp$) has a statistical

distance η with the real transcript, which means that the distributions of $resp$ and $resp'$ are η-close. And the η-random trapdoor collision holds as a result.

Strong collision resistance. This is implied by the commitment recoverability and the strong special soundness of LID. Recall that $\mathsf{Ver}(pk, cmt, ch, resp)$ returns 1 if and only if $cmt = \mathsf{Com}(pk, ch, resp)$, where Com is the determistic algorithm defined in Definition 21. Therefore, a tuple $(cmt, ch, resp, ch', resp)$ that breaks the strong special soundness directly leads to a collision of the constructed LCH.

Indistinguishability. This is directly implied by the indistinguishability of LID.

ϵ-lossiness. This is directly implied by the ϵ-lossiness of LID.

C.2 Construction from Re-Randomizable Encryption

We first recall the definition of re-randomizable encryption.

Definition 23. *A re-randomizable encryption (RPKE) scheme consists of four algorithms $\mathsf{R\text{-}PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$, where the first three algorithms are defined as normal PKE (Definition 12), and*

- *the re-randomize algorithm ReRand that takes as input the public key pk, a ciphertext ct and a randomness r', and outputs a rerandomized ciphertext $ct' \leftarrow \mathsf{ReRand}(pk, ct; r)$.*

We require that for every $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$, every message μ and $\bar{r}$, the following two distributions are identical,

$$\{r \xleftarrow{\$} \mathcal{R} : \mathsf{Enc}(pk, \mu; r)\} \text{ and } \{r \xleftarrow{\$} \mathcal{R} : \mathsf{ReRand}(pk, \mathsf{Enc}(pk, \mu; \bar{r}); r)\},$$

where $\mathcal{R}$ is the randomness space of $\mathsf{R\text{-}PKE}$.

Besides the rerandomize algorithm ReRand, we additionally require an efficient collision algorithm to find our a randomness with which ReRand will rerandomize a ciphertext to a specific one.

Definition 24 (Efficient collision of R-PKE). *Let $\mathsf{R\text{-}PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ be an R-PKE scheme. We say $\mathsf{R\text{-}PKE}$ has efficient collision, if there exists an efficient collision algorithm Coll that, for any m, $r_0, r_1, r_0' \xleftarrow{\$} \mathcal{R}$, $\mathsf{Coll}(pk, m, r_0, r_1, r_0')$ outputs r_1' such that*

$$\mathsf{ReRand}(pk, \mathsf{Enc}(pk, m; r_0); r_0') = \mathsf{ReRand}(pk, \mathsf{Enc}(pk, m; r_1); r_1').$$

Meanwhile, if r_0' is sampled according to some distribution $\mathcal{R}$, then $r_1' \leftarrow \mathsf{Coll}(pk, m, r_0, r_1, r_0')$ also satisfies the distribution $\mathcal{R}$.

Some examples of R-PKE include ElGamal, Paillier, Regev, etc.

Now we describe our construction from R-PKE. The idea mainly follows the construction in [36], but we additionally require an efficient collision algorithm so that the constructed LCH has correctness (i.e., there exists a trapdoor to find collisions efficiently in the collision mode).

Construction of LCH from R-PKE. Let $\mathsf{R\text{-}PKE}$ be a secure R-PKE scheme with efficient collision. Let $\mathcal{R}$ be the randomness space of $\mathsf{R\text{-}PKE}$. The construction of LCH is shown as follows, where the message space is $\mathcal{M} = \{0,1\}^\ell$, and the randomness space is $\mathcal{R}^\ell$.

- $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$. $(pk, sk) \leftarrow \mathsf{R\text{-}PKE.Gen}(1^\lambda)$.
  For $i \in [\ell]$: $\bar{r}_{i,0}, \bar{r}_{i,1} \xleftarrow{\$} \mathcal{R}$, $c_{i,0} := \mathsf{Enc}(pk, 0; \bar{r}_{i,0})$, $c_{i,1} := \mathsf{Enc}(pk, 0; \bar{r}_{i,1})$.
  Return $hk := (pk, c_{1,0}, c_{1,1}, ..., c_{\ell,0}, c_{\ell,1})$ and $td := (\bar{r}_{1,0}, \bar{r}_{1,1}, ..., \bar{r}_{\ell,0}, \bar{r}_{\ell,1})$.
- $hk \leftarrow \mathsf{LGen}(1^\lambda)$. $(pk, sk) \leftarrow \mathsf{R\text{-}PKE.Gen}(1^\lambda)$.
  For $i \in [\ell]$: $\bar{r}_{i,0}, \bar{r}_{i,1} \xleftarrow{\$} \mathcal{R}$, $c_{i,0} := \mathsf{Enc}(pk, 0; \bar{r}_{i,0})$, $c_{i,1} := \mathsf{Enc}(pk, 1; \bar{r}_{i,1})$.
  Return $hk := (pk, c_{1,0}, c_{1,1}, ..., c_{\ell,0}, c_{\ell,1})$.
- $h \leftarrow \mathsf{Hash}(hk, m, r)$. Let $m = (m_1, ..., m_\ell) \in \{0,1\}^\ell$ and $r = (r_1, ..., r_\ell) \in \mathcal{R}^\ell$.
  For $i \in [\ell]$: $h_i := \mathsf{ReRand}(pk, c_{i,m_i}; r_i)$.
  Return $h := (h_1, ..., h_\ell)$.
- $r' \leftarrow \mathsf{TdColl}(td, m, r, m')$. Let $m = (m_1, ..., m_\ell) \in \{0,1\}^\ell$, $m' = (m'_1, ..., m'_\ell) \in \{0,1\}^\ell$, and $r = (r_1, ..., r_\ell) \in \mathcal{R}^\ell$.
  For $i \in [\ell]$:
  - If $m_i = m'_i$, then $r'_i := r_i$.
  - If $m_i \neq m'_i$, then $r'_i \leftarrow \mathsf{Coll}(pk, 0, \bar{r}_{i,0}, \bar{r}_{i,1}, r_i)$.

  Return $r' := (r'_1, ..., r'_\ell)$.

Theorem 6. *If $\mathsf{R\text{-}PKE}$ is a secure R-PKE scheme with efficient collision and κ-min-entropy, then LCH constructed above is a secure lossy chameleon hash scheme.*

Proof. We prove the properties of LCH as follows.

Correctness and (perfect) random trapdoor collision. These two properties follow directly from the efficient collision of $\mathsf{R\text{-}PKE}$.

$(\ell\kappa)$-uniformity. For every $i \in [\ell]$, any $\bar{r}_i$, $\mathsf{ReRand}(pk, \mathsf{Enc}(pk, 0; \bar{r}_i); r_i)$ has a min-entropy κ if r_i is uniformly distributed. And $(\ell\kappa)$-uniformity holds as a result.

Indistinguishability. This follows from the CPA security of $\mathsf{R\text{-}PKE}$. Namely, there exists a PPT algorithm $\mathcal{B}$ such that

$$\mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{cpa}_{\mathsf{R\text{-}PKE},\mathcal{B}}(\lambda).$$

Collision resistance. We first change the generation of hk from $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$ to $hk \leftarrow \mathsf{LGen}(1^\lambda)$, due to the IND property.

If the adversary $\mathcal{A}$ find a collision (m, r, m', r') under this lossy key hk, then there exists at least one i s.t. $m_i \neq m'_i$ (w.l.o.g. we assume $m_i = 0$), and

$$\mathsf{ReRand}(pk, \mathsf{Enc}(pk, 0; \bar{r}_{i,0}); r_i) = h_i = \mathsf{ReRand}(pk, \mathsf{Enc}(pk, 1; \bar{r}_{i,1}); r'_i).$$

Due to the correctness of R-PKE, h_i can be decrypted to exactly either 0 or 1. Therefore collision resistance holds.

$2^{-\ell}$-lossiness. The analysis is the same as that in collision resistance. For any fixed $h = (h_1, ..., h_\ell)$, the message $\tilde{m}$ encrypted in h is uniquely determined, due to the correctness of R-PKE. Therefore, for a random m, as long as $m \neq \tilde{m}$ (which happens with probability $(1 - 2^{-\ell})$), there does not exist randomness r such that $\mathsf{Dec}(sk, \mathsf{ReRand}(pk, c_{i,m_i}; r_i)) = \tilde{m}_i$ for all i.

C.3 Construction from Lossy PKE with Efficient Opening

We first recall the lossy encryption (L-PKE) as follows.

Definition 25. *A lossy encryption (L-PKE) scheme consists of four algorithms $\mathsf{L\text{-}PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{LGen})$, where the first three algorithms are defined as normal PKE (Definition 12), and*

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

We require that for every $pk \leftarrow \mathsf{LGen}(1^\lambda)$, every μ_0, μ_i, the following two distribution are γ-close,

$$\{r \leftarrow \mathcal{R} : \mathsf{Enc}(pk, \mu_0, r)\} \text{ and } \{r \leftarrow \mathcal{R} : \mathsf{Enc}(pk, \mu_1, r)\},$$

where $\mathcal{R}$ is the randomness space of $\mathsf{L\text{-}PKE}$.

Definition 26 (Efficient Openability of L-PKE). *An L-PKE scheme $\mathsf{L\text{-}PKE}$ has efficient openability, if for every $pk \leftarrow \mathsf{LGen}(1^\lambda)$, there exists a trapdoor td generated along with pk in LGen and an efficient algorithm Open, such that given any μ, μ', r, $\mathsf{Open}(td, \mu, r, \mu')$, with overwhelming probability it outputs a randomness r' such that $\mathsf{Enc}(pk, \mu, r) = \mathsf{Enc}(pk, \mu', r')$. Namely, a lossy ciphertext can be opened to any plaintext μ'.*

Moreover, we require that the output $r \leftarrow \mathsf{Open}(td, \mu, r, \mu')$ satisfies the same distribution as r except a negligible probability.

Now we construct a lossy chameleon hash scheme from any lossy PKE scheme.

Construction of LCH from L-PKE. Let L-PKE be a secure L-PKE scheme with efficient openability, and $\mathcal{M}$ and $\mathcal{R}$ be the message space and randomness space of R-PKE, respectively. At a high lever, the injective/lossy mode in L-PKE corresponds to the lossy/collision mode in LCH.

- $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$. $pk \leftarrow \mathsf{L\text{-}PKE.LGen}(1^\lambda)$. Let td be the trapdoor used for efficient openability (Definition 26).
  Return $hk := pk$ and td.

– $hk \leftarrow \mathsf{LGen}(1^\lambda)$. $(pk, sk) \leftarrow \mathsf{L\text{-}PKE.Gen}(1^\lambda)$.
Return $hk := pk$.
– $h \leftarrow \mathsf{Hash}(hk, m, r)$. Let $hk = pk$. $ct \leftarrow \mathsf{Enc}(pk, m; r)$.
Return $h := ct$.
– $r' \leftarrow \mathsf{TdColl}(td, m, r, m')$. Let $ct \leftarrow \mathsf{Enc}(pk, m; r)$. Return $r' \leftarrow \mathsf{Open}(td, m, r, m')$.

Theorem 7. *If L-PKE is a secure L-PKE scheme with efficient openability and κ-min-entropy, then LCH constructed above is a secure lossy chameleon hash scheme.*

Proof. We prove the properties of LCH as follows.

Correctness. Recall that the collision hash key is actually a lossy public key of L-PKE. Due to the efficient openability of L-PKE, the correctness of LCH holds with overwhelming probability.

κ-uniformity. This is directly implied by the κ-min-entropy of L-PKE.

Random trapdoor collision. This follows from the efficient openability of L-PKE.

Indistinguishability. This follows directly from the indistinguishability of L-PKE.

Collision resistance. To prove the collision resistance of LCH, we first change the generation of hk from $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$ to $hk \leftarrow \mathsf{LGen}(1^\lambda)$, due to the indistinguishability property.
If the adversary $\mathcal{A}$ find a collision (m, r, m', r') under this lossy key hk (i.e., under the injective public key pk), then $m \neq m'$ and

$$\mathsf{Enc}(pk, m; r) = \mathsf{Enc}(pk, m'; r').$$

Obviously this cannot happen due to the correctness of L-PKE.

$1/|\mathcal{M}|$-lossiness. In the lossy mode of LCH (i.e., the injective mode of L-PKE), for any fixed h, the message $\tilde{m}$ encrypted in h is uniquely determined due to the correctness of L-PKE. Therefore, for a random m, as long as $m \neq \tilde{m}$ (which happens with probability $(1 - 1/|\mathcal{M}|)$), there does not exist randomness r such that $\mathsf{Enc}(pk, m; r) = h$.

C.4 Construction from LWE

In this subsection we show the LWE-based construction of LWE, which was presented as a dual-mode commitment scheme by Pan and Wagner [50]. Here we provide the proof of collision resistance based on the SIS assumption.

Let n, m, q be positive integers. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times m} (m > n)$ and vector $\mathbf{u} \in \mathbb{Z}_q^n$, define the n-dimensional lattice $\Lambda(\mathbf{A}) := \{\mathbf{y} \in \mathbb{R}^n | \mathbf{y} = \mathbf{A}\mathbf{x}, \mathbf{x} \in \mathbb{Z}^m\}$, the orthogonal lattice $\Lambda_q^\perp(\mathbf{A}) := \{\mathbf{x} \in \mathbb{Z}^m | \mathbf{A}\mathbf{x} = 0^n \mod q\}$ and shifted lattice $\Lambda_\mathbf{u}^\perp(\mathbf{A}) := \{\mathbf{x} \in \mathbb{Z}^m | \mathbf{A}\mathbf{x} = \mathbf{u} \mod q\}$.

The Gaussian function with parameter s and center $\mathbf{c}$ is defined as $\rho_{s,\mathbf{c}} : \mathbb{R}^n \to \mathbb{R}$, $\rho_{s,\mathbf{c}}(\mathbf{x}) := \exp(-\pi||\mathbf{x} - \mathbf{c}||^2/s^2)$. For countable set $\mathcal{S} \in \mathbb{R}^n$, the discrete Gaussian distribution $D_{\mathcal{S},s,\mathbf{c}}(\mathbf{x})$ parameterized with s and $\mathbf{c}$ is defined as

$D_{\mathcal{S},s,\mathbf{c}}(\mathbf{x}) := \rho_{s,\mathbf{c}}(\mathbf{x})/\sum_{\mathbf{x}\in\mathcal{S}}\rho_{s,\mathbf{c}}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{S}$ and $D_{\mathcal{S},s,\mathbf{c}}(\mathbf{x}) := 0$ for $\mathbf{x} \notin \mathcal{S}$. Usually we omit $\mathbf{c}$ if $\mathbf{c} = \mathbf{0}$.

We recall the following lemmas from [50], which are originally presented in [2, 47, 30, 31].

Lemma 2. *Let n, m, q be positive integers and $m \geq 2n\log q$. Consider any $\omega(\sqrt{\log m})$ function and $s \geq \omega(\sqrt{\log m})$. Then for all but negligible (in n) fraction of all $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ the distribution $\{\mathbf{Ae}|\mathbf{e} \leftarrow D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}),s}\}$ is statistically close to uniform distribution over $\mathbb{Z}_q^n$. Moreover, the conditional distribution of $\mathbf{e} \leftarrow D_{\mathbb{Z}^m,s}$ given $\mathbf{u} = \mathbf{Ae} \mod q$ is exactly $D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}),s}$.*

Lemma 3. *Let n, m, q be positive integers and $m \geq 2n\log q$. Consider any $\omega(\sqrt{\log m})$ function and $s \geq \omega(\sqrt{\log m})$. Then for all but at most q^{-n} fraction of $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and any vector $\mathbf{u} \in \mathbb{Z}_q^n$, it holds that $\Pr[||\mathbf{x}|| > s\sqrt{m}|\mathbf{x} \leftarrow D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}),s}] \leq 2^{-m+1}$.*

Let $\mathbf{G}$ be the gadget matrix defined in [46]. Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, n, m, q be positive integers and $m \geq n\lceil\log q\rceil$. A matrix $\mathbf{R} \in \mathbb{Z}^{(m-n\lceil\log q\rceil)\times n\lceil\log q\rceil}$ is a trapdoor for $\mathbf{A}$ is $\mathbf{A}[-\mathbf{R}^\top|\mathbf{I}_{n\lceil\log q\rceil}]^\top = \mathbf{G}$.

Lemma 4. *There exist PPT algorithms $\mathsf{GenTrap}$ and $\mathsf{SampleD}$ and constants $C_0 > 0$, $C_1 \leq 3$ such that for positive integers n, m, q, $q \geq 2$, $m \geq 3n\log q$, $w := n\lceil\log q\rceil$, and any $\omega(\sqrt{\log n})$ function the following holds with overwhelming probability over all random choices:*

1. *For any $s \geq \omega(\log n)$, the algorithm $\mathsf{GenTrap}(n, m, s, q)$ outputs matrices $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{R} \in \mathbb{Z}^{(m-w)\times w}$ such that $\mathbf{A}$ is statistically close to uniform matrix over $\mathbb{Z}_q^{n\times m}$, $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ with entriex sampled from $D_{\mathbb{Z},s}$ and $s_1(\mathbf{R}) \leq s \cdot C_0 \cdot (\sqrt{m-w} + \sqrt{w})$.*
2. *For any matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ with trapdoor $\mathbf{R}$, for any $\mathbf{u} \in \mathbb{Z}_q^n$ and $s \geq C_1 \cdot \sqrt{s_1(\mathbf{R})^2 + 1}\omega(\sqrt{\log n})$, the distribution $\{\mathbf{z}|\mathbf{z} \leftarrow \mathsf{SampleD}(\mathbf{A}, \mathbf{R}, \mathbf{u}, s)\}$ is statistically close to $D_{\lambda_{\mathbf{u}}^{\perp}(\mathbf{A}),s}$.*

Next we present two hardness assumptions on lattices.

Definition 27 (The LWE assumption). *Let $n = n(\lambda), m = poly(n), q$ be positive integers and $\mathbf{x}$ be an error distribution over $\mathbb{Z}$. The learning with errors (LWE) assumption states that for any PPT adversary $\mathcal{A}$, its advantage*

$$\mathsf{Adv}^{lwe}_{[n,q,\chi],\mathcal{A}}(\lambda) := |\Pr[\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}; \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n; \mathbf{e} \leftarrow \chi^m : \mathbf{A}(\mathbf{A}, \mathbf{s}^\top\mathbf{A} + \mathbf{e}^\top) = 1] - \Pr[\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}; \mathbf{b} \xleftarrow{\$} \mathbb{Z}_q^m : \mathbf{A}(\mathbf{A}, \mathbf{b}^\top) = 1]|$$

is negligible in λ.

Definition 28 (The SIS assumption). *Let $n = n(\lambda), m, q$ be positive integers and β be a positive real. The short integer solution (SIS) assumption states that for any PPT adversary $\mathcal{A}$, the advantage*

$$\mathsf{Adv}^{sis}_{[n,q,\beta,m],\mathcal{A}}(\lambda) := \Pr[\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}; \mathbf{x} \leftarrow \mathcal{A}(\mathbf{A}) : \mathbf{Ax} = 0^n \wedge \mathbf{x} \neq 0^m \wedge ||\mathbf{x}|| \leq \beta]$$

is negligible in λ.

$\underline{\mathsf{Gen}(1^\lambda)}$ $(\mathbf{A}, \mathbf{T_A}) \leftarrow TrapdoorGen()$ Return $(hk, td) := (\mathbf{A} \in \mathbb{Z}_q^{(n+\ell)\times m)}, \mathbf{T_A})$ $\underline{\mathsf{LGen}(1^\lambda)}$ $\bar{\mathbf{A}} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ $\mathbf{S} \xleftarrow{\$} \mathbb{Z}_q^{n\times \ell}$; $\mathbf{E} \leftarrow D_{\mathbb{Z},\alpha q}^{m\times \ell}$ $\underline{\mathbf{A}} := \mathbf{S}^\top \mathbf{A} + \mathbf{E}^\top$ Return $hk := \mathbf{A} := \begin{bmatrix} \bar{\mathbf{A}} \\ \underline{\mathbf{A}} \end{bmatrix} \in \mathbb{Z}_q^{(n+\ell)\times m}$	$\underline{\mathsf{Hash}(hk, \mathbf{m} \in \{0,1\}^\ell, \mathbf{r})}$ $\mathbf{z} := \mathbf{Ar} + \begin{bmatrix} \mathbf{0} \\ \lfloor q/2 \rceil \cdot \mathbf{m} \end{bmatrix}$ Return $\mathbf{z}$ $\underline{\mathsf{TdColl}(td, \mathbf{m}, \mathbf{r}, \mathbf{m}')}$ $\mathbf{z}' := \mathbf{Ar} + \begin{bmatrix} \mathbf{0} \\ \lfloor q/2 \rceil \cdot (\mathbf{m} - \mathbf{m}') \end{bmatrix}$ $\mathbf{r}' \leftarrow Resample(\mathbf{A}, \mathbf{T_A}, \mathbf{z}', s)$ If $\|\mathbf{r}'\| > s\sqrt{m}$: return $\perp$ Return $\mathbf{r}'$

Fig. 6. LCH construction from the LWE and SIS assumptions.

Theorem 8 (Security of the Lattice-based Construction [50]). *Under the LWE assumption, the construction in Fig. 6 has completeness, simulatability, min-entropy, indistinguishability, lossiness, and commitment recoverability.*

Theorem 9. *Under the LWE and SIS assumptions, LCH in Fig. 6 is a secure lossy chameleon hash scheme.*

Proof. Thanks to Theorem 5 and Theorem 8, it is sufficient to prove the collision resistance.

Recall that in the generation of hash key, $\mathbf{A}$ is statistically close to a random matrix over $\mathbb{Z}_q^{(n+\ell)\times m}$. Under a random matrix $\mathbf{A} = \begin{bmatrix} \bar{\mathbf{A}} \\ \underline{\mathbf{A}} \end{bmatrix}$, a collision $(\mathbf{m}, \mathbf{r}, \mathbf{m}', \mathbf{r}')$ such that

$$\mathbf{Ar} + \begin{bmatrix} \mathbf{0} \\ \lfloor q/2 \rceil \cdot \mathbf{m} \end{bmatrix} = \mathbf{Ar}' + \begin{bmatrix} \mathbf{0} \\ \lfloor q/2 \rceil \cdot \mathbf{m}' \end{bmatrix}$$

directly implies a SIS solution $(\mathbf{r} - \mathbf{r}')$ to $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n\times m}$.

C.5 Construction from DDH

In this subsection we present a construction of LCH from the DDH assumption, which is the well-known DDH-based lossy identification scheme by Chaum et al [18].

We first recall the background on the discrete logarithm assumption and the DDH assumption.

Let GGen be a group generation algorithm that outputs $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a cyclic group of prime order q with generator g.

Definition 29 (The DL assumption). *The discrete logarithm (DL) assumption states that, for any PPT adversary $\mathcal{A}$, its advantage*

$$\mathsf{Adv}_{\mathbb{G},\mathcal{A}}^{dl}(\lambda) := \Pr[x \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(\mathbb{G}, q, g, g^x) = x]$$

is negligible over λ.

Definition 30 (The DDH assumption). *The decisional Diffie-Hellman (DDH) assumption states that, for any PPT adversary $\mathcal{A}$, its advantage*

$$\begin{aligned}\mathsf{Adv}^{ddh}_{\mathbb{G},\mathcal{A}}(\lambda) := \Pr[x, y \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] \\ - \Pr[x, y, z \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1]\end{aligned}$$

is negligible over λ.

Note that the DL assumption is implied by the DDH assmumption.

Construction. Let $(\mathbb{G}, q, g) \leftarrow \mathbb{G}(1^\lambda)$, and $\tilde{g}$ is another random generator of $\mathbb{G}$. The DDH-based construction of LCH [18] is shown in Fig. .

$\underline{\mathsf{Gen}(1^\lambda)}$ $\alpha \leftarrow \mathbb{Z}_q$ Return $td := x$, $hk := (X, \tilde{X}) := (g^x, \tilde{g}^x)$	$\underline{\mathsf{Hash}(hk, m, r)}$ Return $h := (X^m g^r \|\| \tilde{X}^m \tilde{g}^r)$
$\underline{\mathsf{LGen}(1^\lambda)}$ $x, x' \leftarrow \mathbb{Z}_q$ s.t. $x \neq x'$ Return $hk := (X, \tilde{X}) := (g^x, \tilde{g}^{x'})$	$\underline{\mathsf{TdColl}(td, \mathbf{m}, \mathbf{r}, \mathbf{m}')}$ Return $r' := x(m - m') + r$

Fig. 7. LCH construction from the LWE and SIS assumptions.

Theorem 10 ([18, 24]). *Under the DDH assumption, LCH above is a strongly secure lossy chameleon hash scheme.*

D Proof of Theorem 1

In this section we prove Theorem 1 (the strong unforgeability of ♯AMS).

Proof. For simplicity, we will model the signing process of ♯AMS as a algorithm and ignore the moderator in the following proof.

First, we prove the strong unforgeability under static corruptions via hybrid games $\mathsf{G}_0 - \mathsf{G}_4$. Before describing the hybrid games, we specify some notations used in the proof. Let $(\mathsf{msg}^*, \sigma^* = (t', m_1^*, ..., m_n^*, r_1^*, ..., r_n^*))$ be $\mathcal{A}$'s final forgery and $t^* \leftarrow \mathsf{Ver}(vk, \mathsf{msg}^*, \sigma^*)$. For $\mathcal{A}$ to win, it must hold that $t' = t^*$. Therefore, in the following proof we implicitly assume that $t' = t^*$. Meanwhile, for $i \in [n]$, let $h_i^* \leftarrow \mathsf{Hash}(hk_i, m_i^*, r_i^*)$, and let $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^* || t^*) = u^*$.

Game G_0. This is just the original strong unforgeability experiment.

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \mathsf{Adv}^{s\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda).$$

Game G_1. If $\mathcal{A}$ never asks $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^* || t^*)$ before outputting the final forgery, then G_1 outputs 0 directly.

Since H works as a random oracle, if $\mathcal{A}$ never asks $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*)$ before, then u^* is randomly distributed over $\mathcal{U}$. Recall that for every (n, t) and $(m_1, ..., m_n)$, there exists only one u such that $\mathbf{F}(n, t, m_1, ..., m_n, u) = 1$. Therefore, the probability that $\mathcal{A}$ wins is at most $1/|\mathcal{M}|$, and we have

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq 1/|\mathcal{U}|.$$

Game G_2. In this game $\mathcal{C}$ changes the way of signing oracle's simulation as follows. Upon receiving a signing query $\mathcal{O}(\mathsf{msg}, G)$ ($|G| = t$), $\mathcal{C}$ randomly samples m_i and r_i for all $i \in [n]$, computes $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$, and reprograms the random oracle such that $H(vk, h_1, ..., h_n, \mathsf{msg}||t) = u$, where u is computed according to the forward sample algorithm $s_{fwd}(n, t, G)$ of $\mathbf{F}_\theta$. If $\mathcal{C}$ fails to reprogram, i.e., $H(vk, h_1, ..., h_n, \mathsf{msg}||t)$ has already been defined before, then G_2 outputs $\perp$ and aborts. At last $\mathcal{C}$ returns the signature $\sigma = (t, \{m_i\}_{i\in[n]}, \{r_i\}_{i\in[n]})$ to $\mathcal{A}$.

First, we argue that G_2 and G_1 are statistically close if it does not abort. Notice that the outputs $(m_1, ..., m_n, u)$ of $s_{fwd}(n, t, G)$ and $s_{back}(n, t, G)$ satisfy the identical distribution, where u outputted by $s_{back}(n, t, G)$ enjoys a random distribution. Meanwhile, the distribution of r_i ($i \in G$) is γ-close to the original signing algorithm in G_1, due to the γ-random trapdoor collision property of LCH.

Then, we bound the abort probability in G_2. Recall that LCH has κ-uniformity, i.e., for all $i \in [n]$, $\mathbf{H}_\infty(\mathsf{Hash}(hk_i, m_i, r_i)) \geq \kappa$ for randomly sampled m_i and r_i. Suppose $\mathcal{A}$ asks at most Q_{sign} signing queries and Q_H hash queries. By the union bound, we have

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq (Q_{sign} + Q_H)/2^{n\kappa} + Q_{sign} n \cdot \gamma.$$

Game G_3. We add an extra abort rule in this game. Define by reuse the event that, $\mathcal{A}$ has ever asked $\mathcal{O}(\mathsf{msg}^*, G)$ with some $|G| = t^*$ and gets $\sigma = (t^*, m_1, ..., m_n, r_1, ..., r_n)$ back, and

1. $\mathsf{Hash}(hk_i, m_i, r_i) = \mathsf{Hash}(hk_i, m_i^*, r_i^*)$ for all $i \in [n]$; and
2. There exists i such that $(m_i^*, r_i^*) \neq (m_i, r_i)$, and $\mathcal{A}$ never asks $\mathcal{O}_{corr}(i)$.

If reuse happens, then G_3 outputs $\perp$ and aborts.

We can easily construct a reduction algorithm $\mathcal{B}_1$ that breaks the strong collision resistance of the underlying uncorrupted hk_i if reuse happens. Via a standard hybrid argument, we know that

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq n \cdot \mathsf{Adv}^{s\text{-}cr}_{\mathsf{LCH},\mathcal{B}_1}(\lambda).$$

Game G_4. Let $G' \subseteq [n]$ be the corruption group by $\mathcal{A}$ and $\widetilde{G} := [n] - G'$. In this game, $\mathcal{C}$ generates $hk_i \leftarrow \mathsf{LGen}(1^\lambda)$ instead of $(hk_i, td_i) \leftarrow \mathsf{Gen}(1^\lambda)$ for all $i \in \widetilde{G}$. The simulations of $H^t(\cdot)$ and $\mathcal{O}(\mathsf{msg}, G)$ are the same as G_3. Note that $\mathcal{C}$ can simulate a $\sharp$AMS signature for group G that contains $i \in \widetilde{G}$ without knowing a trapdoor td_i.

We can easily construct a reduction algorithm $\mathcal{B}_2$ to reduce the indistinguishability between G_3 and G_4 into the indistinguishability of LCH. Therefore,

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq n \cdot \mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}_2}(\lambda).$$

Now we argue that in G_4, even an all-powerful adversary $\mathcal{A}$ cannot win with a non-negligible probability due to the ϵ-lossiness of LCH.

Let $(\mathsf{msg}^*, \sigma^* = (t', m_1^*, ..., m_n^*, r_1^*, ..., r_n^*))$ be $\mathcal{A}$'s forgery and $t^* \leftarrow \mathsf{Ver}(vk, \mathsf{msg}^*, \sigma^*)$. Let $|G'| = t'$. Obviously we have $t' < t^* \leq n$. Define by W_4 the event that G_4 outputs 1. We divide W_4 into the following three subevents.

1. Event W_4^1: $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*)$ was defined when $\mathcal{A}$ asking a query $\mathcal{O}(\mathsf{msg}^*, G)$ with $|G| = t^*$ and the response is $\sigma = (t^*, m_1, ..., m_n, r_1, ..., r_n)$, and $m_i = m_i^*$ for all $i \in [n]$.
2. Event W_4^2: $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*)$ was defined when $\mathcal{A}$ asking a query $\mathcal{O}(\mathsf{msg}^*, G)$ with $|G| = t^*$ and the response is $\sigma = (t^*, m_1, ..., m_n, r_1, ..., r_n)$, and there exists at least one i s.t. $m_i \neq m_i^*$.
3. Event W_4^3: $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*)$ was defined upon some hash query by $\mathcal{A}$.

We analyze W_4^1 first. Recall that we require $\mathcal{A}$'s final forgery to be different from all signatures obtained from the signing oracle. That is, there exists some i such that $m_i = m_i^*$, $r_i \neq r_i^*$, and $\mathsf{Hash}(hk_i, m_i, r_i) = \mathsf{Hash}(hk_i, m_i^*, r_i^*)$. Due to the uniqueness of LCH, it is impossible for $i \in G'$ (the corrupted group). Moreover, W_4^1 cannot happen for $i \in \widetilde{G}$ (the uncorrupted/lossy group) due to the extra abort rule reuse in G_3. Therefore, we have $\Pr[W_4^1] = 0$.

Then we analyze W_4^2. According to the interdependency of constraint function $\mathbf{F}_\theta$, if W_4^2 happens, then there must exist at least one $i \in \widetilde{G}$ such that $m_i \neq m_i^*$. Together with the extra abort rule reuse in G_3, we get that $\Pr[W_4^2] = 0$.

Then we bound $\Pr[W_4^3]$ according to the ϵ-lossiness of LCH. Recall that if G_5 does not abort, then all hash keys $\{hk_i\}_{i \in \widetilde{G} = ([n] - G')}$ are generated in the lossy mode. Under the hash query $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}||t^*)$, a random u^* is returned. According to the randomness property of $\mathbf{F}_\theta$, 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. From the analysis above we know, there exists at least one i s.t. hk_i is a lossy hash key. Then according to the ϵ-lossiness, given h_i, for a random m_i, the probability that $\mathcal{A}$ can find a randomness r_i with $\mathsf{Hash}(hk_i, m_i, r_i) = h_i$ is at most ϵ. By the union bound, we have

$$\Pr[\mathsf{G}_4 \Rightarrow 1] \leq (n - t^* + 1)\epsilon \leq n\epsilon,$$

which finishes the proof of strong unforgeability.

It is still left to prove the unconditional strong anonymity. Recall that in the signing, m_i for $i \notin G$ is randomly distributed, and m_i for $i \in G$ is computed from $\{m_i\}_{i \in [n] \setminus G}$ and randomly sampled u according to $\mathbf{F}_\theta$. Since the two sample algorithms s_{fwd} and s_{back} have identical distributions, m_i for $i \in G$ distributes the same as m_i for $i \notin G$. Therefore, to prove the strong anonymity, it is sufficient to prove that for any $m, m' \in \mathcal{M}$, the following two distributions are indistinguishable:

$$\left\{(m,r)\,\middle|\,r \xleftarrow{\$} \mathcal{R}\right\} \text{ and } \left\{(m,r)\,\middle|\,\begin{array}{c}(hk,td) \leftarrow \mathsf{LCH.Gen}(1^\lambda), \bar{r} \xleftarrow{\$} \mathcal{R}, \\ r \leftarrow \mathsf{LCH.TdColl}(td, \bar{m}, \bar{r}, m)\end{array}\right\}.$$

Thanks to the γ-random trapdoor collision property, the above two distributions are γ-close, and unconditional and strong anonymity holds as a result.

Taking the above all together, we obtain the desired result.

E Unforgeability under Adaptive Corruptions

In this section we analyze the (strong) unforgeability of LCH-based ♯AMS scheme in the adaptive corruption model.

Theorem 11. *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 the ♯AMS scheme ♯AMS constructed in Section 5 has strong unforgeability and strong anonymity under adaptive corruptions. More precisely, for any PPT adversary $\mathcal{A}$, we have another PPT algorithm $\mathcal{B}$ such that $Time(\mathcal{B}) \approx Time(A)$, and*

$$\begin{aligned}\mathsf{Adv}^{s\text{-}unforg\text{-}sta\text{-}corr}_{\sharp AMS,\mathcal{A}}(\lambda) \leq n2^n\,\mathsf{Adv}^{s\text{-}cr}_{LCH}(\lambda) + n2^n\,\mathsf{Adv}^{ind}_{LCH,\mathcal{B}}(\lambda) + n2^n \cdot \epsilon \\ + \frac{1}{|\mathcal{M}|} + \frac{Q_{sign}+Q_H}{2^{n\kappa}} + Q_{sign} n \cdot \gamma,\end{aligned}$$

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.

Proof. Similar to that in Appendix D, we prove the strong unforgeability via hybrid games $\mathsf{G}_0, \mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3, \widetilde{\mathsf{G}}_3, \mathsf{G}_4$. All hybrid games except $\widetilde{\mathsf{G}}_3$ are defined the same, and we safely omit the details here.

Game $\widetilde{\mathsf{G}}_3$. In this game $\mathcal{C}$ randomly samples a subgroup $\widetilde{G} \subseteq [n]$ at the beginning, and whenever $\mathcal{A}$ terminates and outputs its forgery, $\mathcal{C}$ checks whether $\mathcal{A}$ asks $\mathcal{O}_{corr}(i)$ for all $i \in ([n] - \widetilde{G})$ exactly, i.e., whether $\widetilde{G}$ is exactly the subgroup of users that $\mathcal{A}$ does not corrupt. If not, $\widetilde{\mathsf{G}}_4$ outputs $\perp$ and aborts.

There are at most 2^n different subgroups for $[n]$. Via a standard complexity argument, we have that

$$\Pr[\mathsf{G}_3 \Rightarrow 1] \leq 2^n \cdot \Pr[\widetilde{\mathsf{G}}_3 \Rightarrow 1].$$

Game G_4. In this game, $\mathcal{C}$ generates $hk_i \leftarrow \mathsf{LGen}(1^\lambda)$ instead of $(hk_i, td_i) \leftarrow \mathsf{Gen}(1^\lambda)$ for all $i \in \widetilde{G}$. And we have

$$|\Pr[\widetilde{\mathsf{G}}_3 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq n \cdot \mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}'}(\lambda).$$

Combined with hybrid games $\mathsf{G}_1, ..., \mathsf{G}_4$, we finish the proof.

Following the same proof steps in Theorem 1, we have the following theorem.

Theorem 12. *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 $\sharp\mathsf{AMS}$ constructed in Section 5 has unforgeability under adaptive corruptions. More precisely, for any PPT adversary $\mathcal{A}$, we have another PPT algorithm $\mathcal{B}$ such that $Time(\mathcal{B}) \approx Time(A)$, and*

$$\mathsf{Adv}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq n2^n \mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}}(\lambda) + n2^n \cdot \epsilon + \frac{1}{|\mathcal{M}|} + \frac{Q_{sign} + Q_H}{2^{n\kappa}} + Q_{sign} n \cdot \gamma,$$

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

F Instantiations of Constraint Functions

Assume $\mathcal{M}$ be a finite field. We show two classical instantiations of constraint functions.

Constraint Function $\mathbf{F}_\mathbf{A}$ by Linear Equation Systems.

Let $\mathcal{U} := \mathcal{M}^t$. Let $\mathbf{A} = (a_{i,j})_{(i,j\in[n])}$ be an invertible matrix such that, for every $t \in [n]$ and subset $G \subseteq [n]$ with $|G| = t$, elements $(a_{i,j})_{(i\in[t],j\in G)}$ form an invertible submatrix. For example, we can set $\mathbf{A}$ as a Vandermonde matrix.

Define $\mathbf{F}_\mathbf{A}(n, t, m_1, ..., m_n, u = (u_1, ..., u_t)) = 1$, if and only if the following linear equation system holds.

$$\begin{cases} a_{1,1}m_1 + a_{1,2}m_2 + ... + a_{1,n}m_n = u_1, \\ a_{2,1}m_1 + a_{2,2}m_2 + ... + a_{2,n}m_n = u_2, \\ ... \\ a_{t,1}m_1 + a_{n,2}m_2 + ... + a_{t,n}m_n = u_t. \end{cases} \tag{3}$$

- The forward sample algorithm $s_{fwd}(n, t, G)$ first randomly samples m_i for all $i \in [n]$, then computes $(u_1, ..., u_t)$ according to (3).
- The backward sample algorithm $s_{bck}(n, t, G)$ first randomly samples m_i for all $i \in [n] \setminus G$, and u_i for all $i \in [t]$, then computes $\{m_i\}_{i\in G}$ according to (3).

Proof. Now we prove that $\mathbf{F}_\mathbf{A}$ defined above is a family of constraint functions.

- Given $n, t, m_1, ..., m_n$ and $u = (u_1, ..., u_t)$, it is easy to evaluate $\mathbf{F}_\mathbf{A}(n, t, m_1, ..., m_n, u)$ by checking the above linear equation system.
- Given $n, t, m_1, ..., m_n$, one can compute the unique u that satisfies the above linear equation system.
- Since $\mathbf{A}$ is a full-rank matrix, the forward sample algorithm and the backward sample algorithm have the identical distribution $(m_1, ..., m_n, u)$.
- Interdependency.

- Assume $\mathbf{F}_{\mathbf{A}}(n, t, m_1, ..., m_n, u) = \mathbf{F}_{\mathbf{A}}(n, t, m'_1, ..., m'_n, u) = 1$. Define $\Delta m = (m_1 - m'_1, ..., m_n - m'_n)^\top \in \mathcal{M}^n$, and we have $\mathbf{A}\Delta m = \mathbf{0}$. Suppose Δm has less than t non-zero elements. Then we can separate a vector $\Delta\bar{m} \in \mathcal{M}^t$ and a full-rank submatrix $\bar{\mathbf{A}} \in \mathcal{M}^{t\times t}$ such that $\bar{\mathbf{A}}\Delta\bar{m} = \mathbf{A}\Delta m = \mathbf{0}$. Thus, $\Delta\bar{m} = \bar{\mathbf{A}}^{-1}\mathbf{0} = \mathbf{0}$, which means that $(m_1, ..., m_n) = (m'_1, ..., m'_n)$.
- Define $\Delta u := u - u'$ and we have $\mathbf{A}\Delta m = \Delta u$. Divide $\mathbf{A}$ into two submatrices $\bar{\mathbf{A}} \in \mathcal{M}^{t\times t}$ of full rank and $\tilde{\mathbf{A}} \in \mathcal{M}^{t\times(n-t)}$. Similarly, divide Δm into two vectors $\Delta\bar{m} \in \mathcal{M}^t$ and $\Delta\tilde{m} \in \mathcal{M}^{n-t}$. Thus we have $\bar{\mathbf{A}}\Delta\bar{m} + \tilde{\mathbf{A}}\Delta\tilde{m} = \Delta u$. Since $\bar{\mathbf{A}}$ has full rank, $\Delta\bar{m} = \bar{\mathbf{A}}^{-1}(\Delta u - \tilde{\mathbf{A}}\Delta\tilde{m})$ is randomly distributed over $\mathcal{M}^t$ if Δu is randomly sampled. Therefore, with overwhelming probability, $\Delta\bar{m}$ contains no zero-elements, which means that there are at least t different $i \in [n]$ such that $m_i \neq m'_i$.

– Randomness. Following the same argument above, we rewrite $\mathbf{F}_{\mathbf{A}}(n, t, m_1, ..., m_n) = 1$ as $\bar{\mathbf{A}}\bar{m} + \tilde{\mathbf{A}}\tilde{m} = u$, where $\bar{m} \in \mathcal{M}^t$, $\tilde{m} \in \mathcal{M}^{n-t}$, and $u \in \mathcal{M}^t$. Then $\bar{m} = \bar{\mathbf{A}}^{-1}(u - \tilde{\mathbf{A}}\tilde{m})$ is randomly distributed over $\mathcal{M}^t$ if u is randomly sampled.

Constraint Function $\mathbf{F}_p$ by Polynomial Interpolation.

Let $\mathcal{H} := \mathcal{M}$. Let $(P_0, ..., P_n) := ((0, u), (1, m_1), ..., (n, m_n))$ be $n + 1$ points of a polynomial.

Define $\mathbf{F}_p(n, t, m_1, ..., m_n, u) = 1$, if and only if $P_0, ..., P_n$ form a polynomial of degree at least $(n - t)$ by polynomial interpolation.

– The forward sample algorithm $s_{fwd}(n, t, G)$ first randomly samples a polynomial of degree $n-t$, then it computes $u := g(0)$ and $m_i := g(i)$ for $i \in [n]$.
– The backward sample algorithm $s_{bck}(n, t, G)$ first randomly samples m_i for $i \in [n] \setminus G$ and u, then forms a polynomial g from the $(n - t + 1)$ points $((0, u), \{(i, m_i)\}_{i\in[n]\setminus G})$ by polynomial interpolation. For $i \in G$, it computes $m_i := g(i)$.

Proof. Now we prove that $\mathbf{F}_p$ defined above is a family of constraint functions.

– Given $n, t, m_1, ..., m_n$ and u, we form an $(n - t)$-degree polynomial from $P_0, ..., P_{n-t}$. And $\mathbf{F}_p(n, t, m_1, ..., m_t, u) = 1$ if and only if $P_{n-t+1}, ..., P_n$ are points on the polynomial.
– Given $n, t, m_1, ..., m_n$, if there exists u satisfying $\mathbf{F}_p(n, t, m_1, ..., m_n, u) = 1$, then from $(n-t+1)$ points $P_1, ..., P_{n-t+1}$, we can construct the polynomial via interpolation and then obtain the constant coefficient u.
– The $(n - t)$-degree polynomial from the backward sample algorithm is randomly distributed as that in the forward sample algorithm.
– Interdependency.
  - Suppose $\mathbf{F}_p(n, t, m_1, ..., m_n, u) = \mathbf{F}_p(n, t, m'_1, ..., m'_n, u) = 1$. If there are at least t different i such that $m_i \neq m'_i$ (i.e., there are more than $(n - t)$ positions i such that $m_i = m'_i$), then more than $(n - t)$ points of the two polynomials g and g' are the same. Along with $P_0 = (0, u)$, we know $g = g'$, and consequently $m_i = m'_i$ for all $i \in [n]$.

- Let g and g' be two $(n-t)$-degree polynomials w.r.t. $(m_1, ..., m_n, u)$ and $(m'_1, ..., m'_n, u')$. If there are more than $(n-t)$ different i such that $m_i = m'_i$, then with these $(n-t+1)$ points one can construct the same polynomial g and g', and consequently $u = u'$. Since u, u' are sampled randomly, this happens with negligible probability.

– Randomness. The equation $\mathbf{F}_p(n, t, m_1, ..., m_n, u) = 1$ indicates an $(n-t)$-degree polynomial $g(X) = u + a_1 X + a_2 X^2 + ... + a_{n-t} X^{n-t}$ for some coefficients $a_1, ..., a_{n-t}$. If u distributes uniformly over $\mathcal{M}$, then for every $i \in [n]$, $m_i = g(i) = u + \sum_{j \in [n-t]} a_j i^{n-t}$ is a uniform distribution over $\mathcal{M}$.

G Proof under (Normal) Chameleon Hashes

Theorem 13. *If CH is strongly secure (i.e., it has κ-uniformity, random trapdoor collision, and strong collision resistance) and unique, and $\mathbf{F}_\theta$ is a constraint function, then the AMS scheme AMS constructed in Section 5 has strong unforgeability and strong anonymity under adaptive corruptions. More precisely, for any PPT adversary $\mathcal{A}$, there exist PPT algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $Time(\mathcal{B}_1) \approx Time(\mathcal{B}_2) \approx Time(\mathcal{A})$, and*

$$\mathsf{Adv}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq \sqrt{n(Q_{sign} + Q_H)\mathsf{Adv}^{s\text{-}cr}_{\mathsf{CH},\mathcal{B}_1}(\lambda) + \epsilon_1} + n\mathsf{Adv}^{s\text{-}cr}_{\mathsf{CH},\mathcal{B}_2}(\lambda) + \epsilon_2,$$

where ϵ_1, ϵ_2 are some negligible functions in λ, and Q_{sign} and Q_H are the numbers of signing queries and hash queries, respective.

Proof. We mainly focus on the proof of strong unforgeability since proof of strong anonymity is the same as that in Section 5.

The theorem is proved via four hybrid games $\mathsf{G}_0 - \mathsf{G}_3$, which are the same as those in Appendix D. And we will use the forking lemma to bound the probability that $\mathcal{A}$ wins in G_3.

Let $(\mathsf{msg}^*, \sigma^* = (t', m_1^*, ..., m_n^*, r_1^*, ..., r_n^*))$ be $\mathcal{A}$'s final forgery and $t^* \leftarrow \mathsf{Ver}(vk, \mathsf{msg}^*, \sigma^*)$. Obviously we have $t' = t^*$ if $\mathcal{A}$ wins. For $i \in [n]$, let $h_i^* \leftarrow \mathsf{Hash}(hk_i, m_i^*, r_i^*)$, and let $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^* || t^*) = u^*$.

Game G_0. This is just the original unforgeability experiment. And we have

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \mathsf{Adv}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda).$$

Game G_1. If $\mathcal{A}$ never asks $H^{t^*}(vk, h_1^*, ..., h_n^*, \mathsf{msg}^* || t^*)$ before outputting the final forgery, then G_1 outputs 0 directly. And we have

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq 1/|\mathcal{M}|.$$

Game G_2. In this game $\mathcal{C}$ changes the way of signing oracle's simulation as follows. Upon receiving a signing query $\mathcal{O}(\mathsf{msg}, G)$ $(|G| = t)$, $\mathcal{C}$ randomly samples m_i and r_i for all $i \in [n]$, computes $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$, and reprograms the random oracle such that $H^t(vk, h_1, ..., h_n, \mathsf{msg} || t) = u^{(j)}$, where $u^{(j)}$ is computed according to the constraint function $\mathbf{F}$. If $\mathcal{C}$ fails to reprogram, i.e.,

$H(vk, h_1, ..., h_n, \mathsf{msg}||t)$ has already been defined before, then G_2 outputs $\perp$ and aborts. At last $\mathcal{C}$ returns the signature $\sigma := (t, \{m_i\}_{i\in[n]}, \{r_i\}_{i\in[n]})$ to $\mathcal{A}$.

We have

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq (Q_{sign} + Q_H)/2^{n\kappa}.$$

Game G_3. We add an extra abort rule in this game. Define by reuse the event that, $\mathcal{A}$ has ever asked $\mathcal{O}(\mathsf{msg}^*, G)$ with some $|G| = t^*$ and gets $\sigma = (t^*, m_1, ..., m_n, r_1, ..., r_n)$ back, and

1. $\mathsf{Hash}(hk_i, m_i, r_i) = \mathsf{Hash}(hk_i, m_i^*, r_i^*)$ for all $i \in [n]$;
2. There exists i such that $(m_i^*, r_i^*) \neq (m_i, r_i)$, and $\mathcal{A}$ never asks $\mathcal{O}_{corr}(i)$.

If reuse happens, then G_3 outputs $\perp$ and aborts.

We have

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq \Pr[\mathsf{reuse}] \leq n \cdot \mathsf{Adv}^{s\text{-}cr}_{\mathsf{LCH},\mathcal{B}_1}(\lambda).$$

Next, we use the forking lemma to bound the probability that $\mathcal{A}$ wins in G_3. The high level idea is to construct an algorithm $\mathcal{B}$ that simulates G_3 for the forger $\mathcal{A}$, and outputs $(j^*, (\mathsf{msg}^*, \sigma^*))$ as long as $\mathcal{A}$ successfully outputs a forgery $(\mathsf{msg}^*, \sigma^*)$, where j denotes that for the j-th query to the random oracle $\mathcal{A}$ asks $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*)$. Then, we show the forking algorithm $\mathcal{F}_{\mathcal{B}}$ associated to $\mathcal{B}$ that finds a collision under some specific chameleon hash key, which completes the proof.

Following the same argument of W_5^1 and W_5^2 in G_5 in Appendix D, we know that if G_3 does not abort, then $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*)$ is defined upon some hash query by $\mathcal{A}$. We construct algorithm $\mathcal{B}$ as follows. Let $(hk', td') \leftarrow \mathsf{CH.Gen}(1^\lambda)$ and $Q := Q_{sign} + Q_H$, where Q_{sign} and Q_H are the numbers of signing queries and hash queries, respective. Given the initial input $(hk', u^{(1)}, ..., u^{(Q)})$ and the randomness coin ρ, $\mathcal{B}$ randomly samples $i^* \xleftarrow{\$} [n]$, and let $hk_{i^*} := hk'$. For other $i \in [n]\backslash i^*$, $\mathcal{B}$ invokes $(hk_i, td_i) \leftarrow \mathsf{CH.Gen}(1^\lambda)$. Then it sets $vk := (hk_1, ..., hk_n)$ and sends vk to $\mathcal{A}$.

$\mathcal{B}$ simulates $\mathcal{O}_{corr}(\cdot)$ and $\mathcal{O}(\cdot, \cdot)$ for $\mathcal{A}$ as follows.

- Simulation of corruption queries $\mathcal{O}_{corr}(i)$. If $i = i^*$ then $\mathcal{B}$ aborts, otherwise it returns td_i.
- Simulation of the j-th query $H^t(\cdot)$. Return $\hat{m}^{(j)}$ directly.
- Simulation of the j-th query $\mathcal{O}(\mathsf{msg}, G)$. Let $t := |G|$. $\mathcal{B}$ randomly samples m_i for $i \in [n]\backslash G$, and computes $\{m_i\}_{i\in G}$ from $\{m_i\}_{i\in[n]\backslash G}$ and $u^{(j)}$ according to the forward sample algorithm of the constraint function $\mathbf{F}$. Obviously the distribution of $\{m_i\}_{i\in[n]}$ is the same as that in G_2 due to. Then, it randomly samples r_i for all $i \in [n]$, computes $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$, and reprograms the random oracle such that $H^t(vk, h_1, ..., h_n, \mathsf{msg}||t) = u^{(j)}$. Finally $\mathcal{B}$ returns the signature $\sigma := (t, \{m_i\}_{i\in[n]}, \{r_i\}_{i\in[n]})$.

Finally, $\mathcal{A}$ outputs its forgery $(\mathsf{msg}^*, \sigma^*)$. If $\mathcal{B}$ does not abort in the simulation, and $\mathcal{A}$'s forgery is valid and related to the j-th query, then $\mathcal{B}$ outputs $(j, (\mathsf{msg}^*, \sigma^*))$. In any other case $\mathcal{B}$ outputs $(0, \perp)$.

First, we know that, for $\mathcal{A}$ to win in G_3, at least one signer is not corrupted. Since i^* is randomly chosen from $[n]$ and totally hidden from the adversary, $\mathcal{B}$ does not abort with probability at least $1/n$. Therefore, the probability that $\mathcal{B}$ outputs $(j, (\mathsf{msg}^*, \sigma^*))$ is at least $acc_{\mathcal{B}} \geq \Pr[\mathsf{G}_3 \Rightarrow 1]/n$.

According to the forking lemma, there exists a forking algorithm $\mathcal{F}_{\mathcal{B}}$ associated to $\mathcal{B}$ that takes input hk', and with probability $frk \geq acc_{\mathcal{B}} \cdot (acc_{\mathcal{B}}/Q - 1/|\mathcal{M}|)$ it outputs $(j, (\mathsf{msg}^*, \sigma^*), (\mathsf{msg}^{*\prime}, \sigma^{*\prime}))$ (recall that j indexes the forking point, i.e., $\hat{m}^{(j)} \neq \hat{m}^{(j)'}$). Let $\sigma^* = (t^*, m_1^*, ..., m_n^*, r_1^*, ..., r_n^*)$ and $\sigma^{*\prime} = (t^{*\prime}, m_1^{*\prime}, ..., m_n^{*\prime}, r_1^{*\prime}, ..., r_n^{*\prime})$. Upon to the point of the j-th hash query, the environment of $\mathcal{A}$ provided by $\mathcal{B}$ in the first and the second run are identical, since $\mathcal{B}$ uses the same inputs, random tape and values $\hat{m}^{(1)}, ..., \hat{m}^{(j-1)}$ to generate $\mathcal{A}$'s inputs and oracle responses. Therefore, the two executions of $\mathcal{A}$ are identical up to this point, and the arguments of both hash queries must be the same, implying that $\mathsf{msg}^*||t^* = \mathsf{msg}^{*\prime}||t^{*\prime}$, and $(h_1^*, ..., h_n^*) = (h_1^{*\prime}, ..., h_n^{*\prime})$, where $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*)$ and $H(vk, h_1^{*\prime}, ..., h_n^{*\prime}, \mathsf{msg}^{*\prime}||t^{*\prime})$ are the j-th queries in the first and the second run, respectively.

For the first running of $\mathcal{F}_{\mathcal{B}}$, we have $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*) = \hat{m}^{(j)}$. And for the second running of $\mathcal{F}_{\mathcal{B}}$, we have $H(vk, h_1^*, ..., h_n^*, \mathsf{msg}^*||t^*) = \hat{m}^{(j)'}$. According to the interdependency of $\mathbf{F}$, there exist at least t^* pairs $(m_i^*, m_i^{*\prime})$ such that $m_i^* \neq m_i^{*\prime}$, $h_i^* \leftarrow \mathsf{Hash}(hk_i, m_i^*, r_i^*)$, and $h_i^* \leftarrow \mathsf{Hash}(hk_i, m_i^{*\prime}, r_i^{*\prime})$. Since $\mathcal{A}$ can corrupt up to $t^* - 1$ signers, with probability at least $1/(n - t^* + 1)$, we successfully find a collision under the specific chameleon hash key hk'.

Taking all together, we obtain the desired result.

H Proof Sketch of Theorem 3 (Fault-Tolerant ♯AMS)

Proof Sketch. In the proof of weak unforgeability, the hybrid games $\mathsf{G}_0 - \mathsf{G}_4$ are defined similarly, except that G_3 and events W_4^1, W_4^2 in G_4 are skipped since we do not consider strong unforgeability. Let $(\mathsf{msg}^*, \sigma^* = (t^*, F, \{m_i^*, r_i^*\}_{[n]\setminus F}, \{h_i^*\}_{i \in F}))$ be $\mathcal{A}$'s final forgery. Let G' be the corruption group and $|G'| = t'$. For $\mathcal{A}$ to win in G_4, it must hold that $t' < (t^* - |F|)$. Recall that in G_4, all $(n - t')$ non-corrupted hash keys are generated in the lossy mode. Since $t' < (t^* - |F|)$ and $t^* > |F|$, according to the randomness property of $\mathbf{F}_\theta$, we know among G there exists at least one i s.t. hk_i is a lossy hash key. Then according to the ϵ-lossiness, given h_i, for a random m_i, the probability that $\mathcal{A}$ can find a randomness r_i with $\mathsf{Hash}(hk_i, m_i, r_i) = h_i$ is at most ϵ, and weak unforgeability holds as a result.

The proof of unconditional strong anonymity (for honest signers) follows directly from the random trapdoor collision property of LCH, i.e., from the statistical indistinguishability of the following distributions for all $m, m' \in \mathcal{M}$:

$$\left\{(m, r) \,\middle|\, r \xleftarrow{\$} \mathcal{R}\right\} \text{ and } \left\{(m, r) \,\middle|\, \begin{array}{c}(hk, td) \leftarrow \mathsf{LCH.Gen}(1^\lambda), \bar{r} \xleftarrow{\$} \mathcal{R}, \\ r \leftarrow \mathsf{LCH.TdColl}(td, \bar{m}, \bar{r}, m)\end{array}\right\},$$

where the left distribution corresponds to users $i \in [n] \setminus G$, and the right distribution corresponds to honest users $i \in G \setminus F$.

Remark 6 (Why does fault-tolerant ♯AMS schemes achieve weak unforgeability only?). Let $F \subset F'$. It is easy to see that if $\sigma = (t, F, \{m_i, r_i\}_{i\in[n]\setminus F}, \{h_i\}_{i\in F})$ is a $(t - |F|)$-valid ♯AMS signature, then

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

is a $(t-|F'|)$-valid ♯AMS signature. That says, a malicious leader can always decrease the "credibility" of a valid ♯AMS signature. Therefore, the fault-tolerant ♯AMS scheme above can achieve weak unforgeability only. However, we emphasize that in the application of blockchain governance, a malicious developer/moderator P earns nothing meaningful from this kind of attacks. Furthermore, P will get punished due to a malicious disclosure in the voting systems presented in the next section, since all transcript messages between P and the signer are signed using their (regular) digital signatures.

I Other Applications

In this section we discuss some other applications of our ♯AMS schemes.

Linked whistle-blowing [44]. Suppose now a citizen wants to reveal a scandal of the government. To avoid the risk of malicious retaliation, he/she may resort to revealing it via anonymous ways, for example, via using a ring signature. But unfortunately, media or journalists may not believe what the whistleblower tells and think that he/she is telling lies. However, they may choose to believe the disclosure if quite a number of citizens confirm it. In this situation, the whistleblower can first gather some supporters and then generate a ♯AMS signature to announce the scandal to the public.

Ad-hoc networks. Ad-hoc network is a decentralized type of wireless network, which does not rely on any preset infrastructure, such as routers or wireless access points. Instead, each node participates in the network by forwarding data for other nodes. Our ♯AMS schemes constructed from chameleon hashes and linkable ring signatures enjoy the advantage of spontaneity, and they are perfectly applicable for the application where several spontaneous nodes (users) want to communicate secretly. By attaching a ♯AMS signature with the message sent out, the receiver is convinced of the authority of the message and the anonymity of senders maintains.

J Further Discussion

We discuss more about our e-voting system / our blockchain governance system.

Infrastucture and authenticated transcription. We assume a (regular) signature scheme $\mathcal{S}$ that is unforgeable, and each node in the network has published a public key of $\mathcal{S}$. For each message to be sent out, the sender also signs a signature using its own secret key to avoid the message being interpolated in the transmission.

On-chain. After generating a ♯AMS signature, the leader of the proposal will upload it on the blockchain so that it cannot be altered further.

Vote-and-go. The (non-leader) voters are not involved in the announcement period, and they can leave the system after completing their own part without any interaction in the signing period. Our round-optimal voting systems V2 and V3 achieve this property.

Unwareness-before-publishing. Voters remain unaware of the current vote count (hence also other voters) until the results are announced. Our round-optimal voting systems V2 and V3 achieve this property.

Receipt-freeness. Receipt-freeness means that a voter cannot claim the authorship of some particular votes (i.e., some ♯AMS signatures). This property helps prevent vote-buying behaviors. Our scheme has strong anonymity, which means that except for the leader of the proposal, any other signers cannot provide proof of having participated in the signing process.