A Two-Party Hierarchical Deterministic Wallets in Practice

ChihYun Chuang[1], IHung Hsu[1], and TingFang Lee[2]

[1] AMIS, Taipei, Taiwan {chihyun, glen}@maicoin.com
[2] NYU Grossman School of Medicine, New York NY, USA Ting-Fang.Lee@nyulangone.org

Abstract. The applications of Hierarchical Deterministic Wallet are rapidly growing in various areas such as cryptocurrency exchanges and hardware wallets. Improving privacy and security is more important than ever. In this study, we proposed a protocol that fully support a two-party computation of BIP32. Our protocol, similar to the distributed key generation, can generate each party's secret share, the common chain-code, and the public key without revealing a seed and any descendant private keys. We also provided a simulation-based proof of our protocol assuming a rushing, static, and malicious adversary in the hybrid model. Our master key generation protocol produces up to total of two bit leakages from a honest party given the feature that the seeds will be re-selected after each execution. The proposed hardened child key derivation protocol leads up to a one bit leakage in the worst situation of simulation from a honest party and will be accumulated with each execution. Fortunately, in reality, this issue can be largely mitigated by adding some validation criteria of boolean circuits and masking the input shares before each execution. We then implemented the proposed protocol and ran in a single thread on a laptop which turned out with practically acceptable execution time. Lastly, the outputs of our protocol can be easily integrated with many threshold sign protocols.

Keywords: Secure Two-party Computation · BIP32 · Wallets.

1 Introduction

Blockchains are well-known for their role in cryptocurrency systems, where they provide a secure and decentralized record of transactions without requiring a trusted third party. Digital signatures are used to authorize and validate transactions in which users must protect their private keys. If a private key is lost, the user loses all assets that are associated with it. This makes key management crucial. However, managing multiple private keys is challenging with the growing demand of blockchains. Hierarchical Deterministic (abbrev. HD) Wallet[3] and $t-n$ threshold signature scheme (abbrev. TSS) are current two methods of managing private keys in blockchains. HD wallet offers convenience, and TSS provides greater security. The goal of this paper is to employ the advantages of these methods, as previously commented in [10, Section 6.4].

HD wallets were introduced in the blockchain world to simplify private key management. With HD wallets, users utilize deterministic algorithms described in BIP32

[3] https://en.bitcoin.it/wiki/Deterministic_wallet

[31] (ref. Figures 1 and 3), a Bitcoin Improvement Proposal, to generate multiple private keys. Ideally, they generate a set of random words called "seed" S, and then use HMAC-SHA512("Bitcoin seed", S) to derive the master private key k. Next, users derive the new private key, which is $k + \text{HMAC-SHA512}(\text{k}, \text{chaincode}, \text{keyID})$. By iterating this process, BIP32 provides a tree structure of private keys generating paths. BIP32 allows users to securely manage and generate a large number of addresses by securing only one random seed.

There are some potential security issues of HD wallets. First, a compromised private key can lead to the loss of all descendant private keys, which becomes a risk with improper ECDSA implementations. For instance, [4] used lattice attacks against ECDSA signatures to demonstrate that repeated or non-uniform generation of the nonce can potentially enable an attacker to compute the long-term signing key. Second, if given a parent public key and any non-hardened child private keys, it is possible to easily recover the parent private key. To address this issue, [13] proposed a new HD wallet (i.e. different from the formula defined by BIP32) that can tolerate the leakage of up to m private keys, with a parent public key size of $O(m)$.

TSS enables t out of n parties to generate signatures on behalf of a group. In most TSS implementations, each party holds the same public key and their own secret, also known as a "share". The private key can only be compromised when all t shares are stolen simultaneously. However, TSS also features "proactive refresh" [26], which voids shares produced from an old epoch. TSS is secure, robust, and flexible key management schemes that allow for signature generation without needing for private keys [21]. In practice, users often use multiple addresses to manage their assets, causing management burden for service providers who must manage numerous shares for different addresses. Another popular solution is multi-signature[4], in which a valid transaction requires multiple keys instead of a single signature from one key. Multi-signature also avoids the risk of a single point of failure and is easier to implement than TSS. However, the main drawback of multi-signature is its significantly higher cost in blockchains.

Current TSS protocols cannot fully support BIP32's key generation protocols. To address this, we employ garbled circuit protocols [34] that allow parties to compute a predetermined function using their inputs while revealing only the output. To ensure security against malicious adversaries, various garbled circuits have been proposed, which we will discuss further in subsection 2.1. For efficiency, we employ the DualEx protocol [14], an enhanced semi-honest protocol with dual execution against malicious adversaries. It involves conducting two separate runs of a garbled-circuit protocol against a semi-honest adversary with the parties swapping roles, followed by a secure equality test that leaks no more than one bit against a malicious adversary. The challenge of this paper is to design a "TSS-type BIP32 protocol".

Unbound Security[5], a pioneer in multi-party computation technology, has developed a two-party HD wallet protocol that utilizes the garbled circuit scheme. Unbound's idea of the master key generation protocol and child key derivations is to produce the results through DualEx that conform to BIP32 standard. To prevent attackers from using incorrect inputs, they added some randomized information such that many shares of a

[4] https://en.bitcoin.it/wiki/Multi-signature

[5] https://github.com/unboundsecurity/blockchain -crypto-mpc

key are outputs. Eventually, both parties receive the correct public key and validate their own shares.

In our study, we also apply the DualEx protocol [14] to BIP32. Our protocol uses two layers of twist masking r_i and n_i of i-th party to prevent cheating, with the DualEx output $s' := \text{secret} + r_i * n_{1-i} + r_{1-i} * n_i$. Each party then announces $r_i G$ and n_i sequentially to ensure the correctness of s'. Our protocol's efficiencies for child key derivation are comparable to Unbound's, while Unbound's master key generation is more efficient. However, it is unclear what security model Unbound considered, whereas our protocols were proven to be secure in the hybrid-world.

As far as we know, there have been no formal papers that study two-party BIP32. Our contributions in this work are as follows: 1) We present a protocol that employs two-party computation, including generating a master private key, hardened private keys, and non-hardened private keys, that complies with all BIP32 regulations (Figures 2 and 4); 2) We provide a security proof for our protocol in the hybrid model using simulation-based methods. We utilize the concepts of real/ideal worlds, as described in [14], to demonstrate that the outputs of the two worlds are indistinguishable. Our master key generation protocol allows attackers to use up to two arbitrary boolean circuit functions, costing up to a two-bit leakage from the honest party in the ideal world. However, this is not a significant security concern because the seeds are re-selected after each execution, preventing the leakage from accumulating. In contrast, our hardened child key derivation protocol permits attackers to use only one arbitrary circuit function, resulting in up to a one-bit leakage from the honest party that accumulates with each execution. Nevertheless, this issue can be mitigated considerably by adding some circuit function validation criteria and masking the input shares before each execution. The leakage of bits in the ideal world is the worst-case scenario, and can be prevented in the real world, as discussed in [14]; 3) We implement the proposed protocol running in a single thread on a laptop, without the need of downloading boolean circuits, as each execution uses the same boolean circuits. The average time spent for the master key generation is 7.37s, and for the hardened child key derivation it is 2.74s. The efficiency can be achieved if network bandwidth allows. However, the number of executions required to generate keys or shares for individual users is usually less than significant. The master key generation and the translation of the child key only need to be performed once in most cases. Therefore, a slightly longer execution time is acceptable and practical. More importantly, the outputs of our protocol can be easily integrated with many sign protocols [5, 7, 9–11, 20, 28, 32] to generate signatures. The complete implementations and benchmarks will be explained in Section 5.

2 Preliminaries

Notation. The sets of natural numbers, integers are denoted by $\mathbb{N}$, $\mathbb{Z}$ respectively. Given a prime integer $p \in \mathbb{N}$, we define $\mathbb{Z}_p$ to be the quotient field of $\mathbb{Z}/p\mathbb{Z}$, and $\mathbb{Z}_p^\times$ to be the multiplicative group of $\mathbb{Z}_p$. Let E be an elliptic curve over $\mathbb{Z}_p$, and G be a point of order q on the curve E. The point $b \cdot G := \overbrace{G + \cdots + G}^{b-\text{times}}$ is the scalar multiplication of E by a non-negative integer b. In the rest of this paper, we will use the commonly used

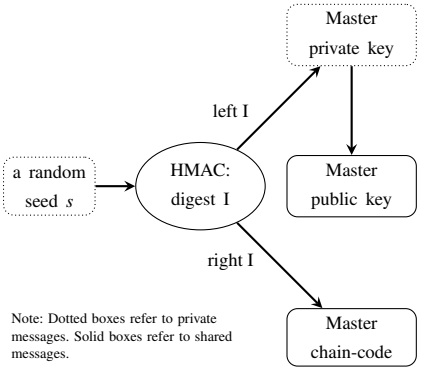

Fig. 1: BIP32: The Master Key Generation.

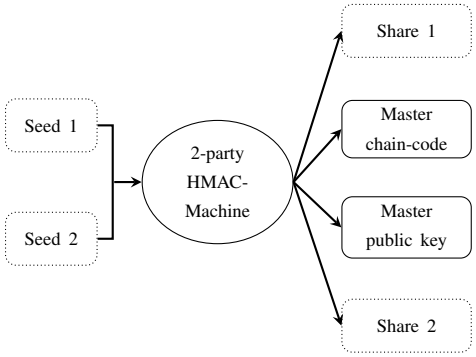

Fig. 2: BIP32: 2-Party Master Key Generation.

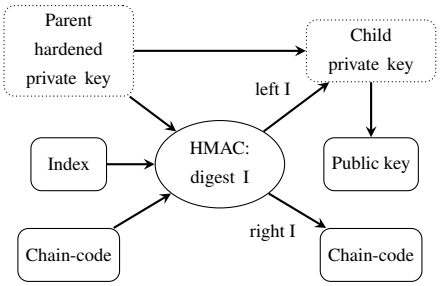

Fig. 3: BIP32: The Child Key Derivation.

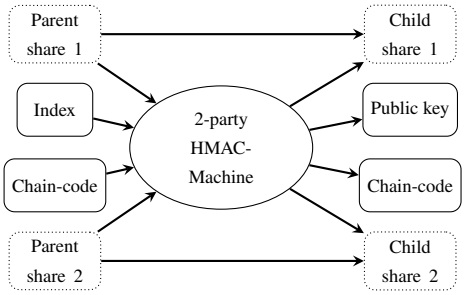

Fig. 4: BIP32: 2-Party Child Key Derivation.

elliptic curve defined by secp256k1[6] in which q is a prime. Given m, $n \in \mathbb{N}$, we let $\mathrm{ser}_m(n)$ serialize a m-bit unsigned integer n as a $\left\lceil \frac{m}{8} \right\rceil$-byte sequence, most significant byte first, $\mathrm{parse}_m(n)$ interpret a $\left\lceil \frac{m}{8} \right\rceil$-byte sequence n as a m-bit number, most significant byte first, and $\mathrm{ser}_P(Q)$ serialize the coordinate pair $Q = (x, y)$ as a byte sequence using SEC1's compressed form. Denote H to be a hash function and HMAC-SHA512 to be the function specified in RFC 4231[22]. Lastly, the notation $\oplus$ is the XOR operation and $\|$ is the concatenation.

2.1 Circuit Garbling

A boolean circuit f can be parameterized by $(n, m, \ell, \mathrm{Gates})$ with a chosen topological order, where n is the number of input, m is the number of output, ℓ is the cardinality of all wires, and Gates is the set of all gates. In our study, the Bristol fashion boolean circuits [1] were employed, and the gates, AND, XOR, EQ, and INV, of boolean circuits are commonly used here.

[6] https://en.bitcoin.it/wiki/Secp256k1

Protocol 1 The Master Key Generation

Input: a positive integer n.
output: a master secret key $\text{parse}_{256}(\text{I}_L)$ and a master chain-code I_R.

1: Sample a seed S with the byte-length n.
2: Compute $\text{I} := \text{HMAC-SHA512}(\text{“Bitcoin seed”}, S)$.
3: Split $\text{I} = \text{I}_L \| \text{I}_R$, where the byte-length of I_L is equal to the byte-length of I_R.
4: If $\text{parse}_{256}(\text{I}_L) \geq q$ or $\text{parse}_{256}(\text{I}_L) = 0$, then restart this protocol.

Protocol 2 Child Key Derivation

Input: a parent private key k_{par}, a chain-code c_{par}, and an index j.
output: a child secret key k_j with index j and the corresponding chain code c_j or $\perp$.

1: Compute I according to the below three cases:
- if $2^{31} \leq j < 2^{32}$(i.e. hardened): $\text{I} := \text{HMAC-SHA512}\big(c_{\text{par}}, \text{0x00} \| \text{ser}_{256}(k_{\text{par}}) \| \text{ser}_{32}(j)\big)$.
- if $0 \leq j < 2^{31}$(i.e. non-hardened): $\text{I} := \text{HMAC-SHA512}\big(c_{\text{par}}, \text{ser}_{\text{P}}(k_{\text{par}} \cdot G) \| \text{ser}_{32}(j)\big)$.
- the others: return $\perp$.

2: Split $\text{I} = \text{I}_L \| \text{I}_R$, where the byte-length of I_L is equal to the byte-length of I_R.
3: Compute the child key $k_j := \text{parse}_{256}(\text{I}_L) + k_{\text{par}} \mod q$
4: If $\text{parse}_{256}(\text{I}_L) \geq q$ or $k_j = 0$, then return $\perp$. Otherwise return k_j and $c_j := \text{I}_R$.

Circuit garbling [34] allows that two semi-honest parties use their own secrets to evaluate a prior agreed boolean circuit function without leaking any information from their inputs beyond what is revealed by the function output itself. One party represents the garbled-circuit generator and the another party represents the garbled-circuit evaluator. The generator begins the garbling process by associating both values of each binary wire with random labels and then passes the garbled circuit to the evaluator. After receiving the garbled circuit, the evaluator then evaluates the circuit without knowing the meaning of the input-wire labels. Evaluator will have the final output-wire labels and be able to learn the meaning. A standard definition of garbling scheme is proposed by Bellare et al. [3] as follows:

Definition 1. *A garbling scheme consists of four algorithms* [35, Section 2]*:*

GB*: On input $\mathbf{1}^k$ and a boolean circuit f, outputs (F,e,d). Here F is a garbled circuit, e is encoding information, and d is decoding information.*
En*: On input (e,x), where e is as above and x is an input suitable for f, outputs a garbled input X.*
Ev*: On input (F,X) stated as above, outputs a garbled output Y.*
De*: On input (d,Y) stated as above, outputs a plain output y.*

The algorithms have the following conditions:

Correctness*: For any circuit f and input x, and sampling $(F,e,d) \leftarrow \mathbf{GB}(\mathbf{1}^k, f)$, we have*

$$\mathbf{De}\left(d, \mathbf{Ev}\big(F, \mathbf{En}(e,x)\big)\right) = f(x).$$

Privacy*: There exists a simulator S that takes input $(\mathbf{1}^k, f, f(x))$ and whose output is indistinguishable from (F,X,d) generated the usual way.*

Obliviousness*: There must exist a simulator S that takes input* $(\mathbf{1}^k, f)$ *and whose output is indistinguishable from* (F, X) *generated the usual way.*
Authenticity*: Given input* (F, X) *alone, no adversary should be able to produce* $\tilde{Y} \neq \mathbf{Ev}(F, X)$ *such that* $\mathbf{De}(d, \tilde{Y}) \neq \perp$*, except with negligible probability.*

A well-known and efficient example of Definition 1 is the half-gates construction [3] which is compatible with free-XOR [17] and point-and-permute [23]. We adapt the Dual-Ex protocol proposed by Huang et al. [14] to defend malicious attackers in the garbling scheme. This protocol suggests that two parties take turns in playing the generator and evaluator to garble the same inputs and boolean circuits. Next, both parties will verify if the results from both sides are the same through a secured validation protocol. Assuming that garbling circuit scheme is able to against semi-honest adversaries, the protocol guaranties that the two parties leak up to 1 bit information.

The current 2-party computation protocols against malicious can be categorized as garbled circuit paradigms, and secret sharing paradigms. Firstly, a secret sharing paradigm (e.g. SPDZ [16]) considers an arithmetic circuit in which the number of rounds is correlated with circuit depth. This is not practical for SHA512 due to huge amount of complicated bit operations [2]. Next, garbled circuit paradigms against malicious include DualEx protocol [14], cut-and-choose to generate garbled circuits [18], and authenticated garbling [30, 15, 33]. During the execution of these protocols, cut-and-choose needs to deliver n garbled circuit to reach statistical security 2^{-n}. Authenticated garbling generates certain number of authenticated shares which is associated with the number of gates during pre-processing stage which cost higher in communication than garble against a semi-honest adversary. However, DualEx protocol [14] only uses two garbled circuits against a semi-honest adversary to enable a nearly active security with a constant number of rounds. Therefore, considering the efficiency, we gave priority to the DualEx protocol in our study.

2.2 Oblivious Transfer

Oblivious transfer (abbrev. OT) is a fundamental cryptographic primitive in the multi-party computation. To be more specific, 1-n OT protocol has two characters called *sender* and *receiver* such that the receiver asks for one of the sender's n messages, which satisfies two conditions: the receiver does not know the other messages except for his select message, and the sender does not know the receiver's choice. So far, OT has become the most widely used building block in the two-party setting, such as garbling scheme. However, garbling scheme requires a large number of OT protocols, which leads to computation and efficiency burden. OT extension protocols are introduced to mitigate the issue by using small number of OTs as seeds and symmetric key encryption to compute a very large number of OTs.

In our implementation, we employed an efficient OT extension against malicious adversaries protocol that was proposed in Canetti et al.[6]. Their protocol is highly efficient, and results in a 3 round extended OT that is UC secure in the observable[7]

[7] Observable means that the challenger can observe the input points at which the adversary makes queries to the random oracle.

random oracle model assuming computational Diffie–Hellman assumption. Additionally, we also add OT index into the key generating process in the i-th base OT instance according to Ian McQuoid et al. [24, Section 3.3] to avoid all OT instances producing identical outputs. More precisely, we replace $\mathrm{H}(\mathrm{rid}, B_i^r)$ and $\mathrm{H}\left(\mathrm{rid}, \left(\frac{B_i}{T}\right)^r\right)$ with $\mathrm{H}(\mathrm{rid}, B_i^r, i)$ and $\mathrm{H}\left(\mathrm{rid}, \left(\frac{B_i}{T}\right)^r, i\right)$ (ref. [6, Fig8]).

2.3 Two-party Computation

The major adversaries in the secure two-party computation are semi-honest and malicious adversaries. Semi-honest adversaries run the protocol honestly, but try to learn as much as possible from the outputs received from other parties. Malicious adversaries may not follow the protocol and execute any computation for stealing and corrupting information. In our study, we focus on static, malicious, and rushing adversaries. Static means that the adversary is restricted to choose a set of parties to corrupt before the protocol execution starts and cannot change this set after. Rushing adversary is allowed to observe the honest parties' messages before modifying its own messages when protocol proceeds in rounds. Our protocol employs symmetric messaging within the same round of parties, without any prescribed order. As a result, the adversary always has the ability to see the honest parties' messages before deciding on their own message, leading us to assume a rushing adversary.

A two-party computation can be modeled as a two variables function f, that is for any $x, y \in \{0,1\}^*$, the function $f(x,y) := (f_0(x,y), f_1(x,y))$, where $f_0, f_1 : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$. One party with the input of the first coordinate x obtains $f_0(x,y)$ and another party with the input of the second coordinate y obtains $f_1(x,y)$. We extended the security definition in Huang et al. [14] to allow that adversary can learn the information from $f_i(x,y)$ where $i = 0$ or 1 and k arbitrary boolean functions $g_1(x,y), \ldots, g_k(x,y)$. The same as security discussion of Huang et al., we consider $k = 2$ in our protocol in which the correctness and input independence still hold. In the k-leakage model, the adversary can learn k extra bit of information about the honest user's input, without being detected.

We adapted the real/ideal world definition in Huang et al. [14, Section V] and [19, Section 6.2]. In the real model, we assume the adversary owns a secret input from one of the parties and any auxiliary input **aux**, and attacks with an arbitrary polynomial-time strategy. The honest party follows the protocol to execute. Let Π be a two-party protocol computing f, and $\mathrm{REAL}_{\Pi,\mathcal{A}(\mathbf{aux}),i}(x,y,n)$ be the output pair of the honest party where $\mathcal{A}$ is the adversary from the real execution of Π.

The ideal model is permitted that the adversary may obtain k additional bit of the information about the honest party's input.As is customary in most two-party computation scenarios, we consider security with abort, meaning that the adversary obtains the output while the honest party does not. Consider $\mathcal{P}_i$ one of the two participants $\mathcal{P}_0$ and $\mathcal{P}_1$ is adversary. An ideal execution for the computation of f proceeds as follows:

1. Let x (resp. y) denote the input of party $\mathcal{P}_0$ (resp. $\mathcal{P}_1$). The adversary $\mathcal{A}$ also has an auxiliary input denoted by **aux**. All parties are initialized with the same value on their security parameter tape.

2. Both parties send inputs to the trusted third party. However, the corrupted party controlled by $\mathcal{A}$ may send any value of its choice. Let the pair of inputs sent to the trusted party be (x', y'). Additionally, the adversary sends k arbitrary boolean function $g_1, \ldots g_k$ to the trusted party[8].
3. The trusted party computes $f(x', y') = (f_0(x', y'), f_1(x', y'))$ and $g_1(x', y'), \ldots, g_k(x', y')$, and sends the outputs to the adversary. At this point, the adversary $\mathcal{P}_i$ may tell the trusted party to **abort**, in which case the honest party receives $\perp$. Otherwise, the adversary tells the trusted party to **continue**, in which case the honest party receives $f_{1-i}(x', y')$.
4. The honest party outputs whatever it received from the trusted party and $\mathcal{A}$ outputs an arbitrary function of its own choice.

Then, the ideal execution of $f = (f_0, f_1)$, with inputs (x, y), security parameter n, and auxiliary input **aux** to $\mathcal{A}$, denoted by $\text{IDEAL}_{f,\mathcal{A}(\mathbf{aux}),i}(x, y, n)$ for $i \in \{0, 1\}$ which is the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

A distribution ensemble $\{X(a,n)\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$. We denote $X \overset{\text{c}}{\equiv} Y$ to be two distribution ensembles $X := \{X(a,n)\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ and $Y := \{Y(a,n)\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ are computationally indistinguishable. We have the following definition.

Definition 2. *Let $k \geq 1$ be a positive integer, f be a two-party functionality, and Π be a two-party protocol that computes f. Protocol Π is said to securely compute f with k-bit leakage if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $i \in \{0, 1\}$,*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(\mathbf{aux}),i}(x, y, n)\right\}_{x,y,\mathbf{aux},n} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}(\mathbf{aux}),i}(x, y, n)\right\}_{x,y,\mathbf{aux},n}.$$

where $x, y \in \{0, 1\}^*$ with an equal bit-length of x and y, $\mathbf{aux} \in \{0, 1\}^*$, and $n \in \mathbb{N}$.

3 Our schemes

We describe the two proposed protocols, 2-party master key generation(abbrev. 2P-MKG) and harden key derivation(abbrev. 2P-CKD), in this section. The 2P-MKG allows that while both parties do not know each other's seed s_i, they can receive their own secret share $x_{\text{mas},i}$, the chain-code I_R, and the corresponding public key Q such that

1. HMAC-SHA512("Bitcoin seed", $s_0 \oplus s_1$) $= \text{I}_L \| \text{I}_R$;
2. $x_{\text{mas},0} + x_{\text{mas},1} = \text{parse}_{256}(\text{I}_L) \mod q$;
3. $\text{parse}_{256}(\text{I}_L) \in (0, q)$;
4. $Q = (x_{\text{mas},0} + x_{\text{mas},1}) \cdot G$.

[8] In our security proof, we consider a slight modification of the ideal model described above: namely, the adversary is allowed to adaptively choose g after learning $f(x', y')$. However, this two models are identical. More discussion can be found in [14, Remark, Section V]

When the two parties do not know each other's parent private share $x_{\text{par},i}$, the 2P-CKD allows that both parties utilize their own $x_{\text{par},i}$, an index $2^{31} \leq j < 2^{32}$ of the child private key, chain-code c_{par}, and public key $K_{\text{par}} := \left(x_{\text{par},0} + x_{\text{par},1}\right) \cdot G$ to receive the translation I_L, the new chain-code c_j, and a new public key K_j, which satisfy

i. $\text{HMAC-SHA512}\left(c_{\text{par}}, \text{value}\right) = \mathrm{I}_L \| c_j$, where

$$\text{value} := \text{0x00} \| \text{ser}_{256}\left(x_{\text{par},0} + x_{\text{par},1} \mod q\right) \| \text{ser}_{32}(j);$$

ii. $K_j := K_{\text{par}} + \text{parse}_{256}(\mathrm{I}_L) \cdot G$;
iii. $\text{parse}_{256}(\mathrm{I}_L) \in [0, q)$;
iv. $K_j \neq 0 \cdot G$.

3.1 A protocol of two-party master key generation

Let k and k' be positive integers and $\mathbb{Z}_{q,k}^{\times}$ be the subset of the finite field $\mathbb{Z}_q^{\times}$ with the bit-length of all elements at most k. Consider the following boolean circuits with security parameters $k < 255$ and k':

$$\begin{aligned} f_{\text{mkg}} \colon & \{0,1\}^{k'} \times \mathbb{Z}_q^{\times} \times \mathbb{Z}_{q,k}^{\times} \times \{0,1\}^{k'} \times \mathbb{Z}_q^{\times} \times \mathbb{Z}_{q,k}^{\times} \to \{0,1\}^{513+k} \\ & s_0, r_0, n_0, s_1, r_1, n_1 \mapsto \text{ser}_{256}(\text{parse}_{256}(\mathrm{I}_L) + r_0 n_1 + r_1 n_0 \bmod q) \| \mathrm{I}_R \| \text{ser}_{256}(n_0 + n_1), \end{aligned} \tag{1}$$

and

$$\begin{aligned} f_{\text{aux}} \colon & \{0,1\}^{k'} \times \mathbb{Z}_q^{\times} \times \{0,1\}^{k'} \times \mathbb{Z}_{q,k}^{\times} \to \{0,1\}^{257} \\ & s_0, r_0, s_1, n_1 \mapsto \mathbf{1}_{[0,q)}(\text{parse}_{256}(\mathrm{I}_L)) \| \text{ser}_{256}\left(\text{parse}_{256}(\mathrm{I}_L) + r_0 n_1 \bmod q\right). \end{aligned} \tag{2}$$

Here $\mathrm{I}_L \| \mathrm{I}_R = \text{HMAC-SHA512}(\text{“Bitcoin seed”}, s_0 \oplus s_1)$, and $\mathbf{1}_{[0,q)}(x)$ is the indicator function of the interval $[0, q)$.

Since k and k' are fixed in the beginning, we omit the notation k and k' in functions f_{mkg} and f_{aux} for simplicity. In the 2P-MKG, neither of two parties know the digest I, so we add the indicator function $\mathbf{1}_{[0,q)}$ and check that the master public key is non-trivial in the step 3 of Protocol 3 to ensure the generating master private key belonging to $(0, q)$. Before introducing our protocol and theorem, we explain our idea briefly.

Idea. The point of 2P-MKG is to make sure that the summation of obtained secret shares, $x_{\text{mas},0}$ and $x_{\text{mas},1}$, is equal to the master private key $s := \text{parse}_{256}(\mathrm{I}_L)$ in which the public key is $s \cdot G$, under the circumstance that neither of them know the master private key and each other's private share. Both parties can access $f_{\text{aux}}(s_0, r_0, s_1, n_1)$ through a garbled circuit to reach an agreement of the public key. During this process, a malicious adversary might attempt to input $n_1 = 0$ in (2) for getting the private key. This issue is resolved by directly sending one non-zero input-wire label of n_1 (ref. step 1 in Protocol 3). Next, both parties use DualEx protocol to verify their outputs of the garbled circuit f_{mkg} are the same. In the mean time, each party uses n_{1-i} that was obtained from the other party to check the algebraic relationship between f_{aux} and f_{mkg}. At this stage, it is very hard for malicious adversaries pass a series of the checking lists through non-consistent inputs without knowing the another party's s_i, r_i, and n_i. The role of r_i is to

mask I_L in f_{mkg} from the other party, and the role of n_i is to temporarily mask r_i through a linear combination within f_{mkg} and f_{aux}. These processes can ensure the consistency of garbled circuit inputs and the correctness of the secret shares.

Let $f = (n, m, \ell, \mathrm{Gates})$ be a boolean circuit. A participant $\mathcal{P}_i$ applies the algorithm $\mathbf{GB}(\mathbf{1}^k, f)$ to output (F_i, e_i, d_i) (resp. $\mathbf{GB}(\mathbf{1}^k, f_*)$ to $(F_{*,i}, e_{*,i}, d_{*,i})$), where $i \in \{0,1\}$. We denote the input-wire matrix to be

$$\mathbb{X}_{F_i} = \begin{bmatrix} X^0_{F_i}[0] & X^0_{F_i}[1] & \dots & X^0_{F_i}[n-1] \\ X^1_{F_i}[0] & X^1_{F_i}[1] & \dots & X^1_{F_i}[n-1] \end{bmatrix}.$$

Here $X^b_{F_i}[k]$ is the k-th input-wire label representing the bit value b. If $v \in \{0,1\}^n$ is a bit-string (i.e. $v = v_0 \| v_1 \| \dots \| v_{n-1}$), then we set $\mathbb{X}^v_{F_i} := (X^{v_1}_{F_i}[0], \dots, X^{v_{n-1}}_{F_i}[n-1])$ which also writes this vector to be $X^v_{F_i}$. Moreover, if v' is a sub-string of v, which means $v' = v_t \| v_{t+1} \| \dots \| v_{t+k-1}$ with k the bit-Length of v', then we set $\mathbb{X}^{v'}_{F_i} := (X^{v_t}_{F_i}[t], \dots, X^{v_{t+k-1}}_{F_i}[t+k-1]) = X^{v'}_{F_i}$. Set $\mathbb{X}_{F_i}(0)$ (resp. $\mathbb{X}_{F_i}(1)$) to be the participant $\mathcal{P}_0$'s (resp. $\mathcal{P}_1$'s) input-wire matrix. Given two $d_1 \times d_2$ matrices $A = [a_{ij}]$, $B = [b_{ij}]$, we define

$$A \| B = \begin{bmatrix} a_{11} & \dots & a_{1d_2} & b_{11} & \dots & b_{1d_2} \\ a_{21} & \dots & a_{2d_2} & b_{21} & \dots & b_{2d_2} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{d_1 1} & \dots & a_{d_1 d_2} & b_{d_1 1} & \dots & b_{d_1 d_2} \end{bmatrix}_{d_1 \times 2d_2}.$$

Therefore, given an input $x = x_0 \| x_1$ with x_j being $\mathcal{P}_i$'s input, one has

$$\mathbf{En}(e_i, x) = \mathbb{X}^x_{F_i} = \mathbb{X}^{x_0}_{F_i} \| \mathbb{X}^{x_1}_{F_i} = \mathbb{X}^{x_0}_{F_i}(0) \| \mathbb{X}^{x_1}_{F_i}(1).$$

For the output-wire matrix

$$\mathbb{Y}_{F_i} := \begin{bmatrix} Y^0_{F_i}[0] & Y^0_{F_i}[1] & \dots & Y^0_{F_i}[m-1] \\ Y^1_{F_i}[0] & Y^1_{F_i}[1] & \dots & Y^1_{F_i}[m-1] \end{bmatrix},$$

we define the hash matrix of $\mathbb{Y}_{F_i}$

$$\mathrm{H}\mathbb{Y}_{F_i} := \begin{bmatrix} \mathrm{H}\big(Y^0_{F_i}[0]\big) & \mathrm{H}\big(Y^0_{F_i}[1]\big) & \dots & \mathrm{H}\big(Y^0_{F_i}[m-1]\big) \\ \mathrm{H}\big(Y^1_{F_i}[0]\big) & \mathrm{H}\big(Y^1_{F_i}[1]\big) & \dots & \mathrm{H}\big(Y^1_{F_i}[m-1]\big) \end{bmatrix}.$$

We consider Protocol 3 runs in a hybrid world where the parties are given access to the trusted party computing two functionalities: oblivious transfer f_{OT} (ref. Figure 5) and validation testing f_{VT} (ref. Figure 6). Then one has the following theorem:

Theorem 1. *Assume that a garbling scheme satisfies the properties: correctness, privacy, and obliviousness, a hash function H is modeled as an random oracle, and the discrete logarithm problem is hard then π_{MKG} is securely compute $f_{\pi_{\mathrm{MKG}}}$ with 2-bit leakage in the hybrid world described above.*[9] *Here for input $s_0, s_1 \in \{0,1\}^{k'}$, the function*

[9] We remark that our protocol securely computes $f_{\pi_{\mathrm{MKG}}}$ with 2-bit leakage if the trusted entities are replaced by any secure protocols against malicious adversaries.

Fig. 5: The ideal functionality f_{OT} for Oblivious Transfer.

- The input of sender: $(m_0, m_1) \in \{0,1\}^k \times \{0,1\}^k$.
- The input of receiver: a bit $b \in \{0,1\}$.
- The output of sender: $\perp$.
- The output of receiver: the message m_b.

Fig. 6: The ideal functionality f_{VT} for Validation Test.

- The input of $\mathcal{P}_0$: x_0.
- The input of $\mathcal{P}_1$: x_1.
- The output of $\mathcal{P}_0$ and $\mathcal{P}_1$: if x_0 equals x_1, return 1; otherwise, return 0.

Fig. 7: The ideal functionality f_{RO} for Random Oracle.

- Secure parameter: n.
- Variables: initially empty list L.
- The input: x.
- The output: Find r such that $(x,r) \in L$, then return r. If such r doesn't exist, uniformly sample $r \in \{0,1\}^n$ and add (x,r) to L and return r.

$f_{\pi_{\mathrm{MKG}}} = (f_0, f_1)$ *with*

$$f_i(s_0, s_1) := \left(\frac{\left(\mathrm{parse}_{256}\left(\mathrm{I}_L\right) + (-1)^i \cdot r\right)}{2} \mod q, \mathrm{I}_R, \mathrm{parse}_{256}\left(\mathrm{I}_L\right) \cdot G \right) \in \mathbb{Z}_q \times \{0,1\}^{256} \times \mathbb{G},$$

for r uniformly random in $\mathbb{Z}_q$ *and* HMAC-SHA512("Bitcoin seed", $s_0 \oplus s_1$) $= \mathrm{I}_L \| \mathrm{I}_R$.

We remark that the most important secret seeds of both parties in our protocol will be re-selected at each execution to assure the accumulative risk of leaking information is up to 2 bits. The correctness of our theorem is briefly explained as follows and the security proof given in Appendix B.1.

Correctness. According to the Protocol of π_{MKG}, for $i \in \{0,1\}$, the plaintext $(w_{\mathrm{mkg},i}, w_{\mathrm{aux},i})$ is $(\mathrm{parse}_{256}(\mathrm{I}_L) + r_i \cdot n_{1-i} + r_{1-i} \cdot n_i \mod q,\ \mathrm{parse}_{256}(\mathrm{I}_L) + r_{1-i} \cdot n_i \mod q)$, and the participant $\mathcal{P}_i's$ $x_{\mathrm{mas},i}$ is $\frac{w_{\mathrm{mkg},i}}{2} - (n - n_i) r_i = \frac{1}{2}\left(\mathrm{parse}_{256}\left(\mathrm{I}_L\right) - r_i n_{1-i} + r_{1-i} n_i\right) \mod q$. Thus, $\mathrm{parse}_{256}\left(\mathrm{I}_L\right) = x_{\mathrm{mas},0} + x_{\mathrm{mas},1} \mod q$, and $Q = w_{\mathrm{aux},i} \cdot G - n_i \cdot R_{1-i} = \mathrm{parse}_{256}(\mathrm{I}_L) \cdot G$. On the other hand, the chain-code I_R has gotten in the phase 5 of Protocol 3.

3.2 A protocol of two-party child key derivation

In the child key derivation, non-hardened key generation is obvious since all inputs are public information (ref. Protocol 2). We will then focus on the case of hardened key in this section. Let k be a positive integer. Consider the following boolean circuit with

Protocol 3 π_{MKG}: 2-party master key generation

Input: two security parameters k, k' and a seed $s_i \in \{0,1\}^{k'}$.
output: a share $x_{\text{mas},i}$, the corresponding public key Q, and the chain code.

1: Each participant $\mathcal{P}_i$:
- randomly chooses $r_i \in \mathbb{Z}_q^\times$ and an odd integer $n_i \in [1, 2^k)$.
- performs **GB** in Definition 1 with f_{mkg} and f_{aux} to obtain $(F_{\text{mkg},i}, e_{\text{mkg},i}, d_{\text{mkg},i})$, and $(F_{\text{aux},i}, e_{\text{aux},i}, d_{\text{aux},i})$.
- computes $\mathbb{X}^{s_i\|\text{ser}_{256}(r_i)\|\text{ser}_k(n_i)}_{F_{\text{mkg},i}} = \mathbf{En}\big(e_{\text{mkg},i}, (s_i\|\text{ser}_{256}(r_i)\|\text{ser}_k(n_i)\big)$ and $\mathbb{X}^{s_i\|\text{ser}_{256}(r_i)}_{F_{\text{aux},i}} = \mathbf{En}\big(e_{\text{aux},i}, (s_i\|\text{ser}_{256}(r_i))\big)$.
- performs the OT protocol with $\mathcal{P}_{1-i}$ to obtain garbled input-wire labels $\mathbb{X}^{s_i\|\text{ser}_{256}(r_i)\|\text{ser}_k(n_i)}_{F_{\text{mkg},1-i}}$ (resp. $\mathbb{X}^{s_i\|\text{ser}_k(n_i)}_{F_{\text{aux},1-i}}$) except for $X^1_{F_{\text{mkg},1-i}}[\dot{n}_{\text{mkg},i}]$ (resp. $X^1_{F_{\text{aux},1-i}}[\dot{n}_{\text{aux},i}]$), where $\dot{n}_{\text{mkg},i}$ (resp. $\dot{n}_{\text{aux},i}$) is the index of all input-wire labels corresponding to the least significant bit of n_i. Sends $X^1_{F_{\text{mkg},i}}[\dot{n}_{\text{mkg},1-i}]$, $X^1_{F_{\text{aux},i}}[\dot{n}_{\text{aux},1-i}]$, and the first message of OT.

2: Each participant $\mathcal{P}_i$:
- After OT completed, sends the garbled message $(F_{\text{aux},i}, \mathbb{X}^{s_i\|\text{ser}_{256}(r_i)}_{F_{\text{aux},i}}, d_{\text{aux},i})$, and the point $r_i \cdot G$ to the participant $\mathcal{P}_{1-i}$.

3: Each participant $\mathcal{P}_i$:
- interprets the coming point to be R_{1-i}.
- uses **Ev** to get $Y_{F_{\text{aux},1-i}}$ and then applies **De** to obtain the binary string $v_{\text{aux},i}$, interpret $v_{\text{aux},i}$ to be the result of the indicator function $\mathbf{1}_{[0,q)}(\text{I}_L)$ and $w_{\text{aux},i}$.
- denotes $Q := w_{\text{aux},i} \cdot G - n_i \cdot R_{1-i}$ which is the shared public key. If $Q = 0 \cdot G$, then sets Q be an arbitrary point of $\mathbb{G}$.
- if the indicator function $\mathbf{1}_{[0,q)}(\text{I}_L) = 0$, then assign $(F_{\text{mkg},i}, \mathbb{X}^{s_i\|\text{ser}_{256}(r_i)\|\text{ser}_k(n_i)}_{F_{\text{mkg},i}}, \text{H}\mathbb{Y}_{F_{\text{mkg},i}})$ to be an arbitrary chosen message with the valid length.
- performs the validation test in Protocol 5 with $\mathcal{P}_{1-i}$ to compare the shared public key Q.

4: Each participant $\mathcal{P}_i$:
- sends $(F_{\text{mkg},i}, \mathbb{X}^{s_i\|\text{ser}_{256}(r_i)\|\text{ser}_k(n_i)}_{F_{\text{mkg},i}}, \text{H}\mathbb{Y}_{F_{\text{mkg},i}})$ to the participants $\mathcal{P}_{1-i}$.

5: Each participant $\mathcal{P}_i$:
- uses **Ev** to get $Y_{F_{\text{mkg},1-i}}$ and the incoming hash table $\text{H}\mathbb{Y}_{F_{\text{mkg},1-i}}$ to obtain the binary string $v_{\text{mkg},i}$. Interpret $v_{\text{mkg},i}$ to be $w_{\text{mkg},i}$, chain-code, and n.
- verifies $w_{\text{mkg},i} = v_{\text{aux},i} + (n - n_i) \cdot r_i \mod q$.
- computes $x_{\text{mas},i} := \dfrac{w_{\text{mkg},i}}{2} - (n - n_i) \cdot r_i \mod q$
- computes
$$h_{\text{mkg},i} := \begin{cases} \text{H}\left(\mathbb{Y}^{v_{\text{mkg},i}}_{F_{\text{mkg},i}} \| Y_{F_{\text{mkg},1-i}}\right), & \text{if } i = 0, \\ \text{H}\left(Y_{F_{\text{mkg},1-i}} \| \mathbb{Y}^{v_{\text{mkg},i}}_{F_{\text{mkg},i}}\right), & \text{if } i = 1. \end{cases}$$
- performs the validation test in Protocol 5 with $\mathcal{P}_{1-i}$ to compare $h_{\text{mkg},i}$, and $h_{\text{mkg},1-i}$. If they are equal, then the protocol is complete. Otherwise, abort.

security parameters $k < 255$:

$$\begin{aligned} f_{\text{ckd}} : \mathbb{Z}_q^\times \times \mathbb{Z}_q^\times \times \mathbb{Z}_q^\times \times \mathbb{Z}_{q,k}^\times \times \{0,1\}^{32} \times \mathbb{Z}_q^\times \times \mathbb{Z}_q^\times \times \mathbb{Z}_q^\times \times \mathbb{Z}_{q,k}^\times \times \{0,1\}^{256} &\to \{0,1\}^{769+k} \\ s_0, r_0, m_0, n_0, j, s_1, r_1, m_1, n_1, c_{\text{par}} &\mapsto \text{I}(c_{\text{par}}, j, s_0, s_1, m_0, m_1) \| \text{ser}_{256}(s_0 + s_1 + m_0 + m_1 + r_0 n_1 + r_1 n_0 \mod q) \\ &\quad \| \text{ser}_{256}(n_0 + n_1). \end{aligned} \tag{3}$$

Protocol 4 π_{CKD}: 2-party child key derivation

Input: a security parameter k, a parent share $x_{\mathrm{par},i}$, an index j, the chain-code c_{par}, and the public parent key K_{par}.

output: the offset I_L of the private child key, child chain-code c_j, and the corresponding public key K_j.

1: Each participant $\mathcal{P}_i$:
- randomly chooses $r_i \in \mathbb{Z}_q^\times$, $m_i \in \mathbb{Z}_q$, and an odd integer $n_i \in [1, 2^k)$.
- computes $\tilde{x}_{\mathrm{par},i} := x_{\mathrm{par},i} - m_i \mod q$.
- performs **GB** in Definition 1 with f_{ckd} to obtain $(F_{\mathrm{ckd},i}, e_{\mathrm{ckd},i}, d_{\mathrm{ckd},i})$.
- computes $\mathbb{X}^{b_i}_{F_{\mathrm{ckd},i}} = \mathbf{En}\big(e_{\mathrm{ckd},i}, (b_i)\big)$ and $\mathrm{H}\mathbb{Y}_{F_{\mathrm{ckd},i}}$, where b_i is the bit string representing the inputs $\tilde{x}_{\mathrm{par},i}$, r_i, m_i, n_i, j, c_{par}.
- performs the OT protocol with $\mathcal{P}_{1-i}$ to obtain garbled input-wire labels $\mathbb{X}^{b'_i}_{F_{\mathrm{ckd},1-i}}$ except for $X^1_{F_{\mathrm{ckd},1-i}}[\dot{n}_{\mathrm{ckd},i}]$, where $\dot{n}_{\mathrm{ckd},i}$ is the index of all input-wire labels corresponding to the least significant bit of n_i and b'_i is the bit string representing the inputs $\tilde{x}_{\mathrm{par},i}$, r_i, m_i, n_i. Send the first message of OT, and $X^1_{F_{\mathrm{ckd},i}}[\dot{n}_{\mathrm{ckd},1-i}]$.

2: Each participant $\mathcal{P}_i$:
- After OT completed, sends $r_i \cdot G$ to $\mathcal{P}_{1-i}$.

3: Each participant $\mathcal{P}_i$:
- interprets the incoming points to be R_{1-i}
- sends the garbled message
$$(F_{\mathrm{ckd},i}, \mathbb{X}^{b_i}_{F_{\mathrm{ckd},i}}, \mathrm{H}\mathbb{Y}_{F_{\mathrm{ckd},i}})$$
to the participant $\mathcal{P}_{1-i}$.

4: Each participant $\mathcal{P}_i$:
- performs **Ev** to get the output-wire labels $Y_{F_{\mathrm{ckd},1-i}}$.
- uses $\mathrm{H}\mathbb{Y}_{F_{\mathrm{ckd},i}}$ to learn the bit-string $v_{\mathrm{ckd},i}$, and interpret $v_{\mathrm{ckd},i}$ to be I_L, the child chain-code c_j, $w_{\mathrm{ckd},i}$, and n.
- verifies $w_{\mathrm{ckd},i} \cdot G = K_{\mathrm{par}} + (n - n_i) \cdot r_i \cdot G + n_i \cdot R_{1-i}$.
- computes
$$h_{\mathrm{ckd},i} := \begin{cases} \mathrm{H}\left(\mathbb{Y}^{v_{\mathrm{ckd},i}}_{F_{\mathrm{ckd},i}} \| Y_{F_{\mathrm{ckd},1-i}}\right), & \text{if } i = 0, \\ \mathrm{H}\left(Y_{F_{\mathrm{ckd},1-i}} \| \mathbb{Y}^{v_{\mathrm{ckd},i}}_{F_{\mathrm{ckd},i}}\right), & \text{if } i = 1. \end{cases}$$
- performs the validation test in Protocol 5 with $\mathcal{P}_{1-i}$ to compare $h_{\mathrm{ckd},i}$, and $h_{\mathrm{ckd},1-i}$. If they are equal, then the protocol is complete. Otherwise, abort.

$$f_{\mathrm{ckd}} : \mathbb{Z}_q^\times \times \mathbb{Z}_q^\times \times \mathbb{Z}_q^\times \times \mathbb{Z}_{q,k}^\times \times \{0,1\}^{32} \times \mathbb{Z}_q^\times \times \mathbb{Z}_q^\times \times \mathbb{Z}_q^\times \times \mathbb{Z}_{q,k}^\times \times \{0,1\}^{256} \to \{0,1\}^{769+k}$$
$$s_0, r_0,\ m_0, n_0, j, s_1, r_1, m_1, n_1, c_{\mathrm{par}} \mapsto \mathrm{I}(c_{\mathrm{par}}, j, s_0, s_1, m_0, m_1) \| \mathrm{ser}_{256}(s_0 + s_1 + m_0 + m_1 + r_0 n_1 + r_1 n_0 \mod q)$$
$$\| \mathrm{ser}_{256}(n_0 + n_1). \tag{4}$$

Here $\mathrm{I}(c_{\mathrm{par}}, j, s_0, s_1, m_0, m_1) := \mathrm{HMAC\text{-}SHA512}(c_{\mathrm{par}}, 0\mathrm{x}00 \| \mathrm{ser}_{256}(s_0 + s_1 + m_0 + m_1 \mod q) \| \mathrm{ser}_{32}(j))$. Similarly, we also omit the notation k in f_{ckd}. Likewise, we explain the idea of our protocol and then introduce our protocol and theorem.

Idea. The goal of our 2P-CKD is establishing the offset I_L from both correct parent private shares under the condition that neither of them know each other's parent private

share and child secret key even though they know the digest of HMAC-SHA512. The main concept is to directly utilize the DualEX protocol to obtain I_L such that both parties receive the same output. It can also confirm that the input of garbled circuit is the expected parent private share through the algebraic relationship between the parent public key and $(s_0+s_1+m_0+m_1+r_0n_1+r_1n_0)\cdot G$. Here $s_i+m_i \mod q$ is the parent secret share where m_i is to cover up the share against selective failure attack[10]. This is because even if the attacker guesses the value of some bit values of s_i and m_i when executing OT, it is still difficult to know the bit value corresponding to s_i+m_i, because s_i+m_i and s_i may differ by q. The role of r_i is to mask secret share from the other party, and the role of n_i is to mask r_i through a linear combination within f_{ckd}.

Theorem 2. *Assume that a garbling scheme satisfies the properties: correctness, privacy, and obliviousness, a hash function H is modeled as an random oracle, and the discrete logarithm problem is hard then* π_{CKD} *is securely compute* $f_{\pi_{\mathrm{CKD}}}$ *with* 1*-bit leakage in the hybrid world described as in the Theorem 1. Here for input shares* $x_{\mathrm{par},i} \in \mathbb{Z}_q$ *respectively and the parent public key*

$$K_{\mathrm{par}} = \left(x_{\mathrm{par},0}+x_{\mathrm{par},1}\right)\cdot G = K_{\mathrm{par},0} = K_{\mathrm{par},1},$$

the function[11] $f_{\pi_{\mathrm{CKD}}}(x_{\mathrm{par},0}, K_{\mathrm{par},0}, x_{\mathrm{par},1}, K_{\mathrm{par},1}) :=$

$$\begin{cases} (\mathrm{I},\mathrm{I}), & \text{if } K_{\mathrm{par},0} = K_{\mathrm{par},1} \text{ and} (x_{\mathrm{par},0}+x_{\mathrm{par},1})\cdot G = K_{\mathrm{par},1}; \\ \text{the adversary gets I, the honest party gets } \perp & \text{if } K_{\mathrm{par},0} = K_{\mathrm{par},1} \text{and} (x_{\mathrm{par},0}+x_{\mathrm{par},1})\cdot G \neq K_{\mathrm{par},1}; \\ (\perp,\perp), & \text{otherwise.} \end{cases}$$

and $\mathrm{I} = \mathrm{HMAC\text{-}SHA512}\left(c_{\mathrm{par}}, 0\mathrm{x}00 \| \mathrm{ser}_{256}\left(x_{\mathrm{par},0}+x_{\mathrm{par},1} \bmod q\right) \| \mathrm{ser}_{32}(j)\right)$.

At first, both parties learn I in this protocol, so they can directly verify if $\mathrm{parse}_{256}(\mathrm{I}_L) \in [0,q)$ and $K_{\mathrm{par}} + \mathrm{parse}_{256}(\mathrm{I}_L)\cdot G$ is non-trivial. Second, our protocol frequently executes the derivations from parent private key to child private key, which will cause higher computational cost. To maintain the efficiency, we decide to assign the same outputs I_L to both parties rather than different outputs like Protocol 4. With this design, the security is still guaranteed because as long as the parent private share is secured and then the child private share remains secured. We also transfer the inputs of the parent secret share $x_{\mathrm{par},i}$ for $i \in \{0,1\}$ from both parties into $x_{\mathrm{par},i} - m_i \mod q$ to prevent selective failure attack. For example, the sender can assign the desired value to a certain bit to steal the secret from the receiver while executing OT.

In addition, the worst scenario is to leak 1 bit in each execution according to the security proof. To be more specific, an attacker can manipulate the output, parent private key (i.e. $s_0+s_1+m_0+m_1 = x_{\mathrm{par},0}+x_{\mathrm{par},1}$ is the input of the circuit function f_{ckd}), of an arbitrary boolean function g to be an assigned bit. Since the parent private key will not be changed or re-selected, the attacker is able to learn the parent private key after attacking certain times. To tackle this issue, we embed a verifying criteria to the garble circuit functions that are given by each party such as number of wires, ADD

[10] A malicious party attempts to learn the inputs of honest party through either putting bad entries into a garbled truth table, or by providing bad input-wire labels in the OT for some values.

[11] With the output I, the parties can easily compute (I_L, c_j, K_j) the output of π_{MKG}.

gates, EQ gates, INV gates, and XOR gates. These fundamental parameters follow a topological order, which induce that the attacker can only execute false function attack by controlling the non-free gates. We anticipate that it is not trivial for the attacker to manipulate g under such restriction in the real world. The correctness of our theorem is briefly explained in below and the security proof is given in Appendix B.2.

Correctness. Taking $s_i = x_{\text{par},i} - m_i \mod q$ in f_{ckd}, two parties both obtain the same offset I_L, so they set the child private share $x_{\text{chd},i} := x_{\text{par},i} + \frac{\text{parse}_{256}(\text{I}_L)}{2} \mod q$. Then

$$\begin{aligned} x_{\text{chd},0} + x_{\text{chd},1} =& x_{\text{par},0} + x_{\text{par},1} + \text{parse}_{256}(\text{I}_L) \mod q \\ =& k_{\text{par}} + \text{parse}_{256}(\text{I}_L) \mod q \\ =& k_j \mod q. \end{aligned}$$

4 Application: Two party Hierarchical Deterministic Wallet

Our protocol is to demonstrate how two parties can collaborate to generate valid shares that support HD wallet, but not the signatures. In most of current key generation protocols, a private key is the linear combination of shares, so we can integrate the shares generated by our protocols in the current sign protocol to produce signatures when $t = n = 2$. Most threshold signature protocols assume that each party's input includes private share s_i and the associated public key P satisfying one of the following two cases: 1) If $t \leq n$, $P = \sum_{i=1}^{t} (\lambda_i s_i) \cdot G$, where λ_i is the corresponding Lagrange coefficient of participants $\mathcal{P}_1, ..., \mathcal{P}_t$. 2) If $t = n$, $P = \sum_{i=1}^{n} s_i \cdot G$. In particular, when $t = n = 2$, two cases are identical with letting $\tilde{s}_i = \lambda_i s_i \mod q$ in case 1).

After performing our protocol 3, the participant $\mathcal{P}_i$ generates a master key share $x_{\text{mas},i}$, the chain-code, and the public key. Each party only needs to memorize its own master key share. Given a derivation path $(m/i_1/.../i_n)$ with depth n, where m is the master key and $(m/i_1/.../i_n)$ is the i_n-th child key of $(m/i_1/..../i_{n-1})$. Let $k_{i_1/.../i_n}$ be the private key of the derivation path $(m/i_1/.../i_n)$ and $\text{I}_{i_1/.../i_n}$ be the offset that satisfies

$$k_{i_1/.../i_n} = k_{i_1/.../i_{n-1}} + \text{I}_{i_1/.../i_n} \mod q.$$

Given a path $(m/i_1/.../i_n)$, the private key $k_{i_1/.../i_n}$ can be written as

$$x_{\text{mas},0} + x_{\text{mas},1} + \left(\text{I}_{i_1} + \text{I}_{i_1/i_2} + \cdots + \text{I}_{i_1/.../i_n}\right) \mod q.$$

In the hardened key case (i.e. $2^{31} \leq i_j < 2^{32}$), the offset $\text{I}_{i_1/.../i_j}$ can be generated iteratively from protocol 4. In the non-hardened key case (i.e. $0 \leq i_j < 2^{31}$), each party derives the offset using the parent public key, the parent chain-code, and the index i_j through Protocol 2. In conclusion, each party only needs to memorize the master key share, $x_{\text{mas},i}$, the set of offsets $\{\text{I}_{i_1/.../i_j} : 2^{31} \leq i_1, ..., i_j < 2^{32}\}$, the chain-codes, and the

public keys associated to each hardened key in the database. Meanwhile, a canonical selection of $\mathcal{P}_i's$ share associated with the key $k_{i_1/\ldots/i_n}$ is given by

$$x_{\text{mas},i} + \frac{1}{2}\left(\mathrm{I}_{i_1} + \mathrm{I}_{i_1/i_2} + \cdots + \mathrm{I}_{i_1/\ldots/i_n}\right) \mod q.$$

Considering the matter of backups, an appropriate setting for current multi-party computation wallets is $t = 2$ and $n = 3$. Our protocol, even though we let $t = 2$ and $n = 2$, can add a new backup share using two known master key shares through "add share" [25].

The above discussion integrates our protocols and hot wallet. Moreover, we also can incorporate cold wallets and hot wallets. We split a private key, that was derived from a path in HD wallet, into the case $t = n = 2$ through Shamir's secret sharing [29]. If a hot wallet supports Protocol 4, then it can take these shares as parent shares to derive child private shares. The advantage is that the derived accounts can be used to manage assets in cold wallets. The assets will not be stolen when the share in mobile wallets was lost, since the cold wallets can still make transactions and the other share is still secured. The roles of hot and cold wallets in our proposal are similar to saving and checking accounts. The integration of hot and cold wallets can ease the backup issues of mobile wallets.

5 Implementation, Benchmarks, and Evaluation

We illustrate how to implement the Protocol 3 and 4[12] and evaluate the efficiency in this section. The garbled scheme were generated using Two Halves Make a Whole [35] to enhance the efficacy. We employ OT that proposed by Canetti et al.[6] and optimize the security according to the suggestions in McQuoid et al. [24, Section 3.3]. In the validation protocol, we let both parties use one-side validation protocol (ref. Protocol 5) to verify that both outputs from garbled circuit are the same. The detailed discussion can be found in Appendix A. Additionally, we follow the Bristol fashion to generate three major boolean circuits of f_{mkg}, f_{aux} and f_{ckd}. To improve the consumption during circuit execution, we follow the process below:

1. We can focus on SHA-512, since HMAC-SHA512 is iteratively derived by two SHA-512. SHA-512 first processes through padding the input message to n 1024 bits length messages $m_0, \ldots, m_{n-1}$, then generates the output using compression function(abbrev. CF) iteratively. More precisely, a compression function has a 1024 bit-length and a 512 bit-length inputs and then produces a 512 bit-length output. Given an initial 512 bit-length message $m'^{(0)}$, let

$$m'^{(i)} := \mathrm{CF}\left(m_{i-1}, m'^{(i-1)}\right),\ i = 1, \ldots, n.$$

   $m'^{(n)}$ is the digest. In our scenario, we consider the padding messages of SHA-512 are 2048 bits and the first 1024 bits are all the same excluding seeds and shares. To lessen the iterations of compression function, we pre-calculate the first state and use it as our circuit input to improve the efficiency of SHA-512.

[12] https://github.com/getamis/alice/tree/AddMoreCircuit

Table 1: Basic information of our circuits including number of gates and wires of f_{mkg}, f_{aux}, and f_{ckd}.

	f_{mkg}	f_{aux}	f_{ckd}
Total number of gates	847,420	796,827	507,581
Total number of wires	850,558	799,676	510,463
Total number of ADD Gates	162,054	145,784	107,442

Table 2: Time consuming for running 20 samples.

	2P-MKG	2P-CKD
Fast time	7.181s	2.688s
Slow time	7.748s	2.839s
Avg time	7.374s $\pm$ 0.14s	2.739s $\pm$ 0.04s

The bit-length of seed = 512; The bit-length of n = 33; The bit-length of Paillier public key = 2048; The number of base OT = 128; The half-gates scheme uses the Hash function $\widehat{\text{MMO}}^{E}$ (ref. [12, Section 7.3]) with the ideal cipher AES-128.

2. In CKD, we only need to compute compression function once to mask both parties' inputs and then get the output of HMAC-SHA512, since both sides will obtain I.
3. We reduce the number of gates for the boolean circuit of multiplication and provide some useful boolean circuits such as evaluating the inequality and mod q.

All benchmarks without downloading boolean circuits were run in a single-threaded on an Intel i5 CPU 2.3GHz and 16GB 2133MHz LPDDR3 of RAM in the 13-inch (2018) macbook pro, because each execution uses the same boolean circuits.

6 Future work

Theoretically, there will be risk in leaking bits while using DualEx protocol since we considered the worst scenario without any restrictions on g in the security proof. We conjecture that, practically, the input of the honest party will not leak one bit while executing the proposed protocol. To overcome the risk of leaking bits, we can potentially utilize garbled circuit schemes against malicious adversaries such as authenticated Garbling method. Because our proof was given in the hybrid world, it is needed to investigate secure instances of the functionality f_{VT}. Lastly, the other direction of future work is to enhance the efficiency and improve the cost of memory through reducing the number of boolean circuits, gates, wires, and to achieve optimal rounds of the proposed protocols.

References

1. 'Bristol Fashion' MPC Circuits, https://homes.esat.kuleuven.be/ nsmart/MPC/
2. Sha512 and its implementation in noir, https://research.aragon.org/sha512-noir.html
3. Bellare, M., Hoang, T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 784–796. CCS '12, Association for Computing Machinery (10 2012)
4. Breitner, J., Heninger, N.: Biased Nonce Sense: Lattice Attacks Against Weak ECDSA Signatures in Cryptocurrencies, pp. 3–20. Springer-Verlag (09 2019)

5. Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., Peled, U.: Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1769–1787. Association for Computing Machinery (10 2020)
6. Canetti, R., Sarkar, P., Wang, X.: Blazing Fast OT for Three-Round UC OT Extension, pp. 299–327. Springer International Publishing (04 2020)
7. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-Efficient Threshold EC-DSA, pp. 266–296. Springer International Publishing (04 2020)
8. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from ddh. In: Topics in Cryptology — CT-RSA 2015. Springer International Publishing (04 2015)
9. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ecdsa from ecdsa assumptions. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 980–997 (05 2018)
10. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ecdsa with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1179–1194. Association for Computing Machinery (10 2018)
11. Gennaro, R., Goldfeder, S.: One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive (2020)
12. Guo, C., Katz, J., Wang, X., Yu, Y.: Efficient and secure multiparty computation from fixed-key block ciphers. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 825–841 (05 2020)
13. Gutoski, G., Stebila, D.: Hierarchical deterministic bitcoin wallets that tolerate key leakage. In: Financial Cryptography. pp. 497–504 (01 2015)
14. Huang, Y., Katz, J., Evans, D.: Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. Proceedings - IEEE Symposium on Security and Privacy pp. 272–284 (05 2012). https://doi.org/10.1109/SP.2012.43
15. Katz, J., Ranellucci, S., Rosulek, M., Wang, X.: Optimizing Authenticated Garbling for Faster Secure Two-Party Computation, pp. 365–391. Lecture Notes in Computer Science, Springer (07 2018)
16. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making spdz great again. In: Advances in Cryptology – EUROCRYPT 2018. pp. 158–189. Springer International Publishing (01 2018)
17. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free xor gates and applications. In: Automata, Languages and Programming. vol. 7, pp. 486–498. Springer Berlin Heidelberg (07 2008)
18. Lindell, Y.: Fast cut-and-choose-based protocols for malicious and covert adversaries. Journal of Cryptology **29**, 456–490 (04 2016)
19. Lindell, Y.: How to Simulate It – A Tutorial on the Simulation Proof Technique, pp. 277–346. Springer International Publishing (04 2017)
20. Lindell, Y.: Fast secure two-party ecdsa signing. Journal of Cryptology **34** (10 2021)
21. Luis, A., Nicky, M., Apostol, V.: Threshold schemes for cryptographic primitives (2019), https://doi.org/10.6028/NIST.IR.8214
22. M., N.: Identifiers and test vectors for hmac-sha-224, hmac-sha-256, hmac-sha-384, and hmac-sha-512 (2005), https://datatracker.ietf.org/doc/html/rfc4231
23. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay—a secure two-party computation system. In: Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13. p. 20. SSYM'04, USENIX Association (2004)
24. McQuoid, I., Rosulek, M., Roy, L.: Batching Base Oblivious Transfers, pp. 281–310. Springer International Publishing (12 2021)
25. Nojoumian, M., Stinson, D., Grainger, M.: Unconditionally secure social secret sharing scheme. IET Information Security (IFS), Special Issue on Multi-Agent and Distributed Information Security **4**, 202 – 211 (12 2010)

26. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing. pp. 51–59. PODC '91, Association for Computing Machinery (01 1991)
27. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology — EUROCRYPT '99. vol. 5, pp. 223–238. Springer Berlin Heidelberg (05 1999)
28. Pettit, M.: Efficient Threshold-Optimal ECDSA, pp. 116–135. Springer-Verlag (12 2021)
29. Shamir, A.: How to share a secret (1979). Commun. ACM pp. 475–478 (02 2021)
30. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 21–37. Association for Computing Machinery (10 2017)
31. Wuille, P.: Hierarchical deterministic wallets (2013), https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
32. Xue, H., Au, M.H., Xie, X., Yuen, T., Cui, H.: Efficient online-friendly two-party ecdsa signature. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 558–573. Association for Computing Machinery (11 2021)
33. Yang, K., Wang, X., Zhang, J.: More efficient mpc from improved triple generation and authenticated garbling. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1627–1646. CCS '20, Association for Computing Machinery (10 2020)
34. Yao, A.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). vol. 10, pp. 162 – 167 (11 1986)
35. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Advances in Cryptology - EUROCRYPT 2015. vol. 9057, pp. 220–250. Springer Berlin Heidelberg (04 2015)

Appendix A Validation Protocol

We focus on two commonly used additive homomorphic encryptions, Paillier cryptosystem[27] and CL homomorphic encryption[8] in this section. Let sk_i be the private key of participant $\mathcal{P}_i$, and pk_i be the public key corresponding to sk_i, $\bigoplus$ be the homomorphic addition, and $\bigotimes$ be the scalar multiplication. We first recall the one side validation protocol in [14, Figure 5].

Protocol 5 One-side validation protocol

Input: a secret value h_i, a private key sk_i and public keys pk_0 and pk_1 from both participants.

Output: $\mathcal{P}_i$: if $h_i = h_{1-i}$ then return true; otherwise return false; $\mathcal{P}_{1-i}$: $\perp$.

1: The participant $\mathcal{P}_i$:
   - computes a ciphertext $C_i := \mathrm{En}_i(-h_i)$. Here En_i is the encryption function of the public key pk_i.
   - sends C_i to $\mathcal{P}_{1-i}$.

2: The participant $\mathcal{P}_{1-i}$:
   - randomly chooses r_{1-i}, s_{1-i} and computes $C_{1-i} := \mathrm{En}_i(r_{1-i}(h_{1-i} - h_i) + s_{1-i}) = (r_{1-i} \bigotimes C_i) \bigoplus \mathrm{En}_i(r_{1-i}h_{1-i} + s_{1-i})$ and $h := \mathrm{H}(s_{1-i}, h_{1-i})$.
   - sends C_{1-i}, and h, to $\mathcal{P}_i$.

3: The participant $\mathcal{P}_i$:
   - computes $s_i := \mathrm{De}_i(C_{1-i})$. Here De_i is the decryption function of the private key sk_i.
   - verifies $h = \mathrm{H}(s_i, h_i)$. If the equality holds, then output **true**; otherwise, output **false**.

As stated in the paper [14, Section 3-C] applying above protocol twice sequentially with the parties swapping roles does not achieve the simulation-based security against malicious adversaries. Even if the one-side validation protocol is resistant to malicious attackers, it is still not an instance of the functionality f_{VT}. It is because that the malicious can insert the different inputs h_i in twice executions, which produces the outputs of two parties are divergent. However, this protocol satisfies that 1) no information leaks from the honest party; 2) and the probability is negligible that the honest party receives the output **true** when the malicious party is cheating. An instance of the functionality f_{VT} is needed for future work.

Appendix B Security Proofs

In this section, we prove Theorem 1 and Theorem 2.

B.1 A Security Proof for Theorem 1

First, we ignore the event $\mathrm{parse}_{256}(\mathrm{I}_L) \notin (0, q)$ since this event occurs with probability lower than 2^{-127}. That is, we will skip the checks, which are the non-trivial public key and the non-zero of indicator function $\mathbf{1}_{[0,q)}$ in the Protocol 3. Without loss of generality, we assume that $\mathcal{P}_1$ is the adversary since the behaviors of $\mathcal{P}_0$ and $\mathcal{P}_1$ are symmetrical.

Let $\mathcal{A}$ be a hybrid-world adversary, which controls the corrupted party $\mathcal{P}_1$. We show that we can construct an adversary $\mathcal{S}$, executing in our ideal world where the parties have access to a trusted entity computing $f_{\pi_{\mathrm{MKG}}}$, that has the same effect as $\mathcal{A}$ in the hybrid world. Note that we consider if $\mathcal{S}$ receives any invalided messages in the simulation, then records an "error". The simulation details are as follows:

Step 1: The adversary $\mathcal{S}$:

- uses its inputs s_1 and auxiliary input **aux** running $\mathcal{A}$ on the same inputs.
- randomly sample an odd number $n_0 \in [1, 2^k)$ and generates $(\widetilde{F_{\mathrm{aux},0}}, \widetilde{X_{F_{\mathrm{aux},0}}})$ and $(\widetilde{F_{\mathrm{mkg},0}}, \widetilde{X_{F_{\mathrm{mkg},0}}})$. Here $\widetilde{X_{F_{\mathrm{aux},0}}} = \mathbb{X}^{v}_{F_{\mathrm{aux},0}}$ and $\widetilde{X_{F_{\mathrm{mkg},0}}} = \mathbb{X}^{v'}_{F_{\mathrm{mkg},0}}$ for some $v, v' \in \{0,1\}^*$.
- sends garbled input-wire labels $\widetilde{X_{F_{\mathrm{mkg},0}}}[\dot{n}_{\mathrm{mkg},1}]$ and $\widetilde{X_{F_{\mathrm{aux},0}}}[\dot{n}_{\mathrm{aux},1}]$ to $\mathcal{A}$ where $\dot{n}_{\mathrm{mkg},1}$ (resp $\dot{n}_{\mathrm{aux},1}$) is the index of all input-wire labels of $\widetilde{X_{F_{\mathrm{mkg},0}}}$ (resp. $\widetilde{X_{F_{\mathrm{aux},0}}}$) corresponding to the least significant bit of n_1.
- receives the input-wire labels $X_{F_{\mathrm{aux},1}}[\dot{n}_{\mathrm{aux},0}]$ and $X_{F_{\mathrm{mkg},1}}[\dot{n}_{\mathrm{mkg},0}]$ from $\mathcal{A}$.
- simulates OT functionality on the inputs $(\hat{s_1}, n_1', r_1) \in \{0,1\}^{k'} \times \{0,1\}^{k-1} \times \mathbb{Z}_q$ of $\mathcal{A}$. Let $n_1 := 2n_1' + 1 \in \{0,1\}^k$. Send garbled input-wire labels $\widetilde{X_{F_{\mathrm{mkg},0}}}(1)$, $\widetilde{X_{F_{\mathrm{aux},0}}}(1)$ to $\mathcal{A}$, (except $\widetilde{X_{F_{\mathrm{aux},0}}}[\dot{n}_{\mathrm{aux},1}]$ and $\widetilde{X_{F_{\mathrm{mkg},0}}}[\dot{n}_{\mathrm{mkg},1}]$). Here $X(i)$ is the $\mathcal{P}_i's$ input-wire labels and X is the all input-wire labels.
- simulates OT functionality to obtain the input-wire labels excluding the $\dot{n}_{\mathrm{aux},0}$-th input-wire label and the $\dot{n}_{\mathrm{mkg},0}$-th input-wire label respectively of $\mathcal{A}$, denoted by $\mathbb{X}'_{F_{\mathrm{aux},1}}$ and $\mathbb{X}'_{F_{\mathrm{mkg},1}}$.

Step 2: If $\mathcal{S}$ has recorded the "error" then $\mathcal{S}$ randomly samples $s_1' \in \{0,1\}^{k'}$. Otherwise, $\mathcal{S}$ sets $s_1' := \hat{s_1}$. The next step, $\mathcal{S}$ provides s_1' to the trust party, and apply the functionality $f_{\pi_{\mathrm{MKG}}}$ to obtain the output

$$((\mathrm{parse}_{256}(\mathrm{I}_L) - r)/2 \mod q, \mathrm{I}_R, \mathrm{parse}_{256}(\mathrm{I}_L) \cdot G).$$

Let

$$v_{\mathrm{mkg},1} := \mathrm{ser}_{256}(\mathrm{parse}_{256}(\mathrm{I}_L) - r + 2n_0 r_1 \mod q) \| \mathrm{I}_R \| \mathrm{ser}_{256}(n_0 + n_1),$$

and

$$v_{\mathrm{aux},1} := \mathrm{ser}_{256}(\mathrm{parse}_{256}(\mathrm{I}_L) - r + n_0 r_1 \mod q).$$

Step 3: $\mathcal{S}$ lets $r_0 \cdot G$ to be

$$n_1^{-1}\Big((\mathrm{parse}_{256}(\mathrm{I}_L) - r) \cdot G - \mathrm{parse}_{256}(\mathrm{I}_L) \cdot G + n_0 \cdot r_1 \cdot G\Big) = n_1^{-1}((-r) \cdot G + n_0 \cdot r_1 \cdot G).$$

Generate $\widetilde{d_{\mathrm{aux},0}}$ such that

$$\mathbf{De}(\widetilde{d_{\mathrm{aux},0}}, \mathbf{Ev}(\widetilde{F_{\mathrm{aux},0}}, \widetilde{X_{F_{\mathrm{aux},0}}})) = v_{\mathrm{aux},1}.$$

Send the point $r_0 \cdot G$ and $(\widetilde{F_{\mathrm{aux},0}}, \widetilde{X_{F_{\mathrm{aux},0}}}(0), \widetilde{d_{\mathrm{aux},0}})$ to $\mathcal{A}$. Receive a point R_1, and a tuple $(F_{\mathrm{aux},1}, X_{F_{\mathrm{aux},1}}(1), d_{\mathrm{aux},1})$.

Step 4: $\mathcal{A}$ submits some input q_1 for the validation test. $\mathcal{S}$ then defines $g_0 : \{0,1\}^{k'} \to \{0,1\}$ as following:

- On input $x \in \{0,1\}^{k'}$, use the bits of x as selector bits to define the input-wire labels $\mathbb{X}^{\prime x}_{F_{\text{aux},1}}$. Let $n'_0 := n_0 >> 1$ be the right shift 1 of n_0 and set the labels $\mathbb{X}^{\prime x}_{F_{\text{aux},1}}$ combining the other input-wire labels $X_{F_{\text{aux},1}}[\dot{n}_{\text{aux},0}]$, $\mathbb{X}^{\prime \text{ser}_{k-1}(n'_0)}_{F_{\text{aux},1}}$ and $X_{F_{\text{aux},1}}(1)$ corresponding the inputs n_0, s_1, r_1 be X. Perform **Ev** on X, $F_{\text{aux},1}$, and **De** on $d_{\text{aux},1}$ to obtain a plain output $v_{\text{aux},0}$, sets $w_{\text{aux},0} := \text{parse}_{256}(v_{\text{aux},0})$ and let $q_0 := w_{\text{aux},0} \cdot G - n_0 \cdot R_1$. If $\mathcal{S}$ has recorded "error", then let q_0 to be an arbitrary value.
- The output of $g_0 = 1$ if $q_1 = q_0$. Otherwise $g_0 = 0$.

$\mathcal{S}$ sends g_0 to the trusted party, receives a return $g_0(s_0) \in \{0,1\}$, and passes $g_0(s_0)$ to $\mathcal{A}$.

Step 5: If $g_0(s_0) = 0$, then $\mathcal{S}$ records "error". Let $\widetilde{Y_{F_{\text{mkg},0}}} := \mathbf{Ev}\left(\widetilde{F_{\text{mkg},0}}, \widetilde{X_{F_{\text{mkg},0}}}\right)$. Generate the output-wire label table

$$\mathbb{Y}_{F_{\text{mkg},0}} := \begin{pmatrix} Y_{00} & Y_{01} & \ldots & Y_{0n-1} \\ Y_{10} & Y_{11} & \ldots & Y_{1n-1} \end{pmatrix},$$

where $Y_{ij} = \widetilde{Y_{F_{\text{mkg},0}}}[j]$ for $j \in \{0, \ldots, n-1\}$. If j-th bit of $v_{\text{mkg},1}$ is i, then the other entry of column j is a random bit-string with the equal bit-length. One can learn $v_{\text{mkg},1}$ by comparing the hash output matrix $\text{H}\mathbb{Y}_{F_{\text{mkg},0}}$. Send $(\widetilde{F_{\text{mkg},0}}, \widetilde{X_{F_{\text{mkg},0}}}(0), \text{H}\mathbb{Y}_{F_{\text{mkg},0}})$ to $\mathcal{A}$. Receive a tuple $(F_{\text{mkg},1}, X_{F_{\text{mkg},1}}(1), \text{H}\mathbb{Y}_{F_{\text{mkg},1}})$ and the second input $h_{\text{mkg},1}$ of the validation test.

Step 6: $\mathcal{S}$ defines the function $g_1 := \{0,1\}^{k'} \to \{0,1\}$ as follows:

- If there exists "error", then $\text{error}_{\text{flag}} = 1$. Otherwise, $\text{error}_{\text{flag}} = 0$. Moreover, if any errors occur in any of the steps below, then $\text{error}_{\text{flag}} = 1$.
- On input $x \in \{0,1\}^{k'}$, let $r_0 := n_1^{-1}(n_0 r_1 - r') \mod q$ where $r' := \text{parse}_{256}(\text{I}'_L) - (\text{parse}_{256}(\text{I}_L) - r) \mod q$ and HMAC-SHA512("Bitcoin seed", $x \oplus s'_1$) $= \text{I}'_L \| \text{I}'_R$.
- Use the bits of x as selector bits to define the input-wire labels $\mathbb{X}^{\prime x}_{F_{\text{mkg},1}}$. Let these labels combining the input-wire labels $X_{F_{\text{mkg},1}}[\dot{n}_{\text{mkg},0}]$, $\mathbb{X}^{\prime \text{ser}_{256}(r_0) \| \text{ser}_{k-1}(n'_0)}_{F_{\text{mkg},1}}$ and $X_{F_{\text{mkg},1}}(1)$ corresponding the inputs n_0, r_0, s_1, n_1, r_1 be X. Perform **Ev** on X, $F_{\text{mkg},1}$ to obtain $Y_{F_{\text{mkg},1}}$ and compare $Y_{F_{\text{mkg},1}}$ with the hash table $\text{H}\mathbb{Y}_{F_{\text{mkg},1}}$ to obtain a plain output $v_{\text{mkg},0}$. If $\text{error}_{\text{flag}} = 1$, set $v_{\text{mkg},0}$ to be a randomly reasonable bit-string.
- Recompute $w_{\text{aux},0}$ by applying the same progress in Step 4. If $\text{error}_{\text{flag}} = 1$, set $w_{\text{aux},0}$ to be a randomly integer in $\mathbb{Z}_q$.
- Interpret $v_{\text{mkg},0}$ to be $w_{\text{mkg},0}$, the chain-code, n and verify if $w_{\text{mkg},0} = w_{\text{aux},0} + (n - n_0) r_0 \mod q$. If the verification successes, then let $h_{\text{mkg},0} := \text{H}\left(\mathbb{Y}^{v_{\text{mkg},0}}_{F_{\text{mkg},0}} \| Y_{F_{\text{mkg},1}}\right)$. Otherwise, let $h_{\text{mkg},0}$ be a randomly reasonable bit-string.
- If $\text{error}_{\text{flag}} = 0$ and $h_{\text{mkg},0} = h_{\text{mkg},1}$, then the output is 1. Otherwise, return 0.

$\mathcal{S}$ sends g_1 to the trusted party, receives a return $g_1(s_0) \in \{0,1\}$, and passes $g_1(s_0)$ to $\mathcal{A}$.

Step 7: If there exists "error" or $g_1(s_0) = 0$, then $\mathcal{S}$ sends **abort** to the trust party and returns the output of $\mathcal{A}$. If $r_1 \cdot G \neq R_1$, then $\mathcal{S}$ outputs "fail". Otherwise, $\mathcal{S}$ sends **continuous** to the trust party and returns the output of $\mathcal{A}$.

To prove that the hybrid world output and the ideal world output are indistinguishable, we consider four simulators $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$.

1. Let $\mathcal{S}_1$ work in the same way as $\mathcal{S}$ except that $\mathcal{S}_1$ chooses r by itself and has the input s_0 of $\mathcal{P}_0$ as its **aux** instead of receiving r externally from the trusted party.
2. $\mathcal{S}_2$ runs in the same way as $\mathcal{S}_1$ except that $\mathcal{S}_2$ let $r := n_0 r_1 - n_1 r_0$ where r_0 is randomly chosen from $\mathbb{Z}_q$ in the step 1 of the above simulation.
3. $\mathcal{S}_3$ executes in the same way as $\mathcal{S}_2$ except that $\mathcal{S}_3$ follows the strategy of the honest party to generate garbled circuits of the function f_{aux} with the input s_0, r_0.
4. All behaviours of $\mathcal{S}_4$ and $\mathcal{S}_3$ are the same except $\mathcal{S}_4$ follows the strategy of the honest party to generate garbled circuits of the function f_{mkg} with the input s_0, r_0, n_0.

Firstly, it is clear that the distribution ensembles of the ideal world and $\mathcal{S}_1$ are identical, since $\mathcal{S}_1$ and the trust party both randomly choose r. Second, recall that $r_0 = n_1^{-1}(n_0 r_1 - r') = n_1^{-1}(n_0 r_1 - r) \mod q$ and $n_1 \neq 0 \mod q$. Note that the input of g_1 is s_0, which implies $r' = r$. Thus, the distribution of r_0 is the uniform distribution on $[0, q-1]$ if and only if the distribution of r is the uniform distribution on $[0, q-1]$, which leads to that the joint distributions of (r_0, r) generated by $\mathcal{S}_1$ and $\mathcal{S}_2$ are identical. Next, the property Obliviousness of garbled circuit scheme implies that $(\widetilde{F_{\text{aux},0}}, \widetilde{X_{F_{\text{aux},0}}})$ and the standard construction $(F_{\text{aux},0}, X_{F_{\text{aux},0}})$ are indistinguishable. Furthermore, they both give the same output, so the outputs of $\mathcal{S}_2$ and $\mathcal{S}_3$ are computationally indistinguishable. Similarly, the above argument also gives the outputs of $\mathcal{S}_3$ and $\mathcal{S}_4$ are computationally indistinguishable.

In order to prove that $\mathcal{S}_4$ generates a transcript that is indistinguishable from a transcript in the hybrid world, we need the following three lemmas.

Lemma 1. *In the transcripts generated by $\mathcal{S}_4$, the probability of $g_1(s_0) = 1$ and the two bit-strings are different*

$$v_{\text{mkg},0} \neq \text{ser}_{256}(\text{parse}_{256}(\text{I}_L) + n_0 r_1 + n_1 r_0 \mod q) \\ \|\text{I}_R\| \text{ser}_{256}(n_0 + n_1)$$

is negligible. Here $v_{\text{mkg},0}$ is the decoding result in the step 6.

Lemma 2. *The probability of the output "fail" in the simulator $\mathcal{S}_4$ is negligible.*

Lemma 3. *If the output generated by $\mathcal{S}_4$ is neither "fail" nor* **abort***, then the output of $\mathcal{P}_0$ is*

$$\left(\frac{w_{\text{mkg},0}}{2} - (n - n_0) r_0, \text{chain-code}, w_{\text{aux},0} \cdot G - n_0 \cdot R_1\right)$$

which is also the output of $\mathcal{P}_0$ in the protocol π_{MKG} without **abort***.*

Note that the joint output distribution of $\mathcal{S}_4's$ output and the hybrid world output are identical before the step 6. Combining this fact, Lemma 1, Lemma 2, and Lemma 3, one has the outputs of $\mathcal{S}_4$ and π_{MKG} executing in the hybrid world are computational indistinguishable.

B.2 A Security Proof for Theorem 2

Similarly, we assume that $\mathcal{P}_1$ is the adversary. The same setting as Appendix B.1. The details are as follows:

Step 1: The adversary $\mathcal{S}$:
- uses its inputs $x_{\text{par},1}$, K_{par} and auxiliary input **aux** running $\mathcal{A}$ on the same inputs.
- randomly samples an odd number $n_0 \in [1, 2^k)$ and generates $\widetilde{X_{F_{\text{ckd},0}}}$ where $\widetilde{X_{F_{\text{ckd},0}}} = \mathbb{X}^v_{F_{\text{ckd},0}}$ for some $v \in \{0,1\}^*$.
- sends garbled input-wire labels $\widetilde{X_{F_{\text{ckd},0}}}[\dot{n}_{\text{ckd},1}]$ to $\mathcal{A}$. Here $\dot{n}_{\text{ckd},1}$ is the index of all input-wire labels of $\widetilde{X_{F_{\text{ckd},0}}}$ corresponding to the least significant bit of n_1.
- receives the input-wire label $X_{F_{\text{ckd},1}}[\dot{n}_{\text{ckd},0}]$ from $\mathcal{A}$.
- simulates OT functionality on the input $(\hat{x}_{\text{par},1},\ m_1,\ n'_1,\ r_1) \in \mathbb{Z}_q \times \mathbb{Z}_q \times \{0,1\}^{k-1} \times \mathbb{Z}_q$ of $\mathcal{A}$. Let $n_1 := 2n'_1 + 1 \in \{0,1\}^k$. Send garbled input-wire labels $\widetilde{X_{F_{\text{ckd},0}}}(1)$ (except $\widetilde{X_{F_{\text{ckd},0}}}[\dot{n}_{\text{ckd},1}]$) to $\mathcal{A}$.
- simulates OT functionality to obtain the input-wire labels excluding the $\dot{n}_{\text{ckd},0}$-th input-wire label of $\mathcal{A}$, denoted by $\mathbb{X}'_{F_{\text{ckd},1}}$.

Step 2: $\mathcal{S}$ sets $x'_{\text{par},1} := \hat{x}_{\text{par},1} + m_1 \mod q$. If $\mathcal{S}$ has recorded the "error" then $\mathcal{S}$ randomly samples $x''_{\text{par},1} \in \mathbb{Z}_q$. Otherwise, $\mathcal{S}$ sets $x''_{\text{par},1} := x'_{\text{par},1}$. The next step, $\mathcal{S}$ provides $(x''_{\text{par},1}, K_{\text{par}})$ to the trust party, and then obtains the output I.

Step 3: $\mathcal{S}$ uniformly samples $r \in \mathbb{Z}_q$ and sends $R_0 := n_1^{-1}\left((r - n_0 r_1 - x'_{\text{par},1}) \cdot G - K_{\text{par}} + x_{\text{par},1} \cdot G\right)$ to $\mathcal{A}$. Receive R_1 from $\mathcal{A}$.

Step 4: Let $\widetilde{Y_{F_{\text{ckd},0}}} := \mathbf{Ev}\left(\widetilde{F_{\text{ckd},0}}, \widetilde{X_{F_{\text{ckd},0}}}\right)$. $\mathcal{S}$ generates the output-wire label table

$$\mathbb{Y}_{F_{\text{ckd},0}} := \begin{pmatrix} Y_{00} & Y_{01} & \dots & Y_{0n} \\ Y_{10} & Y_{11} & \dots & Y_{1n} \end{pmatrix},$$

where $Y_{ij} = \widetilde{Y_{F_{\text{ckd},0}}}[j]$ for $j \in \{0,\dots,n-1\}$. If j-th bit of $v_{\text{ckd},1} := \mathrm{I} \| \text{ser}_{256}(r) \| \text{ser}_{256}(n_0 + n_1)$ is i, then the other entry of column j is a random bit-string with equal bit-length. One can learn $v_{\text{ckd},1}$ by comparing $\mathrm{H}\mathbb{Y}_{F_{\text{ckd},0}}$. Send $(\widetilde{F_{\text{ckd},0}}, \widetilde{X_{F_{\text{ckd},0}}}(0), \mathrm{H}\mathbb{Y}_{F_{\text{ckd},0}})$ to $\mathcal{A}$. Receive tuple $(F_{\text{ckd},1}, X_{F_{\text{ckd},1}}(1), \mathrm{H}\mathbb{Y}_{F_{\text{ckd},1}})$ and the input $h_{\text{ckd},1}$ of the validation test.

Step 5: $\mathcal{S}$ randomly chooses $m_0 \in \mathbb{Z}_q$ and defines the function $g := \mathbb{Z}_q \to \{0,1\}$ as following:
- If there exists "error", then $\mathcal{S}$ sets $\text{error}_{\text{flag}} = 1$. Otherwise, $\text{error}_{\text{flag}} = 0$. Moreover, if any errors occur in any of the steps below, then $\text{error}_{\text{flag}} = 1$.
- On input $x \in \mathbb{Z}_q$, let $r_0 := n_1^{-1}\left(r - n_0 r_1 - x + x'_{\text{par},1}\right)$. Next, uses the bits of $s := \text{ser}_{256}(x - m_0 \mod q)$ as selector bits to define the input-wire labels $\mathbb{X}^{\prime s}_{F_{\text{ckd},1}}$. Let $n'_0 = n_0 >> 1$ be the right 1 of n_0 and the input-wire labels $\mathbb{X}^{\prime s}_{F_{\text{ckd},1}}$ combining the input-wire labels $\mathbb{X}^{\prime \text{ser}_{256}(r_0)\|\text{ser}_{256}(m_0)\|\text{ser}_{k-1}(n'_0)}_{F_{\text{ckd},1}}, X_{F_{\text{ckd},1}}[\dot{n}_{\text{ckd},0}]$

and $X_{F_{\text{ckd},1}}(1)$ be X. Perform **Ev** on X, $F_{\text{ckd},1}$ to obatin $Y_{F_{\text{ckd},1}}$ and compare the hash table $\text{H}\mathbb{Y}_{F_{\text{ckd},1}}$ to obtain a bit-string $v_{\text{ckd},0}$. If $\text{error}_{\text{flag}} = 1$, set $v_{\text{ckd},0}$ to be a randomly reasonable bit-string.

- Interpret $v_{\text{ckd},0}$ to be I_L, the child chain-code c_j, $w_{\text{ckd},0}, n$, and verifies $w_{\text{ckd},0} \cdot G = K_{\text{par}} + (n - n_0) r_0 \cdot G + n_0 \cdot R_1$. If the verification fails, then set $\text{error}_{\text{flag}} = 1$.
- If $\text{error}_{\text{flag}} = 1$, then set $h_{\text{ckd},0} := \text{H}\left(\mathbb{Y}^{v_{\text{ckd},0}}_{F_{\text{ckd},0}} \| Y_{F_{\text{ckd},1}}\right)$. Otherwise, let $h_{\text{ckd},0}$ to be a randomly reasonable bit-string.
- If $\text{error}_{\text{flag}} = 0$ and $h_{\text{ckd},0} = h_{\text{ckd},1}$, then the output is 1. Otherwise, return 0.

Step 6: If there exists "error" or $g\left(x_{\text{par},0}\right) = 0$, then $\mathcal{S}$ sends **abort** to the trust party and returns the output of $\mathcal{A}$. If $r_1 \cdot G \neq R_1$, then $\mathcal{S}$ outputs "fail". Otherwise, $\mathcal{S}$ sends **continuous** to the trust party and returns the output of $\mathcal{A}$.

We now consider two simulators $\mathcal{S}_1$ and $\mathcal{S}_2$.

1. All actions of $\mathcal{S}_1$ and $\mathcal{S}$ are the same except that $\mathcal{S}_1$ has the input $x_{\text{par},0}$ of $\mathcal{P}_0$ as its **aux**. In addition, $\mathcal{S}_1$ randomly chooses $r_0 \in \mathbb{Z}_q$ in the Step 1 of the above simulation and let $r := x_{\text{par},0} + x'_{\text{par},1} + n_0 r_1 + n_1 r_0$.
2. All actions of $\mathcal{S}_2$ and $\mathcal{S}_1$ are the same except that $\mathcal{S}_2$ follows the strategy of the honest party to generate garbled circuits of the function f_{ckd}.

The indistinguishability for the ideal world output, $\mathcal{S}_1's$ output, and $\mathcal{S}_2's$ output are similar to Theorem 1. We only show that the outputs of $\mathcal{S}_2$ and Protocol 4 executing in the hybrid world are indistinguishable. First, we need the following lemma. We will skip the proof, since it is similar to the Lemma 1.

Lemma 4. *In the execution of $\mathcal{S}_2$, the probability of $g(x_{\text{par},0}) = 1$ and the two bit-strings are different*

$$v_{\text{ckd},0} \neq \text{I} \| \text{ser}_{256}(x_{\text{par},0} + x'_{\text{par},1} + n_0 r_1 + n_1 r_0 \mod q) \| \text{ser}_{256}(n_0 + n_1)$$

is negligible.

If $\mathcal{S}_2$ does not abort in the step 6, then Lemma 4 and the equality

$$w_{\text{ckd},0} \cdot G = K_{\text{par}} + (n - n_0) r_0 \cdot G + n_0 R_1$$

imply that $\left(x'_{\text{par},1} - x_{\text{par},1}\right) \cdot G = n_0 (R_1 - r_1 \cdot G)$. Note that $x'_{\text{par},1} = \hat{x}_{\text{par},1} + m_1$. Although the adversary $\mathcal{A}$ knows one bit of n_0 (i.e. the least significant bit), $\mathcal{A}$ still hardly determines $r_1, \hat{x}_{\text{par},1}, m_1$, and R_1 satisfying the equality $\left(x'_{\text{par},1} - x_{\text{par},1}\right) \cdot G = n_0 (R_1 - r_1 \cdot G)$ with $r_1 \cdot G \neq R_1$. Thus, the probability that $\mathcal{S}_2$ outputs "fail" is negligible.

If the output is neither "fail" nor **abort**, the proof is completed by the following:

1. $\mathcal{S}_2$ has to check an additional condition $\left(x_{\text{par},0} + x'_{\text{par},1}\right) \cdot G = K_{\text{par}}$ by the definition of f_{ckd}. If this equation does not hold, then $x'_{\text{par},1} \neq x_{\text{par},1}$. Combining this fact and $\left(x'_{\text{par},1} - x_{\text{par},1}\right) \cdot G = n_0 (R_1 - r_1 \cdot G)$, we have $R_1 \neq r_1 \cdot G$, which implies that $\mathcal{S}_2$ should output "fail". Therefore, the probability of $\left(x_{\text{par},0} + x'_{\text{par},1}\right) \cdot G \neq K_{\text{par}}$ is negligible.

2. Although the outputs of $\mathcal{P}_0$ are different in these two worlds (i.e. the hybrid world gives $\mathrm{I}_L \| c_j$ and $\mathcal{S}_2$ gives I), we have $\mathrm{I}_L \| c_j = \mathrm{I}$ with overwhelming probability by Lemma 4.

Proof (Proof of Lemma 1).
For the circuit $F_{\mathrm{mkg},0}$, $\mathcal{A}$ only knows the output-wire labels $\mathbb{Y}_{F_{\mathrm{mkg},0}}^{v_{\mathrm{mkg},1}}$ corresponding to the bit-string

$$v_{\mathrm{mkg},1} = \mathrm{ser}_{256}(\mathrm{parse}_{256}(\mathrm{I}_L) + n_0 r_1 + n_1 r_0 \mod q) \| \mathrm{I}_R \| \mathrm{ser}_{256}(n_0 + n_1).$$

If $\mathcal{A}$ sets $v_{\mathrm{mkg},0}$ to be another value, $\mathcal{A}$ has to guess the output-wire labels $\mathbb{Y}_{F_{\mathrm{mkg},0}}^{v_{\mathrm{mkg},0}} \neq \mathbb{Y}_{F_{\mathrm{mkg},0}}^{v_{\mathrm{mkg},1}}$ to pass the validation test $h_{\mathrm{mkg},0} = \mathrm{H}\left(\mathbb{Y}_{F_{\mathrm{mkg},0}}^{v_{\mathrm{mkg},0}} \| Y_{F_{\mathrm{mkg},1}}\right) = h_{\mathrm{mkg},1}$. Hence, if

$$v_{\mathrm{mkg},0} \neq \mathrm{ser}_{256}(\mathrm{parse}_{256}(\mathrm{I}_L) + n_0 r_1 + n_1 r_0 \mod q) \| \mathrm{I}_R \| \mathrm{ser}_{256}(n_0 + n_1),$$

then $g_1(s_0) = 0$ with overwhelming probability.

Proof (Proof of Lemma 2). If $\mathcal{S}_4$ does not abort at Step 7 in Appendix B.1, then $g_1(s_0) = 1$. Therefore, we have Lemma 1:

$$v_{\mathrm{mkg},0} = \mathrm{ser}_{256}(\mathrm{parse}_{256}(\mathrm{I}_L) + n_0 r_1 + n_1 r_0 \mod q) \| \mathrm{I}_R \| \mathrm{ser}_{256}(n_0 + n_1).$$

The equality $w_{\mathrm{mkg},0} = w_{\mathrm{aux},0} + (n - n_0) r_0 \mod q$ and Lemma 1 imply that

$$q_0 = w_{\mathrm{aux},0} \cdot G - n_0 \cdot R_1 = \mathrm{parse}_{256}(\mathrm{I}_L) \cdot G + n_0 \cdot (r_1 \cdot G - R_1).$$

For any r_1, R_1 with $r_1 \cdot G \neq R_1$, $\mathcal{A}$ hardly finds $q_1 = q_0$ since $\mathcal{A}$ only knows one bit of n_0.

Proof (Proof of Lemma 3). First, if $\mathcal{S}_4$ does not output **abort**, then $g_1(s_0) = 1$. By Lemma 1 and the equality $w_{\mathrm{aux},0} = w_{\mathrm{mkg},0} - n_1 r_0 \mod q$, we have

$$v_{\mathrm{mkg},0} = \mathrm{ser}_{256}(\mathrm{parse}_{256}(\mathrm{I}_L) + n_0 r_1 + n_1 r_0 \mod q) \| \mathrm{I}_R \| \mathrm{ser}_{256}(n_0 + n_1)$$

and

$$w_{\mathrm{aux},0} = \mathrm{parse}_{256}(\mathrm{I}_L) + n_0 r_1 \mod q.$$

Second, the equality $R_1 = r_1 \cdot G$ holds since $\mathcal{S}$ does not output "fail". We conclude that the output of $\mathcal{P}_0$ generated by $\mathcal{S}_4$ is

$$\left(\frac{\mathrm{parse}_{256}(\mathrm{I}_L) + r}{2} \mod q, \mathrm{I}_R, \mathrm{parse}_{256}(\mathrm{I}_L) \cdot G \right).$$

Moreover, one has

$$\frac{\mathrm{parse}_{256}(\mathrm{I}_L) + r}{2} = \frac{w_{\mathrm{mkg},0}}{2} - (n - n_0) r_0 \mod q,$$

and

$$\mathrm{parse}_{256}(\mathrm{I}_L) \cdot G = w_{\mathrm{aux},0} \cdot G - n_0 \cdot R_1.$$

The proof is complete.