Reconstructing Chameleon Hash: Full Security and the Multi-Party Setting

Kwan Yin Chan
kychan@cs.hku.hk
The University of Hong Kong
Pokfulam, Hong Kong

Liqun Chen
liqun.chen@surrey.ac.uk
University of Surrey
Guildford, United Kingdom

Yangguang Tian
yangguang.tian@surrey.ac.uk
University of Surrey
Guildford, United Kingdom

Tsz Hon Yuen*
john.tszhonyuen@monash.edu
Monash University
Clayton, Australia

ABSTRACT

Chameleon hash (CH) function differs from a classical hash function in a way that a collision can be found with the knowledge of a trapdoor secret key. CH schemes have been used in various cryptographic applications such as sanitizable signatures and redactable blockchains. In this work, we reconstruct CH to ensure advanced security and usability. Our contributions are four-fold. First, we propose the *first* CH scheme, which supports full security, meaning the inclusion of both full indistinguishability and full collision-resistance. These two properties are required in the strongest CH security model in the literature. We achieve this by our innovative design of removing the CH public key during the computation of the hash value. Second, we investigate the security of CH in the multi-party setting and introduce the new properties of *claimability* and *deniability* under this setting. Third, we present and implement two instantiations of our CH scheme: an ECC-based one and a post-quantum lattice-based one. Our implementation demonstrates their practicality. Finally, we discuss the possible use cases in the blockchain.

CCS CONCEPTS

• **Security and privacy → Mathematical foundations of cryptography**.

KEYWORDS

Chameleon hash, Full Security, Multi-party Setting, Post-Quantum Cryptography

ACM Reference Format:
Kwan Yin Chan, Liqun Chen, Yangguang Tian, and Tsz Hon Yuen. 2024. Reconstructing Chameleon Hash: Full Security and the Multi-Party Setting. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3634737.3656291

*Corresponding author

ASIA CCS '24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07
https://doi.org/10.1145/3634737.3656291

1 INTRODUCTION

Chameleon hash (CH) function was introduced by Krawczyk and Rabin at NDSS 2000 [21]. A chameleon-hash CH function is a trapdoor collision-resistant hash function parameterized by a public key. Chameleon hash outputs a hash value h and a randomness (or check string) r for a given message m and a public key $\mathsf{pk}_{\mathsf{ch}}$. The owner of the secret key $\mathsf{sk}_{\mathsf{ch}}$ can use r, m to generate a different randomness r' for another message m'. The hash output pairs (h, r) and (h, r') are valid for the message m and m' respectively. In the classical CH construction [21]:

$$h = g^m \mathsf{pk}_{\mathsf{ch}}^r,$$

where g is a generator and $\mathsf{pk}_{\mathsf{ch}} = g^{\mathsf{sk}_{\mathsf{ch}}}$. In order to adapt h to a different message m', the owner of the secret key just needs to calculate r' such that:

$$m + \mathsf{sk}_{\mathsf{ch}} \cdot r = m' + \mathsf{sk}_{\mathsf{ch}} \cdot r'.$$

The operation of computing (m, r) is referred to as "hashing" and of computing (m', r') as "adapting".

CH functions have been used as building blocks for many cryptographic primitives and real-life applications, including online/offline signatures [31], sanitizable signatures [1], double-authentication-preventing signatures [27], ring signatures [24], asynchronous payment channels [29], redactable blockchains [2] and privacy-preserving payment channel networks [33].

1.1 Full Security in Chameleon Hash

A chameleon hash usually needs to achieve indistinguishability (IND) and collision-resistance (CollRes). IND means that the randomness associated with a chameleon hash does not reveal whether it was derived from hashing or adapting. The full indistinguishability property [30] implies that any third parties cannot break indistinguishability even if the secret key $\mathsf{sk}_{\mathsf{ch}}$ is generated by the adversary. CollRes means that given the public key $\mathsf{pk}_{\mathsf{ch}}$, no third parties can find two pairs (m, r) and (m', r') that are valid under $\mathsf{pk}_{\mathsf{ch}}$ and map to the same chameleon hash value h. The full collision-resistance property [11] implies that the adversary can adaptively query the CHAdapt oracle and use a queried message m as the attack on the collision-resistance property as long as the pair (h, m) was not involved in the CHAdapt oracle query. The definitions of IND and CollRes are various in the literature, including weak [21], enhanced [2], standard [7] and full security [11] for CollRes,

and standard [7], strong [12] and full security [30] for IND. The complete discussions can be found in [11, Section 5 & Appendix A]. A CH scheme supporting IND (or CollRes) but not full IND (or full CollRes) is said that it holds standard IND (or standard CollRes). As discussed in [11], the full indistinguishability and the full collision-resistance are the strongest security models in the literature.[1]

Full indistinguishability. The RSA-based CH construction from Brzuska et al. [4] has full indistinguishability and standard collision-resistance. In this scheme, the hash algorithm picks a randomness r and sets the hash value $h = H(m) \cdot \mathsf{RSA.Enc}_{\mathsf{pk}_{\mathsf{ch}}}(r)$, where H is a full domain hash function. By use of the secret key $\mathsf{sk}_{\mathsf{ch}}$, one can set

$$h = H(m) \cdot \mathsf{RSA.Enc}_{\mathsf{pk}_{\mathsf{ch}}}(r) = H(m') \cdot \mathsf{RSA.Enc}_{\mathsf{pk}_{\mathsf{ch}}}(r'),$$

with different pairs (m, r) and (m', r'). Hence, due to the security of RSA.Enc, it achieves full indistinguishability. In the proof of the standard collision-resistant, the CHAdapt oracle query is answered by (the oracle of) the one-more RSA problem in [30]. In the challenge phase, the adversary returns a new message m^* that has not queried before, and hence the output of the adversary can be used to solve the one-more RSA problem. However, the same proof cannot work in the full collision-resistance model, since the adversary is allowed to return some old message m that is queried to the CHAdapt oracle before.

Full collision-resistance. Derler et al. [11] proposed a generic construction of chameleon hashes (denoted as DSS20) by using public key encryption (or a commitment [10]) and a zero-knowledge proof. In DSS20 [11], the hash value $h = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ch}}}(m; \rho)$, which is the encryption of m under the randomness ρ. The randomness r is the non-interactive zero-knowledge (NIZK) proof of knowing ρ with respect to h, or knowing $\mathsf{sk}_{\mathsf{ch}}$ with respect to $\mathsf{pk}_{\mathsf{ch}}$. They achieve full collision-resistance by the use of NIZK proof in generating r. By using the simulator of the NIZK proof, the CHAdapt oracle query can be answered without the knowledge of the secret key $\mathsf{sk}_{\mathsf{ch}}$. However, DSS20 [11] cannot achieve full indistinguishability, since they define the hash value $h = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ch}}}(m; \rho)$. By using the decryption key, the original message m is decrypted and breaks the full indistinguishability.

As summarized in Table 1, it is an open problem to design a chameleon hash with full indistinguishability and full collision-resistance.

1.2 High-Level Idea of Our Scheme

Before presenting our solution, we first describe a practical attack on CollRes, which is not considered in the existing security model. This attack inspires us to formulate our new generic construction of chameleon hash with full security.

Security in the Multi-key Setting. The existing security models of chameleon-hash only consider a single public key $\mathsf{pk}_{\mathsf{ch}}$. For CollRes, no polynomial time attacker can output h and two pairs (m, r) and (m', r') that are valid under $\mathsf{pk}_{\mathsf{ch}}$. However, we consider a *key replacement attack* for the classical CH construction [21] by an attacker who knows a valid (m, r) under $\mathsf{pk}_{\mathsf{ch}}$. He can pick a random r' and calculate:

$$\mathsf{pk}_A = \mathsf{pk}_{\mathsf{ch}}^{r/r'} \cdot g^{(m-m')/r'},$$

for any message m'. Now we can see that (m', r') and h is valid under pk_A, since:

$$g^{m'} \mathsf{pk}_A^{r'} = g^{m'} (\mathsf{pk}_{\mathsf{ch}}^{r/r'} \cdot g^{(m-m')/r'})^{r'} = g^m \mathsf{pk}_{\mathsf{ch}}^r = h.$$

We observe that this attack is *outside* the existing security model of CollRes. In fact, this attack also applies to schemes in which h is additive/multiplicative homomorphic over $\mathsf{pk}_{\mathsf{ch}}$. Details can be found in the Appendix A. A simple solution to avoid this key replacement attack is to change the hash value to $(h, H'(\mathsf{pk}_{\mathsf{ch}}))$ for some hash function H'. An additional checking over $H'(\mathsf{pk}_{\mathsf{ch}})$ is needed to prevent the key replacement attack. Since the fix is simple, we do not change the existing models to capture this attack, and we also do not write $H'(\mathsf{pk}_{\mathsf{ch}})$ explicitly in the rest of the paper.

From this attack, we observe that even if we use $\mathsf{pk}_{\mathsf{ch}}$ in the calculation of h, it cannot prevent the key replacement attack. On the other hand, it may even harm the full indistinguishability with the use of $\mathsf{pk}_{\mathsf{ch}}$ (the case of [11]).

Our Solution: "Hashing" without CH Public Key. In this paper, we reconstruct the chameleon hash and change the way we calculate the *hash value h*. Surprisingly, we find out that it is not necessary for the "hash value" h to contain any information about the owner public key $\mathsf{pk}_{\mathsf{ch}}$. In contrast, we can use $\mathsf{pk}_{\mathsf{ch}}$ only when we calculate the "randomness" r, i.e.:

$$h = F_h(m, \rho), \quad r = F_r(m, \rho, \mathsf{pk}_{\mathsf{ch}}),$$

for some functions F_h, F_r and internal randomness ρ. The hash value h itself is not bound to any public key, and hence it can bind to anyone only when the creator generates r. By using this new structure, we can achieve full indistinguishability since the knowledge of $\mathsf{sk}_{\mathsf{ch}}$ cannot help to distinguish if m is bound in h. We also choose to use a NIZK proof system for F_r to achieve full collision-resistance.

We give a new generic construction of chameleon hash by using a one-way function F, a collision-resistant hash function H mapping from the message space to the range of F, and a NIZK proof system for disjunctive relation (the so-called *OR proof*). Suppose that $\mathsf{pk}_{\mathsf{ch}}$ is generated from $F(\mathsf{sk}_{\mathsf{ch}})$. In order to hash a message m, we first pick a random ρ and calculate

$$h = F(\rho) \oplus H(m),$$

where $\oplus$ is the bitwise XOR operation. The randomness r is the NIZK proof of knowing ρ such that:

$$F(\rho) = h \oplus H(m) \vee F(\rho) = \mathsf{pk}_{\mathsf{ch}}.$$

The chameleon hash function outputs (h, r). To verify the output (h, r), one just needs to validate the NIZK proof.

To adapt (h, r) to another message m', the owner of $\mathsf{sk}_{\mathsf{ch}}$ calculates another NIZK proof r' for the above relation, but proving the knowledge of $\mathsf{sk}_{\mathsf{ch}}$ instead of ρ, and it is now written as:

$$F(\mathsf{sk}_{\mathsf{ch}}) = h \oplus H(m') \vee F(\mathsf{sk}_{\mathsf{ch}}) = \mathsf{pk}_{\mathsf{ch}}.$$

We can easily instantiate the generic construction in the elliptic curve (ECC) setting and lattice-based setting. They are both practical for real-world use cases.

[1] There is another property called *uniqueness* [7], but it is not considered as a fundamental property.

Table 1: The comparison of security models between various CHs. – indicates IND is not required in [2, 21].

	[21]	[2]	[7]	[12]	[30]	[11]	[10]	Ours
CollRes	Weak	Enhanced	Standard	Standard	Standard	Full	Full	Full
IND	-	-	Standard	Strong	Full	Standard	Strong	Full
Multi-Party	×	✓	×	✓	✓	×	×	✓

1.3 Chameleon Hash in the Multi-Party Setting

Apart from the generic construction, the key replacement attack also inspires us to reconsider the security of chameleon hash, especially in the multi-party setting. Several CH schemes in Table 1 support multi-party setting, meaning that multiple trapdoor holders can perform adapting [12, 30]. Multi-party settings in the CH scheme [9] and sanitizable signatures [5, 8] have been used in real applications, such as content protection and secure routing.

The Creator and the Owner. The existing IND and CollRes security models mainly consider attacks from a third party. In the model of full indistinguishability [30] and strong indistinguishability [12], the adversary can also obtain the secret key $\mathsf{sk}_{\mathsf{ch}}$. However, none of the existing models considers the ability of the original creator of the hash output. To the best of our knowledge, there is not even a formal name for this role. We name the one who created h as the Creator, while the one who owns the secret key $\mathsf{sk}_{\mathsf{ch}}$ as the Owner.

Claimability and Deniability. Next, we consider the ability of the Creator. Consider the application that the hash output (h, r) is written in a redactable blockchain by the Creator, and then it is later modified to (h, r') by the Owner using a smart contract. Theoretically, it is possible that the Creator can claim the authorship of h, or claim that the original message m is associated with h. For example in DSS20 [11], $h = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ch}}}(m; \rho)$. By outputting (m, ρ), the Creator can claim the creation of h and the original message m if the Owner cannot recover valid encryption randomness for any message m' using his secret key (this condition holds for the ElGamal encryption instantiation used in [11]). It implies that the Creator can voluntarily break the IND property based on h. This kind of insider attack is not considered in the literature.

After defining the roles of the Creator and the Owner, we can even consider more complicated situations. The Creator or the Owner may claim the authorship of the randomness r or r' generated by himself. It implies that either party can voluntarily break the IND property based on r or r'. For example in DSS20 [11], the Creator or the Owner can use the randomness used in the NIZK proof to claim such authorship. These situations may have different implications in real-world applications.

When we consider it from the opposite direction, the Creator or the Owner may deny generating the randomness r' or r respectively. If one party denies generating a randomness, then it must be generated by the other party. It implies that the IND property can also be broken by the counterparty. However, it is non-trivial to construct schemes with this *deniability* property.

Multi-Owner Chameleon Hash. After defining the roles of the creator and the owner, it is natural for us to extend CH to a multi-owner setting. It means that multiple parties have the ability to adapt the hash value h. At first glance, it may look easy to extend it from the classical CH construction [21]. For a message m and randomly chosen r_1, r_2, compute:

$$h = g^m (\mathsf{pk}_{\mathsf{ch}}^{(1)})^{r_1} (\mathsf{pk}_{\mathsf{ch}}^{(2)})^{r_2},$$

for two owner public keys $\mathsf{pk}_{\mathsf{ch}}^{(1)}, \mathsf{pk}_{\mathsf{ch}}^{(2)}$. However, if the first owner wants to adapt it, he can only change r_1 to r_1' such that $m + \mathsf{sk}_{\mathsf{ch}}^{(1)} \cdot r_1 = m' + \mathsf{sk}_{\mathsf{ch}}^{(1)} \cdot r_1'$. After adapting, he outputs (m', r_1', r_2). It trivially violates (the multi-owner version of) indistinguishability, since the hash outputs (m, r_1, r_2) and (m', r_1', r_2) are not generated by the second owner $\mathsf{pk}_{\mathsf{ch}}^{(2)}$.

Therefore, it is an open problem to define the security model of the multi-owner version of indistinguishability, and to give a secure construction. Fortunately, our generic construction can be easily extended to the multi-owner setting. Since the construction of h does not involve any public key, we do not need to change it. We just need to extend the disjunctive proof to a multiple one when generating r.

1.4 Our Contributions

In this paper, we study two main open problems in chameleon-hash:

- Achieving full indistinguishability and full collision-resistance at the same time.
- Evaluating the (in)security of chameleon-hash in the multi-party setting.

We propose a new generic construction of chameleon-hash, which is inspired by our key replacement attack on existing schemes. It is the first chameleon-hash that can achieve full indistinguishability and full collision-resistance simultaneously. We define different parties (creator and owners) involved in chameleon hash, and investigate their ability to claim or deny the generation of the chameleon hash. We also extend our generic construction to give the first multi-owner chameleon hash.

2 BACKGROUND

2.1 Chameleon Hash

We show the definition of chameleon hashes [7, 11], which is based on the work done by [2, 4].

- CHPG: It takes a security parameter λ as input, outputs public parameters $\mathsf{pp}_{\mathsf{ch}}$.
- CHKG: It takes public parameters $\mathsf{pp}_{\mathsf{ch}}$ as input, outputs a chameleon key pair $(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}})$.
- CHash: It takes the chameleon public key $\mathsf{pk}_{\mathsf{ch}}$, and a message $m \in \mathcal{M}$ as input, outputs a chameleon hash h, a randomness r. Note that $\mathcal{M} = \{0,1\}^*$ denotes a general message space.

- CHCheck: It takes the chameleon public key $\mathsf{pk}_{\mathsf{ch}}$, a message m, a chameleon hash h and a randomness r as input, outputs a bit $b \in \{0,1\}$.
- CHAdapt: It takes the chameleon secret key $\mathsf{sk}_{\mathsf{ch}}$, messages m, m', chameleon hash h and randomness r as input, outputs a new randomness r'.

Correctness states that a pair (h, r), computed by the CHash algorithm, verifies with overwhelming probability. Also, the CHAdapt algorithm always verifies if the given chameleon hash h is valid, or it outputs $\perp$. In the definition above, hashing is assumed to be randomized. The randomness r (or check string) is a public value, and the implicit random coins (or internal randomness) are secret. This definition is a generalization of standard chameleon hashes [21]. Now, we present two security guarantees: full indistinguishability [30] and full collision-resistance [11], which will be used in the security analysis of our proposed constructions.

Full Indistinguishability. Informally, for a chameleon hash, an adversary cannot decide whether its randomness was freshly generated using CHash algorithm or was created using CHAdapt algorithm. The adversary is allowed to generate the keys which are used for hashing and adapting. We define a formal experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{S}$ in Figure 1. The security experiment allows $\mathcal{A}$ to access a HashOrAdapt oracle which ensures that the randomness does not reveal whether it was obtained from CHash or CHAdapt algorithm.
We define the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathrm{IND}}_{\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathrm{IND}}_{\mathcal{A}}(\lambda) \rightarrow 1] - 1/2|.$$

Definition 2.1. A CH scheme is fully indistinguishable if for any probabilistic polynomial-time PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{IND}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

Full Collision Resistance. Informally, an adversary attempts to find valid collisions without using trapdoors. The adversary can access an CHAdapt oracle: it takes messages (m, m') as input, outputs a collision for the adversarially chosen hash and records (m, m') under a list Q. $\mathcal{A}$ wins if it outputs a collision for an adversarially generated chameleon hash h^*, while (h^*, m^*) was not previously queried to the CHAdapt oracle. We define a formal experiment in Figure 2.
We define the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathrm{CR}}_{\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathrm{CR}}_{\mathcal{A}}(\lambda) \rightarrow 1]|.$$

Definition 2.2. A CH scheme is fully collision-resistant if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{CR}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

2.2 One-Way Function

Definition 2.3. A function $F : D_F \rightarrow R_F$ is one-way, if the advantage of PPT $\mathcal{A}$ defined in the following equation is negligible in λ.

$$\Pr[x \in_R D_F, x' \in_R \mathcal{A}(F(x)) : F(x) = F(x')].$$

We assume that the domain D_F and range R_F are defined by F.

2.3 Non-Interactive Zero-Knowledge

Let $L = \{x | \exists w : R(x, w) = 1\}$ be an NP-language with a relation R. A non-interactive proof system allows to prove membership of some statement x in the language L. The formal definition is shown below.

Experiment $\mathsf{Exp}^{\mathrm{IND}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CHPG}(\lambda), b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}(\cdots, b)}(\mathsf{pp}_{\mathsf{ch}})$
  where $\mathsf{HashOrAdapt}(\cdots, b)$ on input $\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}, m, m', b$:
    $(h_0, r_0) \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m')$
    $(h_1, r'_1) \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m)$
    $r_1 \leftarrow \mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', h_1, r'_1)$
    return $\perp$ if $r_b = \perp \vee r'_1 = \perp$
    return (h_b, r_b)
if $b' = b$, return 1; else, return 0.

Figure 1: Full Indistinguishability.

Experiment $\mathsf{Exp}^{\mathrm{CR}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CHPG}(\lambda), Q \leftarrow \emptyset$
$(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}) \leftarrow \mathsf{CHKG}(\mathsf{pp}_{\mathsf{ch}})$
$(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, \cdots)}(\mathsf{pk}_{\mathsf{ch}})$
  where $\mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, \cdots)$ on input $\mathsf{sk}_{\mathsf{ch}}, m, m', h, r$:
    return $\perp$, if $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, h, m, r) \neq 1$
    $r' \leftarrow \mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', h, r)$
    $Q \leftarrow Q \cup \{(h, m), (h, m')\}$
    return r'
if $1 = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m^*, h^*, r^*) = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m'^*, h^*, r'^*)$
$\wedge (h^*, m^*) \notin Q \wedge m^* \neq m'^*$, return 1;
$//(h*, m*)$ does not appear as first or second pair in CHAdapt query
else, return 0.

Figure 2: Full Collision-Resistance.

Definition 2.4. A non-interactive proof system Π for language L includes the following three algorithms.

- PG: It takes a security parameter λ as input, outputs public common reference string (CRS) crs_{Π}.
- Pf: It takes the CRS crs_{Π}, a statement x and a witness w, outputs a proof π.
- Vfy: It takes the CRS crs_{Π}, a statement x and a proof π, outputs a bit $b \in \{0,1\}$.

Non-interactive proof systems typically satisfy three properties: correctness, zero-knowledge, and soundness. Correctness (or completeness) says that an honest prover should always be able to convince the verifier that $x \in L$. Zero-knowledge says that the receiver of the proof π cannot learn anything except the validity of the statement. Soundness says that a malicious prover cannot generate a proof π for any element $x \notin L$. We use a stronger formulation of the soundness called simulation-sound extractability [11, 13]. It says that every adversary who can generate a proof π^* for a statement x^* must know the witness w^*, even when seeing simulated proofs for adaptively chosen statements not in language L. The distribution of the common reference string crs_{Π} generated by PG and the distribution of the simulated crs_{Π} by SIM_1 or Ext_1 are assumed to be indistinguishable.

Definition 2.5 (Correctness). A non-interactive proof system Π is correct, if for all λ, for all $\mathsf{crs}_{\Pi} \leftarrow \mathsf{PG}(\lambda)$, for all $x \in L$, for all w such that $R(x, w) = 1$, for all $\pi \leftarrow \mathsf{Pf}(\mathsf{crs}_{\Pi}, x, w)$, it holds that $\mathsf{Vfy}(\mathsf{crs}_{\Pi}, x, \pi) = 1$.

Definition 2.6 (Zero-Knowledge). We define a formal experiment between a PPT adversary $\mathcal{A}$ and a simulator $\mathsf{SIM} = (\mathsf{SIM}_1, \mathsf{SIM}_2)$ in Figure 3. Note that τ denotes a simulation trapdoor. We define

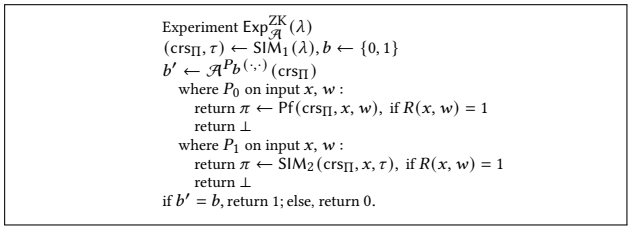

Figure 3: Zero-Knowledge.

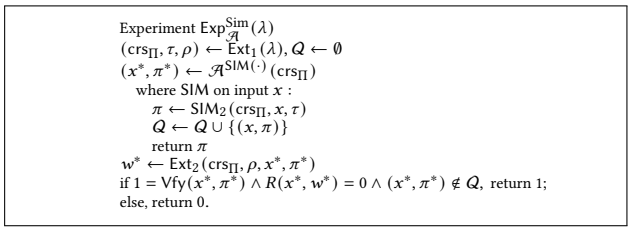

Figure 4: Simulation Sound Extractability.

the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{ZK}}_{\mathcal{A}}(\lambda) \rightarrow 1] - 1/2|.$$

The non-interactive proof system Π for language L is zero-knowledge, if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

Definition 2.7 (Simulation-sound Extractability). We define a formal experiment between a PPT adversary $\mathcal{A}$ and an extractor $\mathsf{Ext} = (\mathsf{Ext}_1, \mathsf{Ext}_2)$ in Figure 4. We define the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{Sim}}_{\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{Sim}}_{\mathcal{A}}(\lambda) \rightarrow 1]|.$$

The non-interactive proof system Π for language L is simulation-sound extractable, if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{Sim}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

3 OUR CONSTRUCTIONS

In this section, we introduce a new generic construction of chameleon hash from a one-way function (OWF) and a non-interactive zero-knowledge (NIZK) proof.

Suppose that F is a one-way function with its output uniformly distributed in the range R_F[2]. We define an efficient NIZK proof of knowledge x for the following language:

$$L := \{(F, y_1, y_2) | x : F(x) = y_1 \vee F(x) = y_2\}.$$

This is known as the disjunctive proof. The NIZK proof system Π consists of three algorithms $(\mathsf{PG}, \mathsf{Pf}, \mathsf{Vfy})$. Denote $\oplus$ as the bitwise XOR operation[3].

3.1 Generic Construction

Our generic CH algorithm is as follows:

- $\mathsf{CHPG}(\lambda)$: It picks a collision-resistant hash function H that maps the message space to the range R_F of F. It runs $\mathsf{crs}_\Pi \leftarrow \mathsf{PG}(\lambda)$. It outputs public parameters $\mathsf{pp}_{\mathsf{ch}} = (F, H, \mathsf{crs}_\Pi)$.
- $\mathsf{CHKG}(\mathsf{pp}_{\mathsf{ch}})$: It outputs a random $\mathsf{sk}_{\mathsf{ch}}$ (randomly chosen from the domain of F) and $\mathsf{pk}_{\mathsf{ch}} = F(\mathsf{sk}_{\mathsf{ch}})$.
- $\mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m)$: It picks a random ρ from the domain of F and calculates $h = F(\rho) \oplus H(m)$. It computes a zero-knowledge proof $r \leftarrow \mathsf{Pf}(\mathsf{crs}_\Pi, (F, h \oplus H(m), \mathsf{pk}_{\mathsf{ch}}), \rho)$. It outputs (h, r). Note that r is a NIZK proof for the witness ρ such that
$$F(\rho) = h \oplus H(m) \vee F(\rho) = \mathsf{pk}_{\mathsf{ch}}.$$
The parameters $(F, h \oplus H(m), \mathsf{pk}_{\mathsf{ch}})$ are public in the NIZK proof.
- $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, h, r)$: It outputs $b \leftarrow \mathsf{Vfy}(\mathsf{crs}_\Pi, (F, h \oplus H(m), \mathsf{pk}_{\mathsf{ch}}), r)$.
- $\mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', h, r)$: It returns $\perp$ if $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, h, r) = 0$. Otherwise, it outputs another zero-knowledge proof $r' \leftarrow \mathsf{Pf}(\mathsf{crs}_\Pi, (F, h \oplus H(m'), \mathsf{pk}_{\mathsf{ch}}), \mathsf{sk}_{\mathsf{ch}})$.
Note that r' is a NIZK proof for $\mathsf{sk}_{\mathsf{ch}}$ such that
$$F(\mathsf{sk}_{\mathsf{ch}}) = h \oplus H(m') \vee F(\mathsf{sk}_{\mathsf{ch}}) = \mathsf{pk}_{\mathsf{ch}}.$$

3.1.1 Security. We prove the security of our generic construction under the strong model of full indistinguishability and full collision-resistance.

Theorem 3.1. *Our CH scheme is fully indistinguishable if* Π *has the zero-knowledge property and the output of* F *is uniformly distributed in its range* R_F.

Proof. The proof is given by a sequence of games. We define the following games:

- Game_0: The same as the IND game.
- Game_1: The output r_0 from $\mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m')$ is given by the simulator of Π.
- Game_2: The output r_1 from $\mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', h_1, r'_1)$ is given by the simulator of Π.
- Game_3: The output h_0 from $\mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m')$ is replaced by a random number in R_F.
- Game_4: The output h_1 from $\mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m)$ is replaced by a random number in R_F.

By the zero-knowledge property of Π, we can see that no PPT adversary can distinguish between Game_0 and Game_1 (and also between Game_1 and Game_2) with non-negligible probability.

If the output of F is uniformly distributed in its range, then $F(\rho)$ is indistinguishable with a random number chosen from the range. Hence, no PPT adversary can distinguish $F(\rho) \oplus H(m')$ (in Game_2) with a random number h (in Game_3) with non-negligible probability. The same argument applies for Game_3 and Game_4.

Finally in Game_4, the pairs (h_0, r_0) and (h_1, r_1) are randomly generated. Hence no PPT adversary can win this game with non-negligible probability. □

We need a new assumption to prove the full collision-resistance property.

Definition 3.2 (Assumption 1). Given a one-way function F and a collision resistant hash function H, no PPT adversary can output x_0, x_1, m_0, m_1 with non-negligible probability such that $m_0 \neq m_1$ and

$$F(x_0) \oplus F(x_1) = H(m_0) \oplus H(m_1).$$

[2]For example, under the discrete logarithm assumption, $F(x) = g^x$ is uniformly distributed in the group $\mathbb{G}$ when g is a generator of $\mathbb{G}$. Under the RSA assumption, $F(x) = x^e \bmod N$ is uniformly distributed when given the RSA public key (e, N).
[3]We can replace the bitwise XOR operation with an algebraic operation over the range of F, with some minor modification in the writing of the security proof and the assumptions used. See section 3.2.

We note that the Assumption 1 should be analyzed in the concrete instantiation. For example in the discrete logarithm(DL)-based setting, if $F(x) = g^x$ and $\oplus$ is point addition, then the hard problem becomes outputting x_0, x_1, m_0, m_1 such that $g^{x_0+x_1} = H(m_0) \cdot H(m_1)$. It can be easily reduced to the DL problem if H is modeled as a random oracle. In the RSA-based setting, if $F(x) = x^e \bmod N$ where (e, N) is a RSA public key, and $\oplus$ is modular multiplication, the hard problem becomes outputting x_0, x_1, m_0, m_1 such that $(x_0x_1)^e = H(m_0)H(m_1) \bmod N$. It can be reduced to the RSA problem if H is modeled as a random oracle.

Theorem 3.3. *Our CH scheme is fully collision resistant if F has the one-wayness property, Π has simulation-sound extractability and assumption 1 holds.*

Proof. The simulator $\mathcal{S}$ runs as the adversary of breaking one-wayness property of F. $\mathcal{S}$ is given y and wants to find $F^{-1}(y)$.

$\mathcal{S}$ runs the extractor Ext_1 of Π to obtain $(\mathsf{crs}_\Pi, \tau, \rho)$. $\mathcal{S}$ sets $\mathsf{pp}_{\mathsf{ch}} = (F, H, \mathsf{crs}_\Pi)$, $\mathsf{pk}_{\mathsf{ch}} = y$ and returns $(\mathsf{pp}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}})$ to the adversary $\mathcal{A}$.

For the CHAdapt oracle query with input (m, m', h, r), $\mathcal{S}$ runs the extractor Ext_2 of Π to obtain $\hat{x} \leftarrow \mathsf{Ext}_2(\mathsf{crs}_\Pi, \rho, (F, h \oplus H(m), \mathsf{pk}_{\mathsf{ch}}), r)$ such that

$$\hat{x} : F(\hat{x}) = h \oplus H(m) \vee F(\hat{x}) = y.$$

If $F(\hat{x}) = y$, $\mathcal{S}$ simply returns $\hat{x}$ as the solution of $F^{-1}(y)$ and aborts. Otherwise, $\mathcal{S}$ runs the simulator of the zero-knowledge proof to generate $r' \leftarrow \mathsf{SIM}_2(\mathsf{crs}_\Pi, \tau, (h \oplus H(m'), \mathsf{pk}_{\mathsf{ch}}))$ and return r' to $\mathcal{A}$.

In the output phase, the adversary returns $(m^*, r^*, m'^*, r'^*, h^*)$ with $m^* \neq m'^*$ and $(h^*, m^*) \notin Q$. We consider two cases: (1) $(h^*, m'^*) \notin Q$; (2)$(h^*, m'^*) \in Q$.

We first consider case 1. Since it passes CHCheck, it means that $\mathcal{S}$ can use the extractor Ext_2 to obtain

$$x_0 : F(x_0) = h^* \oplus H(m^*) \vee F(x_0) = y,$$
$$x_1 : F(x_1) = h^* \oplus H(m'^*) \vee F(x_1) = y.$$

Now we have two possible sub-cases:

- 1a. $F(x_0) = y$ or $F(x_1) = y$.
- 1b. $F(x_0) = h^* \oplus H(m^*)$ and $F(x_1) = h^* \oplus H(m'^*)$.

For the case 1a, $\mathcal{S}$ simply outputs x_0 or x_1 as the solution of $F^{-1}(y)$. For the case 1b, it implies that

$$h^* = F(x_0) \oplus H(m^*) = F(x_1) \oplus H(m'^*).$$

Since H is a collision resistant hash function, $H(m^*) \neq H(m'^*)$ for $m^* \neq m'^*$. By assumption 1, this case happens with a negligible probability.

Next, we consider case 2. If $(h^*, m'^*) \in Q$, there are two sub-cases:

- 2a. There was a CHAdapt query with input $(m'^*, \cdot, h^*, \cdot)$.
- 2b. There was a CHAdapt query with input (m, m'^*, h^*, r).

For case 2a, by the simulation of the CHAdapt query, $\mathcal{S}$ already extracts $\hat{x}_{2a}$ such that $F(\hat{x}_{2a}) = h^* \oplus H(m'^*)$. Similar to case 1, $\mathcal{S}$ can use the extractor Ext_2 to obtain

$$x_0 : F(x_0) = h^* \oplus H(m^*) \vee F(x_0) = y.$$

Hence, $\mathcal{S}$ either returns x_0 as the solution to $F^{-1}(y)$, or obtains

$$h^* = F(x_0) \oplus H(m^*) = F(\hat{x}_{2a}) \oplus H(m'^*),$$

which contradicts the Assumption 1.

For case 2b, by the simulation of the CHAdapt query, $\mathcal{S}$ already extracts $\hat{x}_{2b}$ such that $F(\hat{x}_{2b}) = h^* \oplus H(m)$. Since $(h^*, m) \in Q$ and $(h^*, m^*) \notin Q$, it implies that $m \neq m^*$. Similar to case 1, $\mathcal{S}$ can use the extractor Ext_2 to obtain

$$x_0 : F(x_0) = h^* \oplus H(m^*) \vee F(x_0) = y.$$

Hence, $\mathcal{S}$ either returns x_0 as the solution to $F^{-1}(y)$, or obtains

$$h^* = F(x_0) \oplus H(m^*) = F(\hat{x}_{2b}) \oplus H(m),$$

which contradicts the Assumption 1 since $m \neq m^*$. □

3.2 ECC-based Construction

We instantiate F in an ECC group $\mathbb{G}$ with prime order q and its generator is g. We define $F(x) = g^x$ for any $x \in \mathbb{Z}_q$. F is a one-way function if the DL assumption holds in $\mathbb{G}$. We can easily instantiate our generic construction with a DL-based OR proof.

To further simplify the assumption we need, we change $\oplus$ to the point addition operation. The scheme has to be modified slightly, by introducing some inverse operation.

The algorithm is as follows:

- $\mathsf{CHPG}(\lambda)$: On input the security parameter λ, it outputs public parameters $\mathsf{pp}_{\mathsf{ch}}$, including the elliptic curve group $\mathbb{G}$ of prime order q, a generator $g \in \mathbb{G}$, and collision-resistant hash function $H : \{0,1\}^* \to \mathbb{G}$ and $H' : \{0,1\}^* \to \mathbb{Z}_q$.
- $\mathsf{CHKG}(\mathsf{pp}_{\mathsf{ch}})$: It picks $x \in \mathbb{Z}_q$, and outputs $(\mathsf{sk}_{\mathsf{ch}} = x, \mathsf{pk}_{\mathsf{ch}} = g^x)$.
- $\mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m)$: It picks $\rho \in \mathbb{Z}_q$ and sets $h = g^\rho \cdot H(m)$. Then we compute a NIZK proof r by:
  (1) It picks $t_2, z_1 \in \mathbb{Z}_q$ and computes $T_2 = g^{t_2}, c_1 = H'(T_2, \mathsf{pk}_{\mathsf{ch}}, g^\rho, m)$, $T_1 = g^{z_1} \mathsf{pk}_{\mathsf{ch}}^{c_1}$.
  (2) It computes $c_2 = H'(T_1, \mathsf{pk}_{\mathsf{ch}}, g^\rho, m)$.
  (3) It computes $z_2 = t_2 - c_2\rho$. It returns $r = (z_1, z_2, c_1)$.
  It outputs (h, r).
- $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, h, r)$: It parses $r = (z_1, z_2, c_1)$. It computes $y' = h/H(m)$. It outputs 1 if

$$T_1 = g^{z_1} \mathsf{pk}_{\mathsf{ch}}^{c_1}, \quad c_2 = H'(T_1, \mathsf{pk}_{\mathsf{ch}}, y', m),$$
$$T_2 = g^{z_2} y'^{c_2}, \quad c_1 = H'(T_2, \mathsf{pk}_{\mathsf{ch}}, y', m).$$

  Otherwise, it returns 0.
- $\mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', h, r)$: It returns $\perp$ if $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, h, r) = 0$. Otherwise, it generates another NIZK proof r' with respect to $(\mathsf{pk}_{\mathsf{ch}}, y' = h/H(m'))$, using the secret key $\mathsf{sk}_{\mathsf{ch}} = x$ and the message m'.
  (1) It picks $t'_1, z'_2 \in \mathbb{Z}_q$ and computes $T'_1 = g^{t'_1}, c'_2 = H'(T'_1, \mathsf{pk}_{\mathsf{ch}}, y', m')$, $T'_2 = g^{z_2} y'^{c'_2}$.
  (2) It computes $c'_1 = H'(T'_2, \mathsf{pk}_{\mathsf{ch}}, y', m')$.
  (3) It computes $z'_1 = t'_1 - c'_1 x$. It returns $r' = (z'_1, z'_2, c'_1)$.
  It outputs r'.

Theorem 3.4. *Our ECC-based scheme is fully indistinguishable in the random oracle model (ROM).*

Theorem 3.5. *Our ECC-based scheme is fully collision-resistant if the DL assumption in $\mathbb{G}$ holds in the random oracle model.*

We show the security proofs in the Appendix B.

3.3 Lattice-based construction

We define q as an odd modulus and R_q as a ring $\mathbb{Z}_q[X]/(X^d+1)$ of dimension d. Define $\vec{I}_n$ as the identity matrix with size n, $\mathfrak{U}_k$ as a set of polynomials in $\mathbb{Z}[X]/(X^d+1)$ with infinity norm at most $k \in \mathbb{Z}^+$, and $\mathcal{U}$ as the uniform distribution. The Euclidean $\|\cdot\|$ and infinity $\|\cdot\|_\infty$ norms of a polynomial (or a vector of polynomials) are defined in the standard fashion w.r.t. the coefficient vector of the polynomial. The symbol $x \leftarrow_s \mathcal{X}$ means randomly picks x from $\mathcal{X}$. Define the following challenge space:

$$C = \{\, c \in \mathbb{Z}[X]/(X^d+1) \;:\; \|c\|_\infty = 1 \,\}. \tag{1}$$

Observe that $|C| = 3^d$. That is, for $d = 128$, we have $|C| = 3^{128} > 2^{202}$.

We review the hardness of Module-SIS (M-SIS) and Module-LWE (M-LWE) problems [17].

Definition 3.6 (M-SIS$_{n,m,q,\beta_{SIS}}$ Assumption). For all PPT adversaries $\mathcal{A}$, the probability

$$\Pr\left[\begin{array}{l|c} \vec{A}' \leftarrow_s \mathcal{U}(R_q^{n\times(m-n)}), & \vec{A}\vec{z} = \vec{0} \in R_q^n, \\ \vec{A} = [\vec{I}_n \| \vec{A}'], \vec{z} \leftarrow \mathcal{A}(\vec{A}) & 0 < \|\vec{z}\| \le \beta_{SIS} \end{array}\right]$$

is at most $\mathsf{negl}(\lambda)$.

Definition 3.7 (M-LWE$_{n,m,q,\chi}$ Assumption). Let χ be a distribution over R_q and $\vec{s} \leftarrow_s \chi^n$ be a secret key. Define LWE$_{q,s}$ as the distribution obtained by sampling $\vec{a} \leftarrow_s R_q^n, e \leftarrow_s \chi$ and outputting $(\vec{a}, \langle \vec{a}, \vec{s}\rangle + e)$. For all PPT adversaries $\mathcal{A}$, the probability of distinguishing between m samples from LWE$_{q,s}$ and $\mathcal{U}(R_q^n, R_q)$ is $\mathsf{negl}(\lambda)$.

The algorithm is as follows:

- CHPG(λ): On input the security parameter λ, it sets M-LWE parameters k, m, d, q, picks $\vec{G}' \leftarrow R_q^{k\times(m-k)}$, $\vec{G} = \left[\vec{I}_k \,\|\, \vec{G}'\right]$ and collision-resistant hash function $H : \{0,1\}^* \to R_q^k$ and $H' : \{0,1\}^* \to C$. It returns $\mathsf{pp_{ch}} = (k, m, d, q, \vec{G}, H, H')$
- CHKG($\mathsf{pp_{ch}}$): It picks $\vec{x} \leftarrow \mathfrak{U}_1^m$ and computes $\vec{y} = \vec{G}\cdot\vec{x}$. It outputs $(\mathsf{sk_{ch}} = \vec{x}, \mathsf{pk_{ch}} = \vec{y})$.
- CHash($\mathsf{pk_{ch}}, m$): It picks $\vec{\rho} \leftarrow \mathfrak{U}_1^m$ and computes $\vec{h} = \vec{G}\cdot\vec{\rho} + H(m)$. Then we compute a NIZK proof r by:
  (1) It picks $\vec{t_2} \leftarrow \mathfrak{U}_{md^2}^m$, $\vec{z_1} \leftarrow \mathfrak{U}_{md^2-d}^m$ and computes $\vec{T_2} = \vec{G}\cdot\vec{t_2}$, $c_1 = H'(\vec{T_2}, \mathsf{pk_{ch}}, \vec{G}\cdot\vec{\rho}, m)$, $\vec{T_1} = \vec{G}\cdot\vec{z_1} - \mathsf{pk_{ch}}\cdot c_1$.
  (2) It computes $c_2 = H'(\vec{T_1}, \mathsf{pk_{ch}}, \vec{G}\cdot\vec{\rho}, m)$.
  (3) It computes $\vec{z_2} = \vec{t_2} - c_2\vec{\rho}$. If $\|\vec{z_2}\|_\infty > md^2 - d$, restart from step 1. Else, it returns $r = (\vec{z_1}, \vec{z_2}, c_1)$.

  It outputs $(\vec{h}, r)$.
- CHCheck($\mathsf{pk_{ch}}, m, \vec{h}, r$): It parses $r = (\vec{z_1}, \vec{z_2}, c_1)$. It computes $\vec{y'} = \vec{h} - H(m)$. It outputs 1 if

$$\vec{T_1} = \vec{G}\cdot\vec{z_1} - \mathsf{pk_{ch}}\cdot c_1, \quad c_2 = H'(\vec{T_1}, \mathsf{pk_{ch}}, \vec{y'}, m),$$
$$\vec{T_2} = \vec{G}\cdot\vec{z_2} + \vec{y'}\cdot c_2, \quad c_1 = H'(\vec{T_2}, \mathsf{pk_{ch}}, \vec{y'}, m).$$

  and $\|\vec{z_1}\|_\infty \le md^2 - d$, $\|\vec{z_2}\|_\infty \le md^2 - d$. Otherwise, it returns 0.
- CHAdapt($\mathsf{sk_{ch}}, m, m', \vec{h}, r$): It returns $\perp$ if CHCheck($\mathsf{pk_{ch}}, m, \vec{h}, r$) = 0. Otherwise, it generates another NIZK proof r' with respect to $(\mathsf{pk_{ch}}, \vec{h} - H(m'))$, using the secret key $\mathsf{sk_{ch}} = \vec{x}$ and the message m'. It outputs r'.

Theorem 3.8. *Our lattice-based scheme is fully indistinguishable if the M-LWE$_{m-k,k,q,\mathfrak{U}_1}$ assumption holds in the random oracle model.*

Theorem 3.9. *Our lattice-based scheme is fully collision-resistant if the M-SIS$_{k,m+1,q,\beta_{\mathrm{SIS}}}$ assumption and the M-LWE$_{m-k,k,q,\mathfrak{U}_1}$ assumption hold in the random oracle model, where $\beta_{\mathrm{SIS}} \approx 4d\cdot(2md^2 + \sqrt{md})$.*

The proofs are given in the Appendix C.

3.4 Comparison

Apart from the comparison of IND and CollRes models in Table 1, we further compare various CHs in Table 2. We assume an ECC group $(\mathbb{G}, \mathbb{Z}_q)$, and RSA modulus $\mathbb{Z}_N$. Let $E_{\mathbb{G}}$ be an exponentiation on $\mathbb{G}$, and this notation also applies to $\mathbb{Z}_N$. Let E_{ct} denote exponentiation on an ABE ciphertext ct, and $|ct|$ denote its length. We refer the reader to the listed schemes for explanations on assumptions such as DL, DDH, OM-RSA, DLIN and the models like standard model and ROM. We put [12, 30] together as their efficiency and assumption are very similar. The RSA-based CH in [7] is similar to the tag-based CH in [4]. We do not put the tag-based CHs [4, 23] for comparison since the security model is different from the normal CH. According to Table 2, our ECC-based construction is the shortest one and the fastest one except the classical CH. We use the strongest model of full IND and full CollRes.

By following DSS20 [11], Derler et al. [10] introduced a new framework of chameleon hash scheme with full collision-resistance. The idea is to replace PKE used in [11] with a commitment scheme (e.g., Pedersen commitment). They also present a quantum-secure CH scheme using commitment and zero-knowledge proofs from learning parity with noise [20]. They use a ZK proof with hash output size = 3 bit. Hence it needs to repeat many times to achieve soundness error $\le 2^{-128}$. It implies that their ZK proof size is very large and impractical. In contrast, our lattice-based construction is very practical (4kB for the hash value h, and 15kB for r), and the running time is mainly a multiplication $M_{k\times m}$ of a matrix with size $k = 9, m = 20$ and a vector of length m. The size of parameters are discussed in the Appendix C.3.

Lastly, we implement our instantiations and present our source code at https://github.com/kychancf/CCTY24_cham_hash. We run our implementation in a Azure standard B2s virtual machines with 2vCPUs and 4GB RAM. We implement our ECC-based CH in Rust using the curve secp256k1 from the Zengo-X curv library and SHA256 for the hash function. The running time is 0.189ms for CHash, 0.202ms for CHCheck, and 0.161ms for CHAdapt (excluding the CHCheck part). We implement our lattice-based CH in Python3 using the library SymPy and SHA256 for the hash function. The running time is 2.413s for CHash, 1.617s for CHCheck, and 1.653s for CHAdapt (excluding the CHCheck part).

3.5 Multi-owner CH

In this subsection, we focus on our generic construction in the multi-owner setting. We assume that the scheme involves a creator and n owners. Specifically, the creator generates a chameleon hash, and any owner can adapt the chameleon hash multiple times. We only sketch the difference between our generic construction with a single owner.

Table 2: The comparison between different CHs.

Scheme	$\|h\|$	$\|r\|$	CHash	CHAdapt	Assumption	Model
[21]	$1\mathbb{G}$	$1\mathbb{Z}_q$	$2E_{\mathbb{G}}$	$0E_{\mathbb{G}}$	DL	Standard
[2]	$1\mathbb{G}$	$12\mathbb{G}+7\mathbb{Z}_q$	$17E_{\mathbb{G}}$	$14E_{\mathbb{G}}$	DDH	ROM
[7]	$1\mathbb{Z}_N$	$1\mathbb{Z}_N$	$1E_{\mathbb{Z}_N}$	$1E_{\mathbb{Z}_N}$	OM-RSA	ROM
[12, 30]	$3\mathbb{Z}_N+\|ct\|$	$1\mathbb{Z}_N$	$2E_{\mathbb{Z}_N}+E_{ct}$	$8E_{\mathbb{Z}_N}+E_{ct}$	DLIN	ROM
[11]	$2\mathbb{G}$	$4\mathbb{Z}_q$	$6E_{\mathbb{G}}$	$5E_{\mathbb{G}}$	DDH	ROM
[10]	$1\mathbb{G}$	$5\mathbb{Z}_q$	$6E_{\mathbb{G}}$	$4E_{\mathbb{G}}$	DL	ROM
Ours (1)	$1\mathbb{G}$	$3\mathbb{Z}_q$	$4E_{\mathbb{G}}$	$3E_{\mathbb{G}}$	DL	ROM
Ours (2)	$1R_q^k$	$2R_q^m + 1\|C\|$	$1M_{k\times m}$	$1M_{k\times m}$	M-SIS/LWE	ROM

- $\mathsf{CHash}((\mathsf{pk}_{\mathsf{ch}}^{(1)},\ldots,\mathsf{pk}_{\mathsf{ch}}^{(n)}),m)$: It picks a random ρ from the domain of F and calculates $h = F(\rho)\oplus H(m)$. It computes a zero-knowledge proof $r \leftarrow \mathsf{Pf}(\mathsf{crs}_{\Pi},(F,h\oplus H(m),\mathsf{pk}_{\mathsf{ch}}^{(1)},\ldots,\mathsf{pk}_{\mathsf{ch}}^{(n)}),\rho)$. It outputs (h,r). Note that r is a NIZK proof for ρ such that
$$F(\rho) = h \oplus H(m) \vee F(\rho) = \mathsf{pk}_{\mathsf{ch}}^{(1)} \vee \ldots \vee F(\rho) = \mathsf{pk}_{\mathsf{ch}}^{(n)}.$$
- $\mathsf{CHCheck}((\mathsf{pk}_{\mathsf{ch}}^{(1)},\ldots,\mathsf{pk}_{\mathsf{ch}}^{(n)}),m,h,r)$: It outputs $b \leftarrow \mathsf{Vfy}(\mathsf{crs}_{\Pi},(F,h\oplus H(m),\mathsf{pk}_{\mathsf{ch}}^{(1)},\ldots,\mathsf{pk}_{\mathsf{ch}}^{(n)}),r)$.
- $\mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}^{(i)},m,m',h,r)$: It returns $\perp$ if $\mathsf{CHCheck}((\mathsf{pk}_{\mathsf{ch}}^{(1)},\ldots,\mathsf{pk}_{\mathsf{ch}}^{(n)}),m,h,r)=0$. Otherwise, it outputs another zero-knowledge proof $r' \leftarrow \mathsf{Pf}(\mathsf{crs}_{\Pi},(F,h\oplus H(m),\mathsf{pk}_{\mathsf{ch}}^{(1)},\ldots,\mathsf{pk}_{\mathsf{ch}}^{(n)}),\mathsf{sk}_{\mathsf{ch}}^{(i)})$. Note that r' is a NIZK proof for $\mathsf{sk}_{\mathsf{ch}}^{(i)}$ such that
$$F(\mathsf{sk}_{\mathsf{ch}}^{(i)}) = h \oplus H(m') \vee F(\mathsf{sk}_{\mathsf{ch}}^{(i)}) = \mathsf{pk}_{\mathsf{ch}}^{(1)} \vee \ldots \vee F(\mathsf{sk}_{\mathsf{ch}}^{(i)}) = \mathsf{pk}_{\mathsf{ch}}^{(n)}.$$

The concrete construction can be constructed similarly using the one-out-of-many commitment proof [19], Bulletproof[6]-based OR proof [22, 35]. As compared to DSS20 [11] or DKSS20 [10], we can easily compress the size of r to $O(\log n)$. It could require additional assumptions from the new NIZK proof, such as the discrete logarithm relation assumption in Bulletproof[6].

4 CLAIMABILITY AND DENIABILITY FOR CH

As discussed in section 1.3, we define the creator as the party who generates the hash by running CHash, and the owners as the parties who can run the CHAdapt with their secret keys. These parties are the insider of the chameleon hash function. The existing security models of indistinguishability and collision-resistant only consider the attack from the third party, with the corruption of the owners' secret keys.

In this section, we study the ability of the insider, such as breaking the indistinguishability (IND) or collision-resistant (CollRes) property of his own, or the counterparty. We use different definitions of IND and CollRes to analyze the existing schemes, and our classification below is not limited to full security as some properties have conflicts.

4.1 Classification

We first classify the ability of the creator or the owners to claim or deny the generation of the hash value h or the randomness r. Examples are given based on the existing schemes and also our generic construction. Interestingly, the classical CH scheme and the RSA-based CH scheme fall into Level 0 for all cases. On the other hand, DSS20 [11] (and also DKSS20 [10]) may fall into different levels.

(1) Claimability of creation of h.

(1) Level 0: The creator cannot claim the creation of the hash value h. It is because any owner is able to generate the same claim.
   - e.g., the classical CH scheme $h = g^m\mathsf{pk}_{\mathsf{ch}}^r$ or RSA-based CH.
(2) Level 1: The creator can claim the creation of the hash value h, without disclosing m.
   - e.g., in DSS20, the creator can create a NIZK proof of knowing (m,ρ) such that $h = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ch}}}(m;\rho)$, where ρ represents an internal randomness.
(3) Level 2: The creator can claim that the original message is m given the hash value h. It also implies creation.
   - e.g., in DSS20, the creator can output m and create a NIZK proof of knowing ρ such that $h = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ch}}}(m;\rho)$.

(2) Claimability of hash output r.

(1) Level 0: The creator/the owners cannot claim the authorship of r. It is because the creator/the owners can generate the same claim.
   - e.g., the classical CH scheme or RSA-based CH.
(2) Level 1: The creator/the owner can claim the authorship of r.
   - e.g., in DSS20, the creator/the owner can output the internal randomness ρ used for generating the NIZK proof in r to claim the authorship.

(3) Deniability of hash output r.

(1) Level 0: The creator/the owner cannot deny the authorship of r.
   - e.g., the classical CH scheme or RSA-based CH.
(2) Level 1: The creator/the owner can deny the authorship of r.
   - No known construction achieves Level 1.

(4) Deniability of original message m.

(1) Level 0: The creator cannot deny that the original message is m, in case the owner outputs a valid hash output (h,r') for a message m' later.
   - e.g., the classical CH scheme or RSA-based CH.
(2) Level 1: The creator can deny that the original message is m, in case the owner outputs a valid hash output (h,r') for a message m' later.

- e.g., in DSS20, the creator can create a NIZK proof of knowing (m, ρ) such that $h = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ch}}}(m; \rho)$ and $h' = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ch}}}(m'; \rho)$. Everyone can check that $h \neq h'$.

4.2 New Definitions: (un)claimability and (un)deniability

Based on the classification described above, we present the formal definitions of (un)claimability and (un)deniability in the multi-party setting, which are inspired by [26].

4.2.1 Claimable chameleon hashes. Claimability requires that the creator/owner can claim the creation of h, the authorship of r, and even the original message m.

Definition 4.1 (Claimable chameleon hash). A claimable chameleon hash in the multi-party setting is a chameleon hash (as in Section 2.1) along with a pair of algorithms $(\mathsf{Claim}, \mathsf{VerClaim})$.

- Claim: It takes a set of public keys $\mathsf{pk} = \{\mathsf{pk}_{\mathsf{ch}}^{(1)}, \cdots \mathsf{pk}_{\mathsf{ch}}^{(n)}\}$, a secret key $\mathsf{sk}_{\mathsf{ch}}^{(i)}$, a hash value h, a message m and a randomness r as input, outputs a claim ζ. The claim ζ indicates the claimability of any values in (h, m, r).
- $\mathsf{VerClaim}$: It takes a set of public keys pk, a hash value h, a message m, a randomness r, a claim ζ and a public key $\mathsf{pk}_{\mathsf{ch}}^{(i)}$ as input, outputs a bit $b = \{0, 1\}$, indicating that whether or not ζ is a valid claim of (h, m, r) for public key $\mathsf{pk}_{\mathsf{ch}}^{(i)}$.

Definition 4.2 (Claim Oracle OClaim*).* The oracle OClaim takes an index $i \in [1, n]$, a public-key set pk, and a tuple (h, m, r) as input, outputs $\mathsf{Claim}(\mathsf{pk} \bigcup \{\mathsf{pk}_{\mathsf{ch}}^{(i)}\}, \mathsf{sk}_{\mathsf{ch}}^{(i)}, h, m, r)$. The oracle takes (pk, h, m, r) as input when it is invoked with a single key pair.

We define the oracle $\mathsf{OClaim}(\cdots)$ to output $\perp$ if adversary queries the challenge tuple (h^*, m^*, r^*). Otherwise, the oracle outputs the same response as $\mathsf{OClaim}(\cdots)$. Claimability requires that: (1) honest parties can claim the creation of h or the authorship of r on a message m; (2) adversarial parties cannot claim the creation of h or the authorship of r that they did not produce on a message m; and (3) adversarial parties cannot produce the creation of h or the authorship of r on a message m along with a claim ζ that appears to be produced by an honest party.

Definition 4.3 (Claimability). A chameleon hash is claimable if equipped with Claim and $\mathsf{VerClaim}$ such that the following equations hold

- Equation (1). There exists a negligible function ε such that for any $(\mathsf{sk}_{\mathsf{ch}}^{(1)}, \mathsf{pk}_{\mathsf{ch}}^{(1)}), \cdots, (\mathsf{sk}_{\mathsf{ch}}^{(n)}, \mathsf{pk}_{\mathsf{ch}}^{(n)}) \leftarrow \mathsf{CHKG}(\mathsf{pp}_{\mathsf{ch}}, n), (\mathsf{pp}_{\mathsf{ch}}, n) \leftarrow \mathsf{CHPG}(\lambda)$ and any $i \in [1, n]$, it holds for any message m that
$$\begin{aligned}&\Pr[(h, r) \leftarrow \mathsf{CHash}(\mathsf{pk}, m) :\\ &\mathsf{VerClaim}(\mathsf{pk}, \mathsf{pk}_{\mathsf{ch}}^{(i)}, h, m, r, \mathsf{Claim}(\mathsf{pk}, \mathsf{sk}_{\mathsf{ch}}^{(i)}, h, m, r)) = 1]\\ &> 1 - \varepsilon(\lambda).\end{aligned}$$
where $\mathsf{sk} = \{\mathsf{sk}_{\mathsf{ch}}^{(1)}, \cdots \mathsf{sk}_{\mathsf{ch}}^{(n)}\}$ and $\mathsf{pk} = \{\mathsf{pk}_{\mathsf{ch}}^{(1)} \cdots, \mathsf{pk}_{\mathsf{ch}}^{(n)}\}$.
- Equation (2) & (3). The advantage of $\mathcal{A}$ illustrated in Figure 5 and 6 is negligible in λ, respectively. Q is the set of queries made to oracle $\mathsf{OClaim}(\cdots)$, and $\mathsf{pk}_{\mathsf{ch}}^{(i)}$ is an honest party in Figure 6.

4.2.2 Unclaimable chameleon hashes. Unclaimability requires that the creator/owners cannot convince anyone of its identity. That is, for any function that a party can produce an internal randomness and a chameleon secret key, another party can compute an indistinguishable function.

Definition 4.4 (Unclaimable chameleon hashes). An unclaimable chameleon hash in the multi-party setting is a chameleon hash (as in Section 2.1) along with an algorithm $\mathsf{ExtRand}$. The algorithm $\mathsf{ExtRand}$ takes a set of public keys pk, a secret key $\mathsf{sk}_{\mathsf{ch}}^{(i)}$, a hash value h, a message m and a randomness r as input, outputs an internal randomness ρ if $\mathsf{sk}_{\mathsf{ch}}^{(i)}$ is one of secret keys for pk and $\mathsf{CHCheck}(\mathsf{pk}, h, m, r) = 1$.

$\mathsf{ExtRand}$ satisfies *computational unclaimability*. Let R be the distribution of NIZK internal randomness. For any n, there exists a negligible function ε such that the following condition holds. Let $(\mathsf{pk}_{\mathsf{ch}}^{(i)}, \mathsf{sk}_{\mathsf{ch}}^{(i)}), (\mathsf{pk}_{\mathsf{ch}}^{(j)}, \mathsf{sk}_{\mathsf{ch}}^{(j)}) \leftarrow \mathsf{CHKG}(\mathsf{pp}_{\mathsf{ch}}, n), (\mathsf{pp}_{\mathsf{ch}}, n) \leftarrow \mathsf{CHPG}(\lambda)$, where $i \neq j$. For any message m and any $(\mathsf{sk}_{\mathsf{ch}}^{(1)}, \mathsf{pk}_{\mathsf{ch}}^{(1)}) \cdots, (\mathsf{sk}_{\mathsf{ch}}^{(n)}, \mathsf{pk}_{\mathsf{ch}}^{(n)})$, let $\mathsf{pk} = \{\mathsf{pk}_{\mathsf{ch}}^{(1)}) \cdots, \mathsf{pk}_{\mathsf{ch}}^{(n)}\}$ and $\mathsf{S} = \{(i, \mathsf{sk}_{\mathsf{ch}}^{(i)}, \mathsf{pk}_{\mathsf{ch}}^{(i)})\}_{i \in n}$. Let $\rho_i \leftarrow \mathsf{R}$, $(h_i, r_i) \leftarrow \mathsf{CHash}(\mathsf{pk}, m; \mathsf{sk}_{\mathsf{ch}}^{(i)}, \rho_i)$, and $\rho_i \leftarrow \mathsf{ExtRand}(\mathsf{pk}, \mathsf{sk}_{\mathsf{ch}}^{(j)}, h_i, r_i, m)$. Let $\rho_j \leftarrow \mathsf{R}$ and $(h_j, r_j) \leftarrow \mathsf{CHash}(\mathsf{pk}, m; \mathsf{sk}_{\mathsf{ch}}^{(j)}, \rho_j)$. Then
$$(\mathsf{S}, \rho_i, h_i, r_i) \approx_c (\mathsf{S}, \rho_j, h_j, r_j).$$

4.2.3 Deniable chameleon hashes. Deniability requires that the creator/owner can deny the authorship of r, and even the original message m.

Definition 4.5 (Deniable chameleon hash). A deniable chameleon hash in the multi-party setting is a chameleon hash (as in Section 2.1) along with a pair of algorithms $(\mathsf{Deny}, \mathsf{VerDeny})$.

- Deny: It takes a set of public keys $\mathsf{pk} = \{\mathsf{pk}_{\mathsf{ch}}^{(1)}, \cdots \mathsf{pk}_{\mathsf{ch}}^{(n)}\}$, a secret key $\mathsf{sk}_{\mathsf{ch}}^{(i)}$, a hash value h, a message m and a randomness r as input, outputs a denial ϑ. The denial ϑ indicates the deniability of m or r, even both of them.
- $\mathsf{VerDeny}$: It takes a set of public keys pk, a hash value h, a message m, a randomness r, a denial ϑ and a public key $\mathsf{pk}_{\mathsf{ch}}^{(i)}$ as input, outputs a bit $b = \{0, 1\}$, indicating that whether or not ϑ is a valid denial of (m, r) for public key $\mathsf{pk}_{\mathsf{ch}}^{(i)}$.

Definition 4.6 (Deny Oracle ODeny*).* The oracle ODeny takes an index $i \in [1, n]$, a public-key set pk, and a tuple (h, m, r) as input, outputs $\mathsf{Deny}(\mathsf{pk} \bigcup \{\mathsf{pk}_{\mathsf{ch}}^{(i)}\}, \mathsf{sk}_{\mathsf{ch}}^{(i)}, h, m, r)$. The oracle takes (pk, h, m, r) as input when it is invoked with a single key pair.

We define the oracle $\mathsf{ODeny}(\cdots)$ to output $\perp$ if adversary queries the challenge values (pk^*, h^*). Otherwise, the oracle outputs the same response as $\mathsf{ODeny}(\cdots)$. Deniability requires that: (1) honest parties who did not produce message m and/or randomness r can deny; and (2) adversarial parties possessing a *subset* of secret keys cannot deny (m, r) under all public keys in a set. Equation (2) captured the following cases: 1) If a party generates all keys in a set, s/he may produce denials on $\{(m, r)\}$ under every public key in the set. Given denials under every public key in a set, anyone can infer that all keys in the set were generated dishonestly and

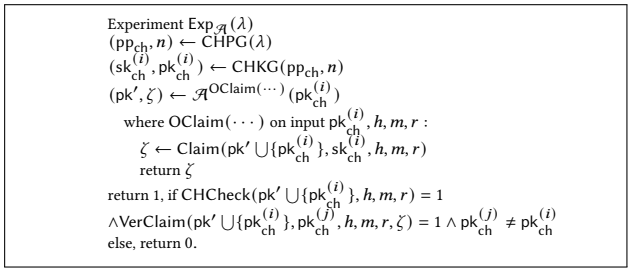

Figure 5: Equation (2) for Claimability.

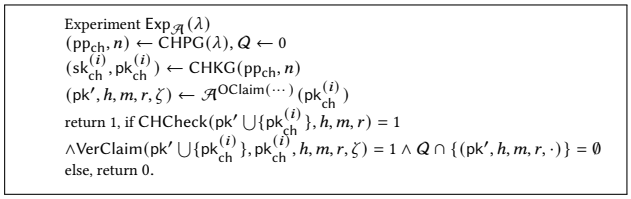

Figure 6: Equation (3) for Claimability.

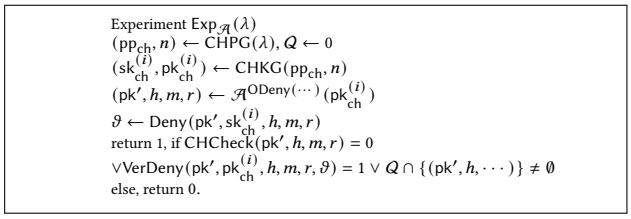

Figure 7: Equation (1) for Deniability.

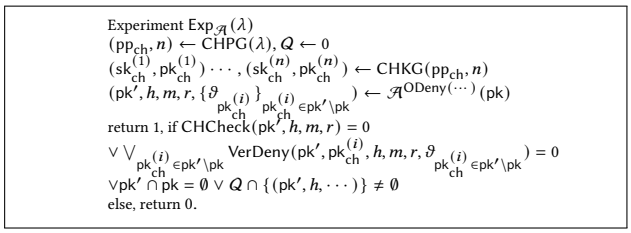

Figure 8: Equation (2) for Deniability.

all parties in the set colluded to produce each tuple (h, m, r) under that set. 2) If there exist denials for a *subset* of public keys in a set, then anyone can infer that *either* one of the remaining public keys in the set (i.e., $\mathsf{pk}_{\mathsf{ch}}^{(i)} \in \mathsf{pk}' \backslash \mathsf{pk}$ in Fig 8) produced the values (m, r) *or* all of the remaining public keys colluded to produce it.

Definition 4.7 (Deniability). A chameleon hash is deniable if the advantage of $\mathcal{A}$ described in Figure 7 and 8 is non-negligible in λ, i.e., $1 - \varepsilon(\lambda)$. $\mathcal{Q}$ is the set of chameleon hash tuples involved in the experiment. $\mathsf{pk}_{\mathsf{ch}}^{(i)} \in \mathsf{pk}' \backslash \mathsf{pk}$ indicates the parties cannot deny (m, r) in Fig 8.

4.2.4 *Undeniable chameleon hashes.*

Claim 1. *A chameleon hash in the multi-party setting is undeniable if it satisfies full indistinguishability.*

We provide two justifications. First, we show that deniable chameleon hashes cannot satisfy full indistinguishability. If an attacker obtains all secret keys in the generation of r, the attacker can produce denial using each secret key. As a result, the attacker can identify the creator of r, as there exists one secret key for which the Deny algorithm could not produce a valid denial ϑ (i.e., adversarial parties cannot deny under all public keys in a set, as in Figure 8).

Second, we show that fully indistinguishable chameleon hashes can imply undeniability. We consider a scenario where an owner Bob should behave indistinguishably from a creator Alice, but the attackers cannot identify the creation of r given their secret keys. That is, for any protocol that Bob could execute with respect to a randomness r and his public key $\mathsf{pk}'_{\mathsf{ch}}$, Alice should engage in the same protocol with respect to her own public key $\mathsf{pk}_{\mathsf{ch}}$ and behave indistinguishably from Bob. In this scenario, we require that if Bob's secret key $\mathsf{sk}'_{\mathsf{ch}}$ is compromised, the attacker cannot decide whether r was produced by Bob or by someone else. Besides, if Bob lends his key to someone else who used it to produce r, Bob could not decide as well. The definition of full indistinguishability represents this scenario: attackers cannot produce a valid denial ϑ of r even if they access all secret keys corresponding r.

Remark. We have so far shown full indistinguishability, full collision-resistance, (un)claimability and (un)deniablity for chameleon hashes. They are not entirely separate properties: one might imply the other (e.g., full indistinguishability implies undeniability). On the other hand, some properties may have conflicts. For example, one cannot create a deniable CH with full indistinguishability. The natural question is whether one can create a secure chameleon hash with a (sub)set of these properties. First, the combination of these properties could be various depending on the application scenarios. Second, it would be a nice conclusion if we could find the relationships among these properties. We leave it for future research, as our main focus in this work is full indistinguishability, full collision-resistance and multi-party setting.

4.3 Our Constructions

4.3.1 *Claimability of creation of h.*

Our scheme is level 2, since the function F is one-way. Also, we do not have an efficient NIZK for the hash function, so no efficient level 1 (theoretically we can use zk-SNARK).

4.3.2 *Claimability of hash output r.*

Our generic construction is flexible for using different NIZK proofs. We can have a Level 0 or Level 1 construction.

Level 0 Construction. We give the multi-owner construction using (uncompressed) Dual Ring [34] to construct CH with unclaimable hash output.

- $\mathsf{CHash}((\mathsf{pk}_{\mathsf{ch}}^{(1)}, \ldots, \mathsf{pk}_{\mathsf{ch}}^{(n)}), m)$: It picks random $\rho \in \mathbb{Z}_q$ and sets $h = g^{\rho} \cdot H(m)$. Then we compute a NIZK proof r by:
  (1) It picks random $t, c_1, \ldots, c_n \in \mathbb{Z}_q$ and computes $T = g^t \prod_{i=1}^{n} (\mathsf{pk}_{\mathsf{ch}}^{(i)})^{-c_i}$.
  (2) It computes $c_0 = H'(T, g^{\rho}, \mathsf{pk}_{\mathsf{ch}}^{(1)}, \ldots, \mathsf{pk}_{\mathsf{ch}}^{(n)}, m) - \sum_{i=1}^{n} c_i$.
  (3) It computes $z = t + c_0 \rho$, and sets $r = (z, c_0, c_1, \ldots, c_n)$.
  It outputs (h, r).

- $\mathsf{CHCheck}((\mathsf{pk}_{\mathsf{ch}}^{(1)}, \ldots, \mathsf{pk}_{\mathsf{ch}}^{(n)}), m, h, r)$: It parses $r = (z, c_0, c_1, \ldots, c_n)$. It computes $y' = h/H(m)$. It outputs 1 if

$$c_0 + \ldots + c_n = H'(g^z y'^{-c_0} \prod_{i=1}^{n} (\mathsf{pk}_{\mathsf{ch}}^{(i)})^{-c_i}, y', \mathsf{pk}_{\mathsf{ch}}^{(1)}, \ldots, \mathsf{pk}_{\mathsf{ch}}^{(n)}, m).$$

Otherwise, it returns 0.

Proof of unclaimability of hash output. For any hash output h^* and $r^* = (z^*, c_1^*, \ldots, c_n^*)$, any owner $\mathsf{sk}_{\mathsf{ch}}^{(i)}$ can claims that he generated this pair (h^*, r^*) by reconstructing the value $t_i = z^* - c_i^* \mathsf{sk}_{\mathsf{ch}}^{(i)}$. The creator can also compute $t_0 = z^* - c_0^* \rho$. The values $t_0, t_1, \ldots, t_n$ can be used to claim the authorship of (h^*, r^*). Hence this scheme is unclaimable.

Level 1 Construction. We use the one-out-of-many commitment proof to give a level 1 construction.

Assume that the public parameters include another generator $\hat{g} \in \mathbb{G}$. Define a commitment scheme for a message $m \in \mathbb{Z}_q$ and randomness $\rho \in \mathbb{Z}_q$ as

$$\mathsf{Com}(m; \rho) = \hat{g}^m g^\rho.$$

In our ECC-based construction, all public keys can be viewed as a commitment of zero: $\mathsf{Com}(0; \mathsf{sk}_{\mathsf{ch}})$. We utilize the one-out-of-many commitment proof to generate r in the chameleon hash. Without loss of generality (WLOG), assume that we want to compute r by using $\mathsf{sk}_{\mathsf{ch}}^{(j)}$. Then we compute a zero-knowledge proof for $\mathsf{sk}_{\mathsf{ch}}^{(j)}$ such that

$$\mathsf{Com}(0; \mathsf{sk}_{\mathsf{ch}}^{(j)}) = \mathsf{pk}_{\mathsf{ch}}^{(j)} \wedge j \in [0, n].$$

In the one-out-of-many proof [19], they express j as a binary number $(\ell_1, \ldots, \ell_{\log n})$ and compute commitment $\mathsf{Com}(\ell_i; \rho_i)$ using randomness ρ_i for $i \in [1, \log n]$. A similar idea applies to the Bulletproof-based OR proof [22, 35].

Proof of claimability of hash output. Since Com is binding, only the author can output a valid pair $(\ell_i; \rho_i)$. Anyone can validate $(\ell_i; \rho_i)$, reconstruct j from $(\ell_1, \ldots, \ell_{\log n})$ and find the index of the author.

4.3.3 Deniability of hash output r. All of the existing constructions are Level 0 in deniability, including our ECC-based construction.

Level 1 Construction. We now give a Level 1, multi-owner construction, inspired from linkable ring signature [24]. Assume that there is another collision-resistant hash function $\hat{H} : \{0,1\}^* \to \mathbb{G}$ in the public parameters.

We also utilize the one-out-of-many commitment proof to generate a zero-knowledge proof π in the chameleon hash. In addition, the creator/owner has to compute an extra value $Z = \hat{H}(m)^{\mathsf{sk}_{\mathsf{ch}}^{(j)}}$. WLOG, we compute a zero-knowledge proof π for $\mathsf{sk}_{\mathsf{ch}}^{(j)}$ such that

$$\mathsf{Com}(0; \mathsf{sk}_{\mathsf{ch}}^{(j)}) = \mathsf{pk}_{\mathsf{ch}}^{(j)} \wedge Z = \hat{H}(m)^{\mathsf{sk}_{\mathsf{ch}}^{(j)}} \wedge j \in [0, n].$$

The hash output is $(h, r = (\pi, Z))$.

Observe that the value Z will not compromise the *indistinguishability* if the DDH assumption holds in $\mathbb{G}$. This level of indistinguishability assumes attackers cannot access creator/owner's secret keys.

Proof of deniability of hash output. Suppose that $(h, r = (\pi, Z))$ is generated by $\mathsf{sk}_{\mathsf{ch}}^{(j)}$ for some $j \in [0, n]$. For party i who wants to deny generating it (where $i \neq j$), he can compute a zero-knowledge proof π for $\mathsf{sk}_{\mathsf{ch}}^{(i)}$ such that:

$$\hat{H}(m)^{\mathsf{sk}_{\mathsf{ch}}^{(i)}} \neq Z \wedge \mathsf{pk}_{\mathsf{ch}}^{(i)} = g^{\mathsf{sk}_{\mathsf{ch}}^{(i)}}.$$

The zero-knowledge proof of inequality of discrete logarithm can be efficiently instantiated.

5 APPLICATION: REDACTABLE BLOCKCHAINS

We consider redactable blockchain as an application scenario. Such a blockchain allows some after-the-fact modifications of its content. This is motivated by the illicit content that might be included in the Bitcoin blockchain, which poses a significant challenge for law enforcement agencies like INTERPOL and legislations like GDPR (General Data Protection Regulation). The existing solutions may either work for the permissioned [12] or the permissionless setting [14], use cryptographic [2] or non-cryptographic tool [14], allow block-level [2] or transaction-level rewriting [12]. This work relies on a cryptographic tool (i.e., a chameleon hash function) to redact mutable transactions in the permissionless setting. We do not consider block insertions and removals as described in [15].

Full indistinguishability and full collision-resistance provide enhanced privacy and security guarantees to the redactable blockchains, as discussed in [11, 30]. Here, we justify the multi-party setting, (un)claimability and (un)deniability. Multiple party setting is more desirable for redactable blockchains, especially considering single-point of failure towards a trapdoor holder. Chameleon hashes in the multi-party setting have been studied in the literature. For example, a threshold number of parties (or miners) are allowed to execute a multi-party computation protocol to redact blocks (or transactions) [2, 28]. Also, the redactable blockchains based on policies [12, 32] allow multiple authorized parties to rewrite the mutable transactions.

Claimability allows an owner to prove the authorship of a modified transaction at a later date, especially in the case of rewriting a mutable transaction containing illicit content to a benign one. Deniability implies that the creator who created a mutable transaction containing illicit content may deny it. The owner also wants to deny the malicious rewriting by one of his members. For example, the owner may suffer serious consequences through no fault of his own, or due to the creator adversarially trying to damage his reputation. Malicious rewriting means either adding illicit or malicious content to mutable transactions, or deleting legitimate or benign mutable transactions from the blockchain. Unclaimability and undeniability are also useful properties. For example, an authority (e.g., an authoritarian government) may coerce the creator/owners to provide proof of authorship or denial for a maliciously written transaction, the provable inability to do so is desired. Undeniability may also be desirable in accountable redactable blockchains [32].

Structures. We consider two types of changes are required in redactable blockchains: block-level and transaction-level. First, we follow the notation used in [2, 11], and describe a block in Bitcoin as $B = \langle s, x, ctr \rangle$, where $s \in \{0,1\}^\lambda$ contains the block header (minus the nonce), $x \in \{0,1\}^*$ contains all the transactions inside a block,

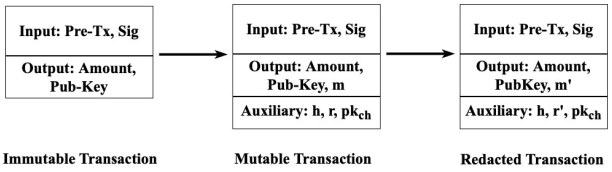

Figure 9: Changes to transactions.

and $ctr \in \mathbb{N}$ denotes a nonce value, and a block is valid if

$$\mathbf{validblock}_q^D(B) := \quad H(ctr, G(s,x)) < D \wedge ctr < q.$$

Here (H, G) are two collision-resistant hash functions. To make a block mutable, we change the description of the block to $B = \langle s, x, ctr, (h, r, \underline{H'(\mathsf{pk}_{\mathsf{ch}})})\rangle$, where the new component is a chameleon hash $(h, r, \underline{H'(\mathsf{pk}_{\mathsf{ch}})})$ with a public key $\mathsf{pk}_{\mathsf{ch}}$, and H' denotes another collision-resistant hash function. The validation predicate changes to

$$\begin{aligned}\mathbf{validblock}_q^D(B) := \quad & H(ctr, h, \underline{H'(\mathsf{pk}_{\mathsf{ch}})}) < D \wedge ctr < q \\ & \wedge \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (s,x), h, r) = 1.\end{aligned}$$

Note that the underlined part is used to prevent the key replacement attack. For rewriting a block, a key holder may change (s, x) to (s', x'), and update r to r'.

Second, we show the modifications to a Bitcoin transaction, and we list immutable, mutable and redacted transactions in Figure 9. The key difference between an immutable transaction and a mutable one is an auxiliary field. This auxiliary field is used to verify the chameleon hash only. Its size is constant due to the hashing of multiple chameleon public keys (i.e., $H'(\mathsf{pk}_{\mathsf{ch}})$). For rewriting a chameleon-hashed message m, a key holder changes m to m', and updates r to r'.

Modification Process. First, we need to clarify the redacted content inside a mutable block or transaction. For rewriting a block, the key holder can rewrite s or x, or both [2, 11]. But, we argue that it is more reasonable to rewrite x. This is because rewriting some content in a block header s may break some rules such as fork. Thus, we focus on redacting some (illicit) content inside a mutable transaction like [14]. We consider two cases in rewriting a mutable transaction. For a Bitcoin transaction, we slightly modify its structure to make it mutable. The Bitcoin transaction's structure mainly includes two fields: 1) the input contains a hash of the previous transaction (Pre-Tx), and a signature (Sig); 2) the output contains an amount, and a public key (Pub-Key) which is used to verify a (redeemer's) signature on a transaction. We add a message m (i.e., arbitrary data) to the output field to create a mutable Bitcoin transaction. For transactions in Ethereum, a submitted (regular) transaction contains similar fields as in Bitcoin, except an optional field including arbitrary data [18]. Like most redactable blockchain solutions, we do not allow anyone to change transaction amount, public key (or address), otherwise it may cause transaction inconsistency [14, 28].

Third, we assume the message m as illicit content, and we take the mutable transaction recording message m in Figure 9 to explain the modification process. Since the message m is hashed under a set of public keys $\{\mathsf{pk}_{\mathsf{ch}}\}$, it can be modified by one of the secret key holders. Thus, a key holder can redact the mutable transaction by replacing (m, r) with (m', r'). After modification, the key holder broadcasts (m', r'), and *id* (helps other parties to identify which transaction needs to be updated) in the blockchain network, each party will perform the following checks.

- whether the message m needs to be modified as m' according to some rules like GDPR.
- whether the message-randomness pairs (m, r) and (m', r') are mapping to the same chameleon hash value h and they are valid under a set of public keys $\{\mathsf{pk}_{\mathsf{ch}}\}$.

If the above requirements are met, all parties are required to update their local copy of the blockchain by replacing (m, r) with (m', r'). Based on Figure 9, each mutable transaction additionally contains one element in the output field, and three elements in the auxiliary field. Considering our concrete ECC-based CH instantiation, the elements in the auxiliary field are 162 bytes. The running time for hashing and verifying are both around 0.6s, which is quite practical.

6 CONCLUSION

In this paper, we proposed a new generic construction of the chameleon hash function supporting enhanced security and usability guarantees. The proposed scheme achieved full indistinguishability, full collision-resistance, and multi-party setting simultaneously with minimal extra overhead. We also discussed (un)claimability and (un)deniability properties for chameleon hashes to satisfy broader chameleon hash-based applications. Lastly, we presented practical ECC and quantum-secure instantiations, and our implementation of these two instantiations showed that they are suitable for real-life applications such as redactable blockchains.

ACKNOWLEDGEMENT

This work is supported by the EU's research and innovation program: 101019645 (SECANT) and 101095634 (ENTRUST). These projects are funded by the UK government Horizon Europe guarantee and administered by UKRI. Yangguang Tian is partially supported by the National Natural Science Foundation of China under Grant No. 62072371 and 61872264.

REFERENCES

[1] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In S. d. C. di Vimercati, P. Syverson, and D. Gollmann, editors, *Computer Security – ESORICS 2005*, pages 159–177, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[2] G. Ateniese, B. Magri, D. Venturi, and E. Andrade. Redactable blockchain – or – rewriting history in bitcoin and friends. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 111–126, 2017.

[3] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 390–399, New York, NY, USA, 2006. Association for Computing Machinery.

[4] C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of sanitizable signatures revisited. In S. Jarecki and G. Tsudik, editors, *Public Key Cryptography – PKC 2009*, pages 317–336, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[5] C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of sanitizable signatures. In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography – PKC 2010*, pages 444–461, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[6] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[7] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Chameleon-hashes with ephemeral trapdoors. In S. Fehr, editor, *Public-Key*

Cryptography – PKC 2017, pages 152–182, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.
[8] S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In A. Mitrokotsa and S. Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012*, pages 35–52, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
[9] S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In S. M. Bellovin, R. Gennaro, A. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security*, pages 258–276, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
[10] D. Derler, S. Krenn, K. Samelin, and D. Slamanig. Fully collision-resistant chameleon-hashes from simpler and post-quantum assumptions. In C. Galdi and V. Kolesnikov, editors, *Security and Cryptography for Networks*, pages 427–447, Cham, 2020. Springer International Publishing.
[11] D. Derler, K. Samelin, and D. Slamanig. Bringing order to chaos: The case of collision-resistant chameleon-hashes. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 462–492, Cham, 2020. Springer International Publishing.
[12] D. Derler, K. Samelin, D. Slamanig, and C. Striecks. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. In *NDSS*, 2019.
[13] D. Derler and D. Slamanig. Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. *Designs, Codes and Cryptography*, 87(6):1373–1413, 2019.
[14] D. Deuber, B. Magri, and S. A. K. Thyagarajan. Redactable blockchain in the permissionless setting. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 124–138, 2019.
[15] M. S. Dousti and A. Küpçü. Tri-op redactable blockchains with block modification, removal, and insertion. *Turkish Journal of Electrical Engineering and Computer Sciences*, 30(2):376–391, 2022.
[16] M. F. Esgin. *Practice-Oriented Techniques in Lattice-Based Cryptography*. PhD thesis, Monash University, 5 2020.
[17] M. F. Esgin, R. Steinfeld, J. K. Liu, and D. Liu. Lattice-based zero-knowledge proofs: New techniques for shorter and faster constructions and applications. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 115–146, Cham, 2019. Springer International Publishing.
[18] ethereum.org. Transactions, 2023. https://ethereum.org/en/developers/docs/transactions/.
[19] J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 253–280, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
[20] A. Jain, S. Krenn, K. Pietrzak, and A. Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 663–680, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
[21] H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS*, 2000.
[22] R. W. F. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang. Omniring: Scaling private payments without trusted setup. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 31–48, New York, NY, USA, 2019. Association for Computing Machinery.
[23] Y. Li and S. Liu. Tagged chameleon hash from lattice and application to redactable blockchain. *Cryptology ePrint Archive*, 2023.
[24] X. Lu, M. H. Au, and Z. Zhang. Raptor: A practical lattice-based (linkable) ring signature. In R. H. Deng, V. Gauthier-Umaña, M. Ochoa, and M. Yung, editors, *Applied Cryptography and Network Security*, pages 110–130, Cham, 2019. Springer International Publishing.
[25] V. Lyubashevsky. Lattice signatures without trapdoors. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 738–755, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
[26] S. Park and A. Sealfon. It wasn't me! repudiability and claimability of ring signatures. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 159–190, Cham, 2019. Springer International Publishing.
[27] B. Poettering and D. Stebila. Double-authentication-preventing signatures. In M. Kutyłowski and J. Vaidya, editors, *Computer Security - ESORICS 2014*, pages 436–453, Cham, 2014. Springer International Publishing.
[28] I. Puddu, A. Dmitrienko, and S. Capkun. μ chain: How to forget without hard forks. Cryptology ePrint Archive, Paper 2017/106, 2017.
[29] T. Ruffing, A. Kate, and D. Schröder. Liar, liar, coins on fire! penalizing equivocation by loss of bitcoins. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 219–230, New York, NY, USA, 2015. Association for Computing Machinery.
[30] K. Samelin and D. Slamanig. Policy-based sanitizable signatures. In S. Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 538–563, Cham, 2020. Springer International Publishing.
[31] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In J. Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 355–367, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
[32] Y. Tian, N. Li, Y. Li, P. Szalachowski, and J. Zhou. Policy-based chameleon hash for blockchain rewriting with black-box accountability. In *Annual Computer Security Applications Conference*, ACSAC '20, page 813–828, New York, NY, USA, 2020. Association for Computing Machinery.
[33] B. Yu, S. K. Kermanshahi, A. Sakzad, and S. Nepal. Chameleon hash time-lock contract for privacy preserving payment channel networks. In R. Steinfeld and T. H. Yuen, editors, *Provable Security*, pages 303–318, Cham, 2019. Springer International Publishing.
[34] T. H. Yuen, M. F. Esgin, J. K. Liu, M. H. Au, and Z. Ding. Dualring: Generic construction of ring signatures with efficient instantiations. In T. Malkin and C. Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 251–281, Cham, 2021. Springer International Publishing.
[35] T. H. Yuen, S.-F. Sun, J. K. Liu, M. H. Au, M. F. Esgin, Q. Zhang, and D. Gu. Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security. In J. Bonneau and N. Heninger, editors, *Financial Cryptography and Data Security*, pages 464–483, Cham, 2020. Springer International Publishing.

A SECURITY IN THE MULTI-KEY SETTING

In this section, we show our key replacement attacks for concrete schemes SS20 [30] and DSS20 [11], and provide a fix.

A.1 Key Replacement Attack on SS20

Suppose that the RSA public key is (N, e). The RSA-based chameleon hash in SS20 [30] is

$$h = H(m) \cdot r^e \mod N,$$

for some randomness r. Suppose (N', e') and d' is the RSA key pair of the attacker. For any message m', he finds r' such that:

$$r' = (h/H(m'))^{d'} \mod N'.$$

It is easy to see that (m', r') is a collision of h with respect to (N', e').

A.2 Key Replacement Attack on DSS20

Suppose that the ElGamal public key is y. The chameleon hash in DSS20 [11] is:

$$h = (C_1 = m \cdot y^\rho, \quad C_2 = g^\rho).$$

Let (x', y') as another key pair of the attacker. The attacker can calculate

$$m' = C_1 C_2^{-x'}.$$

Then we can write $m' \cdot (y')^\rho = C_1 C_2^{-x'} \cdot g^{x'\rho} = C_1$. Hence h is the ElGamal encryption of m' with respect to y'. With the knowledge of x', the attacker can generate a valid NIZK proof of r'. Hence (m', r') is a collision for h with respect to y'.

A.3 Solution to the Key Replacement Attack

Our solution is to change the hash value as $(h, H'(\mathsf{pk}_{\mathsf{ch}}))$, for some collision-resistant hash function H'. In the multi-owner setting, we change the hash value as $(h, H'(\mathsf{pk}))$, where pk contains a creator and multiple owners. We also show how to apply such modification to redactable blockchains in Section 5. We stress that the owners' public keys must be *authenticated* by a certificate authority. The creator must use the *authenticated* public keys to create $\mathsf{H}'(\mathsf{pk})$, and anyone is supposed to check $\mathsf{H}'(\mathsf{pk})$ during $\mathsf{CHCheck}$. This additional check is needed for a secure chameleon hash function.

B SECURITY PROOF OF THE ECC-BASED CONSTRUCTION

To prove Theorem 3.4 and 3.5, we first show that the ECC-based NIZK proof has the desired properties.

Lemma B.1. *The ECC-based NIZK proof has zero-knowledge and simulation-sound extractability in the random oracle model.*

Proof. **Simulation-sound extractability.** Recall that the relation R of the NIZK proof is to know x such that $y_1 = g^x \vee y_2 = g^x$. Denote $\mathcal{A}$ as the adversary of breaking the simulation-sound extractability. $\mathcal{A}$ is given the public parameters. For the SIM oracle with input (y_1, y_2) and a message m, the simulator $\mathcal{S}$ picks random $c_1, z_1, z_2 \in \mathbb{Z}_q$ and computes

$$T_1 = g^{z_1} y_1^{c_1}, \quad c_2 = H'(T_1, y_1, y_2, m), \quad T_2 = g^{z_2} y_2^{c_2}.$$

$\mathcal{S}$ sets $c_1 = H'(T_2, y_1, y_2, m)$. With probability $\epsilon \leq q_H/q$ (q_H is the number of queries to H'), the input (T_2, y_1, y_2, m) is already set in the random oracle H', and $\mathcal{S}$ declares failure and exits in this case. Otherwise, $\mathcal{S}$ returns (z_1, z_2, c_1).

When $\mathcal{A}$ returns (y_1^*, y_2^*, m^*) and a proof (z_1^*, z_2^*, c_1^*), $\mathcal{S}$ calculates $T_1^* = g^{z_1^*} y_1^{*c_1^*}, c_2^* = H'(T_1^*, y_1^*, y_2^*, m^*)$ and $T_2^* = g^{z_2^*} y_2^{*c_2^*}$. Since it is a valid proof, we have $c_1^* = H'(T_2^*, y_1^*, y_2^*, m^*)$. $\mathcal{S}$ checks if c_1^* or c_2^* was queried later to the H' oracle. WLOG, assume that c_1^* was queried later and $c_2^* = H'(T_1^*, y_1^*, y_2^*, m^*)$ was queried before. $\mathcal{S}$ rewinds to the point that c_1^* was queried with input $(T_2^*, y_1^*, y_2^*, m^*)$ and returns a different $c_1' \neq c_1^*$. If $\mathcal{A}$ still wins the game by outputting (z_1', z_2', c_1') (it happens with a non-negligible probability according to the forking lemma [3]), it means that $T_1^* = g^{z_1^*} y_1^{*c_1^*} = g^{z_1'} y_1^{*c_1'}$. Then $\mathcal{S}$ obtains the witness $w^* = (z_1^* - z_1')/(c_1' - c_1^*) = \log_g y_1^*$. Hence, it contradicts the winning condition of $\mathcal{A}$.

Zero-knowledge. Similar to the simulation of the SIM oracle above, no PPT adversary can break the zero-knowledge property with non-negligible probability in the random oracle model. □

Similar to the generic construction, our ECC-based construction has full indistinguishability in the random oracle model since the output of $F(x) = g^x$ is uniformly distributed in $\mathbb{G}$. We can also prove that our ECC-based construction has full collision-resistance in the random oracle model if the DL assumption holds in $\mathbb{G}$, or reduces to the case that the adversary outputs x_0, x_1, m_0, m_1 such that

$$H(m_1)/H(m_0) = g^{x_0}/g^{x_1} = g^{x_0 - x_1}.$$

It can be further reduced to the following Assumption DL-1.

Definition B.2 (Assumption DL-1). Given a generator g and a collision resistant hash function H, no PPT adversary can output x, m_0, m_1 with non-negligible probability such that $m_0 \neq m_1$ and

$$g^x = H(m_1)/H(m_0).$$

Lemma B.3. *The Assumption DL-1 can be reduced to the DL assumption in the random oracle model.*

Proof. Suppose that an algorithm $\mathcal{B}$ is given the DL problem (g, y). If there is a PPT attacker $\mathcal{A}$ that breaks the Assumption DL-1, $\mathcal{B}$ first gives g to $\mathcal{A}$. When $\mathcal{A}$ ask for the hash query $H(m_i)$, $\mathcal{B}$ picks a random, distinct μ_i and returns y^{μ_i}. Finally, $\mathcal{A}$ returns (x, m_0, m_1) such that $g^x = H(m_1)/H(m_0)$. Then we have $g^x = y^{\mu_1 - \mu_0}$. Hence $\mathcal{B}$ can return $x/(\mu_1 - \mu_0)$ as the DL solution of $\log_g y$. □

C SECURITY PROOF OF THE LATTICE-BASED CONSTRUCTION

Because of the "knowledge-gap" in lattice-based NIZK proof, we cannot use the simulation-sound extractability of our lattice-based NIZK proof. We refer the reader to earlier works [17, 25] for explanations on this knowledge gap issue. Hence, we give a direct proof for our lattice-based chameleon hash.

C.1 Proof of Theorem 3.8

Proof. The proof is given by a sequence of games as defined in the proof of Theorem 3.1. In the game hopping to Game_1, the simulator $\mathcal{S}$ can pick random $c_1 \in C$, $\vec{\rho} \leftarrow \mathfrak{U}_1^m$, $\vec{z_1}, \vec{z_2} \leftarrow \mathfrak{U}_{md^2-d}^m$. It calculates $\vec{y'} = \vec{G} \cdot \vec{\rho}$, $\vec{T_1} = \vec{G} \cdot \vec{z_1} - \mathsf{pk}_{\mathsf{ch}} \cdot c_1$, $c_2 = H'(\vec{T_1}, \mathsf{pk}_{\mathsf{ch}}, \vec{y'}, m)$, $\vec{T_2} = \vec{G} \cdot \vec{z_2} - \vec{y'} \cdot c_2$. It sets $c_1 = H'(\vec{T_2}, \mathsf{pk}_{\mathsf{ch}}, \vec{y'}, m)$ in the random oracle. With probability $\epsilon \leq q_H/|C|$ (q_H is the number of queries to H'), the input $(\vec{T_2}, \mathsf{pk}_{\mathsf{ch}}, \vec{y'}, m)$ is already set in the random oracle H', and $\mathcal{S}$ declares failure and exits in this case. The game hopping to Game_2 is handled similarly.

In the game hopping to Game_3, note that we can write $\vec{G} \cdot \vec{\rho} = \vec{\rho}_0 + \vec{G'} \cdot \vec{\rho}_1$ for $\vec{\rho}_0 \in \mathfrak{U}_1^k$ and $\vec{\rho}_1 \in \mathfrak{U}_1^{m-k}$. Therefore, by M-$\mathrm{LWE}_{m-k,k,q,\mathfrak{U}_1}$ assumption, $\vec{G} \cdot \vec{\rho}$ is computationally indistinguishable from a random element in R_q^k and so is $\vec{h} = \vec{G} \cdot \vec{\rho} + H(m')$. The game hopping to Game_4 is handled similarly. □

C.2 Proof of Theorem 3.9

Proof. Suppose that $\mathcal{A}$ is an adversary breaking the full collision-resistance. Suppose that the simulator $\mathcal{S}$ is given $\vec{\hat{G}} = \left[\vec{I_k} \,\|\, \vec{G'} \,\|\, \vec{g} \right] \in R_q^{k \times (m+1)}$ as the M-SIS matrix where $\vec{G'}$ and $\vec{g}$ are sampled uniformly at random. Denote $\vec{G} = \left[\vec{I_k} \,\|\, \vec{G'} \right]$, which is used as the public parameter given to $\mathcal{A}$. $\mathcal{S}$ sets

$$\mathsf{pk}_{\mathsf{ch}} = \vec{G} \cdot \vec{x} + \vec{g} \tag{2}$$

for $\vec{x} \leftarrow \mathfrak{U}_1^m$. Observe that $\|\vec{x'}\| \leq \sqrt{md+1}$ for $\vec{x'} = \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}$. Also, note that we can write $\vec{G} \cdot \vec{x} = \vec{x}_0 + \vec{G'} \cdot \vec{x}_1$ for $\vec{x}_0 \in \mathfrak{U}_1^k$ and $\vec{x}_1 \in \mathfrak{U}_1^{m-k}$. Therefore, by M-$\mathrm{LWE}_{m-k,k,q,\mathfrak{U}_1}$ assumption, $\vec{G} \cdot \vec{x}$ is computationally indistinguishable from a random element in R_q^k and so is $\mathsf{pk}_{\mathsf{ch}} = \vec{G} \cdot \vec{x} + \vec{g}$. $\mathcal{S}$ gives $\mathsf{pp}_{\mathsf{ch}} = (k, m, d, q, \vec{G}, H, H')$ and $\mathsf{pk}_{\mathsf{ch}}$ to $\mathcal{A}$. $\mathcal{S}$ also picks a random number $j \in [1, q_A]$, where q_A is the number of CHAdapt oracle query.

For the CHAdapt oracle query with input $(m, m', \vec{h}, r)$, $\mathcal{S}$ firstly checks if $\mathsf{CHCheck}(m, \vec{h}, r) = 1$. For the j-th query, if r is not the previous CHAdapt oracle with input $(\cdot, m, \vec{h}, \cdot)$, $\mathcal{S}$ additionally rewinds and extracts as follows:

- Denote $r = (\vec{z_1}, \vec{z_2}, c_1)$. $\mathcal{S}$ calculates $\vec{y_2} = \vec{h} - H(m)$, $\vec{T_1} = \vec{G} \cdot \vec{z_1} - \mathsf{pk}_{\mathsf{ch}} \cdot c_1$, $c_2 = H'(\vec{T_1}, \mathsf{pk}_{\mathsf{ch}}, \vec{y_2}, m)$, $\vec{T_2} = \vec{G} \cdot \vec{z_2} - \vec{y_2} \cdot c_2$. Since r is a valid proof, we have $c_1 = H'(\vec{T_2}, \mathsf{pk}_{\mathsf{ch}}, \vec{y_2}, m)$. $\mathcal{S}$ checks if c_1 or c_2 was queried later in the H' oracle.
- If c_1 was queried later, $\mathcal{S}$ rewinds $\mathcal{A}$ to the point that c_1 was queried with input $(\vec{T_2}, \mathsf{pk}_{\mathsf{ch}}, \vec{y_2}, m)$. $\mathcal{S}$ returns a different $c_1' \neq c_1$ for the H' oracle. If $\mathcal{A}$ still query the oracle by returning another proof $r' = (\vec{z_1'}, \vec{z_2'}, c_1')$ (it happens with a non-negligible

probability according to the forking lemma [3]), it means that

$$\vec{T_1} = \vec{G} \cdot \vec{z_1} - \mathsf{pk_{ch}} \cdot c_1 = \vec{G} \cdot \vec{z'_1} - \mathsf{pk_{ch}} \cdot c'_1.$$

Hence we get

$$\mathsf{pk_{ch}} \cdot (c_1 - c'_1) = \vec{G} \cdot (\vec{z_1} - \vec{z'_1}) = \vec{\hat{G}} \cdot \begin{pmatrix} \vec{z_1} - \vec{z'_1} \\ 0 \end{pmatrix}.$$

By multiplying equation (2) by $(c_1 - c'_1)$, we have

$$\begin{aligned} \mathsf{pk_{ch}} \cdot (c_1 - c'_1) &= \vec{G} \cdot \vec{x} \cdot (c_1 - c'_1) + \vec{g} \cdot (c_1 - c'_1) \\ &= \vec{\hat{G}} \cdot (c_1 - c'_1) \cdot \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}. \end{aligned}$$

Therefore, we get

$$\vec{\hat{G}} \cdot \begin{pmatrix} \vec{z_1} - \vec{z'_1} \\ 0 \end{pmatrix} = \vec{\hat{G}} \cdot (c_1 - c'_1) \cdot \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}.$$

That is, $\vec{\hat{G}} \cdot \vec{s} = 0$ over R_q for $\vec{s} = (c_1 - c'_1) \cdot \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix} - \begin{pmatrix} \vec{z_1} - \vec{z'_1} \\ 0 \end{pmatrix}$. Observe that $\vec{s}$ cannot be the zero vector as $c_1 \neq c'_1$ and the last coordinate of $\vec{s}$ is $(c_1 - c'_1)$. Since $\|\vec{z}_1\|_\infty, \|\vec{z}'_1\|_\infty \leq md^2 - d$, we also have $\|\vec{s}\| \leq 2d\sqrt{md+1} + 2md^2$. Hence, $\vec{s}$ is a solution to the M-SIS$_{k,m+1,q,\beta_{\mathrm{SIS}}}$ for $\beta_{\mathrm{SIS}} = 4d \cdot (2md^2 + \sqrt{md})$. $\mathcal{S}$ quits the game in this case.

- If c_2 was queried later, $\mathcal{S}$ rewinds $\mathcal{A}$ to the point that c_2 was queried with input $(\vec{T_1}, \mathsf{pk_{ch}}, \vec{y_2}, m)$. $\mathcal{S}$ returns a different $c'_2 \neq c_2$ for the H' oracle. If $\mathcal{A}$ still queries the oracle by another proof $\tilde{r} = (\vec{\tilde{z}}_1, \vec{\tilde{z}}_2, \tilde{c_2})$, it means that

$$\vec{T_2} = \vec{G} \cdot \vec{z_2} - \vec{y_2} \cdot c_2 = \vec{G} \cdot \vec{\tilde{z}}_2 - \vec{y_2} \cdot \tilde{c}_2.$$

Hence we get

$$\vec{y_2} \cdot (c_2 - \tilde{c}_2) = \vec{G} \cdot (\vec{z_2} - \vec{\tilde{z}}_2) = \vec{\hat{G}} \cdot \begin{pmatrix} \vec{z_2} - \vec{\tilde{z}}_2 \\ 0 \end{pmatrix}. \tag{3}$$

Afterward, $\mathcal{S}$ simulates the CHAdapt oracle using the random oracle H' as in the Game$_1$ of the proof of Theorem 3.8.

For input m_i, the H oracle returns $\vec{G} \cdot \vec{\mu_i} + \vec{g}$, where $\vec{\mu_i} \leftarrow \mathfrak{U}_1^k$. Observe that $\left\|\vec{\mu_i}'\right\| \leq \sqrt{md+1}$ for $\vec{\mu_i}' = \begin{pmatrix} \vec{\mu_i} \\ 1 \end{pmatrix}$. Also, note that we can write $\vec{G} \cdot \vec{\mu_i} = \vec{\mu_{i0}} + \vec{G'} \cdot \vec{\mu_{i1}}$ for $\vec{\mu_{i0}} \in \mathfrak{U}_1^k$ and $\vec{\mu_{i1}} \in \mathfrak{U}_1^{m-k}$. Therefore, by M-LWE$_{m-k,k,q,\mathfrak{U}_1}$ assumption, $\vec{G} \cdot \vec{\mu_i}$ is computationally indistinguishable from a random element in R_q^k and so is $\vec{G} \cdot \vec{\mu_i} + \vec{g}$.

In the output phase, the adversary returns $(m^*, r^*, m'^*, r'^*, \vec{h^*})$ with $m^* \neq m'^*$ and $(\vec{h^*}, m^*) \notin Q$. We consider two cases: (1) $(\vec{h^*}, m'^*) \notin Q$; (2)$(\vec{h^*}, m'^*) \in Q$.

We first consider case 1. $\mathcal{A}$ outputs $r^* = (\vec{z_1^*}, \vec{z_2^*}, c_1^*)$. $\mathcal{S}$ calculates $\vec{y_2^*} = \vec{h^*} - H(m^*)$, $\vec{T_1^*} = \vec{G} \cdot \vec{z_1^*} - \mathsf{pk_{ch}} \cdot c_1^*$, $c_2^* = H'(\vec{T_1^*}, \mathsf{pk_{ch}}, \vec{y_2^*}, m^*)$, $\vec{T_2^*} = \vec{G} \cdot \vec{z_2^*} - \vec{y_2^*} \cdot c_2^*$. Since r^* is a valid proof, we have $c_1^* = H'(\vec{T_2^*}, \mathsf{pk_{ch}}, \vec{y_2^*}, m^*)$. $\mathcal{S}$ checks if c_1^* or c_2^* was queried later in the H' oracle. $\mathcal{S}$ also performs a similar checking for $(r'^* = (\vec{z'^*_1}, \vec{z'^*_2}, c'^*_1), m'^*)$ and we do not repeat the writing here.

Case (1a): if c_1^* was queried later than c_2^*, or c'^*_1 was queried later than c'^*_2, $\mathcal{S}$ rewinds $\mathcal{A}$ to the point that c_1^* or c'^*_1 was queried. Similar to the simulation of the CHAdapt oracle, $\mathcal{S}$ can find a solution to the M-SIS$_{k,m+1,q,\beta_{\mathrm{SIS}}}$ problem.

Case (1b): if c_2^* was queried later than c_1^*, and c'^*_2 was queried later than c'^*_1, $\mathcal{S}$ rewinds $\mathcal{A}$ to the point that c_2^* and c'^*_2 was queried. Similar to the simulation of the CHAdapt oracle, $\mathcal{S}$ gets

$$\vec{y_2^*} \cdot (c_2^* - \tilde{c}_2) = \vec{G} \cdot (\vec{z_2^*} - \vec{\tilde{z}}_2) = \vec{\hat{G}} \cdot \begin{pmatrix} \vec{z_2^*} - \vec{\tilde{z}}_2 \\ 0 \end{pmatrix}, \tag{4}$$

$$\vec{y'^*_2} \cdot (c'^*_2 - \tilde{c'}_2) = \vec{G} \cdot (\vec{z'^*_2} - \vec{\tilde{z'}_2}) = \vec{\hat{G}} \cdot \begin{pmatrix} \vec{z'^*_2} - \vec{\tilde{z'}_2} \\ 0 \end{pmatrix}. \tag{5}$$

$\mathcal{S}$ calculates $(c'^*_2 - \tilde{c'}_2) \times$ Eq. (4) - $(c_2^* - \tilde{c}_2) \times$ Eq. (5):

$$H(m^*)(c'^*_2 - \tilde{c'}_2) - H(m'^*)(c_2^* - \tilde{c}_2) = \vec{\hat{G}} \cdot \begin{pmatrix} \vec{\tau} \\ 0 \end{pmatrix},$$

where $\vec{\tau} = (\vec{z_2^*} - \vec{\tilde{z}}_2)(c'^*_2 - \tilde{c'}_2) - (\vec{z'^*_2} - \vec{\tilde{z'}_2})(c_2^* - \tilde{c}_2)$.

Recall that by the simulation of the H oracle, we have $H(m^*) = \vec{G} \cdot \vec{\mu^*} + \vec{g}$ and $H(m'^*) = \vec{G} \cdot \vec{\mu'^*} + \vec{g}$ for some $\vec{\mu^*}, \vec{\mu'^*} \in \mathfrak{U}_1^k$. Hence, we have

$$\begin{aligned} &H(m^*)(c'^*_2 - \tilde{c'}_2) - H(m'^*)(c_2^* - \tilde{c}_2) \\ =&\vec{G} \cdot [\vec{\mu^*}(c'^*_2 - \tilde{c'}_2) - \vec{\mu'^*}(c_2^* - \tilde{c}_2)] + \vec{g} \cdot [(c'^*_2 - \tilde{c'}_2) - (c_2^* - \tilde{c}_2)] \\ =&\vec{\hat{G}} \cdot \begin{pmatrix} \vec{\mu^*}(c'^*_2 - \tilde{c'}_2) - \vec{\mu'^*}(c_2^* - \tilde{c}_2) \\ (c'^*_2 - \tilde{c'}_2) - (c_2^* - \tilde{c}_2) \end{pmatrix} \end{aligned}$$

That is, $\vec{\hat{G}} \cdot \vec{s} = 0$ over R_q for $\vec{s} = \begin{pmatrix} \vec{\mu^*}(c'^*_2 - \tilde{c'}_2) - \vec{\mu'^*}(c_2^* - \tilde{c}_2) - \vec{\tau} \\ (c'^*_2 - \tilde{c'}_2) - (c_2^* - \tilde{c}_2) \end{pmatrix}$. Observe that the last coordinate of $\vec{s}$ is $(c'^*_2 - \tilde{c'}_2) - (c_2^* - \tilde{c}_2)$, and it is equal to zero only with probability $1/q$. Hence, $\vec{s}$ cannot be the zero vector with overwhelming probability.

Since $\|\vec{z_2^*}\|_\infty, \|\vec{\tilde{z}}_2\|_\infty, \|\vec{z'^*_2}\|_\infty, \|\vec{\tilde{z}'_2}\|_\infty \leq md^2 - d$, we also have

$$\begin{aligned} \|\vec{s}\| &\leq 4d\sqrt{md+1} + 8d \cdot (md^2 - d) + 4d \\ &\leq 4d \cdot \left(2md^2 + \sqrt{md}\right). \end{aligned}$$

Therefore, $\vec{s}$ gives a solution to M-SIS$_{k,m+1,q,\beta_{\mathrm{SIS}}}$ for $\beta_{\mathrm{sis}} \approx 4d \cdot \left(2md^2 + \sqrt{md}\right)$.

Next, we consider case 2. If $(\vec{h^*}, m'^*) \in Q$, there are two subcases:

Case (2a): The i-th CHAdapt oracle query has input $(m'^*, \cdot, \vec{h^*}, r_i)$ for some r_i. If $i \neq j$, $\mathcal{S}$ declares failure and exits. If $i = j$, then the extraction of (r_i, m'^*) has already been done during the CHAdapt oracle query. The solution to the M-SIS problem is calculated as in case (1a) and case (1b).

Case (2b): There was a CHAdapt query with input $(m_i, m'^*, \vec{h^*}, r_i)$ for some m_i, r_i. If further looks for previous query with input $(m_{i-1}, m_i, \vec{h^*}, r_{i-1})$ and output r_i. It runs recursively until it finds a query (denoted as the i^*-th query) with input $(m_0, m_1, \vec{h^*}, r_0)$ and output r_1, such that there is no query with input $(\cdot, m_0, \vec{h^*}, \cdot)$ and output r_0. If $i^* \neq j$, $\mathcal{S}$ declares failure and exits. If $i^* = j$, then the extraction of (r_0, m_0) has already been done during the CHAdapt oracle query. The solution to the M-SIS problem is calculated as in case (1a) and case (1b).

Apart from the probability of rewinding successfully, the success probability for case (2a) and (2b) are at least $1/q_A$, and the success probability for case (1b) is at least $1 - 1/q$. Also, $\mathcal{S}$ has to rewind

once for the j-th CHAdapt oracle query, and twice for case (1b) or case (2b). Hence, $\mathcal{S}$ can solve the M-SIS problem in polynomial time with non-negligible probability. □

C.3 Parameter Selection

The practical security estimations of M-SIS and M-LWE against known attacks can be found in [16, Section 3.2.4]. In particular, we look for a "Root Hermite Factor" of around 1.0045, which is a common metric used in lattice-based cryptography to measure practical hardness. Following Algorithm 3.1 in [16], we have $\log q = 28, d = 128, k = 9, m = 20$.

Observe that the size of the hash value $\vec{h} = dkq$ bits, which is about 4kB. The length of r can be approximated by the following formula:

$$|r| = 2|\vec{z}| + |c_1| \approx 2 * 7541 + 26 = 15108 \text{ bytes.} \quad (6)$$

The above formula stems from the fact that $|c_1| = d \log 3/8$ bytes and $|\vec{z}| = md \log(2md^2)/8$ bytes since $\vec{z} \in R^m$ with $\|\vec{z}\|_\infty \leq md^2$. Plugging in $(d, m) = (128, 20)$ yields (6).