Anonymous probabilistic payment in payment hub

Tatsuo Mitani

Graduate School of Information Security,
Institute of Information Security
Mitsubishi Chemical Systems, Inc.
dgs187101@iisec.ac.jp

Akira Otsuka

Graduate School of Information Security,
Institute of Information Security
otsuka@iisec.ac.jp

June 19, 2020

Abstract

Privacy protection and scalability are significant issues with blockchain. We propose an anonymous probabilistic payment under the general functionality for solving them. We consider the situation that several payers pay several payees through a tumbler. We have mediated the tumbler of the payment channel hub between payers and payees. Unlinkability means that the link, which payer pays which payee via the tumbler, is broken. A cryptographic puzzle plays a role in controlling the intermediation and execution of transactions. Masking the puzzle enables the payer and the payee to unlink their payments. The overview of the proposed protocol is similar to TumbleBit (NDSS 2017). We confirm the protocol realizes the ideal functionalities discussed in TumbleBit. The functionality required for our proposal is the hashed time lock contract that various cryptocurrencies use. This request is general, not restricted to any particular cryptocurrency. Our proposal includes a probabilistic payment. In probabilistic payment, one pays an ordinary mount with a certain probability. One pays a small amount as an expected value. One can run fewer transactions than a deterministic payment. It contributes scalability. We introduce a novel fractional oblivious transfer for probabilistic payment. We call it the ring fractional oblivious transfer (RFOT). RFOT is based on the ring learning with errors (RLWE) encryption. Our trick is based on the fact that an element of the ring is indistinguishable from the circular shifted element. We confirm that RFOT holds the properties of fractional hiding and binding presented in the DAM scheme (Eurocrypt 2017).

1 Introduction

The Bitcoin [Nak08] announced by Nakamoto has spread all over the world. This underlying technology is now called the blockchain. There are many other cryptocurrencies based on blockchain technology. Cryptocurrencies also have spread and usable in the real world. There are many problems that blockchain needs to be solved. Scalability and privacy protection are significant problems in blockchain. However, there are a few proposals to solve these problems at the same time.

Let us state privacy protection. The transaction normally links a payer and a payee. One can figure out their balance and trading frequency by analyzing the links of transactions. Privacy is not protected. Breaking the transaction link is necessary to protect privacy.

Let us describe scalability. It is costly to write all the small transactions into the blockchain. The payment of a small amount of money is called micropayment. For such micropayments, Wheeler [Whe97] and Rivest [Riv97] proposed a probabilistic payment before blockchain appears. It reduces costs for micropayment. In probabilistic payment, one pays an ordinary mount m with a certain probability p. One pays a small amount mp as an expected value. By the probability p (e.g. $p \approx 0.1 \sim 0.001$), one can reduce $1/p$ times transactions compared to deterministic payment. Micropay [Ps15], the DAM scheme [CGL$^+$17] and Microcash [ABCar] have proposed a new micropayment on the blockchain. It is attracting attention as one of the leading solutions for scalability in the blockchain.

1.1 Our contribution

We propose an anonymous probabilistic payment. It aims to solve both scalability and privacy protection. In this work, we realize a kind of unlinkability, which is "k-anonymity in an epoch." This definition is mentioned in TumbleBit [HAB+17] and their earlier work [HBG16]. The link means that which payer pays which payee via a tumbler within an epoch. Anonymity says the link is broken. Even a tumbler never knows the link. The "k" is the number of participants trading via the tumbler. The epoch is the period during which transactions are completed.

Our proposal includes a probabilistic payment. In probabilistic payment, one pays an ordinary mount m with a certain probability p (e.g. $p \approx 0.1 \sim 0.001$). One pays a small amount mp as an expected value. One can reduce $1/p$ times transactions compared to deterministic payment. We introduce a novel fractional oblivious transfer for adopting the probabilistic payment. We call it the ring fractional oblivious transfer. It is based on the ring learning with errors (RLWE) encryption. The functionality required for our proposal is hashed time lock contract (HTLC). Various cryptocurrencies adapt HTLC. This request is general, not restricted to any particular cryptocurrency.

1.2 Related work

In this subsection, we describe the related work regarding anonymity in blockchain, probabilistic payment, fractional oblivious transfer, and the comparison with a concurrent work.

1.2.1 Anonymity in blockchain

There are researches on a new anonymous cryptocurrency. Zerocash [SCG+14] is a famous anonymous cryptocurrency and is implemented as ZCash. Regarding ZCash, their group has proposed continuous researches such as BOLT [GM17] and DAM scheme [CGL+17]. Monero is also a famous anonymous cryptocurrency and provided with incredible works [Noe15, SALY17, YSL+ar, MSRL+19]. There are studies to realize anonymity for existing cryptocurrencies by using off-chain technology. TumbleBit [HAB+17] is compatible with Bitcoin. Zether [BAZBar] is compatible with Ethereum.

1.2.2 Probabilistic payments

Wheeler [Whe97] and Rivest [Riv97] proposed a probabilistic payment. Micropay [Ps15] is compatible with Bitcoin. Microcash [ABCar] can be implemented as a smart contract. The DAM scheme, which is the extension of anonymous ZCash, also realizes a probabilistic payment.

1.2.3 Fractional oblivious transfer over the ring

Bellare and Micali [BM90] and Bellare and Rivest [BR99] proposed the fractional oblivious transfer based on the computational Diffie-Hellman assumption as the early works. The DAM scheme [CGL+17, CGL+16] also proposed a novel fractional oblivious transfer based on the decisional Diffie-Hellman assumption as fractional message transfer. Note that [CGL+16] is the full version of [CGL+17].

Brakerski and Döttling proposed an oblivious transfer based on the learning with errors [BD18]. This work is the first oblivious transfer in the post-quantum cryptography. Liu and Hu first proposed an efficient 1-out-of-2 oblivious transfer (OT) on the RLWE assumption and extends 1-out-of-n OT [LH19]. To the best of our knowledge, we first propose the fractional oblivious transfer over the ring.

1.2.4 Comparison with a concurrent work

The DAM scheme [CGL+17, CGL+16] is an anonymous probabilistic payment, which passes ZCash transaction by a fractional oblivious transfer. Since the scheme includes a ZCash transaction in the message, it is a ZCash specific implementation. Although we also propose an anonymous probabilistic payment, the required functionality is the hashed time lock contract (HTLC). Various cryptocurrencies such as Bitcoin, implement HTLC. Therefore our proposal is not limited to a specific cryptocurrency.

1.3 Our approach

We present the approach of the anonymous probabilistic payment and the ring fractional oblivious transfer, which is a vital ingredient in our proposal.

1.3.1 Anonymous probabilistic payment

We propose an anonymous probabilistic payment that a payer pays a payee via a tumbler. We name the payer Alice and the payee Bob. Suppose that she wants to send one coin to him through the tumbler with a certain probability p. We show the overview of the protocol in Fig. 1. Our protocol mostly follows TumbleBit [HAB+17].

Let us explain each phase in the protocol, setup, puzzle promise, and puzzle solver as follows. In the setup phase, the tumbler generates a one-time key pair consisting of a public key and a private key. The tumbler also attaches proof of the zero-knowledge proof. The tumbler publishes this proof and the public key.

In the promise phase, the tumbler and Bob interact. The tumbler prepares an escrow transaction that pays one coin to him from the tumbler. Both the signatures of the tumbler and he can execute this escrow transaction. The tumbler does not present its signature to him in this phase. Instead, the tumbler creates puzzles and promises from its signature and presents them to him. If he has this puzzle answer, then he obtains the tumbler signature from the promise and puzzle.

To gain the answer, Bob will ask Alice to solve the puzzle. Then, he masks the puzzle to delete his link and sends it to her. She also masks the received puzzle from him to delete her link. She will pay and get the answer to the puzzle with probability p in the next phase.

In the solver phase, Alice interacts with the tumbler to get the puzzle answer. She makes a probability payment to the tumbler, paying one coin with probability p. Then, she sends the double-masked puzzle to the tumbler. She issues a transaction to the tumbler and asks the tumbler to post the puzzle answer. She asks the tumbler to decrypt in exchange for the commitment to the transaction. The tumbler posts the answer and executes the transaction. Alice demasks the decrypted answer and sends the single masked answer to Bob.

In the cash-out phase, Bob receives the single masked answer from Alice. He demasks it and obtains the answer. As the tumbler and he promised in the promise phase, he gets the tumbler signature from the answer. Finally, he executes the escrow transaction and gain one coin.

For the probabilistic payment in the solver phase, we introduce a novel ingredient. It is a fractional oblivious transfer based on the ring learning with errors (RLWE) encryption. We call it the ring fractional oblivious transfer (RFOT).

Alice $\mathcal{A}$	Tumbler $\mathcal{T}$	Bob $\mathcal{B}$
		Puzzle promise protocol
		Obtain puzzle z from $\mathcal{T}$
		$y = \mathsf{mask}(z)$
		Send y to $\mathcal{A}$
Puzzle solver protocol		
$\beta = \mathsf{mask}(y)$		
Send β to $\mathcal{T}$		
Learn s from $\mathcal{T}$		
$m' = \mathsf{demask}(s)$		
Send m' to $\mathcal{B}$		
		$m = \mathsf{demask}(m')$
		Obtain $\mathcal{T}$'s singnature from m and post it

Figure 1: Overview of the proposed protocol

1.3.2 Ring fractional oblivious transfer

Let us describe the ring fractional oblivious transfer (RFOT).

Basic idea. In the RLWE encryption, it is essential to operate on the ring $\mathbf{R}_q = \mathbb{Z}_q[X]/\langle X^{n_d}+1\rangle$. Let us compare an element $a \in \mathbf{R}_q$ with aX^k multiplied by X^k with an integer k. One can choose k randomly from the set $\{1,\ldots,n\}$, where $n < n_d$. Because $X^{n_d} = -1$, the original coefficient of the element a is cyclically shifted regardless of the sign. In key generation or encryption, one samples coefficients of the element a randomly from the uniform distribution or the discrete Gaussian distribution. These distributions are positive and negative symmetric. The element a chosen from the distributions and the shifted aX^k are indistinguishable. Looking at the aX^k, one cannot identify how much the shift is.

Procedure. Suppose that a sender sends a message m to a receiver. We name the sender Alice and the receiver Bob. First, he creates a one-time key pair, a public key and private key. He randomly chooses an integer $k \in \{1,\ldots,n\}$, cyclically shifts the public key by multiplying X^{-k}. Furthermore, he issues this shifted public key to her. She creates the ciphertext by the shifted public key. She does not know the shift amount of k. She also randomly chooses an integer $l \in \{1,\ldots,n\}$, and shifts the ciphertext by multiplying X^l. He receives the shifted ciphertext and decrypts it by the secret key. If both shifted amounts l and k match, he can obtain a message m in the specified format. If they do not match, he has a broken message $\emptyset$. The probability that both l and k match is $1/n$.

Security. We prove that the RFOT satisfies the fractional hiding and fractional binding. The DAM scheme [CGL$^+$17, CGL$^+$16] introduced the two properties. Fractional hiding is a property that a ciphertext created by an honest encryptor can be decrypted exactly with probability p. Fractional binding is a property that a malicious encryptor cannot create valid ciphertext that can be decrypted with probability $p' \neq p$. We prove the security of the fractional hiding and fractional binding with the simulation-based security like [CGL$^+$17, CGL$^+$16]. We adopt a trapdoor to simulate the RFOT. Let us explain the trapdoor for the simulated RFOT. We use the scheme for the trapdoor proposed in the identity-based encryption over NTRU lattice [DLP14]. The trapdoor requires the decisional small polynomial ratio (DSPR) assumption. We regard the one-time public key and private key as a user key and the simulated trapdoor as a master key. It can directly produce the small elements of the RLWE encryption. One can simulate a shifted secret key or plaintext and randomness from the randomly sampled elements s_1, s_2 such that $y = as_1 + 2s_2$ for the given y and the trapped a. A trapdoor using a gadget matrix is known. Micciancio and Peikert proposed a gadget matrix based trapdoor [MP12]. Following this work, Genise and Micciancio proposed an efficient Gaussian sampler for the trapdoor [GM18]. Cousins *et al.* proposed the implementation of the RLWE encryption [CDG$^+$18]. However, this method does not output the small elements required for RLWE encryption.

1.4 Chapter organization

We describe the organization of this work. In Section 2, we state the building blocks. In Section 3, we introduce the ring fractional oblivious transfer. In Section 4, we introduce the puzzle solver protocol and the puzzle promise protocol and discuss the security of the protocols. In Section 5, we conclude this work.

2 Building blocks

In this section, we describe the building blocks.

2.1 Notation

Let $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ and $\mathbb{R}$ be the set of natural numbers, the set of integers, the set of rational numbers and the set of real numbers, respectively. Let the finite field $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z} = \{0, 1, \ldots, q-1\}$, where q is a prime number. Let $\mathbb{Z}[X]$ be the ring of polynomials over the integers. Let $\Phi_l \in \mathbb{Z}[X]$ be the l-th cyclotomic polynomial. We set l to a power of 2. We have the cyclotomic polynomial $\Phi_l = X^{n_d}+1$, where $n_d = \phi(l)$. ϕ is the Euler function. Let the ring of integers $\mathbf{R} = \mathbb{Z}[X]/\langle X^{n_d}+1\rangle$, and let $\mathbf{R}_q = \mathbb{Z}_q[X]/\langle X^{n_d}+1\rangle = \mathbb{Z}[X]/\langle X^{n_d}+1, q\rangle$. $\mathbf{R}_q$ is the ring of integers $\mathbf{R}$ modulo q. We identify a vector $(a_0,\ldots,a_{n_d-1}) \in \mathbb{Z}^{n_d}$ with a polynomial $a_0 + a_1X + \cdots + a_{n_d-1}X^{n_d-1} \in \mathbf{R}$. We denote the norms of $a = a_0 + a_1X + \cdots + a_{n_d-1}X^{n_d-1} \in \mathbf{R}$ as follows. $|a_i|$ is the absolute value of a_i.

- l_2-norm $|a| = \sqrt{a_0^2 + a_1^2 + \cdots + a_{n_d-1}^2}$
- l_∞-norm $|a|_\infty = \max(|a_0|, |a_1|, \ldots, |a_{n_d-1}|)$

If $\forall c \in \mathbb{N}$, $\exists n_c$ such that $\forall n > n_c, f(n) < 1/n^c$, then we denote the function f as a negligible function $\mathrm{negl}(n)$. If an element a is sampled randomly from a distribution A or a uniform distribution over the set A, then we denote $a \xleftarrow{\$} A$.

2.2 Ring learning with errors encryption

We introduce the definitions regarding the ring learning with errors (RLWE) encryption.

2.2.1 Discrete Gaussian distribution

we describe the discrete Gaussian distribution.

- $\rho_{v,\sigma}^{n_d}(x) = (\frac{1}{\sqrt{2\pi}\sigma})^{n_d} e^{\frac{-|x-v|^2}{2\sigma^2}}$ is the continuous normal distribution over $\mathbb{R}^{n_d}$ centered at v with standard deviation σ. If $v = 0$, then we write $\rho_{v,\sigma}^{n_d}$ as $\rho_\sigma^{n_d}$.
- $D_{v,\sigma}^{n_d}(x) = \rho_{v,\sigma}^{n_d}(x)/\rho_\sigma^{n_d}(\mathbb{Z}^{n_d})$ is the discrete normal distribution over $\mathbb{Z}^{n_d}$ centered at $v \in \mathbb{Z}^{n_d}$ with standard deviation σ. In this definition, the quantity $\rho_\sigma^{n_d}(\mathbb{Z}^{n_d}) = \Sigma_{z \in \mathbb{Z}^{n_d}} \rho_\sigma^{n_d}(z)$ is just a normalized quantity. It is necessary to express the function as a probability distribution. We also mention that $\forall v \in \mathbb{Z}^{n_d}$, $\rho_{v,\sigma}^{n_d}(\mathbb{Z}^{n_d}) = \rho_\sigma^{n_d}(\mathbb{Z}^{n_d})$. The scaling factor is the same for all v. If the dimension n_d is clear from the context, then we omit n_d and write $D_\sigma^{n_d}$ as D_σ. We also denote D_σ as χ.

We introduce the below lemma regarding the discrete Gaussian distribution.

Lemma 1 (Lemma 4.4 in [MR07])**.** *Let $n_d \in \mathbb{N}$. For any number $\sigma > \omega(\sqrt{\log n_d})$, we have*

$$\Pr_{x \xleftarrow{\$} D_\sigma}[|x|_\infty > \sigma\sqrt{n_d}] \leq 2^{-n_d+1}.$$

Remark 1. *According to Lemma 1, $|x| \leq B$ with overwhelming probability if $x \xleftarrow{\$} D_\sigma$. B is a constant value.*

2.2.2 Assumption

First, we present the problems over the ring as follows.

Definition 1 (Ring learning with errors ($\mathsf{RLWE}_{\phi,q,\chi}$) problem)**.** *Let $\phi(X) \in \mathbb{Z}[X]$ be a polynomial of degree n_d, let $q \in \mathbb{Z}$ be a prime integer, let χ denote a distribution over the ring $\mathbf{R} = \mathbb{Z}[X]/\langle\phi(X)\rangle$, and let $\mathbf{R}_q = \mathbf{R}/q\mathbf{R}$. The ring learning with errors problem $\mathsf{RLWE}_{\phi,q,\chi}$ is to distinguish between the following two distributions: $(a, y) \in \mathbf{R}_q^2$ such that $a \xleftarrow{\$} \mathbf{R}_q$, $s, e \xleftarrow{\$} \chi$, $y = as + e$, and $(a, y) \xleftarrow{\$} \mathbf{R}_q^2$.*

The RLWE assumption indicates that the RLWE problem is hard for any probabilistic polynomial time algorithm. The RLWE assumption still holds even if we choose the secret s according to the error distribution D_σ rather than uniformly [LPR13].

Definition 2 (Decisional small polynomial ratio ($\mathsf{DSPR}_{\phi,q,\chi}$) problem (Definition 3.4 in [LATV12]))**.** *Let $\phi(X) \in \mathbb{Z}[X]$ be a polynomial of degree n_d, let $q \in \mathbb{Z}$ be a prime integer, and let χ denote a distribution over the ring $\mathbf{R} = \mathbb{Z}[X]/\langle\phi(X)\rangle$. The decisional small polynomial ratio problem $\mathsf{DSPR}_{\phi,q,\chi}$ is to distinguish between the following two distributions: a polynomial $h = g/f$, where f and g are sampled from the distribution χ (conditioned on f being invertible over $\mathbf{R}_q = \mathbf{R}/q\mathbf{R}$), and a polynomial $h \xleftarrow{\$} \mathbf{R}_q$.*

According to [LATV12], let us explain a standard deviation of the discrete Gaussian distribution D_σ and DSPR assumption. It is known that the DSPR problem is hard if the standard deviation is large. For the calculation using homomorphic encryption, we want to take a small one. In this case, it is assumed that the DSPR problem is still hard. This is called DSPR assumption.

2.2.3 RLWE encryption and its syntax

Now we introduce Brakerski-Vaikuntanathan (BV) scheme [BV11]. We describe plaintext space, key generation, encryption and decryption as follows.

- The plaintext space $\mathbf{M} = \mathbf{R}_p = \mathbb{Z}[X]/\langle X^{n_d} + 1, p\rangle$. In this work, we set $p = 2$ in $\mathbf{M}$.
- To generate key, $a, s \xleftarrow{\$} \mathbf{R}_q,\ e_s \xleftarrow{\$} D_\sigma,\ b = as + e_s$, where (a, b) is a public key and s is a secret key.
- To encrypt a plaintext $m \in \mathbf{M}$, choose a set of randomness $v, e, f \xleftarrow{\$} D_\sigma$, and compute the ciphertext $c = (c_1, c_2) = (bv + pe + m, av + pf) \in \mathbf{C}$. We also denote the procedure as $c = \mathsf{RLWE.Enc}(\mathsf{pk}, m)$ or $c = \mathsf{RLWE.Enc}(\mathsf{pk}, m, v, e, f)$. The latter is the case of specifing the randomness.
- To decrypt $c = (c_1, c_2)$ with secret key s and obtain a plaintext m, compute $m = (c_1 - s \cdot c_2 \mod q) \mod p$. We also denote the procedure as $m = \mathsf{RLWE.Dec}(\mathsf{sk}, c)$.

3 Ring fractional oblivious transfer

In this section, we introduce a fractional oblivious transfer over the ring. First, we describe the trapdoor and the zero-knowledge proof. Next, we introduce the syntax and the definition of the scheme and its simulator, and its properties, referring to [CGL+16]. Finally, we present the construction of a novel fractional oblivious transfer and discuss the security.

3.1 Trapdoor

We introduce the trapdoor based on DSPR assumption according to [DLP14]. We use $\mathsf{IBE.MasterKeygen}(n_d, q)$ and $\mathsf{IBE.Extract}(\mathbf{B}, t)$ of [DLP14]. We show the functions in Fig. 2. We state the difference between our functions and [DLP14]. [DLP14] calculates the hash value of the second argument in the function IBE.Extract. However, we use the target value t as it is.

- Master key generation: $\mathsf{IBE.MasterKeygen}(n_d, q) \to (a, \mathbf{B}, g_a, f_a)$

  On input a dimension n_d of the ring $\mathbf{R}$ and a modulo q of the ring $\mathbf{R}_q = \mathbf{R}/q\mathbf{R}$, IBE.MasterKeygen outputs a master public key a, a master secret key $\mathbf{B}$, and small polynomials g_a, f_a such that $a = g_a \cdot f_a^{-1}$.

- Extractor: $\mathsf{IBE.Extract}(\mathbf{B}, t) \to (s_1, s_2)$

  On input a master secret key $\mathbf{B}$ and a target value t, IBE.Extract outputs a pair of small polynomials (s_1, s_2) such that $t = as_1 + 2s_2$, where a is the master public key.

Figure 2: Trapdoor by identity-based encryption [DLP14]

3.2 Non-interactive zero-knowledge proof

Let us state the non-interactive zero-knowledge proof (NIZK). We show the syntax of NIZK in Fig. 3, referring to [CGL+16]. In this work, we represent NIZK according to the syntax. $\mathcal{R}$ is a relation regarding an instance x and a witness w. If x and w satisfy the relation $\mathcal{R}$, then we denote $(x, w) \in \mathcal{R}$.

The non-interactive zero-knowledge proof for the relation $\mathcal{R}$ is a protocol that satisfies the below properties.

1. Completeness For any $(x, w) \in \mathcal{R}$, the conditional probability

$$\Pr[\mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi) = 1 \wedge (x, w) \in \mathcal{R} \mid \mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}), \pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w)] \geq 1 - \mathsf{negl}(\lambda).$$

- Setup: $\mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}) \to \mathsf{crs}$

  On input a security parameter λ and a relation $\mathcal{R}$, NIZK.Setup outputs a common reference strings crs.

- Prove: $\mathsf{NIZK.Prove}(\mathsf{crs}, x, w) \to \pi$

  On input a common reference strings crs, an instance x, and a witness w, NIZK.Prove outputs a proof π.

- Verify: $\mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi) \to 1$ or 0

  On input a common reference strings crs, an instance x, and a proof π, NIZK.Verify outputs 1 if π is valid, otherwise 0.

- Simulated setup: $\mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}) \to (\mathsf{crs}, \mathsf{td})$

  On input a security parameter λ and a relation $\mathcal{R}$, NIZK.SimSetup outputs a common reference strings crs and a trapdoor td.

- Knowledge extractor: $\mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}, x, \pi) \to w$

  On input a common reference strings crs, a trapdoor td, an instance x, and a proof π, NIZK.Extract outputs a witness w.

- Simulator: $\mathsf{NIZK.Simulate}(\mathsf{crs}, \mathsf{td}, x) \to \pi$

  On input a common reference strings crs, a trapdoor td, and an instance x, NIZK.Simulate outputs a proof π.

- Knowledge extracting simulator: $\mathsf{NIZK.ExtSimulate}(\mathsf{crs}, \mathsf{td}, x) \to \pi$

  On input a common reference strings crs, a trapdoor td, and an instance x, NIZK.ExtSimulate outputs a proof π.

Figure 3: Syntax of NIZK [CGL+16]

2. Soundness

   Let a knowledge extractor $\mathsf{NIZK.Extract}$ for any adversary $\mathcal{A}$ exist. For all $(x, w) \notin \mathcal{R}$, the conditional probability

$$\begin{aligned}\Pr[&\mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi') = 1 \wedge (x, w') \notin \mathcal{R} \mid \\ &(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}), \pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w), \\ &w' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}, x, \pi), \pi' \leftarrow \mathcal{A}(\mathsf{crs}, x, w')] \leq \mathsf{negl}(\lambda).\end{aligned}$$

3. Zero-knowledge

   For any adversary, the below two distribution ensembles $\mathcal{E}_Z^{\mathsf{real}}, \mathcal{E}_Z^{\mathsf{ideal}}$ are computationally indistinguishable.

$$\begin{aligned}\mathcal{E}_Z^{\mathsf{real}} &= \{(\mathcal{R}, \mathsf{crs}, x, \pi) \mid \mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}), \pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w)\} \\ \mathcal{E}_Z^{\mathsf{ideal}} &= \{(\mathcal{R}, \mathsf{crs}, x, \pi) \mid (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}), \\ &\qquad \pi \leftarrow \mathsf{NIZK.Simulate}(\mathsf{crs}, \mathsf{td}, x)\}\end{aligned}$$

We introduce the following relations $\mathcal{R}_K, \mathcal{R}_0$ and $\mathcal{R}_E$. We describe the concrete protocols for these relations in Appendix A.

$$\begin{aligned}\mathcal{R}_K &= \{((a, y), (s, e)) \mid y = as + 2e \wedge |s|, |e| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\} \\ \mathcal{R}_0 &= \{((c_1, c_2), (v, e, f)) : (c_1, c_2) = (bv + pe, av + pf) \wedge |v|, |e|, |f| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\} \\ \mathcal{R}_E &= \{((c_1, c_2, a, y), (m, v, e, f)) \mid c_1 = yv + 2e + m \ \wedge \ c_2 = av + 2f \\ &\qquad \wedge \ |m|_\infty \leq 1 \ \wedge \ |v|, |e|, |f| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\}\end{aligned}$$

Let us explain $\mathsf{NIZK.ExtSimulate}$ regarding the relation $\mathcal{R}_E$ for a randomly sampled pair (c_1, c_2) as follows. We adopt the special trapdoor td_I since a pair (c_1, c_2) is randomly sampled.

Definition 3 (Extract Simulate with trapdoor)**.** $\mathsf{NIZK.ExtSimulate}$ *regarding the relation* $\mathcal{R}_E$ *for a given pair* $(c_1, c_2) \xleftarrow{\$} \mathbf{C}$ *is stated as follows.*

- $\mathsf{NIZK.ExtSimulate}(\mathsf{crs}_E, \mathsf{td}_I, x) \rightarrow \pi$

  1. *Parse* x *as* (c_1, c_2, a, y).
  2. *Parse* td_I *as* $(\mathbf{B}, g_a, f_a)$.
  3. *Compute* $(v', f') = \mathsf{IBE.Extract}(\mathbf{B}, c_2)$.
  4. *Compute* $(s_1, s_2) = \mathsf{IBE.Extract}(\mathbf{B}, (c_1 - yv')f_a^{-1})$.
  5. *Compute* m', e' *from* $(c_1 - yv')f_a^{-1} = as_1 + 2s_2$.
  6. *Compute* $\pi = \mathsf{NIZK.Prove}(\mathsf{crs}_E, x, (m', v', e', f'))$.
  7. *Output* π.

Remark 2. *Let us confirm how to obtain* m', e' *from* $(c_1 - yv')f_a^{-1} = as_1 + 2s_2$*. Note that* $a = g_a f_a^{-1}$*, where* f_a *and* g_a *are small polynomials.*

$$\begin{aligned}(c_1 - yv')f_a^{-1} &= as_1 + 2s_2 \\ (c_1 - yv')f_a^{-1} &= s_1 g_a f_a^{-1} + 2s_2 \\ c_1 - yv' &= s_1 g_a + 2s_2 f_a \\ c_1 &= yv' + s_1 g_a + 2s_2 f_a\end{aligned}$$

We have $m' = s_1 g_a \mod 2$ *and* $e' = s_2 f_a$*. We obtain* $c_1 = yv' + 2e' + m'$.

Lemma 2. *Let* NIZK.ExtSimulate *be defined as Definition 3. The below two distribution ensembles* $\mathcal{E}^{\mathsf{real}}_{Z'}$ *and* $\mathcal{E}^{\mathsf{ideal}}_{Z'}$ *are computationally indistinguishable.*

$$\begin{aligned}\mathcal{E}^{\mathsf{real}}_{Z'} &= \{(\mathcal{R}_E, \mathsf{crs}, x, \pi) \mid \mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}_E), \pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w)\}\\ \mathcal{E}^{\mathsf{ideal}}_{Z'} &= \{(\mathcal{R}_E, \mathsf{crs}, x, \pi) \mid (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}_E),\\ &\qquad\qquad \pi \leftarrow \mathsf{NIZK.ExtSimulate}(\mathsf{crs}, \mathsf{td}, x)\}\end{aligned}$$

Proof. Let us confirm Definition 3 of ExtSimulate. A simulated witness $w' = (m', v', e', f')$ is generated from the trapdoor td_I. Then the true witness w is entirely unnecessary. We can choose a small witness w' that fits the norm constraint by IBE.Extract. Besides, we can call the actual proving function NIZK.Prove inline. The simulated proof by NIZK.ExtSimulate is computationally indistinguishable from the proof by NIZK.Prove. We conclude that the two distribution ensembles $\mathcal{E}^{\mathsf{real}}_{Z'}$ and $\mathcal{E}^{\mathsf{ideal}}_{Z'}$ are computationally indistinguishable. □

3.3 Syntax and definition

We introduce the definitions of the scheme, the correctness, and the simulator, fractional hiding, and fractional binding. These definitions appear initially as fractional message transfer in [CGL+16]. We follow the syntax and the definition in [CGL+16], and realize our original scheme and its security in Section 3.4.

3.3.1 Syntax and definition of scheme and simulator

We state the syntax and the definition regarding the scheme and its simulator. We show the syntax and definition of the scheme in Fig. 4.

Definition 4 (Correctness [CGL+16])**.** *A scheme is correct if for every security parameter* λ*, public parameters* $\mathsf{pp} \in \mathsf{RFOT.Setup}(1^\lambda)$*, probability* $p \in [0, 1]$*, key pair* $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{RFOT.Keygen}(\mathsf{pp}, p)$*, and message* $m \in \mathbf{M}$*,*

$$\mathsf{RFOT.Decrypt}(\mathsf{pp}, \mathsf{sk}, \mathsf{RFOT.Encrypt}(\mathsf{pp}, \mathsf{pk}, m)) = \begin{cases} m & \textit{with probability } p \\ \emptyset & \textit{with probability } 1-p \end{cases}$$

Remark 3. *The types of a decrypted message* m' *are as follows. Let us name valid/invalid message hit/miss since we illustrate a probabilistic payment as a lottery ticket.*

- m *: well formatted valid message. We also call it "hit."*
- $\emptyset$ *: broken formatted invalid message. We also call it "miss."*
- $\perp$ *: error message reporting error occurrence*

3.3.2 Fractional hiding and binding

In this subsection, we introduce the definitions of fractional hiding and binding, according to [CGL+16].

Definition 5 (Fractional hiding (Claim A.7 in [CGL+16]))**.** *The hiding property holds when the following distribution ensembles* $\mathcal{E}^{\mathsf{real}}_H$ *and* $\mathcal{E}^{\mathsf{ideal}}_H$ *are computationally indistinguishable for any adversary* $\mathcal{A}_H$*, where*

$$\begin{aligned}\mathcal{E}^{\mathsf{real}}_H = \{\mathsf{out} \mid\ & \mathsf{RFOT.Setup}(1^\lambda) \to \mathsf{pp}, \mathcal{A}_H(\mathsf{pp}) \to (\mathsf{pk}, m),\\ & \mathsf{RFOT.Encrypt}(\mathsf{pp}, \mathsf{pk}, m) \to c, \mathcal{A}_H(c) \to \mathsf{out}\}\end{aligned}$$

and

$$\begin{aligned}\mathcal{E}^{\mathsf{ideal}}_H = \{\mathsf{out} \mid\ & \mathsf{RFOT.SimSetup}(1^\lambda) \to (\mathsf{pp}, \mathsf{td}), \mathcal{A}_H(\mathsf{pp}) \to (\mathsf{pk}, m),\\ & \text{"1 } \textit{with probability } p \textit{ and } 0 \textit{ otherwise"} \to b,\\ & \textit{if } b = 1\textit{, set } m' = m\textit{; else set } m' = \emptyset,\\ & \mathsf{RFOT.SimEncrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{pk}, b, m') \to c, \mathcal{A}_H(c) \to \mathsf{out}\}.\end{aligned}$$

The syntax of the scheme

- Setup: $\mathsf{RFOT.Setup}(1^\lambda) \to \mathsf{pp}$

  On input a security parameter λ, RFOT.Setup outputs a public parameter pp.

- Key generation: $\mathsf{RFOT.Keygen}(\mathsf{pp}, p) \to (\mathsf{pk}, \mathsf{sk})$

  On input a public parameter pp and a probability $p \in [0, 1]$, RFOT.Keygen outputs a public key pk and a secret key sk.

- Encryption: $\mathsf{RFOT.Encrypt}(\mathsf{pp}, \mathsf{pk}, m) \to c$

  On input a public parameter pp, a public key pk, and a message m, RFOT.Encrypt outputs a ciphertext c.

- Decryption: $\mathsf{RFOT.Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to m'$

  On input a public parameter pp, a secret key sk, and a ciphertext c, RFOT.Decrypt outputs a message m'.

The simulator regarding the security for the scheme

- Simulated setup: $\mathsf{RFOT.SimSetup}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})$

  On input a security parameter λ, RFOT.SimSetup outputs a public parameter pp and a trapdoor td.

- Simulated key generation: $\mathsf{RFOT.SimKeygen}(\mathsf{pp}, \mathsf{td}, p) \to (\mathsf{pk}, \mathsf{sk})$

  On input a public parameter pp, a trapdoor td, and a probability $p \in [0, 1]$, RFOT.SimKeygen outputs a public key pk and a secret key sk.

- Simulated encryption: $\mathsf{RFOT.SimEncrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{pk}, b, m') \to c$

  On input a public parameter pp, a trapdoor td, a public key pk, a bit b, and a message m', RFOT.SimEncrypt outputs a ciphertext c.

- Extracting decryption: $\mathsf{RFOT.ExtDecrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{sk}, c) \to m$

  On input a public parameter pp, a trapdoor td, a secret key sk, and a ciphertext c, RFOT.ExtDecrypt outputs a message m.

- Simulated decryption: $\mathsf{RFOT.SimDecrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{sk}, b) \to \mathsf{sk}'$

  On input a public parameter pp, a trapdoor td, a secret key sk, and a bit b, RFOT.SimDecrypt outputs a simulated secret key sk′.

Figure 4: Syntax of the scheme and its simulator [CGL+16]

Definition 6 (Fractional binding (Claim A.8 in [CGL+16]))**.** *The binding property holds when the following distribution ensembles* $\mathcal{E}_B^{\mathsf{real}}$ *and* $\mathcal{E}_B^{\mathsf{ideal}}$ *are computationally indistinguishable for any adversary* $\mathcal{A}_B$*, where*

$$\begin{aligned}\mathcal{E}_B^{\mathsf{real}} = \{(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, c, m) \mid\ & \mathsf{RFOT.Setup}(1^\lambda) \to \mathsf{pp}, \mathsf{RFOT.Keygen}(\mathsf{pp}, p) \to (\mathsf{pk}, \mathsf{sk}),\\ & \mathcal{A}_B(\mathsf{pp}, \mathsf{pk}) \to c, \mathsf{RFOT.Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to m\}\end{aligned}$$

and

$$\begin{aligned}\mathcal{E}_B^{\mathsf{ideal}} = \{(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}', c, m') \mid\ & \text{"1 } \textit{with probability } p \textit{ and } 0 \textit{ otherwise"} \to b,\\ & \mathsf{RFOT.SimSetup}(1^\lambda) \to (\mathsf{pp}, \mathsf{td}),\\ & \mathsf{RFOT.SimKeygen}(\mathsf{pp}, \mathsf{td}, p) \to (\mathsf{pk}, \mathsf{sk}),\\ & \mathcal{A}_B(\mathsf{pp}, \mathsf{pk}) \to c, \mathsf{RFOT.ExtDecrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{sk}, c) \to m,\\ & \mathsf{RFOT.SimDecrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{sk}, b) \to \mathsf{sk}',\\ & \textit{if } b = 1\textit{, set } m' = m\textit{; else set } m' = \emptyset\}.\end{aligned}$$

3.4 Construction

We describe the scheme, the correctness, and simulation-based security.

3.4.1 Scheme

Let us introduce the formal definition of our scheme.

Definition 7 (Ring fractional oblivious transfer)**.** *The ring fractional oblivious transfer (*RFOT*)*

$$\mathsf{RFOT} = (\mathsf{RFOT.Setup}, \mathsf{RFOT.Keygen}, \mathsf{RFOT.Encrypt}, \mathsf{RFOT.Decrypt})$$

is defined in Fig. 5.

Remark 4. *We denote the encryption as* $c = \mathsf{RFOT.Encrypt}(\mathsf{pk}, m; v, e, f)$ *if an encryptor specifies the randomness* v, e, f*.*

3.4.2 Correctness

Let us confirm the correctness of decryption in our scheme. If $l = k$, then

$$\begin{aligned}(c_1 - c_2 s)X^{-k} &= ((yX^{-k}v + 2e + m)X^l - (av + 2f)s)X^{-k}\\ &= ((yX^{l-k} - as)v + 2(eX^l - fs))X^{-k} + mX^{l-k}\\ &= ((y - as)v + 2(eX^l - fs))X^{-k} + m\\ &= 2(e_s v + eX^l - fs)X^{-k} + m\end{aligned}$$

$(e_s v + eX^l - fs)X^{-k}$ is small. We have $m' = (c_1 - c_2 s)X^{-k} \mod 2 = m$.

3.4.3 Security

In this subsection, we state the simulation-based security.

Definition 8 (Simulated ring fractional oblivious transfer)**.** *The simulated ring fractional oblivious transfer*

$$\begin{aligned}\mathsf{RFOT}^{\mathsf{sim}} = (&\mathsf{RFOT.SimSetup}, \mathsf{RFOT.SimKeygen}, \mathsf{RFOT.SimEncrypt},\\ &\mathsf{RFOT.ExtDecrypt}, \mathsf{RFOT.SimDecrypt})\end{aligned}$$

is defined in Fig. 6.

- $\mathsf{RFOT.Setup}(1^\lambda) \to \mathsf{pp}$
  1. Compute $\mathsf{crs}_K \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}_K)$.
  2. Compute $\mathsf{crs}_E \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}_E)$.
  3. Set $a \xleftarrow{\$} \mathbf{R}_q$.
  4. Output $\mathsf{pp} = (\mathsf{crs}_K, \mathsf{crs}_E, a)$.
- $\mathsf{RFOT.Keygen}(\mathsf{pp}, 1/n) \to (\mathsf{pk}, \mathsf{sk})$
  1. Parse pp as $(\mathsf{crs}_K, \mathsf{crs}_E, a)$.
  2. Set $s, e_s \xleftarrow{\$} \chi$.
  3. Compute $y = as + 2e_s$.
  4. Set $k \xleftarrow{\$} \{1, \ldots, n\}$.
  5. Compute $y_0 = yX^{-k}$.
  6. Compute $\pi_K = \mathsf{NIZK.Prove}(\mathsf{crs}_K, (a, y_0), (sX^{-k}, e_sX^{-k}))$.
  7. Set $\mathsf{pk} = (1/n, a, y_0, \pi_K)$.
  8. Set $\mathsf{sk} = (1/n, s, e_s, k, \pi_K)$.
  9. Output key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{RFOT.Encrypt}(\mathsf{pp}, \mathsf{pk}, m) \to c$
  1. Parse pp as $(\mathsf{crs}_K, \mathsf{crs}_E, a)$.
  2. Parse pk as $(1/n, a, y_0, \pi_K)$.
  3. Set $l \xleftarrow{\$} \{1, \ldots, n\}$.
  4. Compute $c_1' = y_0v + 2e + m,\ c_2 = av + 2f$.
  5. Compute $c_1 = c_1'X^l = (y_0v + 2e + m)X^l$.
  6. Compute $\pi_E = \mathsf{NIZK.Prove}(\mathsf{crs}_E, (c_1, c_2, a, y_0), (m, v, e, f))$.
  7. Output $c = (l, c_1, c_2, \pi_E)$.
- $\mathsf{RFOT.Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to m'$
  1. Parse pp as $(\mathsf{crs}_K, \mathsf{crs}_E, a)$.
  2. Parse sk as $(1/n, s, e_s, k, \pi_K)$.
  3. Parse c as (l, c_1, c_2, π_E).
  4. If $l \notin \{1, \ldots, n\}$, then output $\perp$.
  5. If $\mathsf{NIZK.Verify}(\mathsf{crs}_E, (c_1, c_2, a, y_0), \pi_E) = 0$, then output $\perp$.
  6. If $l \neq k$, then output $m' = \emptyset$.
  7. If $l = k$, then output $m' = (c_1 - c_2s)X^{-k} \mod 2$.

Figure 5: Construction of the ring fractional oblivious transfer

- $\mathsf{RFOT.SimSetup}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})$
  1. Compute $(\mathsf{crs}_K, \mathsf{td}_K) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}_K)$.
  2. Compute $(\mathsf{crs}_E, \mathsf{td}_E) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}_E)$.
  3. Compute $(a, \mathbf{B}, g_a, f_a) = \mathsf{IBE.MasterKeygen}(n_d, q)$.
  4. Set $\mathsf{td}_I = (\mathbf{B}, g_a, f_a)$.
  5. Set $\mathsf{pp} = (\mathsf{crs}_K, \mathsf{crs}_E, a)$.
  6. Set $\mathsf{td} = (\mathsf{td}_K, \mathsf{td}_E, \mathsf{td}_I)$.
  7. Output $(\mathsf{pp}, \mathsf{td})$.
- $\mathsf{RFOT.SimKeygen}(\mathsf{pp}, \mathsf{td}, 1/n) \to (\mathsf{pk}, \mathsf{sk})$
  1. Parse pp as $(\mathsf{crs}_K, \mathsf{crs}_E, a)$.
  2. Parse td as $(\mathsf{td}_K, \mathsf{td}_E, \mathsf{td}_I)$.
  3. Set $s, e_s \xleftarrow{\$} \chi$.
  4. Compute $y = as + 2e_s$.
  5. Set $k \xleftarrow{\$} \{1, \ldots, n\}$.
  6. Compute $y_0 = yX^{-k}$.
  7. Compute $\pi_K = \mathsf{NIZK.Simulate}(\mathsf{crs}_K, \mathsf{td}_K, (a, y_0))$.
  8. Set $\mathsf{pk} = (1/n, a, y_0, \pi_K)$.
  9. Set $\mathsf{sk} = (1/n, s, e_s, k, \pi_K)$.
  10. Output key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{RFOT.SimEncrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{pk}, b, m') \to c$
  1. Parse pp as $(\mathsf{crs}_K, \mathsf{crs}_E, a)$.
  2. Parse td as $(\mathsf{td}_K, \mathsf{td}_E, \mathsf{td}_I)$.
  3. Parse pk as $(1/n, a, y_0, \pi_K)$.
  4. Set $l \xleftarrow{\$} \{1, \ldots, n\}$.
  5. If $b = 0$, then
     - Set $(c_1, c_2) \xleftarrow{\$} \mathbf{C}$.
     - Compute $\pi_E = \mathsf{NIZK.ExtSimulate}(\mathsf{crs}_E, \mathsf{td}_I, (c_1, c_2, a, y_0))$.
  6. If $b = 1$, then
     - Compute $c_1' = y_0 v + 2e + m',\ c_2 = av + 2f,\ c_1 = c_1' X^l$.
     - Compute $\pi_E = \mathsf{NIZK.Prove}(\mathsf{crs}_E, (c_1, c_2, a, y_0), (m', v, e, f))$.
  7. Output $c = (l, c_1, c_2, \pi_E)$.
- $\mathsf{RFOT.ExtDecrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{sk}, c) \to m$
  1. Parse sk as $(1/n, s, e_s, k, \pi_K)$.
  2. Parse c as (l, c_1, c_2, π_E).
  3. Output $m = (c_1 X^{-l} - c_2 s X^{-k}) \mod 2$.
- $\mathsf{RFOT.SimDecrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{sk}, b) \to \mathsf{sk}'$
  1. Parse pp as $(\mathsf{crs}_K, \mathsf{crs}_E, a)$.
  2. Parse td as $(\mathsf{td}_K, \mathsf{td}_E, \mathsf{td}_I)$.
  3. Parse td_I as $(\mathbf{B}, g_a, f_a)$.
  4. Parse sk as $(1/n, s, e_s, k, \pi_K)$.
  5. If $b = 0$,
     - Compute $(s', e_s') = \mathsf{IBE.Extract}(\mathbf{B}, y_0)$
     - Set $\mathsf{sk}' = (1/n, s', e_s', k, \pi_K)$.
  6. If $b = 1$, then output sk.

Figure 6: Simulator of the ring fractional oblivious transfer

Remark 5. *Let us confirm that one can extract the plaintext with* RFOT.ExtDecrypt*, even if* $l \neq k$.

$$\begin{aligned} c_1X^{-l} - c_2sX^{-k} &= (y_0v + 2e + m) - (av + 2f)sX^{-k} \\ &= (yX^{-k}v + 2e + m) - (av + 2f)sX^{-k} \\ &= (y - as)vX^{-k} + 2(e - fsX^{-k}) + m \\ &= 2(e_sv + e - fs)X^{-k} + m \end{aligned}$$

$(e_sv + e - fs)X^{-k}$ *is small. We obtain* $(c_1X^{-l} - c_2sX^{-k}) \mod 2 = m$.

Lemma 3. *The below two distribution ensembles are computationally indistinguishable:*

$$\{\mathsf{pp} \mid \mathsf{RFOT.Setup}(1^\lambda) \to \mathsf{pp}\} \text{ and } \{\mathsf{pp} \mid \mathsf{RFOT.SimSetup}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})\}.$$

Proof. We suppose that an adversary distinguishes the two pp. Let us confirm the variable $a \in \mathsf{pp}$. The one is randomly sampled from $\mathbf{R}_q$, and the other is equal to $g_a \cdot f_a^{-1}$. If the adversary distinguishes the two a, then the adversary could break the DSPR assumption regarding Definition 2. This is a contradiction. We conclude that RFOT.Setup and RFOT.SimSetup are computationally indistinguishable. □

Lemma 4. *The below two distribution ensembles are computationally indistinguishable:*

$$\{(\mathsf{pp}, \mathsf{pk}) \mid \mathsf{RFOT.Setup}(1^\lambda) \to \mathsf{pp},\ \mathsf{RFOT.Keygen}(\mathsf{pp}, p) \to (\mathsf{pk}, \mathsf{sk})\} \text{ and}$$
$$\{(\mathsf{pp}, \mathsf{pk}) \mid \mathsf{RFOT.SimSetup}(1^\lambda) \to (\mathsf{pp}, \mathsf{td}),\ \mathsf{RFOT.SimKeygen}(\mathsf{pp}, \mathsf{td}, p) \to (\mathsf{pk}, \mathsf{sk})\}.$$

Proof. The difference between RFOT.Keygen and RFOT.SimKeygen are each function that outputs each proof. One is NIZK.Prove. The other is NIZK.Simulate. The proofs by NIZK.Prove and NIZK.Simulate are computationally indistinguishable because NIZK scheme satisfies zero-knowledge. We conclude that the lemma holds. □

Lemma 5 (Fractional hiding)**.** *The scheme of Definition 7 holds the fractional hiding of Definition 5.*

Proof. Let us compare $\mathcal{E}_H^{\mathsf{real}}$ with $\mathcal{E}_H^{\mathsf{ideal}}$. There are two differences between the two ensembles. The one is NIZK.Prove or NIZK.ExtSimulate. These proofs are indistinguishable from Lemma 2.

The other difference is the encryption or the ramdom sampling. $c_1' = y_0v+2e+m', c_1 = c_1'X^l, c_2 = av+2f$ in $\mathcal{E}_H^{\mathsf{real}}$ or $(c_1, c_2) \xleftarrow{\$} \mathbf{C}$ in $\mathcal{E}_H^{\mathsf{ideal}}$. If an adversary $\mathcal{A}$ can distinguish $\mathcal{E}_H^{\mathsf{real}}$ and $\mathcal{E}_H^{\mathsf{ideal}}$, then $\mathcal{A}$ could distinguish the true ciphertext (c_1, c_2) from the randomly sampled element $(c_1, c_2) \xleftarrow{\$} \mathbf{C}$. This situation goes against the RLWE assumption regarding Definition 1. We conclude that $\mathcal{E}_H^{\mathsf{real}}$ and $\mathcal{E}_H^{\mathsf{ideal}}$ are indistinguishable. The scheme holds the fractional hiding.

□

Lemma 6 (Fractional binding)**.** *The scheme of Definition 7 holds the fractional hiding of Definition 6.*

Proof. Let us compare $\mathcal{E}_B^{\mathsf{real}}$ with $\mathcal{E}_B^{\mathsf{ideal}}$. RFOT.Keygen and RFOT.SimKeygen are computationally indistinguishable because of Lemma 4. RFOT.ExtDecrypt outputs a unique plaintext m for a valid ciphertext even if $l \neq k$. The simulated secret key sk' is distributed independently of the actual secret key sk. When $b = 1$, output the actual secret key itself. When $b = 0$, a different secret key (s', e_s') is output, satisfying the relationship $y_0 = as' + 2e_s'$ with the fixed public key (a, y_0) with IBE.Extract. We conclude that the scheme holds the fractional binding. □

4 Protocol and security

In this section, we present the protocols, the ideal functionalities, the theorems that each protocol realizes each ideal functionality, and the proofs regarding the puzzle solver and puzzle promise. Let us outline the security discussion. We present a simulation-based proof of the real/ideal world paradigm. We build simulators in the cases of each corrupt participant. We discuss the indistinguishability of the game sequence by using a hybrid argument. The discussion is based on TumbleBit [HAB+17, HAB+16]. We also combine the DAM scheme [CGL+17, CGL+16] regarding the puzzle solver protocol related to RFOT.

4.1 Puzzle solver protocol

We present the puzzle solver protocol, the ideal functionality, the theorem, and the proof. We show the puzzle solver protocol in Fig. 7 and the ideal functionality in Fig. 8. We also show the simulators in Fig. 9 and Fig. 10. The simulator in Fig. 9 corresponds to the corrupt Alice. The simulator in Fig. 10 corresponds to the corrupt the tumbler. Let us present the quick look at the protocol, the overview of the ideal functionality, and proof sketch as the below paragraphs.

Quick look at the protocol. We change the procedure by the RSA encryption in TumbleBit [HAB+17, HAB+16], into the one by the RLWE encryption. Also, we combine RFOT with the procedure in TumbleBit. The tumbler and Alice interact at the protocol. We use a cut-and-choose technique in the procedure. Alice creates real puzzles and fake puzzles, respectively. At Step 1, She further masks the masked puzzle (ciphertext) she receives for the real puzzles. She creates ciphertext by randomly choosing a plaintext by herself. She adds this with the puzzle. At Step 2, she creates a ciphertext from randomly chosen plaintext for the fake puzzles. She does not add the received puzzle to the ciphertext. At Step 3, she mixes these puzzles and executes the RFOT encryption. She permutes and mixes real puzzles and fake puzzles. These puzzles are just RLWE ciphertexts. She chooses the integer l randomly and shifts all puzzles by the single amount l. In this way, she produces the RFOT ciphertext from the RLWE ciphertexts. She passes the puzzles to the tumbler. At Step 4, the tumbler decrypts all the puzzles it receives. If the puzzles are miss/error, then the tumbler cannot obtain the correct message. In this case, the tumbler aborts. (Even if the tumbler chooses s randomly, she will notice that it does not match her plaintext at Step 7. Moreover, the tumbler does not know which puzzle is in the fake set at this stage.) The tumbler creates a ciphertext c for the decrypted message by using symmetric key encryption. The tumbler also selects the hash value h of k randomly. The tumbler sends these (c, h) to her for commitment. At Step 5, upon receiving the commitment (c, h), she informs the tumbler of which puzzle belongs to the fake set. She sends a message and randomness to the tumbler for opening. At Step 6, the tumbler verifies the received message and randomness, by matching the ciphertext received earlier. If all can be verified correctly, the tumbler sends k belonging to the fake set to her. At Step 7, she obtains s by the decryption process from the received k and the ciphertext c of the symmetric key encryption. If s matches the original plaintext, she continues. Otherwise, she aborts. At Step 8, she fills in the transaction on the blockchain. Both the preimages of h and the tumbler's signature can fulfill this transaction. She sends the received puzzle to the tumbler and the message and the randomness in the real set. At Step 9, the tumbler verifies the ciphertext from the received message, randomness, and puzzle. If it is OK, the tumbler fills k in the transaction offered at Step 10. At Step 11, she gets k from the transaction, executes the symmetric key encryption, and gets s. She removes her message by XOR operation and gets the message of the received puzzle y.

Overview of the ideal functionality. We combine the functionality of TumbleBit and the functionality of fractional message transfer (FMT) in the DAM scheme for the ideal functionality. First, we incorporate mutual fairness into functionalities, following TumbleBit. Fairness for Alice means that the tumbler earns one coin if and only if she gets the correct answer. Fairness for the tumbler means that the protocol executes decrypting the puzzle selected by her. Upon setup, the functionality receives the key from the tumbler and verifies if it is valid. If the verification is successful, the functionality sends the key to her and Simulator. At the evaluation, the functionality receives the request containing the ciphertext from her and sends it to the tumbler. Also, the functionality receives the plaintext result from the tumbler, sends it to Alice, and pays to the tumbler.

Next, we append a probabilistic payment to the functionality, following the DAM scheme. The functionality conveys a valid message with probability p. Alice sends to the functionality at the request, including bit b and probability $p_{\mathcal{A}}$. The functionality confirms the bit b to determine whether there is an error. The tumbler sends the key to the functionality together with the probability $p_{\mathcal{T}}$. The functionality judges that a lottery is an error in the case that $b = 0$ or the probabilities do not match. If $b = 1$ and the probabilities are equal, the functionality runs that the ciphertext is a hit with probability p and is a failure with probability $1 - p$. The functionality stores the judged value in $\mathcal{Q}$ as a pair $(\mathsf{sid}, \mathsf{mid})$. Let us confirm the behavior after the functionality receives the evaluation result from the tumbler. The functionality searches $\mathcal{Q}$ for mid paired with sid. If mid is a hit and the message x from the tumbler is valid, the functionality sends x to her and

pays the tumbler. Otherwise, the tumbler refunds to her as a missing lottery or an error.

Proof sketch. In the case of corrupt Alice, she cannot learn more than the decrypted message.

Let us confirm the case of corrupt the tumbler. The protocol adopts a cut-and-choose technique like TumbleBit. Therefore, the tumbler must present invalid ciphertexts to the real set while correctly responding to the fake set. The tumbler cannot obtain the information regarding the fake set and the real set before her opening. The probability of corrupt tumbler's success is negligible.

Theorem 1. *Let λ be a security parameter. Let $n_d \geq 2\lambda$. Assume that H and H^{prg} are independent random oracles, and the RLWE and DSPR problems are hard. Then, the protocol in Fig. 7 securely realizes the functionality $\mathcal{F}_{\mathsf{solver}}$ in Fig. 8 with the following security guarantees. The security for $\mathcal{T}$ is $1 - \mathsf{negl}(\lambda)$ and the security for $\mathcal{A}$ is $1 - 1/\binom{\mu+\eta}{\eta} - \mathsf{negl}(\lambda)$.*

Proof. We divide the proof into two cases. One is the case of the corrupt Alice. The other is the case of the corrupt the tumbler. The simulator S in Fig. 9 corresponds to the corrupt Alice $\mathcal{A}^*$. The simulator S in Fig. 10 corresponds to the corrupt the tumbler $\mathcal{T}^*$.

Case that Alice is corrupt. We adopt a hybrid argument and confirm the indistinguishability between the real world and the ideal world. The simulator S in Fig. 9 plays a role of the corrupt Alice $\mathcal{A}^*$.

$\ni_0$: This is the real game.

$\ni_1$: The difference between $\ni_0$ and $\ni_1$ is how the key k_i is computed. If $x' = \mathsf{RFOT.Decrypt}(\mathsf{sk}, \beta_i) \neq \emptyset$ with salt, the simulator in this hybrid $\ni_1$ sends $k_i' = \mathsf{Equivocate}(c_i, x, h_i)$, instead of k_i. If $x' = \emptyset$, then both games receive refund and halt. From the assumption regarding the random oracle, c_i and h_i statistically hide the encrypted message and the preimage. The probability of the event $\mathsf{Collision}$ is negligible. So the view in $\ni_1$ is indistinguishable from the view in $\ni_0$.

$\ni_2$: The difference between $\ni_1$ and $\ni_2$ is what one inputs $\mathsf{Equivocate}$. The simulator in this hybrid $\ni_2$ computes $\mathsf{Equivocate}(c_i, \rho_i, h_i)$ instead of $\mathsf{Equivocate}(c_i, x + r_i, h_i)$. Then x is a valid message of y. x is taken from $\mathcal{F}_{\mathsf{solver}}$ after Adv has sent T_{puzzle}. Both games receive refund instead of x and halt. The view in $\ni_2$ is generated by just a single value x. The view in $\ni_2$ is indistinguishable from the view in $\ni_1$. The simulator in $\ni_2$ corresponds to Simulator S in Fig. 9.

Case that the tumbler is corrupt. The simulator S in Fig. 10 plays a role of the corrupt the tumbler $\mathcal{T}^*$. Because of the RLWE assumption, all ciphertexts $\beta_1, \ldots, \beta_{\mu+\eta}$ are uniformly distributed. They reveal no information about the sets F, R. Furthermore, H, H^{prg} are modeled as a random oracle. The encryption of the key is binding. It is impossible for an adversary Adv to change the values after the sets F, R are revealed. Then, we denote the event that an adversary presumes the set F as the event BAD. The probability of the event BAD is as follows.

$$\Pr[\mathsf{BAD}] = \frac{1}{\binom{\mu+\eta}{\eta}} + \frac{1}{2^{\lambda_1}}$$

The difference between the transcript by the simulator S in Fig. 10 in the ideal world and the one in the real world is if the event BAD happens or not. The two worlds are distinguishable with the negligible probability $\Pr[\mathsf{BAD}]$.

We conclude that the protocol in Fig. 7 securely realizes the functionality $\mathcal{F}_{\mathsf{solver}}$ in Fig. 8. □

4.2 Puzzle promise protocol

We present the puzzle promise protocol, the ideal functionality, the theorem, and the proof. We show the protocol in Fig. 11 and the ideal functionality in Fig. 12. We also show the simulators in Fig. 13 and Fig. 14. The simulator in Fig. 13 corresponds to the corrupt Bob. The simulator in Fig. 14 corresponds to the corrupt the tumbler. Let us present the quick look at the protocol, the overview of the ideal functionality, and proof sketch as the below paragraphs.

Public input: pk. π_K in pk proves validity of pk in a one time setup phase.

Alice $\mathcal{A}$. Input: puzzle y		the tumbler $\mathcal{T}$. Secret input: sk
1. Prepare real puzzles R $\mathsf{salt} \xleftarrow{\$} \{0,1\}^\lambda$ For $j = 1, \ldots, \mu$: $r_j \xleftarrow{\$} \{0,1\}^\lambda, v_j, e_j, f_j \xleftarrow{\$} \chi$ $d_j = \mathsf{RLWE.Enc}(\mathsf{pk}, \mathsf{salt}\|\|r_j; v_j, e_j, f_j) + y$		
2. Prepare fake values F For $i = 1, \ldots, \eta$: $\rho_i \xleftarrow{\$} \{0,1\}^\lambda, v_i, e_i, f_i \xleftarrow{\$} \chi$ $\delta_i = \mathsf{RLWE.Enc}(\mathsf{pk}, \mathsf{salt}\|\|\rho_i; v_i, e_i, f_i)$		
3. Mix sets and RFOT Permute $\{d_1, \ldots, d_\mu, \delta_1, \ldots, \delta_\eta\}$ randomly and obtain $\{\beta'_1, \ldots, \beta'_{\mu+\eta}\}$ $l \xleftarrow{\$} \{1, \ldots, n\}$ For $i = 1, \ldots, \mu + \eta$: Parse β'_i as (c'_1, c_2) and set $c_1 = c'_1 X^l$ Set $\beta_i = (c'_1, c_2)$ Let R be the indices of the d_i Let F be the indices of the δ_i	$\xrightarrow{\mathsf{salt}, l, \beta_1, \ldots, \beta_{\mu+\eta}}$	
		4. Evaluation. For $i = 1, \ldots, \mu + \eta$: Evaluate β_i: $s_i = \mathsf{RFOT.Decrypt}(\mathsf{sk}, \beta_i)$ Check if $s_i = \emptyset$ with salt. If not, abort. Encrypt the result s_i: $k_i \xleftarrow{\$} \{0,1\}^{\lambda_1}$ $c_i = H^{\mathsf{prg}}(k_i) \oplus s_i$ Commit to the keys: $h_i = H(k_i)$
	$\xleftarrow{(c_1,h_1),\ldots,(c_{\mu+\eta},h_{\mu+\eta})}$	
5. Identify fake set.	$\xrightarrow{F, \rho_i, v_i, e_i f_i \ \forall i \in F}$	
		6. Check fake set. For all $i \in F$: Verify $\beta_i =$ $\mathsf{RFOT.Encrypt}(\mathsf{pk}, \mathsf{salt}\|\|\rho_i; v_i, e_i, f_i, l)$ If yes, reveal k_i $\forall i \in F$. Else abort.
	$\xleftarrow{k_i \ \forall i \in F}$	
7. Check fake set. For all $i \in F$: Verify $h_i = H(k_i)$ Decrypt $s_i = H^{\mathsf{prg}}(k_i) \oplus c_i$ Verify $s_i = \rho_i$ Abort if any check fails		
8. Post transaction T_{puzzle} T_{puzzle} offers one coin within timewindow tw_1 under condition "the fulfilling transaction is signed by $\mathcal{T}$ and has preimages of h_j $\forall j \in R$".	$\xrightarrow{y, r_j, v_j, e_j, f_j \ \forall j \in R}$	
		9. Check β_i unblind to y $\forall j \in R$ For all $j \in R$: Verify $\beta_i =$ $\mathsf{RFOT.Encrypt}(\mathsf{pk}, \mathsf{salt}\|\|r_j; v_j, e_j, f_j, l) + y$ If not, abort.
		10. Post transaction T_{puzzle} T_{puzzle} contains k_j $\forall j \in R$
11. Obtain puzzle solution. For $j \in R$: Learn k_j from T_{puzzle} Decrypt $s_j = H^{\mathsf{prg}}(k_j) \oplus c_j$ Verify $\beta_j = \mathsf{RFOT.Encrypt}(\mathsf{pk}, \mathsf{salt}\|\|s_j; v_j, e_j, f_j, l) + y$ Obtain solution $s_j + r_j \mod 2$		

Figure 7: Puzzle solver protocol. We model H and H^{prg} as random oracles.

Parties.

- $\mathcal{A}, \mathcal{T}$ and adversary $\mathcal{S}$.

Setup.

- Receive $(\mathsf{Setup}, \mathsf{pk}, \mathsf{sk}, p_{\mathcal{T}})$ from $\mathcal{T}$.
- If pk or sk are invalid, then
  - do nothing.
- Else,
  - Send $(\mathsf{Setup}, \mathsf{pk})$ to $\mathcal{A}$ and $\mathcal{S}$.

Evaluation.

- On input $(\mathsf{request}, \mathsf{sid}, y, 1\mathsf{coin}, p_{\mathcal{A}}, b)$ from $\mathcal{A}$:
  - If $b = 1$ and $p_{\mathcal{A}} = p_{\mathcal{T}}$,
    - Set $\mathsf{mid} = \mathsf{hit}$ with probability $p_{\mathcal{A}}$ or $\mathsf{mid} = \mathsf{miss}$ with probability $1 - p_{\mathcal{A}}$.
  - Else if $b = 0$ or $p_{\mathcal{A}} \neq p_{\mathcal{T}}$, set $\mathsf{mid} = \mathsf{error}$.
  - Send $(\mathsf{request}, \mathsf{sid}, \mathcal{A}, y)$ to $\mathcal{T}$.
  - Append $(\mathsf{sid}, \mathsf{mid})$ to $\mathcal{Q}_{\mathsf{id}}$.
  - Start counter $tw_{\mathsf{sid}} = 0$.
- On input $(\mathsf{evaluate}, \mathsf{sid}, \mathcal{A}, x)$ from $\mathcal{T}$:
  - Obtain mid corresponding sid from $\mathcal{Q}_{\mathsf{id}}$.
  - If $\mathsf{mid} = \mathsf{hit}$ and $x \neq \emptyset$, then
    - Send (sid, x) to $\mathcal{A}$.
    - Send $(\mathsf{payment}, \mathsf{sid}, 1\mathsf{coin})$ to $\mathcal{T}$.
  - Else, send $(\mathsf{refund}, \mathsf{sid}, 1\mathsf{coin})$ to $\mathcal{A}$.
- If $tw_{\mathsf{sid}} = tw$, send $(\mathsf{refund}, \mathsf{sid}, 1\mathsf{coin})$ to $\mathcal{A}$.

Figure 8: Ideal functionality $\mathcal{F}_{\mathsf{solver}}$

1. Receive $\mathsf{salt}, l, \beta_1, \ldots, \beta_{\mu+\eta}$ from Adv. Choose $c_i \xleftarrow{\$} \{0,1\}^{\lambda}$ and $h_i \in \{0,1\}^{\lambda_2}$ for $i \in [\mu+\eta]$. Send them to Adv.
2. Receive F, ρ_i, v_i, e_i, f_i for $i \in F$. For all $i \in F$, check if $\beta_i = \mathsf{RFOT.Encrypt}(\mathsf{pk}, \mathsf{salt}||\rho_i; v_i, e_i, f_i, l)$. If check fails, output whatever Adv outputs, send $(\mathsf{request}, \mathsf{sid}, y, \mathsf{1coin}, p_{\mathcal{A}}, b=0)$ to $\mathcal{F}_{\mathsf{solver}}$ and halt. Else, run $k_i' = \mathsf{Equivocate}(c_i, \rho_i, h_i)$. Send k_i' for $i \in F$ to Adv.
3. Receive y, r_i, v_i, e_i, f_i for $i \in R$. Check if $\beta_i = \mathsf{RFOT.Encrypt}(\mathsf{pk}, \mathsf{salt}||r_i; v_i, e_i, f_i, l) + y$ for all $i \in R$. If the check succeeds, transaction T_{puzzle} is correctly formed, execute as follows.
   - Send $(\mathsf{request}, \mathsf{sid}, y, \mathsf{1coin}, p_{\mathcal{A}}, b=1)$ to $\mathcal{F}_{\mathsf{solver}}$ and obtain x or refund.
   - If obtain x,
     - Run $k_i' = \mathsf{Equivocate}(c_i, x + r_i, h_i)$ with $i \in R$.
     - Send transaction T_{solve} with values k_i'.
   - Else if obtain refund, halt.

   Else, checks have failed so output whatever Adv outputs, send $(\mathsf{request}, \mathsf{sid}, y, \mathsf{1coin}, p_{\mathcal{A}}, b=0)$ to $\mathcal{F}_{\mathsf{solver}}$ and halt.

Procedure random oracle simulation for $\mathcal{Q}_H, \mathcal{Q}_{H^{\mathsf{prg}}}$ is as follows:
Receive query q for $H(\mathsf{resp.}, H^{\mathsf{prg}})$:

1. If query $q \in \mathcal{Q}_H(\mathsf{resp.}, H^{\mathsf{prg}})$, retrieve entry (q, a) from the set and output a.
2. Else $a \xleftarrow{\$} \{0,1\}^{\lambda_2}(\mathsf{resp.}, \lambda_1)$, and (q, a) to $\mathcal{Q}_H(\mathsf{resp.}, H^{\mathsf{prg}})$ and output a.

Procedure $\mathsf{Equivocate}(c_i, m_i, h_i)$ is stated as below:

1. $k_i' \xleftarrow{\$} \{0,1\}^{\lambda_1}$. If $k_i' \in \mathcal{Q}_H$ or $\mathcal{Q}_{H^{\mathsf{prg}}}$, output $\mathsf{Collision}$ and abort.
2. Compute $a_i = c_i \oplus m_i$, then append (k_i', a_i) to $\mathcal{Q}_{H^{\mathsf{prg}}}$.
3. Append (k_i', h_i) to $\mathcal{Q}_H$.
4. Output k_i'.

Figure 9: Simulator for the puzzle solver protocol in the case that Alice is corrupt

1. If pk and sk are valid, S receives $(\mathsf{request}, \mathsf{sid}, \mathcal{A}, y)$ from $\mathcal{F}_{\mathsf{solver}}$.
2. Compute $x' = \mathsf{RFOT.ExtDecrypt}(\mathsf{pp}, \mathsf{td}, \mathsf{sk}, y)$ and extract x from $x' = \mathsf{salt}||x$.
3. Receiving $\{k_i\}$ for all $i \in F$ from Adv, S checks if all $\{c_i\}_{i \in F}$ are correct. If yes, S sends message $(\mathsf{evaluate}, \mathsf{sid}, \mathcal{A}, x)$ to $\mathcal{F}_{\mathsf{solver}}$. Then, $\mathcal{F}_{\mathsf{solver}}$ sends the puzzle solution x or refund to $\mathcal{A}$. Meanwhile, S sends T_{puzzle} to Adv.
4. S receives T_{solve} from Adv. If all keys $\{k_i\}$ $\forall i \in R$ decrypt ciphertexts c_i, not containing valid puzzle solutions, then S outputs BAD and aborts. Else, S outputs whatever Adv outputs and halts.

Figure 10: Simulator for the puzzle solver protocol in the case that the tumbler is corrupt

Quick look at the protocol. The tumbler and Bob interact at the protocol. As with the puzzle solver protocol, we also use the cut-and-choose technique. At Step 1, the tumbler sets up the escrow transaction. From Step 2 to Step 4, Bob creates real and fake hash values and sends the shuffled hash values to the tumbler. At Step 5, the tumbler signs everything does corresponding puzzles and sends these with promises. At Step 6, he opens fake set to the tumbler. At Step 7, the tumbler verifies them and presents to him the puzzle answers corresponding to the fake values. At Step 8, he verifies these answers. Note that we must arrange a situation where it is sufficient for him to send one of the real puzzles to Alice. At Step 9, the tumbler sends the difference from other elements and proves that the ciphertext is zero. The tumbler uses the non-interactive zero-knowledge proof for the relation $\mathcal{R}_0$. At Step 10 and 12, he can agree that it is sufficient to send either one to her by verifying the proof and the difference. At Step 11, the tumbler posts the transaction.

The difference from TumbleBit is the adoption of the RLWE encryption instead of RSA encryption. The RLWE encryption is probabilistic encryption, unlike the RSA encryption. One can verify the ciphertext by its plaintext and randomness by reproducing and verifing it with the encryption algorithm. We make the tumbler encrypt this randomness with symmetric key encryption and pass it to Bob. Let the key of the symmetric key encryption be the plaintext itself. Even if we adopt the RLWE encryption, we can use the only plaintext as the answer to the puzzle by this trick.

Overview of the ideal functionality. We show the ideal functionality in Fig. 12. This ideal functionality is the same as TumbleBit. Let us outline the ideal functionality briefly. The ideal functionality plays a role as a trusted third party. The functionality signs a transaction by calling the signature oracle. It is fairness for Bob that he obtains a promise that contains a valid signature for at least one genuine transaction. It is fairness for the tumbler that he has no knowledge of anything but the fake transaction signature. Upon receiving fake and real transactions from him, the functionality stores them and sends fake transactions to the tumbler. Upon receiving a promise from the tumbler, the functionality signs the fake transactions. The functionality records fake transaction signatures and the promises to real transactions. Finally, the functionality sends the promises to him. Upon receiving the signature verification from any party, the functionality tells them that the functionality has already recorded the signatures on the fake transactions. Since the functionality does not store real transactions, the functionality records them.

Proof sketch. Regarding corrupt Bob, we change RSA encryption in the simulator in TumbleBit into RLWE encryption. We show the simulator in Fig. 13. The simulator in Fig. 14 is the same as TumbleBit for corrupt the tumbler. The proof in TumbleBit holds in the same way.

Theorem 2. *Let λ be a security parameter. Let $n_d \geq 2\lambda$. Assume that H, H' and H^{shk} are independent random oracles, and the RLWE and DSPR problems are hard. Then, the protocol in Fig. 11 securely realizes the functionality $\mathcal{F}_{\mathsf{promise\ sign}}$ in Fig. 12 with the following security guarantees. The security for $\mathcal{T}$ is $1-\mathsf{negl}(\lambda)$ and the security for $\mathcal{B}$ is $1 - 1/\binom{\mu+\eta}{\eta} - \mathsf{negl}(\lambda)$.*

Proof. We divide the proof into two cases. One is the case of corrupt Bob. The other is the case of the corrupt the tumbler.

Case that Bob is corrupt. We confirm the indistinguishability between the real world and the ideal world with a hybrid argument. The simulator S in Fig. 13 plays a role of the corrupt Bob $\mathcal{B}^*$.

$\ni_0$: This is the real world.

$\ni_{0.5}$: When the simulator receives h_R, h_F and $\{\beta_1, \ldots, \beta_{\mu+\eta}\}$ from $\mathcal{B}^*$, the simulator checks the set of queries $\mathcal{Q}_H$ and extracts the pair $(\mathsf{salt}||R, h_R)$ and $(\mathsf{salt}||F, h_F)$. If there is no pair regarding h_R, h_F, the simulator aborts. This abort is the difference between $\ni_0$ and $\ni_{0.5}$. The probability of the abort is $1/2^{\lambda_2}$, so the two games are statistically close.

$\ni_1$: Instead of computing $c_i = H^{\mathsf{shk}}(\epsilon_i) \oplus \sigma_i$, the simulator chooses $c_i \xleftarrow{\$} \{0,1\}^s$ and stores the pair $(\epsilon_i, c_i \oplus \sigma_i)$ in $\mathcal{Q}_{H^{\mathsf{shk}}}$ in $\ni_1$. The both games $\ni_{0.5}$ and $\ni_1$ are statistically close because H^{shk} is unpredictable.

$\ni_2$: Instead of computing $z_{j_i} = \mathsf{RLWE.Enc}(\mathsf{pk}, \epsilon_{j_i})$, the simulator randomly chooses $z_{j_1}, d_2, \ldots, d_\mu$ from $\{0,1\}^\lambda$ and computes $z_{j_i} = \mathsf{RLWE.Enc}(\mathsf{pk}, d_i) + z_{j_{i-1}}$. Let us check for $\mathsf{RLWE\ failure}$. According to the RLWE assumption, the ciphertext z_{j_i} cannot be distinguished from a uniformly random value. The probability that

Public input: $(\mathsf{pk}, \mathsf{PK}_{\mathcal{T}}^{eph})$. π_K in pk proves validity of pk in a one time setup phase.
$\mathcal{T}$ chooses a fresh ephemeral ECDSA-Secp256k1 key $(\mathsf{SK}_{\mathcal{T}}^{eph}, \mathsf{PK}_{\mathcal{T}}^{eph})$.

Bob $\mathcal{B}$	the tumbler $\mathcal{T}$. Secret input: sk
	1. Set up $T_{\mathsf{escr}(\mathcal{T},\mathcal{B})}$ Sign but do not post transaction $T_{\mathsf{escr}(\mathcal{T},\mathcal{B})}$ timelocked for tw_2 offering one coin under the condition: "the fulfilling transaction
$\xleftarrow{T_{\mathsf{escr}(\mathcal{T},\mathcal{B})}}$	is signed under key $\mathsf{PK}_{\mathcal{T}}^{eph}$ and key $\mathsf{PK}_{\mathcal{B}}$"
2. Prepare μ real unsigned $T_{\mathsf{escr}(\mathcal{T},\mathcal{B})}$. For $i \in 1, \ldots, \mu$: $\rho_i \xleftarrow{\$} \{0,1\}^\lambda,\ T^i_{\mathsf{escr}(\mathcal{T},\mathcal{B})} = \mathsf{CashOutFormat}(\rho_i),\ ht_i = H'(T^i_{\mathsf{fulfill}})$.	
3. Prepare fake set. For $i \in 1, \ldots, \eta$: $r_i \xleftarrow{\$} \{0,1\}^\lambda,\ ft_i = H'(\mathsf{FakeFormat}\|\|r_i)$.	
4. Mix sets. Permute $\{ft_1, \ldots, ft_\eta, ht_1, \ldots, ht_\mu\}$ randomly and obtain $\{\beta_1, \ldots, \beta_{\mu+\eta}\}$ $\xrightarrow{\beta_1,\ldots,\beta_{\mu+\eta}}$ Let R be the indices of the ht_i Let F be the indices of the ft_i Choose $\mathsf{salt} \xleftarrow{\$} \{0,1\}^\lambda$ $h_R = H(\mathsf{salt}\|\|R),\ h_F = H(\mathsf{salt}\|\|F)$ $\xrightarrow{h_R, h_F}$	
	5. Evaluation. For $i = 1, \ldots, \mu + \eta$: ECDSA sign β_i to get $\sigma_i = \mathsf{Sig}(\mathsf{SK}_{\mathcal{T}}^{eph}, \beta_i)$ $\epsilon_i \xleftarrow{\$} \{0,1\}^\lambda, v_i, e_i, f_i \xleftarrow{\$} \chi$. Create promise $c_i = H^{\mathsf{shk}}(\epsilon_i) \oplus \sigma_i$ Encrypt $\bar{v}_i = \mathsf{Enc}(v_i; \epsilon_i), \bar{e}_i = \mathsf{Enc}(e_i; \epsilon_i)$ and $\bar{f}_i = \mathsf{Enc}(f_i; \epsilon_i)$
$(c_1, z_1, \bar{v}_1, \bar{e}_1, \bar{f}_1), \ldots,$ $\xleftarrow{(c_{\mu+\eta}, z_{\mu+\eta}, \bar{v}_{\mu+\eta}, \bar{e}_{\mu+\eta}, \bar{f}_{\mu+\eta})}$	Create puzzle $z_i = \mathsf{RLWE.Enc}(\mathsf{pk}, 0^\lambda \|\| \epsilon_i; v_i, e_i, f_i)$
6. Identify fake set. $\xrightarrow{R, F, r_i\ \forall i \in F, \mathsf{salt}}$	
	7. Check fake set. Check $h_R = H(\mathsf{salt}\|\|R)$ and $h_F = H(\mathsf{salt}\|\|F)$ For all $i \in F$: Verify $\beta_i = H'(\mathsf{FakeFormat}\|\|r_i)$
$\xleftarrow{\epsilon_i\ \forall i \in F}$	Abort if any check fails
8. Check fake set. For all $i \in F$: Validate $\epsilon_i \in \{0,1\}^\lambda$ Decrypt $v_i = \mathsf{Dec}(\bar{v}_i; \epsilon_i), e_i = \mathsf{Dec}(\bar{e}_i; \epsilon_i), f_i = \mathsf{Dec}(\bar{f}_i; \epsilon_i)$ Validate puzzle $z_i = \mathsf{RLWE.Enc}(\mathsf{pk}, 0^\lambda \|\| \epsilon_i; v_i, e_i, f_i)$ Validate promise c_i: Decrypt $\sigma_i = H^{\mathsf{shk}}(\epsilon_i) \oplus c_i$ Verify σ_i such that $(\mathsf{PK}_{\mathcal{T}}^{eph}, H'(ft_i), \sigma_i) = 1$ Abort if any check fails	
	9. Prepare differences For $R = \{j_1, \ldots, j_\mu\}$: Set $d_2 = \epsilon_{j_2} - \epsilon_{j_1}, \ldots, d_\mu = \epsilon_{j_\mu} - \epsilon_{j_{\mu-1}}$
$\xleftarrow{d_2,\ldots,d_\mu,\pi_2,\ldots,\pi_\mu}$	Set π_j proving $z_j + d_j \in \mathbf{C}_0$
10. Difference test. For $R = \{j_1, \ldots, j_\mu\}$: Verify π_j Abort if any check fails	
	11. Post transaction $T_{\mathsf{escr}(\mathcal{T},\mathcal{B})}$
12. Start Payment Phase. Set $z_t = z_{j_1}$. $m_b \xleftarrow{\$} \{0,1\}^\lambda$. Send $z_{tb} = z_t + \mathsf{RLWE.Enc}(\mathsf{pk}, 0^\lambda \|\| m_b)$ to Alice $\mathcal{A}$.	

Figure 11: Puzzle promise protocol. We model H, H' and H^{shk} as random oracles.

Parties.

- $\mathcal{B}, \mathcal{T}$ and adversary $\mathcal{S}$.

Setup.

- Inform $\mathcal{F}_{\mathsf{promise\ sign}}$ if $\mathcal{T}$ is corrupt or honest.

Key generation.

- Receive message $(\mathsf{Keygen}, \mathcal{B})$ from $\mathcal{B}$.
- Send $(\mathsf{Keygen}, \mathcal{B})$ to $\mathcal{S}$.
- Receive response $(\mathsf{PK}_{\mathcal{T}}^{eph}, \mathsf{Sig})$ from $\mathcal{S}$.
- Send $(\mathsf{Setup}, \mathsf{PK}_{\mathcal{T}}^{eph})$ to $\mathcal{B}$.
- Record $(\mathsf{PK}_{\mathcal{T}}^{eph}, \mathsf{Sig})$.

Signature request.

- Receive message $(\mathsf{sign\ request}, \mathsf{PK}_{\mathcal{T}}^{eph'}, \{\mathsf{FkTxn}_i\}_{i\in[\eta]}, \{m_i\}_{i\in[\mu]})$ from $\mathcal{B}$.
- If $\mathsf{PK}_{\mathcal{T}}^{eph'} \neq \mathsf{PK}_{\mathcal{T}}^{eph}$, then do nothing.
- If $\forall i, \mathsf{FkTxn}_i$ compiles with FakeFormat, then send $(\mathsf{sign\ request}, \mathcal{B}, \mathsf{PK}_{\mathcal{T}}^{eph}, \{\mathsf{FkTxn}_i\}_{i\in[\eta]})$ to $\mathcal{T}$.
- Else, do nothing.

Promise.

- Receive $(\mathsf{promise}, \mathcal{B}, \mathsf{ans}, \mathsf{Set})$ from $\mathcal{T}$.
- If $\mathsf{ans} = \mathsf{no}$, then set all signatures to $\perp$.
- Else, if $\mathsf{Set} \neq \emptyset$, compute signatures as follows:
  - If $\mathcal{T}$ is honest, Set $\mathsf{FkSign}_i = \mathsf{Sig}(\mathsf{FkTxn}_i, \mathsf{PK}_{\mathcal{T}}^{eph})$ for $i \in [\eta]$.
  - Else $\mathcal{T}$ is corrupt, Send $(\mathsf{Sign}, \mathsf{FkTxn}_i, \mathcal{B})$ to adversary $\mathcal{S}$, and obtain respective signatures.
  - Abort if there is a recorded entry $(\mathsf{FkTxn}_i, \mathsf{FkSign}_i, \mathsf{PK}_{\mathcal{T}}^{eph}, 0)$.
  - Record entries $(\mathsf{FkTxn}_i, \mathsf{FkSign}_i, \mathsf{PK}_{\mathcal{T}}^{eph}, 1)$ and $(m_j, \mathsf{PK}_{\mathcal{T}}^{eph}, \mathsf{promise})$.
- Send $(\mathsf{sign\ promise}, \mathsf{ans})$ to $\mathcal{B}$.

Signature verification.

- Receive $(\mathsf{Verify}, \mathsf{sid}, m, \sigma, \mathsf{PK}_{\mathcal{T}}^{eph'})$ from any party $\mathcal{P}$:
  - If $\mathsf{PK}_{\mathcal{T}}^{eph'} \neq \mathsf{PK}_{\mathcal{T}}^{eph}$, then do nothing.
  - Else, if $\mathcal{T}$ is honest:
    - If there is a recorded entry $(m, \sigma, \mathsf{PK}_{\mathcal{T}}^{eph}, 1)$, then set $\mathsf{ver} = 1$. (completeness condition)
    - If there is no recorded entry $(m, \sigma, \mathsf{PK}_{\mathcal{T}}^{eph}, 1)$, then set $\mathsf{ver} = 0$ and set the entry $(m, \sigma, \mathsf{PK}_{\mathcal{T}}^{eph}, 0)$. (unforgeability condition)
  - Else, if $\mathcal{T}$ is corrupt, then let ver be set by $\mathcal{S}$. (corrupt signer case)
- Send $(\mathsf{Verify}, \mathsf{sid}, m, \sigma, \mathsf{ver})$ from party $\mathcal{P}$.

Figure 12: Ideal functionality $\mathcal{F}_{\mathsf{promise\ sign}}$ (Fig. 8 in [HAB+16])

- Simulator S simulates the messages that Adversary $\mathcal{B}^*$ expects from $\mathcal{T}$ as follows.
  1. Inform $\mathcal{F}_{\mathsf{promise\ sign}}$ that $\mathcal{T}$ is honest.
  2. Compute $(\mathsf{PK}_{\mathcal{T}}^{eph}, \mathsf{SK}_{\mathcal{T}}^{eph}) = \mathsf{Keygen}(1^\lambda)$.
  3. Send $\mathsf{PK}_{\mathcal{T}}^{eph}$ to $\mathcal{B}^*$ and $\mathcal{F}_{\mathsf{promise\ sign}}$.
- Receive h_R, h_F and $\{\beta_1, \ldots, \beta_{\mu+\eta}\}$ from $\mathcal{B}^*$, runs as follows:
  1. Extracts the sets F, R from the random oracle, checking the set of queries $\mathcal{Q}_H$ and extracting the pair $(\mathsf{salt}||R, h_R)$ and $(\mathsf{salt}||F, h_F)$. If there is no pair regarding h_R, h_F, then set $R = F = \perp$.
  2. Send $\mathcal{B}^*$ the pair (c_i, z_i) which is made as:
     (a) For all i, $c_i \xleftarrow{\$} \{0,1\}^s$.
     (b) For $i \in F$, $\epsilon_i \xleftarrow{\$} \{0,1\}^\lambda$ and $z_i = \mathsf{RLWE.Enc}(\mathsf{pk}, 0^\lambda||\epsilon_i)$.
     (c) For $R = \{j_1, \ldots, j_\mu\}, z_{j_1}, d_2, \ldots, d_\mu \xleftarrow{\$} \{0,1\}^\lambda$ and $z_{j_i} = \mathsf{RLWE.Enc}(\mathsf{pk}, 0^\lambda||d_i) + (0^\lambda||z_{j_{i-1}})$
- Receive (F', R', r_i) from $\mathcal{B}^*$, runs as follows:
  1. If $F' \neq F$ or $R' \neq R$, then abort.
  2. If any $(\mathsf{FakeFortmat}||r_i, \beta_i) \notin \mathcal{Q}_{H'}$ for $i \in F$, then abort.
  3. For $j \in R$, set $m_j = \gamma$ if $(\gamma, \beta_i) \in \mathcal{Q}_{H'}$. Else set $m_j = \perp$.
  4. Send to $\mathcal{F}_{\mathsf{promise\ sign}}$ the message $(\mathsf{sign\ request}, \mathsf{PK}_{\mathcal{T}}^{eph}, \{\mathsf{FakeFormat}||r_i\}_{i\in F}, \{m_j\}_{j\in R})$.
  5. Obtain response $(\mathsf{promise}, \mathcal{B}^*, \mathsf{ans}, \{\mathsf{FkSign}\}_{i\in[\eta]})$.
  6. If $\mathsf{ans} = \mathsf{no}$, then halt and output whatever $\mathcal{B}^*$ outputs.
  7. Compute $h_{j_l} = c_{j_l} \oplus \mathsf{FkSign}_l$.
  8. Store the pair $(\epsilon_{j_l}, h_{j_l})$ in $\mathcal{Q}_{H^{\mathsf{shk}}}$.
  9. Send ϵ_i for $i \in F$ and the differences $d_2, \ldots, d_\mu$.
- Finally, output whatever $\mathcal{B}^*$ outputs and halt.

Procedure RO1, which is the random oracle simulation for H is as follows:

1. Receive query q for H.
2. If $q \in \mathcal{Q}_H$, retrieve (γ, a) for $\mathcal{Q}_H$.
3. Else $a \xleftarrow{\$} \{0,1\}^{\lambda_2}$.
4. Append (γ, a) to $\mathcal{Q}_H$.
5. Output a.

Procedure RO2, which is the random oracle simulation for H^{shk} is as follows:

1. Receive query q for H^{shk}.
2. If $q \in \mathcal{Q}_{H^{\mathsf{shk}}}$, retrieve (γ, a) for $\mathcal{Q}_{H^{\mathsf{shk}}}$.
3. If $\gamma = \mathsf{RLWE.Dec}(\mathsf{sk}, z_i)$ for some $i \in R$ and $(\gamma, a) \notin \mathcal{Q}_{H^{\mathsf{shk}}}$, then output $\mathsf{RLWE\ failure}$.
4. Else $a \xleftarrow{\$} \{0,1\}^{\lambda_2}$.
5. Append (γ, a) to $\mathcal{Q}_{H^{\mathsf{shk}}}$.
6. Output a.

Figure 13: Simulator for the puzzle promise protocol in the case that Bob is corrupt

The simulator S executes $\mathcal{T}^*$ internally.
First, we introduce the algorithm $\mathsf{Sig}(m_i, \mathsf{PK}_{\mathcal{T}}^{eph})$ as follows.

- $L_{\mathsf{fake}}, L_{\mathsf{real}}$ are internal variables.
- If $(m_i, \beta_i, \sigma_i) \in L_{\mathsf{fake}}$, output signature (β_i, σ_i).
- Else if $(m_i, \beta_i, \sigma_i) \in L_{\mathsf{real}}$, then append (m_i, β_i) to $\mathcal{Q}_{H'}$, and output signature (β_i, σ_i).
- Else, abort.

Receive $(\mathsf{Keygen}, \mathcal{B})$. Send request to $\mathcal{T}^*$. Obtain $\mathsf{PK}_{\mathcal{T}}^{eph}$. Send $(\mathsf{PK}_{\mathcal{T}}^{eph}, \mathsf{Sig})$ to $\mathcal{F}_{\mathsf{promise\ sign}}$.
Receive $(\mathsf{sign\ request}, \mathcal{B}, \mathsf{PK}_{\mathcal{T}}^{eph}, \{\mathsf{FkTxn}_i\}_{i\in[\eta]})$.
Pick randomly F, R with $F \cap R = \emptyset$. Compute the RO outputs h_F, h_R and $\beta_1, \ldots, \beta_{\mu+\eta}$. Send them to $\mathcal{T}^*$.
Receive a pair (c_i, z_i) from $\mathcal{T}^*$:

1. Extract ϵ_i by queries to H^{shk}
2. Let σ_i be the signature decrypted from c_i with ϵ_i
3. If $c_i, z_i, \epsilon_i, \beta_i, \sigma_i$ for $i \in F$ are valid, store $(\mathsf{FkTxn}_i, \beta_i, \sigma_i)$ in L_{fake}.
4. If $c_i, z_i, \epsilon_i, \beta_i, \sigma_i$ for $i \in F$ are valid, append i to set Set and append (β_i, σ_i) to L_{real}. If such i does not exist, set $\mathsf{real} = \mathsf{no}$.

For all $i \in F$, $r_i \xleftarrow{\$}$. Send r_i to $\mathcal{T}^*$. Append the pair $(\mathsf{FkTxn}_i || r_i, \beta_i)$ to $\mathcal{Q}_{H'}$.
Receive the openings ϵ_i' to fake messages $i \in F$. Obtain σ_i' with ϵ_i', c_i. If any $(i, c_i, z_i, \epsilon_i', \beta_i, \sigma_i')$ is invalid, send $(\mathsf{promise}, \mathcal{B}, \mathsf{no}, \perp)$ to $\mathcal{F}_{\mathsf{promise\ sign}}$. Else, send $(\mathsf{promise}, \mathcal{B}, \mathsf{yes}, \mathsf{Set})$ to $\mathcal{F}_{\mathsf{promise\ sign}}$.
Now let us confirm the below two cases:
Case 1: If any $i \in F$ such that $(i, \epsilon_i', \beta_i, \sigma_i')$ is valid but $(\cdot, \beta_i, \sigma_i') \notin L_{\mathsf{fake}}$, then abort and output $\mathsf{binding\ fail}$.
Case 2: Suppose that all $i \in F$ such that $(i, \epsilon_i', \beta_i, \sigma_i')$ is valid and $(\cdot, \beta_i, \sigma_i') \in L_{\mathsf{fake}}$.

1. If $\mathsf{real} = \mathsf{no}$, abort and output $\mathsf{cut\ and\ choose\ fail}$.
2. Else, set variables $L_{\mathsf{fake}}, L_{\mathsf{real}}$ for the algorithm Sig.

Figure 14: Simulator for the puzzle promise protocol in the case that the tumbler is corrupt (Appendix F in [HAB+16])

the plaintext presented will match is $1/2^{\lambda}$. It is negligible. Note that $\eth_2$ computes neither ϵ_i nor σ_i for real messages m_i with $i \in R$.

$\eth_3$: Instead of the actual signature, the simulator sends the message

$$(\mathsf{sign\ request}, \mathsf{PK}_{\mathcal{T}}^{eph}, \{\mathsf{FakeFormat}||r_i\}_{i\in F}, \{m_j\}_{j\in R})$$

to $\mathcal{F}_{\mathsf{promise\ sign}}$. Then, S obtains the response $(\mathsf{promise}, \mathcal{B}^*, \mathsf{ans}, \{\mathsf{FkSign}\}_{i\in[\eta]})$ and uses $\sigma_i = \mathsf{FkSign}_i$. $\eth_2$ and $\eth_3$ is identical from the view of $\mathcal{B}^*$. $\eth_3$ is the simulator S in Fig. 13 itself. We conclude that the transcript by the simulator S in Fig. 13 is indistinguishable from the one in the real world.

Remark 6. *The below lemma shows that $\mathcal{B}^*$ cannot forge a valid signature σ for a valid message $T_{\mathsf{cash}(\mathcal{T},\mathcal{B})}$.*

Lemma 7 (Lemma 3 in [HAB+16])**.** *If ECDSA is an existentially unforgeable signature scheme, then* $\Pr[E_{\mathsf{forge}}]$ *is negligible.*

Case that the tumbler is corrupt. The simulator S in Fig. 14 plays a role of the corrupt the tumbler $\mathcal{T}^*$. The difference between the real world and the ideal world is whether the simulator S in Fig. 14 stops. Let us confirm the probability that the simulator will stop for each event.

- Event cut and choose fail:

$$\Pr[\mathsf{cut\ and\ choose\ fail}] = \frac{1}{\binom{\mu+\eta}{\eta}} + \frac{1}{2^{\lambda_1}}$$

- Event binding fail:

$$\Pr[\mathsf{binding\ fail}] = \frac{1}{2^{\lambda_2}}$$

We conclude that the protocol in Fig. 11 securely realizes the functionality $\mathcal{F}_{\mathsf{promise\ sign}}$ in Fig. 12. □

5 Conclusion

Scalability and privacy protection are significant problems with blockchain. In this work, we have proposed an anonymous probabilistic payment to solve these simultaneously. Our proposal is not restricted to any particular cryptocurrency. We have mediated the tumbler of the payment channel hub between a payer and a payee. A cryptographic puzzle plays a role in controlling the intermediation and the execution of transactions. Masking the puzzle enables the payer and payee to unlink their payments. The proposed protocol realizes the ideal functionalities discussed in TumbleBit (NDSS 2017). Besides, we have introduced a novel fractional oblivious transfer based on the RLWE encryption. We confirm that RFOT holds the properties of fractional hiding and binding presented in the DAM scheme (Eurocrypt 2017). We have adopted it for a probabilistic payment.

References

[ABCar] Ghada Almashaqbeh, Allison Bishop, and Justin Cappos. Microcash: Practical concurrent processing of micropayments. *International Conference on Financial Cryptography and Data Security 2020*, to appear.

[BAZBar] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. *International Conference on Financial Cryptography and Data Security 2020*, to appear.

[BCK+14] Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 551–572, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BD18] Zvika Brakerski and Nico Döttling. Two-message statistically sender-private ot from lwe. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography*, pages 370–390, Cham, 2018. Springer International Publishing.

[BM90] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 547–557, New York, NY, 1990. Springer New York.

[BR99] Mihir Bellare and Ronald L Rivest. Translucent cryptography—an alternative to key escrow, and its implementation via fractional oblivious transfer. *Journal of cryptology*, 12(2):117–139, 1999.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 505–524, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[CDG+18] D. B. Cousins, G. Di Crescenzo, K. D. Gür, K. King, Y. Polyakov, K. Rohloff, G. W. Ryan, and E. Savas. Implementing conjunction obfuscation under entropic ring lwe. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 354–371, 2018.

[CGL+16] Alessandro Chiesa, Matthew Green, Jingcheng Liu, Peihan Miao, Ian Miers, and Pratyush Mishra. Decentralized anonymous micropayments. Cryptology ePrint Archive, Report 2016/1033, 2016. `https://eprint.iacr.org/2016/1033`.

[CGL+17] Alessandro Chiesa, Matthew Green, Jingcheng Liu, Peihan Miao, Ian Miers, and Pratyush Mishra. Decentralized anonymous micropayments. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 609–642, Cham, 2017. Springer International Publishing.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over ntru lattices. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 22–41, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[GM17] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 473–489, New York, NY, USA, 2017. Association for Computing Machinery.

[GM18] Nicholas Genise and Daniele Micciancio. Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 174–203, Cham, 2018. Springer International Publishing.

[HAB+16] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. 2016. `https://eprint.iacr.org/2016/575`.

[HAB+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017.

[HBG16] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 43–60, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 1219–1234, New York, NY, USA, 2012. Association for Computing Machinery.

[LH19] Momeng Liu and Yupu Hu. Universally composable oblivious transfer from ideal lattice. *Front. Comput. Sci.*, 13(4):879–906, August 2019.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):43, 2013.

[Lyu09] Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pages 598–616, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 738–755, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MO20] T. Mitani and A. Otsuka. Traceability in permissioned blockchain. *IEEE Access*, 8:21573–21588, 2020.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 700–718, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

[MSRL+19] Pedro Moreno-Sanchez, Randomrun, Duc V. Le, Sarang Noether, Brandon Goodell, and Aniket Kate. Dlsag: Non-interactive refund transactions for interoperable payment channels in monero. Technical report, 2019. `https://eprint.iacr.org/2019/595`.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Retrieved from Website: https://bitcoin.org/bitcoin.pdf*, 2008.

[Noe15] Shen Noether. Ring signature confidential transactions for monero. 2015. `https://eprint.iacr.org/2015/1098`.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[Ps15] Rafael Pass and abhi shelat. Micropayments for decentralized currencies. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 207–218, New York, NY, USA, 2015. Association for Computing Machinery.

[Riv97] Ronald L. Rivest. Electronic lottery tickets as micropayments. In Rafael Hirschfeld, editor, *Financial Cryptography*, pages 307–314, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[SALY17] Shi-Feng Sun, Man Ho Au, Joseph K. Liu, and Tsz Hon Yuen. Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 456–474, Cham, 2017. Springer International Publishing.

[SCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[Whe97] David Wheeler. Transactions using bets. In Mark Lomas, editor, *Security Protocols*, pages 89–92, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[YSL+ar] Tsz Hon Yuen, Shi-feng Sun, Joseph K Liu, Man Ho Au, Muhammed F Esgin, Qingzhao Zhang, and Dawu Gu. Ringct 3.0 for blockchain confidential transaction: shorter size and stronger security. Technical report, to appear.

[ZZD+15] Jiang Zhang, Zhenfeng Zhang, Jintai Ding, Michael Snook, and Özgür Dagdelen. Authenticated key exchange from ideal lattices. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 719–751, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

A Zero-knowledge proof

In this section, we present the zero-knowledge proofs of knowledge for RLWE encryption.

A.1 Pedersen commitments

Let us introduce Pedersen commitments [Ped92]. Given a family of prime order groups $\{\mathbb{G}(\lambda)\}_{\lambda\in\mathbb{N}}$ such that the discrete logarithm problem is hard in $\mathbb{G}(\lambda)$ with security parameter λ, let $\tilde{q} = \tilde{q}(\lambda)$ be the order of $\mathbb{G} = \mathbb{G}(\lambda)$. We denote all elements with order $\tilde{q}$ with a tilde in the following. We write the group $\mathbb{G}(\lambda)$ additively.

- CSetup: This algorithm chooses $\tilde{g}, \tilde{h} \xleftarrow{\$} \mathbb{G}$ and outputs $cpars = (\tilde{g}, \tilde{h})$.
- Commit: To commit to a message $m \in \mathbb{Z}_{\tilde{q}}$, it first chooses $r \xleftarrow{\$} \mathbb{Z}_{\tilde{q}}$. It then outputs a pair $(\widetilde{cmt}, o) = (m\tilde{g} + r\tilde{h}, r)$.
- COpen: Given a commitment $\widetilde{cmt}$, an opening o, a public key $cpars$ and a message m, it outputs accept if and only if $(\widetilde{cmt}, o) \stackrel{?}{=} (m\tilde{g} + r\tilde{h}, r)$.

Lemma 8 (Theorem 2.1. in [BCK+14])**.** *Under the discrete logarithm assumption for $\mathbb{G}$, the given commitment scheme is perfectly hiding and is computationally binding.*

In the protocols of this chapter, we make use of the above scheme as an auxiliary commitment scheme. We denote it as (aCSetup, aCCommit, aCOpen).

A.2 Rejection sampling

To realize zero-knowledge proof, the technique of rejection sampling is useful [Lyu09, Lyu12]. The technique can conceal the information of the witness. Therefore, we apply the technique when the prover sends the response to the verifier.

Lemma 9 (Theorem 4.6 in [Lyu12])**.** *Let V be a subset of $\mathbb{Z}^m$ in which all elements have norms less than T, σ be some element in $\mathbb{R}$ such that $\sigma = \omega(T\sqrt{\log m})$, and $h : V \to \mathbb{R}$ be a probability distribution. Then, there exists a constant $M = \tilde{\mathcal{O}}(1)$ such that the distribution of the following algorithm $\mathcal{A}$:*

1. *$v \xleftarrow{\$} h$*
2. *$z \xleftarrow{\$} D^m_{v,\sigma}$*
3. *output (z, v) with probability $\min(\frac{D^m_\sigma(z)}{MD^m_{v,\sigma}(z)}, 1)$*

is within statistical distance $\frac{2^{-\omega(\log m)}}{M}$ of the distribution of the following algorithm $\mathcal{F}$:

1. *$v \xleftarrow{\$} h$*

2. $z \xleftarrow{\$} D_\sigma^m$

3. *output* (z, v) *with probability* $1/M$

Moreover, the probability that $\mathcal{A}$ *outputs something is at least* $\frac{1-2^{-\omega(\log m)}}{M}$.

More concretely, if $\sigma = \alpha T$ *for any positive* α*, then* $M = e^{12/\alpha + 1/(2\alpha^2)}$*, the output of algorithm* $\mathcal{A}$ *is within statistical distance* $\frac{2^{-100}}{M}$ *of the output of* $\mathcal{F}$*, and the probability that* $\mathcal{A}$ *outputs something is at least* $\frac{1-2^{-100}}{M}$.

A.3 Definition

We describe the formal definition of the Σ'-protocol, the protocol and its theorem proving this relationship.

Definition 9 (Definition 2.5. in [BCK+14])**.** *Let* (P, V) *be a two-party protocol, where* V *is a probabilistic polynomial time algorithim, and let* $L, L' \subseteq \{0,1\}^*$ *be languages with witness relations* R, R' *such that* $R \subseteq R'$*. Then,* (P, V) *is called a* Σ'*-protocol for* L, L' *with completeness error* α*, a challenge set* $\mathbb{C}$*, a public input* x *and a private input* w*, if and only if it satisfies the following conditions:*

- *Three-move form: The prover* P*, on input* (x, w)*, computes a commitment* t *and sends it to* V*. The verifier* V*, on input* x*, then draws a challenge* $c \xleftarrow{\$} \mathbb{C}$ *and sends it to* P*. The prover sends a response* s *to the verifier. Depending on the protocol transcript* (t, c, s)*, the verifier finally accepts or rejects the proof. The protocol transcript* (t, c, s) *is called accepting, if the verifier accepts the protocol run.*
- *Completeness: Whenever* $(x, w) \in R$*, the verifier* V *accepts with probability at least* $1 - \alpha$*.* α *is the completeness error.*
- *Special soundness: There exists a PPT algorithm* E *(the knowledge extractor) that takes two accepting transcripts* $(t, c', s'), (t, c'', s'')$ *satisfying* $c' \neq c''$ *as inputs, and outputs* w' *such that* $(x, w') \in R'$*. The knowledge error denotes the probability that the verifier accepts the proof even if the prover does not know a witness.*
- *Special honest verifier zero-knowledge (HVZK): There exists a PPT algorithm* S *(the simulator) taking* $x \in L$ *and* $c \in \mathbb{C}$ *as inputs. Moreover, the simulator outputs* (t, s) *so that the triple* (t, c, s) *is indistinguishable from an accepting protocol transcript generated by a real protocol run.*

Moreover, let us present the following useful property.

- High-entropy commitments: For all $(y, w) \in R$ and for all t, the probability that an honestly generated commitment by P takes on the value t is negligible.

A.4 Relation for the key generation

We want to show that we know a private key of a public key without revealing the private key itself. We introduce the zero-knowledge proof for the purpose in this subsection. Let us confirm Theorem 3.3. in [BCK+14].

Lemma 10 (Theorem 3.3. in [BCK+14])**.** *The protocol in Fig. 15 is an HVZK* Σ'*-protocol for the following relations:*

$$\mathcal{R} = \{((a, y), (s, e)) : y = as + e \wedge |s|, |e| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\}$$
$$\mathcal{R}' = \{((a, y), (s, e)) : 2y = 2as + 2e \wedge |2s|, |2e| \leq \tilde{\mathcal{O}}(n_d^2\alpha)\}$$

where $2s$ *and* $2e$ *are reduced modulo* q*. The protocol has a knowledge error of* $1/(2n_d)$*, a completeness error of* $1 - 1/M$*, and high-entropy commitments.*

Common input: the open value (a, y)
Relation: $R = \{((a, y), (s, e)) : y = as + e \wedge |s|, |e| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\}$

Prover		Verifier
$r_s, r_e \xleftarrow{\$} D_{\tilde{\mathcal{O}}(\sqrt{n_d}\alpha)}$		
$t = ar_s + r_e$		
$(c_{\text{aux}}, d_{\text{aux}}) = \mathsf{aCommit}(t)$		
	$\xrightarrow{c_{\text{aux}}}$	
		$c \xleftarrow{\$} \mathbb{C} = \{0, \ldots, 2n_d - 1\}$
	$\xleftarrow{c}$	
$s_s = r_s + X^c s$		
$s_e = r_e + X^c e$		
accept with probability $\frac{D_{\tilde{\mathcal{O}}(\sqrt{2n_d}\alpha)}((s_e, s_s))}{MD_{(X^c e, X^c s), \tilde{\mathcal{O}}(\sqrt{2n_d}\alpha)}((s_e, s_s))}$		
	$\xrightarrow{t, d_{\text{aux}}, (s_s, s_e)}$	
		$X^c y + t \stackrel{?}{=} as_s + s_e$
		$\mathsf{aCOpen}(t, c_{\text{aux}}, d_{\text{aux}}) \stackrel{?}{=} \mathsf{accept}$
		$\lvert s_s\rvert, \lvert s_e\rvert \leq \tilde{\mathcal{O}}(n_d\alpha)$

Figure 15: Zero-knowledge proof of knowledge of RLWE secrets s, e such that $y = as + e$ (Protocol 3.2. in [BCK+14])

We can use Lemma 10 for the below relation $\mathcal{R}_K$ by replacing $y \to y/p, a \to a/p$.

$$\mathcal{R}_K = \{((a, y), (s, e)) \mid y = as + 2e \wedge |s|, |e| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\}$$

Let us describe the non-interactive zero-knowledge proof for the relation $\mathcal{R}_K$. The protocol in Fig. 15 realizes the zero-knowledge proof for the relation $\mathcal{R}_K$. Moreover, we can obtain the non-interactive zero-knowledge proof by using Fiat-Shamir transform [FS87] and running the protocol in parallel. Let us confirm the simulator for the relation $\mathcal{R}_K$. The technique of the rejection sampling is used for zero-knowledge in the protocol. According to Lemma 9, the algorithm $\mathcal{F}$ includes no witness and is statistically indistinguishable from the algorithm $\mathcal{A}$. We can implement the simulator for $\mathcal{R}_K$ by replacing the algorithm $\mathcal{A}$ with the algorithm $\mathcal{F}$.

A.5 Relation for the ciphertext of zero

We want to show that we know that the plaintext of a given ciphertext is zero without decryption or revealing randomness of the ciphertext. We introduce the zero-knowledge proof for the purpose in this subsection. We show a non-interactive zero-knowledge proof for ciphertext of zero in Fig. 16. h is a cryptographic hash function. We make the interactive proof non-interactive by using the Fiat-Shamir heuristic [FS87]. This protocol satisfies Lemma 11. The parallel protocol satisfies Theorem 3.

Lemma 11 (Lemma 5 in [MO20])**.** *The protocol in Fig. 16 is an HVZK Σ'-protocol for the following relations:*

$$\begin{aligned}
\mathcal{R}_0 &= \{((c_1, c_2), (v, e, f)) : (c_1, c_2) = (bv + pe, av + pf) \wedge |v|, |e|, |f| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\} \\
\mathcal{R}'_0 &= \{((c_1, c_2), (v, e, f)) : (2c_1, 2c_2) = (2bv + 2pe, 2av + 2pf) \\
&\qquad \wedge |2v|, |2e|, |2f| \leq \tilde{\mathcal{O}}(n_d^2\alpha)\}
\end{aligned}$$

where $2v, 2e$ and $2f$ are reduced modulo q. The protocol has a knowledge error of $1/(2n_d)$, a completeness error of $1 - 1/M$, and high-entropy commitments.

Common input: the public key (a, b), the ciphertext (c_1, c_2)
Relation: $R_0 = \{((c_1, c_2), (v, e, f)) : (c_1, c_2) = (bv + pe, av + pf)$
$\wedge |v|, |e|, |f| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\}$

Prover	Verifier
$r_v, r_e, r_f \xleftarrow{\$} D_{\tilde{\mathcal{O}}(\sqrt{n_d}\alpha)}$	
$t_1 = br_v + r_e$	
$t_2 = ar_v + r_f$	
$(c^{(1)}_{\text{aux}}, d^{(1)}_{\text{aux}}) = \mathsf{aCommit}(t_1)$	
$(c^{(2)}_{\text{aux}}, d^{(2)}_{\text{aux}}) = \mathsf{aCommit}(t_2)$	
	$\xrightarrow{c^{(1)}_{\text{aux}}, c^{(2)}_{\text{aux}}}$
$c = h(t_1, t_2, c^{(1)}_{\text{aux}}, d^{(1)}_{\text{aux}}, c^{(2)}_{\text{aux}}, d^{(2)}_{\text{aux}})$	
$s_v = r_v + X^c v$	
$s_e = r_e + X^c pe$	
$s_f = r_f + X^c pf$	
accept with probability $\frac{D_{\tilde{\mathcal{O}}(\sqrt{3n_d}\alpha)}((s_v, s_e, s_f))}{MD_{(X^c v, X^c e, X^c f), \tilde{\mathcal{O}}(\sqrt{3n_d}\alpha)}((s_v, s_e, s_f))}$	
	$\xrightarrow{t_1, t_2, d^{(1)}_{\text{aux}}, d^{(2)}_{\text{aux}}, (s_v, s_e, s_f)}$
	$c = h(t_1, t_2, c^{(1)}_{\text{aux}}, d^{(1)}_{\text{aux}}, c^{(2)}_{\text{aux}}, d^{(2)}_{\text{aux}})$
	$X^c c_1 + t_1 \stackrel{?}{=} bs_v + s_e$
	$X^c c_2 + t_2 \stackrel{?}{=} as_v + s_f$
	$\mathsf{aCOpen}(t_1, c^{(1)}_{\text{aux}}, d^{(1)}_{\text{aux}}) \stackrel{?}{=} \mathsf{accept}$
	$\mathsf{aCOpen}(t_2, c^{(2)}_{\text{aux}}, d^{(2)}_{\text{aux}}) \stackrel{?}{=} \mathsf{accept}$
	$\lvert s_v\rvert, \lvert s_e\rvert, \lvert s_f\rvert \leq \tilde{\mathcal{O}}(n_d\alpha)$

Figure 16: Non-interactive zero-knowledge proof of a ciphertext of zero regarding RLWE encryption (Figure 3 in [MO20])

Theorem 3 (Theorem 6 in [MO20]). *Let us apply the protocol in Fig. 16 for λ times in parallel (the parallel protocol). Let the parallel protocol be accepting if and only if at least $\lambda/2M$ out of λ proofs were valid under the condition that an honest verifier rejects no proofs. Then, the parallel protocol has both a completeness error and knowledge error of* $\mathrm{negl}(\lambda)$ *under the condition $n_d \geq 2M$.*

A.6 Relation for valid ciphertext

We want to show that we know that a given ciphetext is valid without decryption or revealing a message and randomness of the ciphertext. We introduce the zero-knowledge proof for the purpose in this subsection.

We introduce the following technical lemmas.

Lemma 12 (Lemma 3.1 in [BCK+14]). *Let n_d be a power of 2 and let $0 < i, j < 2n_d - 1$. Then, $2(X^i - X^j)^{-1}$ mod $(X^{n_d}+1)$ only has coefficients in $\{-1, 0, 1\}$.*

Lemma 13 (Lemma 2 in [ZZD+15]). *For any $s, t \in \mathbf{R}$, we have $|s \cdot t| \leq \sqrt{n_d} \cdot |s| \cdot |t|$ and $|s \cdot t|_\infty \leq n_d \cdot |s|_\infty \cdot |t|_\infty$.*

Let us introduce the relation $\mathcal{R}_E$ as follows. In the relation $\mathcal{R}_E$, let a message space $\mathbf{M} = \mathbf{R}_2$.

$$\begin{aligned}\mathcal{R}_E = \{((c_1, c_2, a, y), (m, v, e, f)) \mid\ & c_1 = yv + 2e + m \ \wedge\ c_2 = av + 2f \\ & \wedge\ |m|_\infty \leq 1 \ \wedge\ |v|, |e|, |f| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\}\end{aligned}$$

We construct a non-interactive zero-knowledge proof for the RLWE ciphertext, referring to Protocol 4.1 and Theorem 4.2 in [BCK+14]. We present the protocol in Fig. 17. This protocol in Fig. 17 satisfies Lemma 14. We can make the completeness error and knowledge error in Lemma 14 negligible, by the parallelization as Theorem 3. Let us explain the notation $a_{\ll l}$. One obtains $a_{\ll}$ by an anti-cyclic shift a vector a by l. That is, $a_{\ll l} = aX^l$ in $\mathbf{R}$. If we write the elements of the vector explicitly, we have

$$a_{\ll l} = (a_0, \ldots, a_{n_d-1})_{\ll l} = (-a_{n_d-l}, \ldots, -a_{n_d-1}, a_0, \ldots, a_{n_d-l-1}).$$

Lemma 14. *The protocol in Fig. 17 is an honest verifier zero-knowledge Σ'-protocol for the following relations:*

$$\begin{aligned}\mathcal{R} = \{((c_1, c_2, \tilde{g}, \tilde{h}, (\widetilde{cmt}_i)_{i=0}^{n_d-1}, p, a, b), (m, v, e, f, (r_i)_{i=0}^{n_d-1})) :& \\ (c_1, c_2) = (bv + pe + m, av + pf) \wedge \bigwedge_{i=0}^{n_d-1} \widetilde{cmt}_i = r_{m,i}\tilde{g} + r_{r,i}\tilde{h}& \\ \wedge |m|_\infty \leq 1 \wedge |v|, |e|, |f| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\}& \\ \mathcal{R}' = \{((c_1, c_2, \tilde{g}, \tilde{h}, (\widetilde{cmt}_i)_{i=0}^{n_d-1}, p, a, b), (m, v, e, f, (r_i)_{i=0}^{n_d-1})) :& \\ (2c_1, 2c_2) = (2bv + 2pe + 2m, 2av + 2pf) \wedge \bigwedge_{i=0}^{n_d-1} 2\widetilde{cmt}_i = (2m \mod q)_i\tilde{g} + 2r_{r,i}\tilde{h}& \\ \wedge |2m| \leq 2n_d^2 \wedge |2v|, |2e|, |2f| \leq \tilde{\mathcal{O}}(n_d^2\alpha)\}&\end{aligned}$$

where $2v, 2e$ and $2f$ are reduced modulo q and $(2m \mod q)_i$ is the i-coefficient of $2m \in \mathbf{R}_q$. The protocol has a knowledge error of $1/(2n_d)$, a completeness error of $1 - 1/M$, and high-entropy commitments.

Proof. Let us confirm the proof from the following points: completeness, honest verifier zero-knowledge, special soundness, and high-entropy commitments.

Completeness From Lemma 9 of the rejection sampling, a prover responds with a probability $1/M$. If the prover does not abort, then

$$\begin{aligned}bs_v + ps_e + s_m &= b(r_v + X^c v) + p(r_e + X^c e) + (r_m + X^c m) \\ &= X^c(bv + pe + m) + (br_v + pr_e + r_m) \\ &= X^c c_1 + t_1\end{aligned}$$

Common input: the ciphertext (c_1, c_2), the integer p, the public key (a, b), the public key of commitment $(\tilde{g}, \tilde{h})$, the commitment $(\widetilde{cmt}_i)_{i=0}^{n_d-1}$
Relation $\mathcal{R} = \{((c_1, c_2, \tilde{g}, \tilde{h}, (\widetilde{cmt}_i)_{i=0}^{n_d-1}, p, a, b), (m, v, e, f, (r_i)_{i=0}^{n_d-1})) :$
$(c_1, c_2) = (bv + pe + m, av + pf) \wedge \bigwedge_{i=0}^{n_d-1} \widetilde{cmt}_i = r_{m,i}\tilde{g} + r_{r,i}\tilde{h}$
$\wedge |m|_\infty \leq 1 \wedge |v|, |e|, |f| \leq \tilde{\mathcal{O}}(\sqrt{n_d}\alpha)\}$

Prover		Verifier
$r_v, r_e, r_f \xleftarrow{\$} D_{\tilde{\mathcal{O}}(\sqrt{n_d}\alpha)}$		
$r_m \xleftarrow{\$} D_{\tilde{\mathcal{O}}(\sqrt{n_d})}$		
$r_{r,i} \xleftarrow{\$} \mathbb{Z}_{\tilde{q}}$ for $i = 0, \ldots, n_d - 1$		
$t_1 = br_v + pr_e + r_m$		
$t_2 = ar_v + pr_f$		
$\tilde{t}_i = r_{m,i}\tilde{g} + r_{r,i}\tilde{h}$ for $i = 0, \ldots, n_d - 1$		
$(c_{\text{aux}}, d_{\text{aux}}) = \mathsf{aCommit}(t_1, t_2, (\tilde{t}_i)_{i=0}^{n_d-1})$		
	$\xrightarrow{c_{\text{aux}}}$	
$c = h(t_1, t_2, (\tilde{t}_i)_{i=0}^{n_d-1}, c_{\text{aux}}, d_{\text{aux}})$		
$s_v = r_v + X^c v$		
$s_e = r_e + X^c e$		
$s_f = r_f + X^c f$		
$s_m = r_m + X^c m$		
$s_r = r_r + X^c r$		
accept with probability $\frac{D_{\tilde{\mathcal{O}}(2\sqrt{n_d}\alpha)}((s_v, s_e, s_f, s_m))}{MD_{(X^c v, X^c e, X^c f, X^c m), \tilde{\mathcal{O}}(2\sqrt{n_d}\alpha)}((s_v, s_e, s_f, s_m))}$		
	$\xrightarrow{t_1, t_2, (\tilde{t}_i)_{i=0}^{n_d-1}, d_{\text{aux}}, (s_v, s_e, s_f, s_m, s_r)}$	
		$c = h(t_1, t_2, (\tilde{t}_i)_{i=0}^{n_d-1}, c_{\text{aux}}, d_{\text{aux}})$
		$X^c c_1 + t_1 \stackrel{?}{=} bs_v + ps_e + s_m$
		$X^c c_2 + t_2 \stackrel{?}{=} as_v + ps_f$
		$(\widetilde{cmt}_0, \ldots, \widetilde{cmt}_{n_d-1})_{\ll c} + (\tilde{t}_0, \ldots, \tilde{t}_{n_d-1}) \stackrel{?}{=} s_m\tilde{g} + s_r\tilde{h}$
		$\mathsf{aCOpen}((t_1, t_2, (\tilde{t}_i)_{i=0}^{n_d-1}), c_{\text{aux}}, d_{\text{aux}}) \stackrel{?}{=} \mathsf{accept}$
		$\lvert s_v\rvert, \lvert s_e\rvert, \lvert s_f\rvert \leq \tilde{\mathcal{O}}(n_d\alpha)$
		$\lvert s_m\rvert \leq \tilde{\mathcal{O}}(n_d)$

Figure 17: Non-interactive zero-knowledge proof of plaintext knowledge regarding RLWE encryption

and

$$\begin{aligned} as_v + ps_f &= a(r_v + X^c v) + p(r_f + X^c f) \\ &= X^c(av + pf) + (ar_v + pr_f) \\ &= X^c c_2 + t_2. \end{aligned}$$

Let us confirm the equation of the commitment

$$(\widetilde{cmt}_0, \ldots, \widetilde{cmt}_{n_d-1})_{\ll c} + (\tilde{t}_0, \ldots, \tilde{t}_{n_d-1}) \stackrel{?}{=} s_m \tilde{g} + s_r \tilde{h}.$$

Let us expand the first term of the left hand side.

$$\begin{aligned} (\widetilde{cmt}_0, \ldots, \widetilde{cmt}_{n_d-1})_{\ll c} &= (m_0\tilde{g} + r_0\tilde{h}, \ldots, m_{n_d-1}\tilde{g} + r_{n_d-1}\tilde{h})_{\ll c} \\ &= (m_0, \ldots, m_{n_d-1})_{\ll c}\tilde{g} + (r_0, \ldots, r_{n_d-1})_{\ll c}\tilde{h} \\ &= X^c m\tilde{g} + X^c r\tilde{h} \end{aligned}$$

We also expand the second term of the left hand side.

$$\begin{aligned} (\tilde{t}_0, \ldots, \tilde{t}_{n_d-1}) &= (r_{m,0}\tilde{g} + r_{r,0}\tilde{h}, \ldots, r_{m,n_d-1}\tilde{g} + r_{r,n_d-1}\tilde{h}) \\ &= (r_{m,0}, \ldots, r_{m,n_d-1})\tilde{g} + (r_{r,0}, \ldots, r_{r,n_d-1})\tilde{h} \\ &= r_m\tilde{g} + r_r\tilde{h} \end{aligned}$$

Therefore we have

$$\begin{aligned} (\widetilde{cmt}_0, \ldots, \widetilde{cmt}_{n_d-1})_{\ll c} + (\tilde{t}_0, \ldots, \tilde{t}_{n_d-1}) &= (X^c m\tilde{g} + X^c r\tilde{h}) + (r_m\tilde{g} + r_r\tilde{h}) \\ &= (r_m + X^c m)\tilde{g} + (r_r + X^c r)\tilde{h} \\ &= s_m\tilde{g} + s_r\tilde{h}. \end{aligned}$$

Let us confirm the norms of s_v, s_e, s_f and s_m. The standard deviations of r_v, r_e, r_f are $\tilde{\mathcal{O}}(\sqrt{n_d}\alpha)$. Also, the standard deviations of r_m is $\tilde{\mathcal{O}}(\sqrt{n_d})$. We have

$$\begin{aligned} |s_v| &\leq |r_v| + |v| \leq \tilde{\mathcal{O}}(n_d\alpha) \\ |s_e| &\leq |r_e| + |e| \leq \tilde{\mathcal{O}}(n_d\alpha) \\ |s_f| &\leq |r_f| + |f| \leq \tilde{\mathcal{O}}(n_d\alpha) \\ |s_m| &\leq |r_m| + |m| \leq \tilde{\mathcal{O}}(n_d) \end{aligned}$$

with the overwhelming probability.

Honest verifier zero-knowledge A challenge value c is randomly chosen from the set $\mathbb{C} = \{0, \ldots, 2n_d - 1\}$. A simulator outputs the tuple $(\mathsf{aCommit}(0), c, \perp)$ with the probability $1 - 1/M$. With the probability $1/M$, the simulator runs as follows.

$$\begin{aligned} r_v, r_e, r_f &\stackrel{\$}{\leftarrow} D_{\tilde{\mathcal{O}}(\sqrt{n_d}\alpha)} \\ r_m &\stackrel{\$}{\leftarrow} D_{\tilde{\mathcal{O}}(\sqrt{n_d})} \\ t_1 &= br_v + pr_e + r_m - X^c c_1 \\ t_2 &= ar_v + pr_f - X^c c_2 \\ \tilde{t}_i &= r_{m,i}\tilde{g} + r_{r,i}\tilde{h} \quad \text{for } i = 0, \ldots, n_d - 1 \\ (c_{\mathrm{aux}}, d_{\mathrm{aux}}) &= \mathsf{aCommit}(t_1, t_2, (\tilde{t}_i)_{i=0}^{n_d-1}) \end{aligned}$$

Finally, the simulator outputs

$$(c_{\mathrm{aux}}, d_{\mathrm{aux}}, c, t_1, t_2, (\tilde{t}_i)_{i=0}^{n_d-1}, (s_v, s_e, s_f, s_m)).$$

From Lemma 9, the outputs s_v, s_e, s_f and s_m without abort do not depend on v, e, f and m as the witness. Therefore, the simulator and the actual protocol are indistinguishable. If the protocol aborts, then it is indistinguishable because of the hiding property of aCommit. For any c, the abort occurs in the same way.

Special soundness Let us consider knowledge extractor and suppose that both

$$(c_{\mathrm{aux}}, d'_{\mathrm{aux}}, c', t'_1, t'_2, (\tilde{t}'_i)_{i=0}^{n_d-1}, (s'_v, s'_e, s'_f, s'_m, s'_r))$$

and

$$(c_{\mathrm{aux}}, d''_{\mathrm{aux}}, c'', t''_1, t''_2, (\tilde{t}''_i)_{i=0}^{n_d-1}, (s''_v, s''_e, s''_f, s''_m, s''_r))$$

pass the verification by the verifier. From the binding property of aCommit, we have

$$t_1 := t'_1 = t''_1,\ t_2 := t'_2 = t''_2,\ (\tilde{t}_i)_{i=0}^{n_d-1} := (\tilde{t}'_i)_{i=0}^{n_d-1} = (\tilde{t}''_i)_{i=0}^{n_d-1}.$$

Let us confirm $(\tilde{t}_i)_{i=0}^{n_d-1}$. From the verification, we have

$$\begin{aligned}(\widetilde{cmt}_0, \ldots, \widetilde{cmt}_{n_d-1})_{\ll c'} + (\tilde{t}_0, \ldots, \tilde{t}_{n_d-1}) &= s'_m\tilde{g} + s'_r\tilde{h}\\ (\widetilde{cmt}_0, \ldots, \widetilde{cmt}_{n_d-1})_{\ll c''} + (\tilde{t}_0, \ldots, \tilde{t}_{n_d-1}) &= s''_m\tilde{g} + s''_r\tilde{h}.\end{aligned}$$

From the subtraction between these two equations, we have

$$(\widetilde{cmt}_0, \ldots, \widetilde{cmt}_{n_d-1})(X^{c'} - X^{c''}) = (s'_m - s''_m)\tilde{g} + (s'_r - s''_r)\tilde{h}.$$

Multiplying by $2(X^{c'} - X^{c''})^{-1}$ to this equation, we have

$$(\widetilde{cmt}_0, \ldots, \widetilde{cmt}_{n_d-1}) = \frac{2(s'_m - s''_m)}{X^{c'} - X^{c''}}\tilde{g} + \frac{2(s'_r - s''_r)}{X^{c'} - X^{c''}}\tilde{h} =: 2\hat{m}\tilde{g} + 2\hat{r}\tilde{h}.$$

Then, let us confirm the upper bound of the norm $|2\hat{m}|$. By adopting Lemma 12 and 13, and the triangle inequality $|a \pm b| \leq |a| + |b|$, we have

$$\begin{aligned}|2\hat{m}| &\leq \sqrt{n_d} \cdot |s'_m - s''_m| \cdot \left|\frac{2}{X^{c'} - X^{c''}}\right|\\ &\leq \sqrt{n_d} \cdot |s'_m - s''_m| \cdot \sqrt{n_d}\\ &\leq n_d(|s'_m| + |s''_m|) = 2n_d^2.\end{aligned}$$

We can have a similar discussion with regard to t_1 and t_2. Let us confirm t_1. From the verification equation, we have

$$\begin{aligned}X^{c'}c_1 + t_1 &= bs'_v + ps'_e + m'\\ X^{c''}c_1 + t_1 &= bs''_v + ps''_e + m''.\end{aligned}$$

From the subtraction between these two equations, we obtain

$$(X^{c'} - X^{c''})c_1 = b(s'_v - s''_v) + p(s'_e - s''_e) + (m' - m'').$$

Multiplying by $2(X^{c'} - X^{c''})^{-1}$ to the equation,

$$2c_1 = b\frac{2(s'_v - s''_v)}{X^{c'} - X^{c''}} + p\frac{2(s'_e - s''_e)}{X^{c'} - X^{c''}} + \frac{2(s'_m - s''_m)}{X^{c'} - X^{c''}} =: 2b\hat{v} + 2p\hat{e} + 2\hat{m}.$$

From the similar discussion in the upper bound of $|2\hat{m}|$, we obtain

$$\begin{aligned} |2\hat{v}| &\leq \sqrt{n_d} \cdot |s'_v - s''_v| \cdot \left| \frac{2}{X^{c'} - X^{c''}} \right| \\ &\leq n_d(|s'_v| + |s''_v|) \\ &\leq \tilde{\mathcal{O}}(n_d^2 \alpha). \end{aligned}$$

Since a similar discussion also holds for $\hat{e}$, we have $|2\hat{e}| \leq \tilde{\mathcal{O}}(n_d^2\alpha)$.

In addition, dealing with the verification equation regarding t_2 in a similar way of t_1, we obtain

$$2c_2 = a\frac{2(s'_v - s''_v)}{X^{c'} - X^{c''}} + p\frac{2(s'_f - s''_f)}{X^{c'} - X^{c''}} =: 2a\hat{v} + 2p\hat{f}.$$

In the same way, we obtain $|2\hat{f}| \leq \tilde{\mathcal{O}}(n_d^2\alpha)$.

High-entropy commitments This property directly follows from the security of the auxiliary commitment scheme. □