One-Time Delegation of Unlinkable Signing Rights and Its Application *

Takashi Nishide †

University of Tsukuba, Japan

Abstract. Delegation of signing rights can be useful to promote effective resource sharing and smooth cooperation among participants in distributed systems, and in many situations, we often need restricted delegation such as one-timeness and unlinkability rather than simple full delegation. Particularly, one-timesness cannot be achieved just by deploying cryptographic measures, and one needs to resort to some form of tamper-proofness or the assistance from external cloud servers for "key-disabling". In this work, we extend the latter such that a delegatee can sign a message without the delegator's involvement with the assumption that there exists at least one honest cloud server with secure erasure to achieve one-timeness. In this setting, if the delegator just shares their signing key between the delegatee and cloud servers, it may be problematic. It is because in the worst case, the delegator cannot know whether or not a signing key theft occurred because the signatures generated illegally are indistinguishable from the ones generated legally. To solve this, first we propose an efficient one-time delegation scheme of Okamoto-Schnorr signing. Further we combine the basic delegation scheme with anonymous credentials such that the delegator can detect the signing key theft even if one-time delegation is broken while also achieving unlinkability for both the delegator and cloud servers. Further we show its application to an e-cash scheme, which can prevent double-spending.

Keywords: Signature, Delegation, Anonymous Credential, E-Cash

1 Introduction

Delegation of rights to services and resources is old (e.g., [18, 58, 70]), but still relevant in distributed applications (e.g., [69, 63]), and it can often be realized via delegation of *signing* rights. Signing rights are unlinkable if a delegatee can sign a message unlinkably, where "unlinkably" means that when a delegator delegated their signing rights to multiple delegatees, no entities including the delegator can know which delegatee signed a message from its resulting signature as in group signatures [14]. This type of delegation is useful in applications needing privacy-preserving access control. Further we consider one-time delegation,

*This is the full version of the paper which appears in ProvSec 2020 [72], https://doi.org/10.1007/978-3-030-62576-4_6.

†nishide@risk.tsukuba.ac.jp

where "one-time" means that a delegator outsources their one-time signing capability. More specifically, after the delegator outsources their signing capability to the delegatee, the following properties hold without the delegator's involvement: (1) a delegatee can sign a message, (2) the verifier can verify the signature, and (3) the delegatee is prevented from signing more than once rather than detected after the fact. This type of one-timeness is often useful for electronic one-show tokens such as e-cash. In this work, we aim for such one-time delegation of unlinkable signing rights. In general, we cannot achieve one-timeness just by deploying cryptographic measures, and one viable approach is to resort to tamper-proofness such as smartcards (e.g., [47, 22, 48, 23],[24, Sect.6.3],[16],[25, Sect. 8]), one-time programs (OTPs) [59, 12, 83], or trusted execution environments (TEEs) (e.g., [1, 2, 69, 61]). However, such hardware-based solutions may sometimes be undesirable due to various side-channel attacks or cumbersome to use or deploy in practice (e.g., as pointed out in [31]). As a practical alternative to achieve one-timeness (except a feasibility result using quantum objects [6]), there is another line of research, e.g., [66, 51, 31, 68, 67, 26] (called *password-authenticated server-aided signatures* in [31]). In this line of research, roughly speaking, signing operations are made more secure by employing threshold signing and external clouds (or hardware devices), so that signatures can be generated only when the original signer authenticates to the clouds and cooperate with the clouds having shares of a signing key. Such a technique can make one-time delegation possible by letting a delegator secret-share the signing key between a delegatee and clouds, and the clouds erase the shares (i.e., key-disabling or rate-limiting the signing requests) after the delegatee accessed the clouds for signing operations. However, such an existing approach may be insufficient in the context of delegation (where a delegatee is also a potential adversary) due to the following: When one-timeness is broken in the worst case (i.e., a delegatee signed more than once illegally, given only the one-time signing right, e.g., via cloud breaches), it would be desirable for the delegator to be able to detect the fact from the generated and collected signatures. However, the existing works do not enable the delegator to detect it because all the signing rights use the same key and the signatures generated illegally are indistinguishable from the legal ones. This drawback may deter the delegators from relying on such delegation because no perfect protection against cloud breaches exists. Further, simply using different signing keys for each delegation may make it difficult to realize delegation of unlinkable signing rights.

Our Contributions. We propose efficient delegation of signing rights such that it is one-time, unlinkable for clouds as well as the delegator and *multi-run-detectable* even if one-timeness is broken, which means that the delegator can know the fact that one-time signing right was exerted more than once illegally from the generated and collected signatures[1)]. To this end, we extend the existing approach using one cloud (e.g., [51, 31, 26]) to the setting with multiple clouds and Okamoto-Schnorr blind signatures, and combine it with anonymous

[1)] As in e-cash schemes, we believe that it is a natural assumption that the signatures generated by the delegatees are eventually collected by the delegator.

credentials such that no master signing key needs to be shared among the delegatee and the clouds. In the setting of one cloud, the delegatee does not need to check the validity of the response from the cloud because it can be checked by verifying the resulting signature, but in our setting of multiple clouds, we need an efficient way of checking the validity of each response, and for that, we use a variant of the MACs based on secret sharing in [50]. Further we show a natural application of our one-time delegation to e-cash where double-spending is prevented and even if one-timeness is broken, double-spenders are identified. Our scheme does not require clouds to interact with each other, so will enjoy easy deployment as well.

Other Related Work. Group signatures (e.g., [14, 20, 60, 21]) and its generalized version (also called anonymous proxy signatures, e.g., [55, 56, 53, 54] [2)]) can also be viewed as delegation of unlinkable signing rights, but one-timeness has not been much explored. In the area of (delegatable) anonymous credentials (e.g., [41, 44, 76, 32, 35, 65, 39, 5, 11, 54, 78, 27, 17]), there exists a notion called k-times credential (e.g., [81, 71, 28, 82, 7, 49]), which allows a user to show a credential unlinkably up to k times, but the user is not prevented from showing the same credential more than k times (although identified after $k+1$ showings). Revocation of anonymous credentials is possible (e.g., [33, 30, 4, 3]). For example, in [33, 4, 3], revocation is realized with accumulators (e.g., [10]) that maintain non-revoked or revoked users in an anonymous way efficiently, and in [30], an attribute in the credential corresponds to an expiration date, and the credential issuer puts update values for each non-revoked user on a public bulletin board periodically such that only a non-revoked user can retrieve their corresponding value and update their credential for the new time period. Although this kind of revocation is useful in many situations, it will be insufficient for our purpose because we need revocation immediately after one showing of the credential.

The systems like [69, 63] also address delegation of credentials such as passwords and signing keys with TEEs [3)]. In, e.g., [69], a delegator just sends their credential to a TEE residing on the delegatee's computer (or TEE on the centrally brokered system) such that the delegatee can later use the delegated credential inside the TEE with appropriate authentication. Compared with [69, 63], our approach (1) avoids putting the (master) signing key of a delegator directly in clouds, and (2) tries to reduce the reliance on TEEs by using distributed clouds such that the security is guaranteed even if part of clouds are corrupted.

We construct e-cash by applying our one-time delegation scheme. In the offline e-cash model [46], a bank does not need to be involved in the payment, but double-spending can only be detected without being prevented. In the online e-cash model [40, 41, 42], double-spending is prevented by the bank being online to be involved in the payment. Our e-cash scheme based on one-time delegation

[2)] In anonymous proxy signatures, anyone can act as a group manager by delegating its signing rights to others who can then unlinkably sign, and in addition, received rights can be re-delegated.

[3)] One-timeness is not a main theme in [69, 63], but it will be possible if the delegator specifies a delegation policy enforcing one-timeness for TEEs.

can also prevent double-spending without the online bank in the payment, but with the increased communication overhead due to payers' access to clouds [4], which we believe can be alleviated, e.g., by the emerging 5G technology. In the e-cash scheme of [23, 22, 24], a tamper-proof device such as smartcards (sometimes called *observer* [43]) is issued and delivered to a user by the bank. Roughly speaking, one-timeness is realized by the fact that spending e-cash needs the assistance of the device (i.e., part of computation needs to be done by the device holding partial secret values, and the device is supposed to refuse to reuse the same e-cash). Although the scheme in [22, 24] is efficient, as pointed out in e.g., [9], its core building block, blind signature, does not have a proof of security, and the exculpability property is not achieved when applied to e-cash.

2 Preliminaries

Notation. We use $\lambda \in \mathbb{N}$ as a security parameter. We assume a random oracle (RO) [13] which can be viewed as an idealized hash function, and denote it by $H\colon \{0,1\}^* \to \{0,1\}^{2\lambda}$ (actually the range varies according to the context). We denote string concatenation by $\|$.

Bilinear Groups. Bilinear groups consist of three cyclic groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ of prime order p, and have a bilinear pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the properties: (1) $\forall g \in \mathbb{G}_1,\ \widetilde{g} \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p,\ e(g^a,\ \widetilde{g}^b) = e(g,\ \widetilde{g})^{ab}$, (2) $\forall g \neq 1_{\mathbb{G}_1}$ and $\widetilde{g} \neq 1_{\mathbb{G}_2},\ e(g,\ \widetilde{g}) \neq 1_{\mathbb{G}_T}$, (3) the pairing e can be computed efficiently. In this work, we use type-3 pairings where DDH holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$ [57] [5].

Okamoto-Schnorr (OS) ***Signature.*** The OS signature scheme [74] is obtained by applying the Fiat-Shamir transform [52] to the OS proof of knowledge, and the OS scheme enjoys witness indistinguishability[7]. The construction is given in Fig. 1. We also show the OS blind signatures [75, 74, 36] used in Sect. 3. In the OS blind signature scheme, OS.KeyGen and OS.Vrfy remain the same, but OS.Sign is replaced with OS.SigIssue protocol between a signer $\mathcal{S}$ and user $\mathcal{U}$ (Fig. 1).

Proof of Knowledge for Pedersen Commitment. We show a proof of knowledge (PK) of the discrete logarithm representation [45] in a Pedersen commitment [77] in Fig. 2, which is a Σ protocol and can be viewed as a generalization of the Schnorr proof of knowledge. We assume that the protocol in Fig. 2 is made non-interactive in the RO model by applying the Fiat-Shamir transform [52], and use the Camenisch-Stadler [34] notation such as $\mathsf{PK}\{(x_1, \ldots, x_k)\colon h = g_1^{x_1} \cdots g_k^{x_k}\}$ to denote this non-interactive zero-knowledge proof of knowledge of

[4] In the context of online e-cash, our approach can be viewed as the bank's outsourcing double-spending checks securely to the clouds.

[5] Type-3 pairings are considered to be the most efficient [57, 78].

[6] Here we can see that $\mathcal{U}$ randomizes $g^{r_1}h^{r_2}$ and $c'(= H(s \cdot g^{\alpha_1}h^{\alpha_2}y^{\beta} \parallel m))$. We note that the hash function is computed by $\mathcal{U}$ instead of $\mathcal{S}$ in OS blind signatures.

[7] We need this property in the proof of Theorem 2, and this is why we need OS instead of plain "Schnorr".

- $\mathsf{OS.KeyGen}(1^\lambda) \to (sk, pk)$: Choose a prime p, cyclic group $\mathbb{G}$ of order p, and generators $g, h \in \mathbb{G}$. The secret (signing) key sk is $(x_1, x_2) \in \mathbb{Z}_p^2$, and public key is $pk = (g,\ h,\ p,\ y)$ where $y = g^{x_1} h^{x_2}$.
- $\mathsf{OS.Sign}(sk,\ m \in \{0,1\}^*) \to (c, z_1, z_2)$: Choose random $r_1, r_2 \in \mathbb{Z}_p$, and compute

$$s \leftarrow g^{r_1} h^{r_2},\ c \leftarrow H(s \parallel m),\ z_i \leftarrow r_i - c \cdot x_i \bmod p \text{ for } i = 1, 2$$

  where $H\colon \{0,1\}^* \to \mathbb{Z}_p$ is modeled as an RO. The signature is $(c,\ z_1,\ z_2)$.
- $\mathsf{OS.Vrfy}(pk, m, (c,\ z_1, z_2))$: Check whether $c \stackrel{?}{=} H(g^{z_1} h^{z_2} y^c \parallel m)$, and if it holds, the output is 1, and otherwise 0.
- $\mathsf{OS.SigIssue}(\mathcal{S}(x_1, x_2),\ \mathcal{U}(pk, m))$: // for blind signatures
  1. $\mathcal{S}$ chooses random $r_1, r_2 \in \mathbb{Z}_p$, and sends $s \leftarrow g^{r_1} h^{r_2}$ to $\mathcal{U}$.
  2. $\mathcal{U}$ computes $c' \leftarrow H(s \cdot g^{\alpha_1} h^{\alpha_2} y^{\beta} \parallel m)$ with random $\alpha_1, \alpha_2, \beta \in \mathbb{Z}_p$, and sends $c \leftarrow c' - \beta \bmod p$ to $\mathcal{S}$ [6].
  3. $\mathcal{S}$ sends $z_i \leftarrow r_i - c \cdot x_i \bmod p$ (for $i = 1, 2$) to $\mathcal{U}$.
  4. If $s = g^{z_1} h^{z_2} y^c$ does not hold, $\mathcal{U}$ aborts, and otherwise $\mathcal{U}$ computes $z_i' \leftarrow z_i + \alpha_i \bmod p$ for $i = 1, 2$, and the signature on m is $(c',\ z_1',\ z_2')$.

Fig. 1. Okamoto-Schnorr (Blind) Signature

$(x_1, \ldots, x_k)$. [8]. Similarly we use the notation $\mathsf{SPK}\{(x_1, \ldots, x_k)\colon h = g_1^{x_1} \cdots g_k^{x_k}\}(m)$ to denote the signature on m (i.e., signature based on a PK like OS signatures).

It is well known that, for this PK, there exist a knowledge extractor $\mathcal{E}_\Sigma$ able to extract witnesses and zero-knowledge simulator $\mathcal{S}_\Sigma$ able to generate indistinguishable views by controlling the RO, which are used in security proofs.

Authenticated Secret Sharing. We explain the MACs we use in the context of secret sharing (see Sect. 3.1). This was proposed in [50], and it enables us to efficiently check whether a reconstructed secret is correct. In Fig. 3, we show the case where parties P_1, P_2 additively secret-share two secrets $r,\ x$, and reconstruct a linear combination $z = c'r + cx \bmod p$ of r and x (where c', c are known to P_1, P_2). In Sect. 3.1, we use a variant of this where first the delegator distributes the shares of secrets, MAC key, and MAC tags, and later the delegatee can check whether the shares sent by clouds are correct in reconstructing a secret.

Anonymous Credential. An anonymous credential scheme (e.g., [32, 9, 78, 80]) consists of the following algorithms:

- $\mathsf{AC.Setup}(1^\lambda) \to \mathsf{params}$: Generate public parameters params.
- $\mathsf{AC.IKeyGen}(\mathsf{params},\ w) \to (\mathsf{pk}_\mathcal{I},\ \mathsf{sk}_\mathcal{I})$: Generate a public key $\mathsf{pk}_\mathcal{I}$ and secret key $\mathsf{sk}_\mathcal{I}$ of the credential issuer $\mathcal{I}$ where w is the number of attributes. We implicitly assume that $\mathsf{pk}_\mathcal{I}$ includes params, and $\mathsf{sk}_\mathcal{I}$ includes $\mathsf{pk}_\mathcal{I}$.

[8] In the Camenisch-Stadler notation [34], Greek letters are used to represent secret witnesses (e.g., x_i) known only to $\mathcal{P}$, but here we deviate from that convention.

[9] The computation of a Pedersen commitment with multiple bases can be done efficiently by using simultaneous multiple exponentiation [62, Sect. 14.6].

- **Inputs:** A prover $\mathcal{P}$ has commitment $h = \prod_{i=1}^{k} g_i^{x_i}$ and gives the zero-knowledge proof of knowledge of $\{x_i \in \mathbb{Z}_p\}_{1 \leq i \leq k}$ to a verifier $\mathcal{V}$.
- **Auxiliary inputs:** The commitment h and generators $g_1, \ldots, g_k$ are public information and known to $\mathcal{V}$.
- **The protocol:**
  1. $\mathcal{P}$ chooses random $r_i \in \mathbb{Z}_p$ and sends $R \leftarrow \prod_{i=1}^{k} g_i^{r_i}$ to $\mathcal{V}$ [9].
  2. $\mathcal{V}$ sends random $c \in \mathbb{Z}_p$ to $\mathcal{P}$.
  3. $\mathcal{P}$ computes and sends $z_i \leftarrow r_i + c \cdot x_i \bmod p$ to $\mathcal{V}$.
  4. If $Rh^c = \prod_{i=1}^{k} g_i^{z_i}$ holds, $\mathcal{V}$ accepts the proof, and otherwise rejects.

Fig. 2. Proof of Knowledge for Pedersen Commitment [45]

- **Inputs:** The secrets r, x, MAC key α, and MAC tags αr, αx are additively secret-shared as

$$r = r_1 + r_2 \bmod p, \quad x = x_1 + x_2 \bmod p, \quad \alpha = \alpha_1 + \alpha_2 \bmod p,$$
$$\alpha \cdot r = m_1^{(r)} + m_2^{(r)} \bmod p, \quad \alpha \cdot x = m_1^{(x)} + m_2^{(x)} \bmod p$$

  where p is a prime and the shares with subscript i are held by P_i.
- **Output:** P_1 and P_2 reconstruct $z = c'r + cx \bmod p$ if the MAC verification is successful, and otherwise abort.
- **The protocol:**
  1. Each P_i publishes its share $z_i = c'r_i + cx_i \bmod p$ of z (we note that malicious P_i may publish incorrect z_i).
  2. Each P_i computes a candidate value $z' = z_1 + z_2 \bmod p$ of z.
  3. Each P_i computes $v_i = \alpha_i z' - (c' m_i^{(r)} + c m_i^{(x)}) \bmod p$, and publishes the commitment of v_i, $\mathsf{Com}(v_i)$.
  4. Each P_i publishes the opening of $\mathsf{Com}(v_i)$, and checks whether $v_1 + v_2 \stackrel{?}{=} 0 \bmod p$. If $v_1 + v_2 = 0 \bmod p$, each P_i accepts z' as a correctly reconstructed value, and otherwise aborts.

Fig. 3. Reconstruction of Secret in Authenticated Secret Sharing [50]

- $\mathsf{AC.CredIssue}(\mathcal{I}(\mathsf{sk}_{\mathcal{I}}), \mathcal{U}(\mathsf{att}_{\mathcal{U}}))$ is an *issuance protocol* between $\mathcal{I}$ and $\mathcal{U}$. At the end of the protocol, $\mathcal{U}$ obtains the credential $\mathsf{cr}_{\mathcal{U}}$ on attributes $\mathsf{att}_{\mathcal{U}}$.
- $\mathsf{AC.CredShow}(\mathcal{U}(\mathsf{att}_{\mathcal{U}}, \mathsf{cr}_{\mathcal{U}}), \mathcal{V}(\mathsf{pk}_{\mathcal{I}}))$ is a *show protocol* between $\mathcal{U}$ and a verifier $\mathcal{V}$. At the end of the protocol, if the $\mathsf{cr}_{\mathcal{U}}$ is a valid credential on $\mathsf{att}_{\mathcal{U}}$ issued by $\mathcal{I}$, $\mathcal{V}$ accepts the fact that $\mathcal{U}$ possesses the valid $\mathsf{cr}_{\mathcal{U}}$. If necessary, $\mathcal{U}$ can also disclose part of $\mathsf{att}_{\mathcal{U}}$ to $\mathcal{V}$ in this protocol.

Ideal Functionality $\mathcal{F}_{\text{CA}}$. We assume a public key infrastructure where the delegator and clouds register their public keys, modeled by $\mathcal{F}_{\text{CA}}$ [38]. The formal definition of $\mathcal{F}_{\text{CA}}$ is given in Appendix A.
Ideal Functionality $\mathcal{F}_{\text{AUTH}}$. We assume parties communicate via authenticated (but public) channels modeled by $\mathcal{F}_{\text{AUTH}}$ [37]. The formal definition of the simplified version of $\mathcal{F}_{\text{AUTH}}$ for our use is given in Appendix A.

3 One-Time Delegation of Okamoto-Schnorr Signing

Basic Idea. We consider one-time delegation of Okamoto-Schnorr (OS) signing. Here the OS signing key itself is shared among the delegatee and clouds, so it is not multi-run-detectable, but this serves as an important building block for one-time multi-run-detectable delegation scheme in Sect. 4. Conceptually one-time delegation can be viewed as if a delegator hands a delegatee a signing program that can be run only once with a message to be signed. Making abstraction of how such a signing program is implemented for now, we call it an OS signing one-time program (OTP), and running the OTP actually involves interaction between the delegatee and clouds, but does not need interaction with the delegator (i.e., the delegator can be offline when the OTP is run). In this context, we call a delegator an OTP generator G_{otp}, and a delegatee an OTP executor E_{otp}. We consider how to construct the OS signing OTP (sOTP) from the algorithm $\mathsf{OS.Sign}(sk, m)$ (Fig. 1). To make the computation performed in the OTP as small as possible, we avoid simply embedding the whole computation of $\mathsf{OS.Sign}(sk, m)$ including the hash function into the OTP, and embed only the computation part $z_i \leftarrow r_i - cx_i \bmod p$ into the OTP, thus leading to much better efficiency. Here an input variable of an OTP specified by E_{otp} is denoted by, e.g., $\bar{m}$, and a hardcoded secret variable is denoted as is. If we let $[f]_{\text{otp}}$ denote the OTP of f, the OTP of $\mathsf{OS.Sign}(sk, m)$ can be like a tuple

$$[\mathsf{OS.Sign}(sk, \bar{m})]_{\text{otp}} = \langle g, h, y, p, s, \{[r_i - \bar{c} \cdot x_i \bmod p]_{\text{otp}}\}_{i=1}^{2} \rangle$$
$$\text{where } y = g^{x_1} h^{x_2},\ s = g^{r_1} h^{r_2},\ \bar{c} \leftarrow H(s \parallel \bar{m}).$$

I.e., E_{otp} specifies the variable $\bar{m}$ and obtains the output by:

1. compute $c \leftarrow H(s \parallel m)$,
2. run $[r_i - \bar{c} \cdot x_i \bmod p]_{\text{otp}}$ by substituting c for $\bar{c}$ for $i = 1, 2$.

We can notice that actually this can be viewed as an OS *blind* signing operation because the hash calculation is not done by the signer (OTP). Therefore, the

Algorithm 1 $[\mathsf{OS.BSign}(x_1, x_2, r_1, r_2, \bar{m})]_{\mathrm{otp}}$

Require: $m \in \{0,1\}^*$, $\langle g, h, y = g^{x_1}h^{x_2}, p, s = g^{r_1}h^{r_2}, \{[r_i - \bar{c} \cdot x_i \bmod p]_{\mathrm{otp}}\}_{i=1}^2\rangle$
Ensure: signature (c', z_1', z_2') on m specified by $\mathsf{E}_{\mathrm{otp}}$ under the public key y
1: $\mathsf{E}_{\mathrm{otp}}$ chooses random $\alpha_1, \alpha_2, \beta \in \mathbb{Z}_p$, and computes $c' \leftarrow H(s \cdot g^{\alpha_1}h^{\alpha_2}y^{\beta} \parallel m)$
2: $\mathsf{E}_{\mathrm{otp}}$ computes $c \leftarrow c' - \beta \bmod p$
3: $\mathsf{E}_{\mathrm{otp}}$ runs $z_i \leftarrow [r_i - \bar{c} \cdot x_i \bmod p]_{\mathrm{otp}}$ with clouds by substituting c for $\bar{c}$ for $i = 1, 2$
4: $\mathsf{E}_{\mathrm{otp}}$ computes $z_i' \leftarrow z_i + \alpha_i \bmod p$ for $i = 1, 2$, and obtains (c', z_1', z_2')

randomization of $s = g^{r_1}h^{r_2}$ and the hash value c is also possible as $\mathcal{U}$ does in $\mathsf{OS.SigIssue}$ (Fig. 1). As a result, the basic building block for an OS sOTP $[\mathsf{OS.BSign}(x_1, x_2, r_1, r_2, \bar{m})]_{\mathrm{otp}}$ is Algorithm 1.

3.1 Instantiating OS Signing OTP with Clouds

Now we focus on $[r_i - \bar{c} \cdot x_i \bmod p]_{\mathrm{otp}}$ in the OS signing OTP (sOTP), and consider how to instantiate this OTP with clouds. The basic idea is simple and efficient. The main part $[r_i - \bar{c} \cdot x_i \bmod p]_{\mathrm{otp}}$ is just computing $r_i - cx_i \bmod p$ with input c specified later by $\mathsf{E}_{\mathrm{otp}}$, so $\mathsf{G}_{\mathrm{otp}}$ can secret-share $\{r_i, x_i\}_{i=1}^2$ with the delegatee $\mathsf{E}_{\mathrm{otp}}$ and clouds, and let them compute $r_i - cx_i \bmod p$ distributively [12] with the shares and let the clouds erase the shares later. Although corrupted clouds will not erase the shares correctly, one-timesness can be achieved if there exists at least one honest cloud with secure erasure. We construct our protocol such that $\mathsf{E}_{\mathrm{otp}}$ needs to authenticate to clouds by using a password in running an sOTP. First we define the ideal functionality $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$ (Fig. 4) corresponding to our real protocol, which allows $\mathsf{E}_{\mathrm{otp}}$ to specify c and compute $r_i - cx_i \bmod p$ for $i = 1, 2$ only once, and for $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$, we assume the following:

- The existence of clouds $\{\mathsf{S}_j\}_{j=1}^m$ is public information.
- An (ideal) adversary (i.e., simulator) $\mathcal{S}$ is static (i.e., non-adaptive).
- *sid* is a common globally unique session ID both $\mathsf{G}_{\mathrm{otp}}$ and $\mathsf{E}_{\mathrm{otp}}$ have previously agreed upon (i.e., each OTP has its own unique *sid*) as in [31].
- qid_j is a common globally unique query ID both S_j and $\mathsf{E}_{\mathrm{otp}}$ have previously agreed upon each time $\mathsf{E}_{\mathrm{otp}}$ tries to retrieve a share from S_j as in [31].

Then we give the full description of OTP generation and OTP execution in Fig. 5, 6 respectively. For password authentication, we extend the method in [31] such that $\mathsf{E}_{\mathrm{otp}}$ can specify the password for multiple clouds. To let the simulation-based security proof go through, here we need to use additive secret

[10] When we say that a functionality "looks up a record", we mean that if the record is not found, the functionality just ignores the input.

[11] In this case, $\mathcal{S}$ can only prevent $\mathsf{E}_{\mathrm{otp}}$ from obtaining correct shares by responding with incorrect shares.

[12] No interaction among clouds is needed here thanks to the simplicity of OS signing.

1. `OTP Generation Request`. On input (OTP-GENREQ, *sid*, G_{otp}, *pwd*) from E_{otp}:
   - Record (otpgen-req, *sid*, $\mathsf{E}_{\text{otp}}, \mathsf{G}_{\text{otp}}$, *pwd*).
   - Send (OTP-GENREQ, *sid*, $\mathsf{E}_{\text{otp}}, \mathsf{G}_{\text{otp}}$) to $\mathcal{S}$. Upon receiving ok from $\mathcal{S}$, output (OTP-GENREQ, *sid*, E_{otp}) to G_{otp}.
2. `OTP Generation`. On input (OTP-GEN, *sid*, E_{otp}, g, h, p, x_1, x_2) from G_{otp}:
   - Look up a record (otp-genreq, *sid*, $\mathsf{E}_{\text{otp}}, \mathsf{G}_{\text{otp}}$, *pwd*)[10].
   - Choose random $r_1, r_2 \in \mathbb{Z}_p$ and record (otp, *sid*, $\mathsf{E}_{\text{otp}}, \mathsf{G}_{\text{otp}}, pwd, p, x_1, x_2, r_1, r_2$, *run-flg*), where *run-flg* = not-run.
   - Delete (otp-genreq, *sid*, $\mathsf{E}_{\text{otp}}, \mathsf{G}_{\text{otp}}$, *pwd*).
   - If E_{otp} is corrupted, send (OTP-GEN-LEAK1, *sid*, $\mathsf{G}_{\text{otp}}, \mathsf{E}_{\text{otp}}, g, h, p, g^{x_1}h^{x_2}$, $g^{r_1}h^{r_2}$) to $\mathcal{S}$. If at least one S_j is corrupted, send (OTP-GEN-LEAK2, *sid*, G_{otp}, p) to $\mathcal{S}$.
   - Send (OTP-GEN, *sid*, G_{otp}, E_{otp}) to $\mathcal{S}$. Upon receiving ok from $\mathcal{S}$, output (OTP, *sid*, G_{otp}) to E_{otp}.
   - For each S_j, send (OTP-GEN, *sid*, G_{otp}, S_j) to $\mathcal{S}$. Upon receiving ok from $\mathcal{S}$, output (OTP-SHARE, *sid*, G_{otp}) to each S_j.
3. `Running OTP`. On input (OTP-RUN, *sid*, $\{qid_j\}_{j=1}^m$, *pwd'*, c') from E_{otp}:
   - Look up records (otp, *sid*, E_{otp}, $*$, *pwd*, p, x_1, x_2, r_1, r_2, *run-flg*).
   - If the record (otp-running, *sid*, E_{otp}) already exists, wait until it is deleted.
   - Record (otp-running, *sid*, E_{otp}).
   - Set *ath-flg* $\leftarrow$ pwdok, $rslt_i \leftarrow r_i - c'x_i \bmod p$ for $i = 1, 2$ if *pwd* = *pwd'*, and otherwise *ath-flg* $\leftarrow$ pwdwrong, $rslt_i \leftarrow \perp$.
   - Set *otp-finish* $\leftarrow$ true. // Initialization
   - // OTP was already run
     If *run-flg* = run, output (OTP-RUN, *sid*, $\{qid_j\}_{j=1}^m, \perp$) to E_{otp}
   - Else, if E_{otp} is corrupted, output (OTP-RUN, *sid*, $\{qid_j\}_{j=1}^m$, $\{rslt_i\}_{i=1}^2$) to E_{otp} // OTP can be run
   - Else, if no S_j is corrupted, // E_{otp} is honest
     - send (OTP-RUN, *sid*, $\{qid_j\}_{j=1}^m$, E_{otp}) to $\mathcal{S}$. Upon receiving ok from $\mathcal{S}$, output (OTP-RUN, *sid*, $\{qid_j\}_{j=1}^m$, $\{rslt_i\}_{i=1}^2$) to E_{otp}
   - Else, send (OTP-RUN-LEAK, *sid*, $\{qid_j\}_{j=1}^m$, E_{otp}, c', *ath-flg*) to $\mathcal{S}$[11], set *otp-finish* $\leftarrow$ false, and record (otp-running-wait-share, *sid*, $\{qid_j\}_{j=1}^m$, E_{otp}, c', *ath-flg*). // E_{otp} is honest and at least one of $\{\mathsf{S}_j\}_{j=1}^m$ is corrupted
   - If *otp-finish* = true and *ath-flg* = pwdok, update the record (otp, *sid*, E_{otp}, $*$, *pwd*, p, x_1, x_2, r_1, r_2, *run-flg*) such that *run-flg* $\leftarrow$ run.
   - If *otp-finish* = true, delete (otp-running, *sid*, E_{otp}).
4. `Corrupted Server Proceeds`. On input (OTP-SH-PROC, *sid*, $\{qid_j\}_{j=1}^m$, E_{otp}, *sh-flg*) from $\mathcal{S}$ where *sh-flg* $\in \{$correct-sh, $\perp\}$:
   - Look up records (otp, *sid*, E_{otp}, $*$, *pwd*, p, x_1, x_2, r_1, r_2, *run-flg*), (otp-running, *sid*, E_{otp}), (otp-running-wait-share, *sid*, $\{qid_j\}_{j=1}^m, \mathsf{E}_{\text{otp}}, c'$, *ath-flg*).
   - Set $rslt_i \leftarrow \perp$ for $i = 1, 2$. // Initialization
   - // At least one S_j is honest, so if the password is wrong, E_{otp} cannot
     // obtain the correct result. Note that $\mathcal{S}$ can ignore *ath-flg* if it wants.
     If *ath-flg* = pwdok and *sh-flg* = correct-sh, $rslt_i \leftarrow r_i - c'x_i \bmod p$ for $i = 1, 2$.
   - Send (OTP-RUN, *sid*, $\{qid_j\}_{j=1}^m$, E_{otp}) to $\mathcal{S}$. Upon receiving ok from $\mathcal{S}$, output (OTP-RUN, *sid*, $\{qid_j\}_{j=1}^m$, $\{rslt_i\}_{i=1}^2$) to E_{otp}.
   - If *ath-flg* = pwdok, update the record (otp, *sid*, E_{otp}, $*$, *pwd*, p, x_1, x_2, r_1, r_2, *run-flg*) such that *run-flg* $\leftarrow$ run.
   - Delete (otp-running-wait-share, *sid*, $\{qid_j\}_{j=1}^m, \mathsf{E}_{\text{otp}}, c'$, *ath-flg*), (otp-running, *sid*, E_{otp}).

Fig. 4. $\mathcal{F}_{\text{otp}}^{\text{BOS}}$ for OS Signing OTP ("//" means comments and "$*$" is a wildcard)

sharing rather than Shamir's secret sharing [13] as in [67, 66, 31, 26, 68]. We explain the overview of how to adapt and embed the MAC scheme in [50] into our construction such that $\mathsf{E}_{\rm otp}$ can check the correctness of the shares sent by the clouds. Suppose the cloud S_j holds the shares $\{r'_{i,j},\ x'_{i,j}\}_{i=1}^2$, and sends $\{r'_{i,j} - c'x'_{i,j} \bmod p\}_{i=1}^2$ to $\mathsf{E}_{\rm otp}$ in response to c' specified by $\mathsf{E}_{\rm otp}$. First we outline the process for $\mathsf{G}_{\rm otp}$.

1. $\mathsf{G}_{\rm otp}$ chooses a random MAC key $\alpha^{(j)} \in \mathbb{Z}_p$ and prepares its additive sharing $\alpha^{(j)} = \alpha^{(j)}_{\mathsf{E}_{\rm otp}} + \alpha^{(j)}_{\mathsf{S}_j} \bmod p$.
2. $\mathsf{G}_{\rm otp}$ prepares the additive sharings of MAC tags $\{\alpha^{(j)}r'_{i,j},\ \alpha^{(j)}x'_{i,j}\}_{i=1}^2$ as
$$\{\alpha^{(j)}r'_{i,j} = m^{(j,r'_i)}_{\mathsf{E}_{\rm otp}} + m^{(j,r'_i)}_{\mathsf{S}_j} \bmod p,\ \ \alpha^{(j)}x'_{i,j} = m^{(j,x'_i)}_{\mathsf{E}_{\rm otp}} + m^{(j,x'_i)}_{\mathsf{S}_j} \bmod p\}_{i=1}^2.$$
3. $\mathsf{G}_{\rm otp}$ sends $\langle \alpha^{(j)}_{\mathsf{S}_j}, \{r'_{i,j},\ x'_{i,j},\ m^{(j,r'_i)}_{\mathsf{S}_j},\ m^{(j,x'_i)}_{\mathsf{S}_j}\}_{i=1}^2\rangle$ to S_j, and $\langle \alpha^{(j)}_{\mathsf{E}_{\rm otp}}, \{m^{(j,r'_i)}_{\mathsf{E}_{\rm otp}},\ m^{(j,x'_i)}_{\mathsf{E}_{\rm otp}}\}_{i=1}^2\rangle$ to $\mathsf{E}_{\rm otp}$.

Next we outline the protocol between S_j and $\mathsf{E}_{\rm otp}$ running an sOTP.

1. Given c' by $\mathsf{E}_{\rm otp}$ [15], for $i = 1, 2$, S_j computes $z'_{i,j} = r'_{i,j} - c'x'_{i,j} \bmod p$ and similarly $v_{i,\mathsf{S}_j} = \alpha^{(j)}_{\mathsf{S}_j} z'_{i,j} - (m^{(j,r'_i)}_{\mathsf{S}_j} - c' \cdot m^{(j,x'_i)}_{\mathsf{S}_j}) \bmod p$, and sends $\{z'_{i,j},\ v_{i,\mathsf{S}_j}\}_{i=1}^2$ to $\mathsf{E}_{\rm otp}$.
2. For $i = 1, 2$, $\mathsf{E}_{\rm otp}$ computes $v^{(j)}_{i,\mathsf{E}_{\rm otp}} = \alpha^{(j)}_{\mathsf{E}_{\rm otp}} z'_{i,j} - (m^{(j,r'_i)}_{\mathsf{E}_{\rm otp}} - c' \cdot m^{(j,x'_i)}_{\mathsf{E}_{\rm otp}})$, and checks whether $v^{(j)}_{i,\mathsf{E}_{\rm otp}} + v_{i,\mathsf{S}_j} \stackrel{?}{=} 0 \bmod p$. If 0, $\mathsf{E}_{\rm otp}$ accepts $\{z'_{i,j}\}_{i=1}^2$, and otherwise aborts.

For malicious S_j to cheat here such that $\mathsf{E}_{\rm otp}$ accepts $z'_{i,j} + \Delta = r'_{i,j} - c'x'_{i,j} + \Delta$ (where $\Delta \neq 0$), S_j needs to compute the following v^*_{i,S_j} by guessing $\alpha^{(j)}_{\mathsf{E}_{\rm otp}}$ unknown to S_j, but the probability that the guess is correct is only $1/p$.

$$\begin{aligned} v^*_{i,\mathsf{S}_j} &= -v^{(j)}_{i,\mathsf{E}_{\rm otp}} = -(\alpha^{(j)}_{\mathsf{E}_{\rm otp}}(r'_{i,j} - c'x'_{i,j} + \Delta) - (m^{(j,r'_i)}_{\mathsf{E}_{\rm otp}} - c' \cdot m^{(j,x'_i)}_{\mathsf{E}_{\rm otp}})) \\ &= -\alpha^{(j)}_{\mathsf{E}_{\rm otp}}\Delta + \alpha^{(j)}_{\mathsf{S}_j}(r'_{i,j} - c'x'_{i,j}) - (m^{(j,r'_i)}_{\mathsf{S}_j} - c'm^{(j,x'_i)}_{\mathsf{S}_j}) \end{aligned}$$

Although discrete-log based commitments can also be used to check the response $\{z'_{i,j}\}_{i=1}^2$ from S_j here, the above MACs are much more efficient in that $\mathsf{E}_{\rm otp}$ needs only modular addition/multiplication rather than exponentiations.

---

[13] The adversarial $\mathsf{E}_{\rm otp}$ can send different c's to $\{\mathsf{S}_j\}_{j=1}^m$ in Step 1 of Fig. 6 maliciously because we do not require $\{\mathsf{S}_j\}_{j=1}^m$ to coordinate with each other to reject such requests (because we tried to keep the protocol as simple as possible). This makes the use of Shamir's secret sharing non-trivial in terms of the simulation in the proof.

[14] We assume that *pwd* exists only in $\mathsf{E}_{\rm otp}$'s brain (not stored in $\mathsf{E}_{\rm otp}$'s computer).

[15] We note that the value c' does not need to be hidden from $\{\mathsf{S}_j\}_{j=1}^m$ because it is randomized in OS blind signing.

We assume the following:
- $\mathsf{G}_{\mathrm{otp}}$ has a signing key $(x_1, x_2) \in \mathbb{Z}_p^2$ and public key $y = g^{x_1}h^{x_2}$ with a group $\mathbb{G}$ of prime order p and generator $g, h \in \mathbb{G}$.
- Each cloud is denoted by S_j where $1 \le j \le m$.
- $\mathsf{G}_{\mathrm{otp}}$ and each S_j register their public keys $pk_{\mathsf{G}_{\mathrm{otp}}}$, pk_{S_j} with $\mathcal{F}_{\mathrm{CA}}$ at the beginning of the protocol.
- $\mathsf{G}_{\mathrm{otp}}$ and $\mathsf{E}_{\mathrm{otp}}$ have already agreed upon generating an sOTP.
- The communication between parties is done via $\mathcal{F}_{\mathrm{AUTH}}$.

1. On input ($\mathsf{OTP\text{-}GENREQ}$, *sid*, $\mathsf{G}_{\mathrm{otp}}$, *pwd*), $\mathsf{E}_{\mathrm{otp}}$ chooses random $seed \in \{0,1\}^\lambda$, and generates $\{salt_j,\ h_j\}_{j=1}^m$ such that

$$salt_j \leftarrow H(seed \,\|\, \mathsf{S}_j), \quad h_j \leftarrow H(salt_j \,\|\, pwd).$$

   $\mathsf{E}_{\mathrm{otp}}$ obtains $pk_{\mathsf{G}_{\mathrm{otp}}}$, $\{pk_{\mathsf{S}_j}\}_{j=1}^m$ from $\mathcal{F}_{\mathrm{CA}}$, and sends $(sid, \mathsf{Enc}(pk_{\mathsf{G}_{\mathrm{otp}}}, pk_{\mathsf{E}_{\mathrm{otp}}}, \{\mathsf{Enc}(pk_{\mathsf{S}_j}, h_j)\}_{j=1}^m))$ to $\mathsf{G}_{\mathrm{otp}}$ where $\mathsf{Enc}(pk, m)$ denotes the ciphertext of m under the key pk.
2. $\mathsf{G}_{\mathrm{otp}}$ decrypts and receives $(sid, pk_{\mathsf{E}_{\mathrm{otp}}}, \{\mathsf{Enc}(pk_{\mathsf{S}_j}, h_j)\}_{j=1}^m)$ from $\mathsf{E}_{\mathrm{otp}}$ and outputs ($\mathsf{OTP\text{-}GENREQ}$, *sid*, $\mathsf{E}_{\mathrm{otp}}$).
   On input ($\mathsf{OTP\text{-}GEN}$, *sid*, $\mathsf{E}_{\mathrm{otp}}$, g, h, p, x_1, x_2), $\mathsf{G}_{\mathrm{otp}}$ chooses random $r_1, r_1 \in \mathbb{Z}_p$, generates $s = g^{r_1}h^{r_2}$, and splits $\{r_i,\ x_i\}_{i=1}^2$ into $r_i = r_i' + r_i'' \bmod p$, $x_i = x_i' + x_i'' \bmod p$ at random.
3. To (m, m)-secret-share $\{r_i',\ x_i'\}_{i=1}^2$ among $\{\mathsf{S}_j\}_{j=1}^m$, $\mathsf{G}_{\mathrm{otp}}$ generates the random shares $\{r_{i,j}',\ x_{i,j}'\}_{i=1,2}^{1\le j\le m}$ such that

$$r_i' = \sum_{j=1}^m r_{i,j}' \bmod p \ \text{ and } \ x_i' = \sum_{j=1}^m x_{i,j}' \bmod p \ \text{ for } i = 1, 2.$$

4. For $1 \le j \le m$, $\mathsf{G}_{\mathrm{otp}}$ chooses random MAC keys $\{\alpha^{(j)} \in \mathbb{Z}_p\}_{j=1}^m$, and prepares the following additive sharings at random: $\alpha^{(j)} = \alpha_{\mathsf{E}_{\mathrm{otp}}}^{(j)} + \alpha_{\mathsf{S}_j}^{(j)} \bmod p$,
   $\alpha^{(j)} r_{i,j}' = m_{\mathsf{E}_{\mathrm{otp}}}^{(j,r_i')} + m_{\mathsf{S}_j}^{(j,r_i')} \bmod p$, $\alpha^{(j)} x_{i,j}' = m_{\mathsf{E}_{\mathrm{otp}}}^{(j,x_i')} + m_{\mathsf{S}_j}^{(j,x_i')} \bmod p$ for $i = 1, 2$.
5. $\mathsf{G}_{\mathrm{otp}}$ obtains $\{pk_{\mathsf{S}_j}\}_{j=1}^m$ from $\mathcal{F}_{\mathrm{CA}}$, and sends to each S_j

$$(sid, \mathsf{Enc}(pk_{\mathsf{S}_j}, h_j), \mathsf{Enc}(pk_{\mathsf{S}_j}, \langle \alpha_{\mathsf{S}_j}^{(j)}, \{r_{i,j}', x_{i,j}', m_{\mathsf{S}_j}^{(j,r_i')}, m_{\mathsf{S}_j}^{(j,x_i')}\}_{i=1}^2 \rangle)).$$

6. An OS sOTP P is as follows:

$$P = \langle sid, g, h, \underbrace{y}_{g^{x_1}h^{x_2}}, p, \underbrace{s}_{g^{r_1}h^{r_2}}, \{r_i'', x_i''\}_{i=1}^2, \{\alpha_{\mathsf{E}_{\mathrm{otp}}}^{(j)}\}_{j=1}^m, \{m_{\mathsf{E}_{\mathrm{otp}}}^{(j,r_i')}, m_{\mathsf{E}_{\mathrm{otp}}}^{(j,x_i')}\}_{i=1,2}^{1\le j\le m} \rangle$$

   $\mathsf{G}_{\mathrm{otp}}$ sends $(sid, \mathsf{Enc}(pk_{\mathsf{E}_{\mathrm{otp}}}, P))$ to $\mathsf{E}_{\mathrm{otp}}$.
7. S_j receives $(sid, \mathsf{Enc}(pk_{\mathsf{S}_j}, h_j), \mathsf{Enc}(pk_{\mathsf{S}_j}, \langle \alpha_{\mathsf{S}_j}^{(j)}, \{r_{i,j}', x_{i,j}', m_{\mathsf{S}_j}^{(j,r_i')}, m_{\mathsf{S}_j}^{(j,x_i')}\}_{i=1}^2 \rangle))$ from $\mathsf{G}_{\mathrm{otp}}$, stores $\langle sid, h_j, \alpha_{\mathsf{S}_j}^{(j)}, \{r_{i,j}', x_{i,j}', m_{\mathsf{S}_j}^{(j,r_i')}, m_{\mathsf{S}_j}^{(j,x_i')}\}_{i=1}^2 \rangle$, and outputs ($\mathsf{OTP\text{-}SHARE}$, *sid*, $\mathsf{G}_{\mathrm{otp}}$).
8. $\mathsf{E}_{\mathrm{otp}}$ receives $(sid, \mathsf{Enc}(pk_{\mathsf{E}_{\mathrm{otp}}}, P))$ from $\mathsf{G}_{\mathrm{otp}}$, stores $(seed, P)$[14], and outputs (OTP, *sid*, $\mathsf{G}_{\mathrm{otp}}$).

Fig. 5. Protocol for Generating an OS Signing OTP

- In this protocol for running an OS sOTP, $\mathsf{E}_{\mathrm{otp}}$ interacts with each S_j.
- $\mathsf{E}_{\mathrm{otp}}$ has an OS sOTP P (Fig. 5) and is given input c' where c' is computed as $c \leftarrow H(s \cdot g^{\alpha_1} h^{\alpha_2} y^{\beta} \parallel m)$, $c' \leftarrow c - \beta \bmod p$ with s in P, random $\alpha_1, \alpha_2, \beta \in \mathbb{Z}_p$ [16], and message $m \in \{0,1\}^*$ to be signed.

1. On input ($\mathsf{OTP\text{-}RUN}$, *sid*, $\{qid_j\}_{j=1}^m$, *pwd*, c'), to retrieve a share from each S_j, $\mathsf{E}_{\mathrm{otp}}$ computes, for authentication,

$$ath_j \leftarrow H(sid \parallel qid_j \parallel \overbrace{H(\underbrace{H(seed \parallel \mathsf{S}_j)}_{salt_j} \parallel pwd)}^{h_j}),$$

   and sends (sid, qid_j, $\mathsf{Enc}(pk_{\mathsf{S}_j}, pk_{\mathsf{E}_{\mathrm{otp}}}, c', ath_j)$) to each S_j.
2. Each S_j decrypts and obtains (sid, qid_j, $pk_{\mathsf{E}_{\mathrm{otp}}}$, c', ath_j) from $\mathsf{E}_{\mathrm{otp}}$. If S_j does not have a data tuple corresponding to sid or it does not hold that $ath_j = H(sid \parallel qid_j \parallel h_j)$, S_j sets $rslt_i \leftarrow \perp$ for $i = 1, 2$. Otherwise S_j computes, for $i = 1, 2$,

$$z'_{i,j} \leftarrow r'_{i,j} - c' \cdot x'_{i,j} \bmod p, \quad v_{i,\mathsf{S}_j} \leftarrow \alpha^{(j)}_{\mathsf{S}_j} z'_{i,j} - (m^{(j,r'_i)}_{\mathsf{S}_j} - c' \cdot m^{(j,x'_i)}_{\mathsf{S}_j}) \bmod p,$$

   erases $\langle sid, h_j, \alpha^{(j)}_{\mathsf{S}_j}, \{r'_{i,j}, x'_{i,j}, m^{(j,r'_i)}_{\mathsf{S}_j}, m^{(j,x'_i)}_{\mathsf{S}_j}\}_{i=1}^2 \rangle$ [17]since it is no longer needed, and sets $rslt_i \leftarrow \langle z'_{i,j}, v_{i,\mathsf{S}_j} \rangle$. S_j sends (sid, qid_j, $\mathsf{Enc}(pk_{\mathsf{E}_{\mathrm{otp}}}, \{rslt_i\}_{i=1}^2)$) to $\mathsf{E}_{\mathrm{otp}}$.
3. If $\mathsf{E}_{\mathrm{otp}}$ decrypts and receives $\{rslt_i\}_{i=1}^2$ from S_j and $rslt_i \neq \perp$, $\mathsf{E}_{\mathrm{otp}}$ computes

$$v^{(j)}_{i,\mathsf{E}_{\mathrm{otp}}} = \alpha^{(j)}_{\mathsf{E}_{\mathrm{otp}}} z'_{i,j} - (m^{(j,r'_i)}_{\mathsf{E}_{\mathrm{otp}}} - c' \cdot m^{(j,x'_i)}_{\mathsf{E}_{\mathrm{otp}}}) \bmod p,$$

   and verifies $v^{(j)}_{i,\mathsf{E}_{\mathrm{otp}}} + v_{i,\mathsf{S}_j} \stackrel{?}{=} 0 \bmod p$ for $i = 1, 2$. If the verification fails or $rslt_i = \perp$, $\mathsf{E}_{\mathrm{otp}}$ outputs ($\mathsf{OTP\text{-}RUN}$, sid, $\{qid_j\}_{j=1}^m$, $\perp$).
   Otherwise $\mathsf{E}_{\mathrm{otp}}$ has $\{z'_{i,j}\}_{i=1,2}^{1 \le j \le m}$, computes, for $i = 1, 2$,

$$z'_i = r_i - c' x_i = (\sum_{j=1}^{m} z'_{i,j}) + (r''_i - c' x''_i) \bmod p,$$

   and outputs ($\mathsf{OTP\text{-}RUN}$, sid, $\{qid_j\}_{j=1}^m$, $\{z'_i\}_{i=1}^2$).

Fig. 6. Protocol for Running an OS Signing OTP

Theorem 1. *The protocol in Fig. 5, 6 securely realizes the functionality* $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$ *(Fig. 4) in the* $(\mathcal{F}_{\mathrm{CA}}, \mathcal{F}_{\mathrm{AUTH}})$*-hybrid and RO model, assuming that the static adversary corrupts* $\mathsf{E}_{\mathrm{otp}}$ *and at most* $(m-1)$ *clouds and the unforgeability of the MAC scheme based on authenticated secret sharing.*

We prove the above theorem in Appendix A by following the ideal/real simulation paradigm. I.e., an execution in the real model is compared with an execution in an ideal model where an incorruptible trusted party computes the functionality for the parties, and an entity called environment $\mathcal{Z}$ tries to distinguish between the two models by specifying the initial inputs for the involved parties and interacting with the execution. In the proof, basically we show that there exists a simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$ and the adversary $\mathcal{A}$ and can generate an indistinguishable view for $\mathcal{A}$ by using only leakage from $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$ without knowing $\{r_i, x_i\}_{i=1}^2$.

4 One-Time Multi-Run-Detectable Delegation Based on Anonymous Credentials

Now we construct a one-time multi-run-detectable delegation scheme of unlinkable signing rights with OS sOTPs and anonymous credentials (ACs) such that a delegator does not need to embed their master signing key directly into the OTP. First we construct a one-time AC (OAC) scheme, in which the credential issuer can issue a one-time unlinkable credential which can be shown to a verifier only once in the sense that the credential holder is prevented from showing the credential more than once based on the security of our OTPs rather than detected when multiple shows are performed. As an underlying AC scheme, we use the PS scheme [78], which consists of the following:

- randomizable blind signatures with the message space of multiple attributes,
- zero-knowledge proof of knowledge of a signature.

Key Idea to combine OS ***sOTP and*** PS ***scheme:*** In the PS scheme, the credential requestor $\mathcal{U}$ obtains a blind signature on the commitment to $\mathsf{att}_{\mathcal{U}}$ in the issuance protocol, and in the show protocol, gives a zero-knowledge proof of knowledge of $\mathsf{att}_{\mathcal{U}}$. In our OAC scheme, the issuer adds hidden extra random

[16] This randomizability makes the resulting signature unlinkable even for the clouds as well as the delegator because c' (visible to the clouds) is independent of the signature.

[17] We assume that while a thread running in S_j is accessing a data tuple $\langle sid, h_j, \alpha_{\mathsf{S}_j}^{(j)}, \{r'_{i,j}, x'_{i,j}, m_{\mathsf{S}_j}^{(j,r'_i)}, m_{\mathsf{S}_j}^{(j,x'_i)}\}_{i=1}^2\rangle$, the access to this tuple (with tuple ID *sid*) by other threads is prevented with appropriate mutual exclusion.

[18] Depending on applications, $\mathcal{U}$ can send part of $\mathsf{att}_{\mathcal{U}}$ in the clear to $\mathcal{I}$, and $\mathcal{I}$ will judge that $\mathcal{U}$ is qualified as the portion of $\mathsf{att}_{\mathcal{U}}$. In the underlying non-blind PS multi-message signature [78, Sect. 4.2], these clear $\mathsf{att}_{\mathcal{U}}$ corresponds to multiple messages.

[19] The details of how running an OTP is combined with a (signature based on a) PK can be found in Fig. 10, which is similar to this PK of a signature.

- $\mathsf{OAC.Setup}_{\mathrm{otp}}(1^\lambda)$: generate public parameters $\mathsf{params} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ of a type-3 bilinear group.
- $\mathsf{OAC.IKeyGen}_{\mathrm{otp}}(\mathsf{params}, w)$: choose generators $g \in \mathbb{G}_1$, $\widetilde{g} \in \mathbb{G}_2$ and random values $(x, y_1, \ldots, y_w, y_{1,\mathrm{otp}}, y_{2,\mathrm{otp}}, y_{\mathsf{s}}) \in \mathbb{Z}_p^{w+4}$, and compute
  $(X, Y_1, \ldots, Y_w, Y_{1,\mathrm{otp}}, Y_{2,\mathrm{otp}}, Y_{\mathsf{s}}) \leftarrow (g^x, g^{y_1}, \ldots, g^{y_w}, g^{y_{1,\mathrm{otp}}}, g^{y_{2,\mathrm{otp}}}, g^{y_{\mathsf{s}}})$,
  $(\widetilde{X}, \widetilde{Y}_1, \ldots, \widetilde{Y}_w, \widetilde{Y}_{1,\mathrm{otp}}, \widetilde{Y}_{2,\mathrm{otp}}, \widetilde{Y}_{\mathsf{s}}) \leftarrow (\widetilde{g}^x, \widetilde{g}^{y_1}, \ldots, \widetilde{g}^{y_w}, \widetilde{g}^{y_{1,\mathrm{otp}}}, \widetilde{g}^{y_{2,\mathrm{otp}}}, \widetilde{g}^{y_{\mathsf{s}}})$
  where $\mathsf{pk}_{\mathcal{I}} = (g, \{Y_i\}_{i=1}^w, Y_{1,\mathrm{otp}}, Y_{2,\mathrm{otp}}, Y_{\mathsf{s}}, \widetilde{g}, \widetilde{X}, \{\widetilde{Y}_i\}_{i=1}^w, \widetilde{Y}_{1,\mathrm{otp}}, \widetilde{Y}_{2,\mathrm{otp}}, \widetilde{Y}_{\mathsf{s}})$ and $\mathsf{sk}_{\mathcal{I}} = X$. The values $(Y_i, \widetilde{Y}_i)$ are related to attribute i, and $(g, \widetilde{g})$ are said to be related to dummy attribute 0.
- $\mathsf{OAC.CredIssue}_{\mathrm{otp}}(\mathcal{I}(\mathsf{sk}_{\mathcal{I}}), \mathcal{U}(\mathsf{att}_{\mathcal{U}}))$ is the following protocol between $\mathcal{I}$ and $\mathcal{U}$.
  1. To obtain a signature on the attributes $\mathsf{att}_{\mathcal{U}} = (a_1, \ldots, a_w) \in \mathbb{Z}_p^w$, $\mathcal{U}$ generates $C \leftarrow g^r \cdot Y_{\mathsf{s}}^{s_{\mathcal{U}}} \cdot \prod_{i=1}^w Y_i^{a_i}$ with random $r, s_{\mathcal{U}} \in \mathbb{Z}_p$, and sends C to $\mathcal{I}$.
  2. $\mathcal{U}$ gives to $\mathcal{I}$ the following proof of knowledge PK (Fig. 2) regarding C [18],
  $$\mathsf{PK}\{(r, s_{\mathcal{U}}, a_1, \ldots, a_w) : C = g^r \cdot Y_{\mathsf{s}}^{s_{\mathcal{U}}} \cdot \prod_{i=1}^w Y_i^{a_i}\}.$$
  3. If the PK regarding C is valid, $\mathcal{I}$ chooses random values $u, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, r_{1,\mathrm{otp}}, r_{2,\mathrm{otp}}\, s_{\mathcal{I}} \in \mathbb{Z}_p$, and sends $\mathcal{U}$ the following:
  $$\sigma' = (\sigma'_1, \sigma'_2) = (g^u, (X \cdot C \cdot Y_{1,\mathrm{otp}}^{a_{1,\mathrm{otp}}} \cdot Y_{2,\mathrm{otp}}^{a_{2,\mathrm{otp}}} \cdot Y_{\mathsf{s}}^{s_{\mathcal{I}}})^u),$$
  $$\mathrm{OTP},\ P = \langle \{\widetilde{Y}_{i,\mathrm{otp}}\}_{i=1}^2,\ \widetilde{Y}_{1,\mathrm{otp}}^{a_{1,\mathrm{otp}}} \cdot \widetilde{Y}_{2,\mathrm{otp}}^{a_{2,\mathrm{otp}}},\ s_{\mathcal{I}},\ p,\ \widetilde{Y}_{1,\mathrm{otp}}^{r_{1,\mathrm{otp}}} \cdot \widetilde{Y}_{2,\mathrm{otp}}^{r_{2,\mathrm{otp}}},\ \{[r_{i,\mathrm{otp}} - \bar{c} \cdot a_{i,\mathrm{otp}} \bmod p]_{\mathrm{otp}}\}_{i=1}^2 \rangle$$
  4. $\mathcal{U}$ obtains the signature σ on $(\mathsf{att}_{\mathcal{U}}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, \mathsf{sn})$ where $\mathsf{sn} = s_{\mathcal{U}} + s_{\mathcal{I}}$ as
  $$\sigma = (\sigma_1, \sigma_2) = \left(\sigma'_1,\ \sigma'_2/\sigma'^{\,r}_1\right) = (g^u,\ (X \cdot Y_{1,\mathrm{otp}}^{a_{1,\mathrm{otp}}} \cdot Y_{2,\mathrm{otp}}^{a_{2,\mathrm{otp}}} \cdot Y_{\mathsf{s}}^{\mathsf{sn}} \cdot \prod_{i=1}^w Y_i^{a_i})^u).$$

  The signature on $(\mathsf{att}_{\mathcal{U}}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, \mathsf{sn})$ can be verified as
  $$\sigma_1 \neq 1_{\mathbb{G}_1} \ \text{ and } \ e(\sigma_1,\ \widetilde{X} \cdot \widetilde{Y}_{1,\mathrm{otp}}^{a_{1,\mathrm{otp}}} \cdot \widetilde{Y}_{2,\mathrm{otp}}^{a_{2,\mathrm{otp}}} \cdot \widetilde{Y}_{\mathsf{s}}^{\mathsf{sn}} \cdot \prod_{i=1}^w \widetilde{Y}_i^{a_i}) \stackrel{?}{=} e(\sigma_2,\ \widetilde{g}).$$

  By viewing $\sigma = (\sigma_1, \sigma_2)$ as a signature on $(0, \mathsf{att}_{\mathcal{U}}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, \mathsf{sn})$ where the first entry is the value of dummy attribute 0, $\mathcal{U}$ can randomize σ to obtain another fresh signature on $(t, \mathsf{att}_{\mathcal{U}}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, \mathsf{sn})$ by computing new $\sigma \leftarrow (\sigma_1^s,\ (\sigma_1^t \cdot \sigma_2)^s)$ with random $s, t \in \mathbb{Z}_p$. We note that the portion corresponding to $(\mathsf{att}_{\mathcal{U}}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, \mathsf{sn})$ cannot be changed.
  $\langle \sigma, (t, \mathsf{att}_{\mathcal{U}}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, \mathsf{sn}), P \rangle$ corresponds to the credential $\mathsf{cr}_{\mathcal{U}}$.
- $\mathsf{OAC.CredShow}_{\mathrm{otp}}(\mathcal{V}(\mathsf{pk}_{\mathcal{I}}, \sigma, \mathsf{sn}), \mathcal{U}(\mathsf{cr}_{\mathcal{U}}))$ is a show protocol between $\mathcal{U}$ and verifier $\mathcal{V}$. What $\mathcal{U}$ does is to prove knowledge of a (randomized) signature σ. Since the verification of the randomized signature $\sigma = (\sigma_1, \sigma_2)$ on $(t, \mathsf{att}_{\mathcal{U}}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, \mathsf{sn})$ can be done as
  $$e(\sigma_1,\ \widetilde{X} \cdot \widetilde{g}^t \cdot \widetilde{Y}_{1,\mathrm{otp}}^{a_{1,\mathrm{otp}}} \cdot \widetilde{Y}_{2,\mathrm{otp}}^{a_{2,\mathrm{otp}}} \cdot \widetilde{Y}_{\mathsf{s}}^{\mathsf{sn}} \cdot \prod_{i=1}^w \widetilde{Y}_i^{a_i}) \stackrel{?}{=} e(\sigma_2,\ \widetilde{g}),$$
  this verification can also be viewed as
  $$e(\sigma_1, \widetilde{g})^t \cdot \prod_{i=1}^2 e(\sigma_1, \widetilde{Y}_{i,\mathrm{otp}})^{a_{i,\mathrm{otp}}} \cdot e(\sigma_1, \widetilde{Y}_{\mathsf{s}})^{\mathsf{sn}} \cdot \prod_{i=1}^w e(\sigma_1, \widetilde{Y}_i)^{a_i} \stackrel{?}{=} \frac{e(\sigma_2, \widetilde{g})}{e(\sigma_1, \widetilde{X})},$$
  so with bases $\{e(\sigma_1, \widetilde{g}), \{e(\sigma_1, \widetilde{Y}_{i,\mathrm{otp}})\}_{i=1}^2, e(\sigma_1, \widetilde{Y}_{\mathsf{s}}), \{e(\sigma_1, \widetilde{Y}_i)\}_{i=1}^w\}$, giving the following PK leads to proving knowledge of a signature:
  $$\mathsf{PK}\Big\{(t, a_1, \ldots, a_w, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}, \mathsf{sn}) : e(\sigma_2, \widetilde{g})/\{e(\sigma_1, \widetilde{X}) \cdot e(\sigma_1, \widetilde{Y}_{\mathsf{s}})^{\mathsf{sn}}\} = e(\sigma_1, \widetilde{g})^t \cdot \prod_{i=1}^w e(\sigma_1, \widetilde{Y}_i)^{a_i} \cdot \prod_{i=1}^2 e(\sigma_1, \widetilde{Y}_{i,\mathrm{otp}})^{a_{i,\mathrm{otp}}}\Big\}.$$
  Here $\mathcal{V}$ requires $\mathcal{U}$ to disclose sn, and $\mathcal{U}$ runs the OS sOTP P [19].

Fig. 7. One-Time Anonymous Credential Scheme

attributes $a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}} \in \mathbb{Z}_p$ to the commitment before signing it, and also hands an OS sOTP including $\{a_{i,\mathrm{otp}}\}_{i=1}^{2}$ to $\mathcal{U}$[20]. As a result, $\mathcal{U}$ is forced to use the sOTP to prove knowledge of a signature in the show protocol (as in Fig. 2) because $\mathcal{U}$ cannot know $\{a_{i,\mathrm{otp}}\}_{i=1}^{2}$ directly, and thus it leads to a one-time credential.

Building on the PS scheme and the above idea, our construction is given in Fig. 7. The main differences between the PS scheme and ours are:

- how to issue a credential in $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$,
- part of the prover's process in the show protocol is replaced with a run of an OS sOTP.

Although an OS sOTP is run in $\mathsf{OAC.CredShow}_{\mathrm{otp}}$, the prover's process is the same as that of the PS scheme. Thus it is sufficient for us to prove that $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$ is a blind signature scheme, i.e., its blindness and unforgeability with the following theorem (the proof is given in Appendix B).

Theorem 2. *The OAC scheme in Fig. 7 is one-time, blind, and unforgeable based on the security of OTPs and the underlying* PS *scheme in the RO model.*

Now we can see that the OAC scheme in Fig. 7 can be turned into an sOTP because the PK in $\mathsf{OAC.CredShow}_{\mathrm{otp}}$ can be turned into an SPK as OS signatures by using the Fiat-Shamir transform. Thus the credential issuer and holder can be viewed as a delegator and delegatee respectively. The value sn in $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$ cannot be changed by $\mathcal{U}$ in $\mathsf{OAC.CredShow}_{\mathrm{otp}}$ because of the unforgeability of the PS scheme, and needs to be disclosed in the resulting signature, so if the delegator finds more than one same sn in the collected signatures, the delegator can detect the fact that an sOTP was run more than once, thus achieving multi-run-detectability (if necessary, the delegator can announce that sn is blocked). The PS scheme and OS sOTPs are unlinkable because of randomizability, so our resulting sOTP also enjoys unlinkability for both the delegator and clouds.

- $\mathsf{EC.Setup}(1^\lambda)$ is the same as $\mathsf{OAC.Setup}_{\mathrm{otp}}(1^\lambda)$, and $\mathcal{B}$ obtains $\mathsf{params} = (p,\ \mathbb{G}_1,\ \mathbb{G}_2,\ \mathbb{G}_T,\ e)$.
- $\mathsf{EC.BKeyGen}(\mathsf{params})$: first run $\mathsf{OAC.IKeyGen}(\mathsf{params}, 3)$, and $\mathcal{B}$ obtain the signing key $\mathsf{sk}_\mathcal{B} = X$, and partial public key
  $\mathsf{ppk}_\mathcal{B} = (\mathsf{params},\ g,\ g_\mathsf{v},\ g_\mathsf{u},\ f_\mathsf{u},\ g_{1,\mathrm{otp}},\ g_{2,\mathrm{otp}},\ g_\mathsf{s},\ \widetilde{g},\ \widetilde{X},\ \widetilde{g}_\mathsf{v},\ \widetilde{g}_\mathsf{u},\ \widetilde{f}_\mathsf{u},\ \widetilde{g}_{1,\mathrm{otp}}, \widetilde{g}_{2,\mathrm{otp}}, \widetilde{g}_\mathsf{s})$.
  Next $\mathcal{B}$ computes an additional key $H(\mathsf{ppk}_\mathcal{B}) \to g_{\mathsf{u}'} \in \mathbb{G}_1$ with an appropriate hash function $H : \{0,1\}^* \to \mathbb{G}_1$,and lets $\mathsf{pk}_\mathcal{B} = (\mathsf{ppk}_\mathcal{B},\ g_{\mathsf{u}'})$.
- $\mathsf{EC.UKeyGen}(\mathsf{params},\ \mathsf{pk}_\mathcal{B})$: $\mathcal{U}_i$ chooses their secret key $\mathsf{sk}_{\mathcal{U}_i} \in \mathbb{Z}_p$ for EC and resultant public keys $(\mathsf{pk}_{\mathcal{U}_i} = g_\mathsf{u}^{\mathsf{sk}_{\mathcal{U}_i}},\ \mathsf{pk}'_{\mathcal{U}_i} = g_{\mathsf{u}'}^{\mathsf{sk}_{\mathcal{U}_i}})$. Moreover $\mathcal{U}_i$ generates

  $$\mathsf{PK}\{(\mathsf{sk}_{\mathcal{U}_i}) : \mathsf{pk}_{\mathcal{U}_i} = g_\mathsf{u}^{\mathsf{sk}_{\mathcal{U}_i}} \wedge \mathsf{pk}'_{\mathcal{U}_i} = g_{\mathsf{u}'}^{\mathsf{sk}_{\mathcal{U}_i}}\}$$

  and a signature $\sigma(\mathsf{pk}_{\mathcal{U}_i},\ \mathsf{pk}'_{\mathcal{U}_i})$ on $(\mathsf{pk}_{\mathcal{U}_i},\ \mathsf{pk}'_{\mathcal{U}_i})$ under their PKI key $\mathsf{pk}^{(\mathrm{pki})}_{\mathcal{U}_i}$, and sends $(\mathsf{pk}_{\mathcal{U}_i},\ \mathsf{pk}'_{\mathcal{U}_i},\ \mathsf{pk}^{(\mathrm{pki})}_{\mathcal{U}_i},\ \sigma(\mathsf{pk}_{\mathcal{U}_i},\ \mathsf{pk}'_{\mathcal{U}_i}),\ \mathsf{PK})$ to $\mathcal{B}$ [21], which stores them in the user DB.
- $\mathsf{EC.Withdraw}(\mathcal{B}(\mathsf{sk}_\mathcal{B},\ \mathsf{pk}_{\mathcal{U}_i}), \mathcal{U}_i(\mathsf{sk}_{\mathcal{U}_i},\ \mathsf{pk}_\mathcal{B}))$ is the following protocol between $\mathcal{B}$ and $\mathcal{U}_i$ where $\mathcal{U}_i$ obtains an e-coin corresponding to v dollars [22].
  1. $\mathcal{U}_i$ sends $v,\ \mathsf{pk}_{\mathcal{U}_i},\ \mathsf{pk}'_{\mathcal{U}_i}$ to $\mathcal{B}$, and also gives $\mathsf{PK}\{(\mathsf{sk}_{\mathcal{U}_i}) : \mathsf{pk}_{\mathcal{U}_i} = g_\mathsf{u}^{\mathsf{sk}_{\mathcal{U}_i}}\}$ (Fig. 2) [23]. $\mathcal{B}$ rejects the request if $\mathsf{pk}_{\mathcal{U}_i}$ is not found in the user DB.
  2. $\mathcal{U}_i$ chooses random $r,\ s_{\mathcal{U}_i},\ \omega_\mathsf{u} \in \mathbb{Z}_p$, computes the following commitment C, and gives the following PK to $\mathcal{B}$.

     $$\mathsf{PK}\{(r,\ s_{\mathcal{U}_i},\ \omega_\mathsf{u}) : C = g^r \cdot g_\mathsf{s}^{s_{\mathcal{U}_i}} \cdot f_\mathsf{u}^{\omega_\mathsf{u}}\}$$

  3. If the PK is valid, $\mathcal{B}$ chooses random $u,\ s_\mathcal{B},\ x_{1,\mathrm{otp}}, x_{2,\mathrm{otp}}, r_{1,\mathrm{otp}}, r_{2,\mathrm{otp}} \in \mathbb{Z}_p$, computes

     $$\begin{aligned} C' &\leftarrow C \cdot \mathsf{pk}_{\mathcal{U}_i} \cdot g_\mathsf{v}^v \cdot g_\mathsf{s}^{s_\mathcal{B}} \cdot g_{1,\mathrm{otp}}^{x_{1,\mathrm{otp}}} \cdot g_{2,\mathrm{otp}}^{x_{2,\mathrm{otp}}} \\ &= g^r \cdot g_\mathsf{v}^v \cdot g_\mathsf{s}^{s_\mathcal{B}+s_{\mathcal{U}_i}} \cdot g_\mathsf{u}^{\mathsf{sk}_{\mathcal{U}_i}} \cdot f_\mathsf{u}^{\omega_\mathsf{u}} \cdot g_{1,\mathrm{otp}}^{x_{1,\mathrm{otp}}} \cdot g_{2,\mathrm{otp}}^{x_{2,\mathrm{otp}}}, \\ \sigma' &\leftarrow (\sigma'_1,\ \sigma'_2) = (g^u,\ (X \cdot C')^u), \end{aligned}$$

     and generates the OTP P as follows

     $$P = \langle \widetilde{g}_{1,\mathrm{otp}},\ \widetilde{g}_{2,\mathrm{otp}},\ \widetilde{g}_{1,\mathrm{otp}}^{x_{1,\mathrm{otp}}} \cdot \widetilde{g}_{2,\mathrm{otp}}^{x_{2,\mathrm{otp}}},\ p,\ \widetilde{g}_{2,\mathrm{otp}}^{r_{2,\mathrm{otp}}} \cdot \widetilde{g}_{2,\mathrm{otp}}^{r_{2,\mathrm{otp}}}, \{[r_{i,\mathrm{otp}} - \bar{c} \cdot x_{i,\mathrm{otp}} \bmod p]_\mathrm{otp}\}_{i=1}^2 \rangle.$$

     $\mathcal{B}$ sends σ', $s_\mathcal{B}$, P to $\mathcal{U}_i$, and debits $\mathcal{U}_i$'s account v dollars.
  4. $\mathcal{U}_i$ obtains the signature σ'' on $(v,\ \mathsf{sn},\ \mathsf{sk}_{\mathcal{U}_i},\ \omega_\mathsf{u},\ x_{1,\mathrm{otp}},\ x_{2,\mathrm{otp}})$ where $\mathsf{sn} = s_\mathcal{B} + s_{\mathcal{U}_i}$ (called *serial number*) as

     $$\sigma'' = (\sigma''_1,\ \sigma''_2) = (\sigma'_1,\ \sigma'_2/{\sigma'_1}^r),$$

     which can be verified as

     $$e(\sigma''_1,\ \widetilde{X} \cdot \widetilde{g}_\mathsf{v}^v \cdot \widetilde{g}_\mathsf{s}^\mathsf{sn} \cdot \widetilde{g}_\mathsf{u}^{\mathsf{sk}_{\mathcal{U}_i}} \cdot \widetilde{f}_\mathsf{u}^{\omega_\mathsf{u}} \cdot \widetilde{g}_{1,\mathrm{otp}}^{x_{1,\mathrm{otp}}} \cdot \widetilde{g}_{2,\mathrm{otp}}^{x_{2,\mathrm{otp}}}) \stackrel{?}{=} e(\sigma''_2,\ \widetilde{g}).$$

     $\mathcal{U}_i$ can obtain a randomized signature σ_co on $(t,\ v,\ \mathsf{sn},\ \mathsf{sk}_{\mathcal{U}_i},\ \omega_\mathsf{u},\ x_{1,\mathrm{otp}},\ x_{2,\mathrm{otp}})$ with random $s,\ t \in \mathbb{Z}_p$ and computing, $\sigma_\mathsf{co} = (\sigma_1^s,\ (\sigma_1^t \cdot \sigma_2)^s)$.
     The obtained e-coin co consists of $(\sigma_\mathsf{co},\ t,\ v,\ \mathsf{sn},\ \mathsf{sk}_{\mathcal{U}_i},\ \omega_\mathsf{u},\ P)$.

Fig. 8. E-Cash based on signing OTPs (1/2)

– $\mathsf{EC.Spend}(\mathcal{M}_j(\mathsf{pk}_{\mathcal{B}}, \sigma_{\mathsf{co}}, v, \mathsf{sn}), \mathcal{U}_i(\mathsf{pk}_{\mathcal{B}}, \mathsf{co}))$ is the following protocol between a merchant $\mathcal{M}_j$ who has a signing key pair $(\mathsf{sk}_{\mathcal{M}_j}, \mathsf{pk}_{\mathcal{M}_j})$ and $\mathcal{U}_i$ where $\mathcal{U}_i$ spends an e-coin co corresponding to v dollars.
  1. $\mathcal{M}_j$ sends $\mathsf{pk}_{\mathcal{M}_j}$, info_j to $\mathcal{U}_i$ where info_j is a random bit string.
  2. $\mathcal{U}_i$ computes $c_{\mathsf{ds}} \leftarrow H(\mathsf{pk}_{\mathcal{M}_j} \parallel \mathsf{info}_j)$ called *double-spending challenge* where H is a hash function (modeled as an RO), and the following commitments

$$\begin{aligned}
&\mathsf{Com}_{\mathsf{co}} \leftarrow e(\sigma_2, \widetilde{g})/e(\sigma_1, \widetilde{X} \cdot \widetilde{g}_{\mathsf{v}}^{v} \cdot \widetilde{g}_{\mathsf{s}}^{\mathsf{sn}}) \text{ (for proof of knowledge of a signature)} \\
&\quad = e(\sigma_1, \widetilde{g})^{t} \cdot e(\sigma_1, \widetilde{g}_{\mathsf{u}})^{\mathsf{sk}_{\mathcal{U}_i}} \cdot e(\sigma_1, \widetilde{f}_{\mathsf{u}})^{\omega_{\mathsf{u}}} \cdot e(\sigma_1, \widetilde{g}_{1,\mathrm{otp}})^{x_{1,\mathrm{otp}}} \cdot e(\sigma_1, \widetilde{g}_{2,\mathrm{otp}})^{x_{2,\mathrm{otp}}}, \\
&\mathsf{Com}_{\mathsf{ds}} \leftarrow g_{\mathsf{u}'}^{\mathsf{sk}_{\mathcal{U}_i}} \cdot (g_{\mathsf{u}'}^{\omega_{\mathsf{u}}})^{c_{\mathsf{ds}}} = g_{\mathsf{u}'}^{\mathsf{sk}_{\mathcal{U}_i}} \cdot (g_{\mathsf{u}'}^{c_{\mathsf{ds}}})^{\omega_{\mathsf{u}}} \quad \text{(called } \textit{double-spending tag}\text{)}
\end{aligned}$$

  and generates the following signature $\sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j)$ on $(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j)$ (Sect. 2) by using the OTP P as well, and sends $(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j))$ to $\mathcal{M}_j$:

$$\begin{aligned}
&\sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j) = \mathsf{SPK}\{(t, \mathsf{sk}_{\mathcal{U}_i}, \omega_{\mathsf{u}}, x_{1,\mathrm{otp}}, x_{2,\mathrm{otp}}) : \\
&\quad \mathsf{Com}_{\mathsf{co}} = e(\sigma_1, \widetilde{g})^{t} e(\sigma_1, \widetilde{g}_{\mathsf{u}})^{\mathsf{sk}_{\mathcal{U}_i}} e(\sigma_1, \widetilde{f}_{\mathsf{u}})^{\omega_{\mathsf{u}}} e(\sigma_1, \widetilde{g}_{1,\mathrm{otp}})^{x_{1,\mathrm{otp}}} e(\sigma_1, \widetilde{g}_{2,\mathrm{otp}})^{x_{2,\mathrm{otp}}} \\
&\quad \wedge \mathsf{Com}_{\mathsf{ds}} = g_{\mathsf{u}'}^{\mathsf{sk}_{\mathcal{U}_i}} \cdot (g_{\mathsf{u}'}^{c_{\mathsf{ds}}})^{\omega_{\mathsf{u}}}\}(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j). \text{ [24)}
\end{aligned}$$

  3. $\mathcal{M}_j$ accepts the e-coin if $\sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j)$ is a valid signature, and stores the following tuple which will be deposited later

$$\mathsf{dpst} = \langle \sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j, \sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j) \rangle$$

  where sn is the serial number of this e-coin.

– $\mathsf{EC.Deposit}(\mathsf{dpst})$: $\mathcal{B}$ does the following after receiving from $\mathcal{M}_j$, $\mathsf{dpst} = \langle \sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j, \sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j) \rangle$.
  1. If $\sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j)$ in dpst is invalid, $\mathcal{B}$ rejects the deposit.
  2. If the verification is successful and the serial number sn is fresh in the deposit DB, $\mathcal{B}$ requires $\mathcal{M}_j$ to send a signature $\sigma_{\mathcal{M}_j}(\mathsf{dpst})$ on dpst under $\mathsf{pk}_{\mathcal{M}_j}$. If $\sigma_{\mathcal{M}_j}(\mathsf{dpst})$ is invalid, $\mathcal{B}$ rejects the deposit, and otherwise $\mathcal{B}$ stores $(\mathsf{dpst}, \sigma_{\mathcal{M}_j}(\mathsf{dpst}))$ in the deposit DB, and credits v dollars to $\mathcal{M}_j$'s account.
  3. If a tuple exists in the deposit DB which has the same sn, $\mathsf{pk}_{\mathcal{M}_j}$, info_j as dpst, $\mathcal{B}$ rejects this invalid deposit (i.e., $\mathcal{M}_j$ is cheating).
  4. If $\mathsf{dpst}' = \langle \sigma'_{\mathsf{co}}, v', \mathsf{sn}, \mathsf{Com}'_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}'_j}, \mathsf{info}'_j, \sigma(\sigma'_{\mathsf{co}}, v', \mathsf{sn}, \mathsf{Com}'_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}'_j}, \mathsf{info}'_j) \rangle$ exists in the deposit DB which has the same sn as dpst, but different $\mathsf{pk}_{\mathcal{M}'_j}$ or info'_j [25)], then $\mathcal{B}$ can have the proof $\Pi_{\mathsf{ds}} = (\mathsf{dpst}, \mathsf{dpst}')$ which can be used to identify the double-spender's public key in $\mathsf{EC.Identify}$.

– $\mathsf{EC.Identify}(\mathsf{params}, \mathsf{pk}_{\mathcal{B}}, \Pi_{\mathsf{ds}})$: $\mathcal{B}$ identifies a double-spender as follows:
  1. If dpst and dpst' in Π_{ds} have the same serial number, $\mathcal{B}$ obtains the double-spending tags $(\mathsf{Com}_{\mathsf{ds}}, c_{\mathsf{ds}})$, $(\mathsf{Com}'_{\mathsf{ds}}, c'_{\mathsf{ds}})$ from Π_{ds}.
  2. The double-spender's public key $\mathsf{pk}'_{\mathsf{ds}}$ can be computed as

$$\mathsf{pk}'_{\mathsf{ds}} = (\mathsf{Com}_{\mathsf{ds}}^{c'_{\mathsf{ds}}} / \mathsf{Com}'^{\,c_{\mathsf{ds}}}_{\mathsf{ds}})^{1/(c'_{\mathsf{ds}} - c_{\mathsf{ds}})}.$$

– $\mathsf{EC.VrfyGuilt}(\mathsf{params}, \mathsf{pk}_{\mathcal{B}}, \mathsf{sn}, \mathsf{pk}'_{\mathsf{ds}}, \Pi_{\mathsf{ds}})$: anyone can publicly verify the proof Π_{ds} that the user with $\mathsf{pk}'_{\mathsf{ds}}$ is guilty of double-spending the e-coin whose serial number is sn. The verification can be done by $\mathsf{EC.Identify}(\mathsf{params}, \mathsf{pk}_{\mathcal{B}}, \Pi_{\mathsf{ds}}) \stackrel{?}{=} \mathsf{pk}'_{\mathsf{ds}}$.

Fig. 9. E-Cash based on signing OTPs (2/2)

5 E-Cash based on Signing OTPs

Building on our sOTPs based on OACs (Fig. 7), we construct an e-cash scheme. In the traditional e-cash originating from [40], the following protocols exist:

- Withdraw protocol: A user $\mathcal{U}$ communicates with bank $\mathcal{B}$, and receives electronic data (called *e-coin*), and $\mathcal{B}$ debits $\mathcal{U}$'s account the corresponding value.
- Spend protocol: $\mathcal{U}$ spends an e-coin by sending it to a merchant $\mathcal{M}$.
- Deposit protocol: $\mathcal{M}$ deposits the e-coin spent by $\mathcal{U}$ to $\mathcal{B}$, and $\mathcal{B}$ credits the corresponding amount to the $\mathcal{M}$'s account.

Employing our sOTP, our EC (Fig. 8, 9) prevents double-spending[26] and further identifies a double-spender even if sOTPs are broken. We adopt the elegant framework [29] such that $\mathcal{B}$ can issue an e-coin including an sOTP and user's ID in the Withdraw protocol without embedding its master signing key, and two signatures originating from the same e-coin (collected in the Deposit protocol) can reveal the user's ID. The scheme in [24, Sect.6.3] takes a similar approach, but the double-spender's secret key is revealed, so "exculpability" is not achieved (i.e., $\mathcal{B}$ can frame users), while ours reveals only the double-spender's public key according to [29], thus achieving exculpability. Following the e-cash security model [29, 8, 64, 19], we give the proofs in Appendix C, and discuss the additional possible extensions.

Batch Spending. What happens in the Spend protocol can be viewed as:

- $\mathcal{U}_i$ has an e-coin that can be viewed as a kind of public key certified by $\mathcal{B}$.
- $\mathcal{U}_i$ signs the message from $\mathcal{M}_j$ with e-coin, and sends the signature to $\mathcal{M}_j$.
- Then the value of the e-coin is transferred to $\mathcal{M}_j$.

Hence, e.g., if $\mathcal{U}_i$ has e-coins σ_{co_1}, σ_{co_2} corresponding to v_1, v_2 dollars respectively, and signs σ_{co_1} by σ_{co_2}, then we can think that the value v_2 in σ_{co_2} is transferred to σ_{co_1}, and that signing a message with σ_{co_1} yields $v_1 + v_2$ dollars. This way of

[20] $\{a_{i,\mathrm{otp}}\}_{i=1}^{2}$ corresponding to a signing key in the OS scheme are fresh random and used only once in our OAC, so the attack [15] on OS blind signatures does not apply here because [15] needs concurrent $\mathsf{polylog}(\lambda)$ signing queries with the same signing key.

[21] We take the approach similar to group signatures in [78]. This is needed to identify the user in the real world when disputes related to double-spending occur.

[22] We define EC.Withdraw such that $\mathcal{U}_i$ can specify v, but in practice, v may be a constant or chosen from a set of predefined e-coin denominations to reduce linkability.

[23] This PK will be interactive or a signature on a fresh nonce to avoid replay attacks.

[24] This includes proofs of knowledge of equality of discrete logs $(\mathsf{sk}_{\mathcal{U}_i}, \omega_{\mathsf{u}})$ [47], and its full description is given in Fig. 10.

[25] If this occurs, it means the adversary ran an OTP more than once by breaking the security of OTPs. In this case, it is possible to distribute a list of blocked sn.

[26] Our e-cash is somewhat incomparable to existing e-cash since we assume there exist distributed partially trusted clouds as in [51, 31], while other schemes do not.

thinking can reduce the number of signatures that need to be generated during the Spend protocol, and we give the overview of this method (which we call *batch spending*) as follows:

- Suppose $\mathcal{U}_i$ has e-coins, e.g., σ_{co_1}, σ_{co_2}, σ_{co_3} corresponding to v_1, v_2, v_3 dollars respectively, and wants to spend $v_1 + v_2 + v_3$ dollars for $\mathcal{M}_j$.
- Then $\mathcal{U}_i$ signs σ_{co_1} with σ_{co_2}, σ_{co_3} in advance, obtaining 2 signatures on σ_{co_1}.
- In the Spend protocol with $\mathcal{M}_j$, $\mathcal{U}_i$ signs the message from $\mathcal{M}_j$ with σ_{co_1}, and sends 3 signatures to $\mathcal{M}_j$.
- $\mathcal{M}_j$ verifies the 2 signatures on σ_{co_1}, and another signature generated by σ_{co_1} [27]. If all the verifications are successful and the amount of e-coins suffices, $\mathcal{M}_j$ accepts the e-coins.
- Similarly $\mathcal{B}$ also verifies all the signatures in the Deposit protocol, and checks freshness of all the serial numbers.

As we can see, $\mathcal{U}_i$ has only to generate 1 signature during the Spend protocol although actually it spends 3 e-coins. To sign σ_{co_1} with σ_{co}, as in Step 2 of EC.Spend (Fig. 9), a double-spending challenge $c_{\mathsf{ds}} \in \mathbb{Z}_p$ is necessary, for which $H(\sigma_{\mathsf{co}_1})$ can be used here. To make the difference clear between the signature on the double-spending tag (i.e., $\sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j)$ in Fig. 9) and signature on the e-coin, we modify the hash calculation of Eq. (1) (Fig. 10) by adding a simple tag as

$$c = H(R_1 \parallel R_2 \parallel \sigma_{\mathsf{co}} \parallel v \parallel \mathsf{sn} \parallel \mathsf{Com}_{\mathsf{ds}} \parallel 0 \parallel \mathsf{pk}_{\mathcal{M}_j} \parallel \mathsf{info}_j)$$
(case of signature on $(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j)$),

$$c = H(R_1 \parallel R_2 \parallel \sigma_{\mathsf{co}} \parallel v \parallel \mathsf{sn} \parallel \mathsf{Com}_{\mathsf{ds}} \parallel 1 \parallel \sigma_{\mathsf{co}_1})$$
(case with additional e-coin σ_{co_1}).

Thus if $\mathcal{B}$ or $\mathcal{M}_j$ receives a signature with $H(R_1 \parallel R_2 \parallel \sigma_{\mathsf{co}} \parallel v \parallel \mathsf{sn} \parallel \mathsf{Com}_{\mathsf{ds}} \parallel 1 \parallel \sigma_{\mathsf{co}_1})$, $\mathcal{B}$ or $\mathcal{M}_j$ also requires another signature by σ_{co_1} to accept the e-coins.

Transferring E-Coin. By further extending batch spending, $\mathcal{U}_i$ can transfer an e-coin to another, but with somewhat less anonymity as mentioned later. In batch spending, $\mathcal{U}_i$ signs its own e-coins, whereas, in transferring $\mathcal{U}_i$'s e-coin σ_{co_i} to another e-coin σ_{co_j} of $\mathcal{U}_j$, $\mathcal{U}_i$ signs σ_{co_j} with σ_{co_i}, and the value of σ_{co_i} is transferred to σ_{co_j}. Here the value of σ_{co_j} can be zero. To receive the transfer of many e-coins, we assume that $\mathcal{U}_j$ can obtain e-coins whose values are zero (i.e., they function as placeholders of the transferred e-coins) from $\mathcal{B}$ for free in advance. In this case, however, we have incomplete anonymity in the following sense: Suppose $\mathcal{U}_i$ transferred σ_{co_i} to σ_{co_j}, and the several transfers continued, and the e-coin originating from σ_{co_i} returned to $\mathcal{U}_i$, then $\mathcal{U}_i$ can recognize that $\mathcal{U}_i$ used to hold the e-coin. To obtain complete anonymity, the technique from [8] may be applicable, but its efficient instantiation will be non-trivial.

[27] More exactly $\mathcal{M}_j$ will also need to check that the serial numbers of σ_{co_1}, σ_{co_2}, σ_{co_3} are all different.

– In Fig. 9, $\mathcal{U}_i$ generates the following SPK in spending v dollars with serial number sn by running sOTP P,

$$\sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j) = \mathsf{SPK}\{(t, \mathsf{sk}_{\mathcal{U}_i}, \omega_{\mathsf{u}}, x_{1,\mathrm{otp}}, x_{2,\mathrm{otp}}) :$$
$$\mathsf{Com}_{\mathsf{co}} = e(\sigma_1, \widetilde{g})^t e(\sigma_1, \widetilde{g}_{\mathsf{u}})^{\mathsf{sk}_{\mathcal{U}_i}} e(\sigma_1, \widetilde{f}_{\mathsf{u}})^{\omega_{\mathsf{u}}} e(\sigma_1, \widetilde{g}_{1,\mathrm{otp}})^{x_{1,\mathrm{otp}}} e(\sigma_1, \widetilde{g}_{2,\mathrm{otp}})^{x_{2,\mathrm{otp}}}$$
$$\wedge\ \mathsf{Com}_{\mathsf{ds}} = g_{\mathsf{u}'}^{\mathsf{sk}_{\mathcal{U}_i}} \cdot (g_{\mathsf{u}'}^{c_{\mathsf{ds}}})^{\omega_{\mathsf{u}}}\}(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j),$$

where $c_{\mathsf{ds}} = H(\mathsf{pk}_{\mathcal{M}_j} \parallel \mathsf{info}_j)$,
$$P = \langle \widetilde{g}_{1,\mathrm{otp}},\ \widetilde{g}_{2,\mathrm{otp}},\ \widetilde{g}_{1,\mathrm{otp}}^{x_{1,\mathrm{otp}}} \cdot \widetilde{g}_{2,\mathrm{otp}}^{x_{2,\mathrm{otp}}},\ p,\ \widetilde{g}_{1,\mathrm{otp}}^{r_{1,\mathrm{otp}}} \cdot \widetilde{g}_{2,\mathrm{otp}}^{r_{2,\mathrm{otp}}},\ \{[r_{i,\mathrm{otp}} - \bar{c} \cdot x_{i,\mathrm{otp}} \bmod p]_{\mathrm{otp}}\}_{i=1}^{2} \rangle.$$

1. $\mathcal{U}_i$ chooses random $\alpha_1, \alpha_2, \beta, \beta_1, \beta_2, \beta_3 \in \mathbb{Z}_p$ and computes

$$R \leftarrow \widetilde{g}_{1,\mathrm{otp}}^{r_{1,\mathrm{otp}}} \cdot \widetilde{g}_{2,\mathrm{otp}}^{r_{2,\mathrm{otp}}} \cdot \widetilde{g}_{1,\mathrm{otp}}^{\alpha_1} \cdot \widetilde{g}_{2,\mathrm{otp}}^{\alpha_2} \cdot (\widetilde{g}_{1,\mathrm{otp}}^{x_{1,\mathrm{otp}}} \cdot \widetilde{g}_{2,\mathrm{otp}}^{x_{2,\mathrm{otp}}})^{\beta}$$
$$R_1 \leftarrow e(\sigma_1, \widetilde{g})^{\beta_1} \cdot e(\sigma_1, \widetilde{g}_{\mathsf{u}})^{\beta_2} \cdot e(\sigma_1, \widetilde{f}_{\mathsf{u}})^{\beta_3} \cdot e(\sigma_1, R)$$
$$(= e(\sigma_1,\ \widetilde{g}^{\beta_1} \cdot \widetilde{g}_{\mathsf{u}}^{\beta_2} \cdot \widetilde{f}_{\mathsf{u}}^{\beta_3} \cdot R))$$
$$R_2 \leftarrow g_{\mathsf{u}'}^{\beta_2} \cdot (g_{\mathsf{u}'}^{c_{\mathsf{ds}}})^{\beta_3} \quad \text{(here } g_{\mathsf{u}'} \text{ and } g_{\mathsf{u}'}^{c_{\mathsf{ds}}} \text{ are bases of the commitment)}$$
$$c \leftarrow H(R_1 \parallel R_2 \parallel \sigma_{\mathsf{co}} \parallel v \parallel \mathsf{sn} \parallel \mathsf{Com}_{\mathsf{ds}} \parallel \mathsf{pk}_{\mathcal{M}_j} \parallel \mathsf{info}_j) \tag{1}$$
$$z_1 \leftarrow \beta_1 - c \cdot t \bmod p$$
$$z_2 \leftarrow \beta_2 - c \cdot \mathsf{sk}_{\mathcal{U}_i} \bmod p$$
$$z_3 \leftarrow \beta_3 - c \cdot \omega_{\mathsf{u}} \bmod p$$
$$z_4 \leftarrow r_{1,\mathrm{otp}} - (c - \beta) \cdot x_{1,\mathrm{otp}} + \alpha_1 \bmod p \quad \text{(run with sOTP } P\text{)}$$
$$z_5 \leftarrow r_{2,\mathrm{otp}} - (c - \beta) \cdot x_{2,\mathrm{otp}} + \alpha_2 \bmod p \quad \text{(run with sOTP } P\text{)}$$

where $\{r_{i,\mathrm{otp}} - (c-\beta) \cdot x_{i,\mathrm{otp}}\}_{i=1}^{2}$ can be obtained from P by inputting $c - \beta$ (this is the same randomization as $\mathsf{OS.BSign}$ of Algorithm 1).

2. The signature $\sigma(\sigma_{\mathsf{co}}, v, \mathsf{sn}, \mathsf{Com}_{\mathsf{ds}}, \mathsf{pk}_{\mathcal{M}_j}, \mathsf{info}_j)$ consists of $(c, z_1, z_2, z_3, z_4, z_5)$ and can be verified by

$$c \stackrel{?}{=} H(e(\sigma_1, \widetilde{g})^{z_1} \cdot e(\sigma_1, \widetilde{g}_{\mathsf{u}})^{z_2} \cdot e(\sigma_1, \widetilde{f}_{\mathsf{u}})^{z_3} \cdot e(\sigma_1, \widetilde{g}_{1,\mathrm{otp}})^{z_4} \cdot e(\sigma_1, \widetilde{g}_{2,\mathrm{otp}})^{z_5} \cdot \mathsf{Com}_{\mathsf{co}}^{c}$$
$$\parallel g_{\mathsf{u}'}^{z_2} \cdot (g_{\mathsf{u}'}^{c_{\mathsf{ds}}})^{z_3} \cdot \mathsf{Com}_{\mathsf{ds}}^{c} \parallel \sigma_{\mathsf{co}} \parallel v \parallel \mathsf{sn} \parallel \mathsf{Com}_{\mathsf{ds}} \parallel \mathsf{pk}_{\mathcal{M}_j} \parallel \mathsf{info}_j).$$

Fig. 10. Signature based on Proof of Knowledge (SPK) in Our E-Cash

Acknowledgments. The author thanks Jacob Schuldt for his valuable comments on the early draft and anonymous reviewers of Financial Cryptography'20 and ProvSec'20 for their helpful comments. This work was supported in part by JSPS KAKENHI Grant Number 20K11807.

References

1. Intel Software Guard Extensions (Intel SGX). `https://software.intel.com/en-us/sgx`.
2. ARM Consortium. ARM Trustzone, Available at `https://www.arm.com/products/security-on-arm/trustzone`, 2017.
3. Tolga Acar, Sherman Chow, and Lan Nguyen. Accumulators and U-Prove revocation. In *Financial Cryptography and Data Security*, pages 189–196. Springer, 2013.
4. Tolga Acar and Lan Nguyen. Revocation for delegatable anonymous credentials. In *PKC*, pages 423–440. Springer, 2011.
5. Norio Akagi, Yoshifumi Manabe, and Tatsuaki Okamoto. An efficient anonymous credential system. In *Financial Cryptography and Data Security*, pages 272–286. Springer, 2008.
6. Ryan Amos, Marios Georgiou, Aggelos Kiayias, and Mark Zhandry. One-shot signatures and applications to hybrid quantum/classical authentication. In *STOC*, pages 255–268, 2020.
7. Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In *SCN*, pages 111–125. Springer, 2006.
8. Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. Anonymous transferable e-cash. In *PKC*, pages 101–124. Springer, 2015.
9. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *CCS*, pages 1087–1098. ACM, 2013.
10. Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Eurocrypt*, pages 480–494. Springer, 1997.
11. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO*, pages 108–125. Springer, 2009.
12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *Asiacrypt*, pages 134–153. Springer, 2012.
13. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73. ACM, 1993.
14. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, pages 136–153. Springer, 2005.
15. Fabrice Benhamouda, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. Cryptology ePrint Archive, Report 2020/945, 2020.
16. Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous credentials on a standard Java card. In *CCS*, pages 600–610. ACM, 2009.
17. Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In *ACNS*, pages 221–239. Springer, 2018.

18. Nikita Borisov and Eric A Brewer. Active certificates: A framework for delegation. In *NDSS*, 2002.
19. Florian Bourse, David Pointcheval, and Olivier Sanders. Divisible e-cash from constrained pseudo-random functions. In *Asiacrypt*, pages 679–708. Springer, 2019.
20. Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In *Eurocrypt*, pages 427–444. Springer, 2006.
21. Xavier Boyen and Brent Waters. Full-domain subgroup hiding and constant-size group signatures. In *PKC*, pages 1–15. Springer, 2007.
22. Stefan Brands. Untraceable off-line cash in wallets with observers. In *CRYPTO*, pages 302–318. Springer, 1993.
23. Stefan Brands. Off-line electronic cash based on secret-key certificates. In *Latin American Symposium on Theoretical Informatics*, pages 131–166. Springer, 1995.
24. Stefan Brands. *Rethinking public key infrastructures and digital certificates: building in privacy*. MIT Press, 2000.
25. Stefan Brands. A technical overview of digital credentials. *Available online, Feb*, 20:145–8, 2002. `http://www.credentica.com/overview.pdf`.
26. Ahto Buldas, Aivo Kalu, Peeter Laud, and Mart Oruaas. Server-supported RSA signatures for mobile devices. In *ESORICS*, pages 315–333. Springer, 2017.
27. Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In *CCS*, pages 683–699. ACM, 2017.
28. Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clone wars: efficient periodic n-times anonymous authentication. In *CCS*, pages 201–210. ACM, 2006.
29. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Eurocrypt*, volume 3494, pages 302–321. Springer, 2005.
30. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. Solving revocation with efficient update of anonymous credentials. In *SCN*, pages 454–471. Springer, 2010.
31. Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. Virtual smart cards: How to sign with a password and a server. In *SCN*, pages 353–371. Springer, 2016.
32. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Eurocrypt*, pages 93–118. Springer, 2001.
33. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76. Springer, 2002.
34. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *CRYPTO*, pages 410–424. Springer, 1997.
35. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *CCS*, pages 21–30. ACM, 2002.
36. Jan L Camenisch, Jean-Marc Piveteau, and Markus A Stadler. Blind signatures based on the discrete logarithm problem. In *Eurocrypt*, pages 428–432. Springer, 1994.
37. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000.
38. Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, pages 219–233. IEEE, 2004.
39. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO*, pages 78–96. Springer, 2006.

40. David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203. Springer, 1983.
41. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
42. David Chaum. Online cash checks. In *Eurocrypt*, pages 288–293. Springer, 1989.
43. David Chaum. Achieving electronic privacy. *Scientific American*, 267(2):96–101, 1992.
44. David Chaum and Jan-Hendrik Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In *CRYPTO*, pages 118–167. Springer, 1986.
45. David Chaum, Jan-Hendrik Evertse, and Jeroen Van De Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *Eurocrypt*, pages 127–141. Springer, 1987.
46. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *CRYPTO*, pages 319–327. Springer, 1988.
47. David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105. Springer, 1992.
48. Ronald Cramer and Torben P Pedersen. Improved privacy in wallets with observers. In *Eurocrypt*, pages 329–343. Springer, 1993.
49. Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. In *Eurocrypt*, pages 555–572. Springer, 2006.
50. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662. Springer, 2012.
51. Adam Everspaugh, Rahul Chaterjee, Samuel Scott, Ari Juels, and Thomas Ristenpart. The Pythia PRF service. In *USENIX Security Symposium*, pages 547–562, 2015.
52. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194. Springer, 1986.
53. Georg Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. Cryptology ePrint Archive, Report 2009/320, 2009.
54. Georg Fuchsbauer. Commuting signatures and verifiable encryption. In *Eurocrypt*, pages 224–245. Springer, 2011.
55. Georg Fuchsbauer and David Pointcheval. Anonymous proxy signatures. In *SCN*, pages 201–217. Springer, 2008.
56. Georg Fuchsbauer and David Pointcheval. Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. In *Pairing*, pages 132–149. Springer, 2009.
57. Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.
58. Morrie Gasser and Ellen McDermott. An architecture for practical delegation in a distributed system. In *Computer Society Symposium on Security and Privacy*, page 20. IEEE, 1990.
59. Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. One-time programs. In *CRYPTO*, pages 39–56. Springer, 2008.
60. Jens Groth. Fully anonymous group signatures without random oracles. In *Asiacrypt*, pages 164–180. Springer, 2007.
61. Gabriel Kaptchuk, Ian Miers, and Matthew Green. Giving state to the stateless: Augmenting trustworthy computation with ledgers. In *NDSS*, 2019.

62. Jonathan Katz, Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996. `http://cacr.uwaterloo.ca/hac/`.
63. Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N Asokan. Keys in the clouds: Auditable multi-device access to cryptographic credentials. In *ARES*, pages 40:1–40:10. ACM, 2018.
64. Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based PRFs and applications to e-cash. In *Asiacrypt*, pages 304–335. Springer, 2017.
65. Anna Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2002.
66. Philip MacKenzie and Michael K Reiter. Delegation of cryptographic servers for capture-resilient devices. *Distributed Computing*, 16(4):307–327, 2003.
67. Philip MacKenzie and Michael K Reiter. Networked cryptographic devices resilient to capture. *International Journal of Information Security*, 2(1):1–20, 2003.
68. Antonio Marcedone, Rafael Pass, and Abhi Shelat. Minimizing trust in hardware wallets with two factor signatures. In *Financial Cryptography and Data Security*, pages 407–425. Springer, 2019.
69. Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. DelegaTEE: Brokered delegation using trusted execution environments. In *USENIX Security Symposium*, pages 1387–1403, 2018.
70. B Clifford Neuman. Proxy-based authorization and accounting for distributed systems. In *Proceedings, The 13th International Conference on Distributed Computing Systems*, pages 283–291. IEEE, 1993.
71. Lan Nguyen and Rei Safavi-Naini. Dynamic k-times anonymous authentication. In *International Conference on Applied Cryptography and Network Security*, pages 318–333. Springer, 2005.
72. Takashi Nishide. One-time delegation of unlinkable signing rights and its application. In *ProvSec*, pages 103–123. Springer, 2020.
73. Kazuo Ohta and Tatsuaki Okamoto. On concrete security treatment of signatures derived from identification. In *CRYPTO*, pages 354–369. Springer, 1998.
74. Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*, pages 31–53. Springer, 1992.
75. Tatsuaki Okamoto and Kazuo Ohta. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In *Eurocrypt*, pages 134–149. Springer, 1989.
76. Christian Paquin. U-Prove Technology Overview V1.1 (revision 2), 2013. `https://www.microsoft.com/en-us/research/project/u-prove/`.
77. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140. Springer, 1991.
78. David Pointcheval and Olivier Sanders. Short randomizable signatures. In *CT-RSA*, pages 111–126. Springer, 2016.
79. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.
80. Sietse Ringers, Eric Verheul, and Jaap-Henk Hoepman. An efficient self-blindable attribute-based credential scheme. In *Financial Cryptography and Data Security*, pages 3–20. Springer, 2017.
81. Isamu Teranishi, Jun Furukawa, and Kazue Sako. k-times anonymous authentication. In *Asiacrypt*, pages 308–322. Springer, 2004.

82. Isamu Teranishi and Kazue Sako. k-times anonymous authentication with a constant proving cost. In *PKC*, pages 525–542. Springer, 2006.
83. Lianying Zhao, Joseph I Choi, Didem Demirag, Kevin RB Butler, Mohammad Mannan, Erman Ayday, and Jeremy Clark. One-time programs made practical. In *Financial Cryptography and Data Security*. Springer, 2019.

A Proof of Theorem 1

1. `Registration`. On input (REGISTER, *sid*, *pk*) from a party $\mathcal{P}$:
   - Send (REGISTER, *sid*, *pk*) to adversary $\mathcal{A}$. Upon receiving ok from $\mathcal{A}$, and if $sid = \mathcal{P}$, and this is the first request from $\mathcal{P}$, record (*sid*, *pk*).
2. `Retrieve`. Upon receiving a message (RETRIEVE, *sid*) from a party $\mathcal{P}'$:
   - Send (RETRIEVE, *sid*, $\mathcal{P}'$) to $\mathcal{A}$. Upon receiving ok from $\mathcal{A}$:
     - If a record (*sid*, *pk*) exists,
       - then, output (*sid*, *pk*) to $\mathcal{P}'$,
       - else, output (*sid*, $\perp$) to $\mathcal{P}'$.

Fig. 11. Functionality $\mathcal{F}_{\text{CA}}$ for Certificate Authority [38]

The functionalities $\mathcal{F}_{\text{CA}}$, $\mathcal{F}_{\text{AUTH}}$ are given in Fig. 11, 13. To prove the security of the construction in Sect. 3.1, we now proceed to the proof of Theorem 1.

Proof. We give the proof by constructing a simulator $\mathcal{S}$ that runs $\mathcal{A}$ and can generate an indistinguishable view for $\mathcal{A}$ in the ideal world. We note that we can assume the probablity that collisions occur for RO calls $H(\cdot)$ or $\mathcal{A}$ can predict correct outputs of $H(\cdot)$ without calling $H(\cdot)$ is negligible because of the random choice by $\mathcal{S}$ controlling $H(\cdot)$.

First we consider the case where $\mathcal{A}$ corrupts only a subset of $\{\mathsf{S}_j\}_{1\leq j\leq m}$. In this case, $\mathcal{S}$ needs to do the following:

- By playing a role of honest G_{otp}, $\mathcal{S}$ gives corrupted S_j a data tuple including the shares and data to compute MAC tags without knowing r_1, r_2, x_1, x_2.
- By playing a role of honest E_{otp}, $\mathcal{S}$ sends the share retrieval request to and obtains the computed share and MAC tag from S_j.
- According to the computed share and MAC tag corrupted S_j responded with, $\mathcal{S}$ interacts with $\mathcal{F}_{\text{otp}}^{\text{BOS}}$ by using the interface OTP-SH-PROC.

The details of how $\mathcal{S}$ works are given in Fig. 12. Because additive sharings are used, the simulation by $\mathcal{S}$ is indistinguishable from the real view. Also the probability that $\mathcal{A}$ (corrupted S_j) can send the forged share which can be accepted by $\mathcal{S}$ is negligible ($1/p$) because of the information-theoretic security of authenticated secret sharing, so the simulation by $\mathcal{S}$ is indistinguishable.

Next we consider the case where $\mathcal{A}$ corrupts E_{otp}. $\mathcal{S}$ works as in Fig. 14, which is based on the intuition that in the additive sharing, the shares can be

Simulator $\mathcal{S}$ for $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$ (case for uncorrupted $\mathsf{E}_{\mathrm{otp}}$)

// $\mathcal{A}$ receives share retrieval requests from $\mathsf{E}_{\mathrm{otp}}$.

1. $\mathcal{S}$ runs $\mathcal{A}$, and generates public key pairs of honest $\mathsf{G}_{\mathrm{otp}}$, $\mathsf{E}_{\mathrm{otp}}$, and S_j's. and receives, as $\mathcal{F}_{\mathrm{CA}}$, public keys of corrupted S_j's from $\mathcal{A}$.
2. Upon receiving ($\mathsf{OTP\text{-}GEN\text{-}LEAK1}$, *sid*, $\mathsf{G}_{\mathrm{otp}}$, p) from $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$, for each corrupted S_j, $\mathcal{S}$ chooses random $\alpha^{(j)}$, $r'_{1,j}$, $x'_{1,j}$, $r'_{2,j}$, $x'_{2,j} \in \mathbb{Z}_p$ and $h_j \in \{0,1\}^{2\lambda}$, and prepares the following additive sharings at random

$$\alpha^{(j)} = \alpha^{(j)}_{\mathsf{E}_{\mathrm{otp}}} + \alpha^{(j)}_{\mathsf{S}_j} \bmod p,,$$
$$\alpha^{(j)} r'_{i,j} = m^{(j,r'_i)}_{\mathsf{E}_{\mathrm{otp}}} + m^{(j,r'_i)}_{\mathsf{S}_j} \bmod p,\ \alpha^{(j)} x'_{i,j} = m^{(j,x'_i)}_{\mathsf{E}_{\mathrm{otp}}} + m^{(j,x'_i)}_{\mathsf{S}_j} \bmod p \text{ for } i = 1, 2.$$

   $\mathcal{S}$ sends to corrupted S_j

$$(sid, \mathsf{Enc}(pk_{\mathsf{S}_j}, h_j), \mathsf{Enc}(pk_{\mathsf{S}_j}, \langle \alpha^{(j)}_{\mathsf{S}_j}, \{r'_{i,j}, x'_{i,j}, m^{(j,r'_i)}_{\mathsf{S}_j}, m^{(j,x'_i)}_{\mathsf{S}_j}\}_{i=1}^{2} \rangle)).$$

3. Upon receiving ($\mathsf{OTP\text{-}RUN\text{-}LEAK}$, *sid*, $\{qid_j\}_{j=1}^{m}$, $\mathsf{E}_{\mathrm{otp}}$, c', *ath-flg*) from $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$, for each corrupted S_j, $\mathcal{S}$ does the following as $\mathsf{E}_{\mathrm{otp}}$:
   (a) $\mathcal{S}$ chooses random $ath_j \in \{0,1\}^{2\lambda}$ and if *ath-flg* = pwdok, $\mathcal{S}$ programs the random oracle such that $ath_j = H(sid \parallel qid_j \parallel h_j)$. $\mathcal{S}$ sends (sid, qid_j, $\mathsf{Enc}(pk_{\mathsf{S}_j}, pk_{\mathsf{E}_{\mathrm{otp}}}, c', ath_j)$) to S_j.
   (b) $\mathcal{S}$ waits for the response (sid, qid_j, $\mathsf{Enc}(pk_{\mathsf{E}_{\mathrm{otp}}}, \{rslt_i\}_{i=1}^{2})$) from corrupted S_j. If $rslt_i = \langle z'_{i,j}, v_{i,\mathsf{S}_j} \rangle$ for $i = 1, 2$, $\mathcal{S}$ computes

$$v^{(j)}_{i,\mathsf{E}_{\mathrm{otp}}} = \alpha^{(j)}_{\mathsf{E}_{\mathrm{otp}}} z'_{i,j} - (m^{(j,r'_i)}_{\mathsf{E}_{\mathrm{otp}}} - c' \cdot m^{(j,x'_i)}_{\mathsf{E}_{\mathrm{otp}}}) \bmod p,$$

   and verifies that $v^{(j)}_{i,\mathsf{E}_{\mathrm{otp}}} + v_{i,\mathsf{S}_j} = 0 \bmod p$,
   (c) If $rslt_i = \perp$ or the verification fails, $\mathcal{S}$ sets *sh-flg*$_j \leftarrow \perp$, and otherwise *sh-flg*$_j \leftarrow \mathsf{correct\text{-}sh}$.
4. If all *sh-flg*$_j$'s of corrupted S_j are $\mathsf{correct\text{-}sh}$, $\mathcal{S}$ sets *sh-flg* $\leftarrow$ $\mathsf{correct\text{-}sh}$, and otherwise *sh-flg* $\leftarrow \perp$. $\mathcal{S}$ calls the interface ($\mathsf{OTP\text{-}SH\text{-}PROC}$, *sid*, $\{qid_j\}_{j=1}^{m}$, $\mathsf{E}_{\mathrm{otp}}$, *sh-flg*) of $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$.

Fig. 12. Simulator $\mathcal{S}$ for Proof of Theorem 1 (case for uncorrupted $\mathsf{E}_{\mathrm{otp}}$)

just random except the share retrieved last that must be adjusted according to the output of $\mathcal{F}_{\text{otp}}^{\text{BOS}}$ if $\mathcal{A}$ obtains the correct output. □

1. `Send`. On input (SEND, *sid*, $\mathcal{S}$, $\mathcal{R}$, m) from a party $\mathcal{S}$:
   - Send (SENT, *sid*, $\mathcal{S}$, $\mathcal{R}$, m) to adversary $\mathcal{A}$. Upon receiving ok from $\mathcal{A}$, send (SEND, *sid*, $\mathcal{S}$, m) to $\mathcal{R}$.

Fig. 13. Functionality $\mathcal{F}_{\text{AUTH}}$ for Authenticated Channel [37]

B Proof of Theorem 2

First we recall the syntax of a signature scheme, which consists of the following algorithms.

- $\mathsf{Sig.Setup}(1^\lambda) \to \mathsf{params}$, on a security parameter λ, outputs public parameters params.
- $\mathsf{Sig.KeyGen}(\mathsf{params}) \to (\mathsf{pk}, \mathsf{sk})$ outputs a pair of verification and signing keys (pk, sk). We implicitly assume that sk includes pk, and pk includes params.
- $\mathsf{Sig.Sign}(\mathsf{sk}, m) \to \sigma$ takes a signing key sk and message m, and outputs a signature σ.
- $\mathsf{Sig.Vrfy}(\mathsf{pk}, m, \sigma)$ outputs 1 if σ is a correct signature on m under pk, and otherwise 0.

Next we give the standard security notion for a signature scheme called *existential unforgeability under chosen message attacks* (EUF-CMA).

Definition 1 (EUF-CMA). *A signature scheme is EUF-CMA secure if for every PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins in the following game is negligible. The game is defined between a challenger $\mathcal{C}$ and adversary $\mathcal{A}$.*

1. *(**Setup**) $\mathcal{C}$ runs* Sig.Setup, Sig.KeyGen *to obtain* pk, sk. *$\mathcal{A}$ is given* pk.

[28] Actually $\mathcal{S}$ is given $\mathcal{A}$'s input from the environment $\mathcal{Z}$.

[29] This simulation is justified as follows. Suppose that there exist at least two honest clouds $\mathsf{S}_{j'}$, S_j and that $\mathcal{A}$ queried to S_j last where $\mathcal{A}$ retrieved $z'_{i,j'} = r'_{i,j'} - c'_{j'}x'_{i,j'} \bmod p$, $z'_{i,j} = r'_{i,j} - c'_j x'_{i,j} \bmod p$ with $c'_{j'} \neq c'_j$ from $\mathsf{S}_{j'}$, S_j. When $z'_{i,j'}$ is queried, $x'_{i,j'}$ is not determined yet (i.e., $\mathcal{S}$ can set free variable $x'_{i,j'}$ to be any value later), but $r'_{i,j'}$ is dependent on $x'_{i,j'}$ because $r'_{i,j'} = z'_{i,j'} + c'_{j'}x'_{i,j'}$ with $c'_{j'}$ known to $\mathcal{A}$. Because $z'_{i,j}$ can be expressed as

$$\begin{aligned} z'_{i,j} = r'_{i,j} - c'_j x'_{i,j} &= (r'_i - (\text{other shares}) - r'_{i,j'}) - c'_j(x'_i - (\text{other shares}) - x'_{i,j'}) \\ &= r'_i - (\text{other shares}) - c'_j(x'_i - (\text{other shares})) - z'_{i,j'} + (c'_j - c'_{j'})x'_{i,j'}, \end{aligned}$$

$z'_{i,j}$ can be chosen at random because of the randomness from $x'_{i,j'}$.

Simulator $\mathcal{S}$ for $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$ (case for corrupted $\mathsf{E}_{\mathrm{otp}}$)

// Generating an OTP

1. $\mathcal{S}$ runs $\mathcal{A}$ with input ($\mathsf{OTP\text{-}GENREQ}$, *sid*, $\mathsf{G}_{\mathrm{otp}}$, *pwd*)[28], generates public key pairs of honest $\mathsf{G}_{\mathrm{otp}}$, and S_j's, and receives, as $\mathcal{F}_{\mathrm{CA}}$, public keys of corrupted $\mathsf{E}_{\mathrm{otp}}$ and S_j's from $\mathcal{A}$. $\mathcal{S}$ initializes $\mathcal{Q}_{\mathrm{cor}} \leftarrow \emptyset$.
2. $\mathcal{S}$ receives (*sid*, $\mathsf{Enc}(pk_{\mathsf{G}_{\mathrm{otp}}}, pk_{\mathsf{E}_{\mathrm{otp}}}, \{\mathsf{Enc}(pk_{\mathsf{S}_j}, h_j)\}_{j=1}^{m})$) from $\mathcal{A}$ (note that $\mathcal{S}$ can obtain h_j's of honest S_j's).
3. $\mathcal{S}$ calls the interface ($\mathsf{OTP\text{-}GENREQ}$, *sid*, $\mathsf{G}_{\mathrm{otp}}$, *pwd*) of $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$ where $\mathcal{S}$ sets *pwd* ="1" (we note that $\mathcal{S}$ does not need to extract the password from $\mathcal{A}$).
4. Upon receiving ($\mathsf{OTP\text{-}GEN\text{-}LEAK1}$, *sid*, $\mathsf{G}_{\mathrm{otp}}$, $\mathsf{E}_{\mathrm{otp}}$, $g, h, p, g^{x_1}h^{x_2}, g^{r_1}h^{r_2}$) from $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$, $\mathcal{S}$ chooses random $\{r_i'', x_i''\}_{i=1}^{2}$, $\{\alpha_{\mathsf{E}_{\mathrm{otp}}}^{(j)}\}_{j=1}^{m}$, $\{m_{\mathsf{E}_{\mathrm{otp}}}^{(j,r_i')}, m_{\mathsf{E}_{\mathrm{otp}}}^{(j,x_i')}\}_{i=1,2}^{1\le j\le m}$, and sends (*sid*, $\mathsf{Enc}(pk_{\mathsf{E}_{\mathrm{otp}}}, P)$) to $\mathcal{A}$ (corrupted $\mathsf{E}_{\mathrm{otp}}$) where the simulated OTP P is $P = \langle sid, g, h, g^{x_1}h^{x_2}, p, g^{r_1}h^{r_2}, \{r_i'', x_i''\}_{i=1}^{2}, \{\alpha_{\mathsf{E}_{\mathrm{otp}}}^{(j)}\}_{j=1}^{m}, \{m_{\mathsf{E}_{\mathrm{otp}}}^{(j,r_i')}, m_{\mathsf{E}_{\mathrm{otp}}}^{(j,x_i')}\}_{i=1,2}^{1\le j\le m}\rangle$.
5. Upon receiving ($\mathsf{OTP\text{-}GEN\text{-}LEAK2}$, *sid*, $\mathsf{G}_{\mathrm{otp}}$, p) from $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$, for each corrupted S_j, $\mathcal{S}$ chooses random $\{r_{i,j}', x_{i,j}'\}_{i=1}^{2}$, $\alpha_{\mathsf{S}_j}^{(j)}$ and computes $\{m_{\mathsf{S}_j}^{(j,r_i')}, m_{\mathsf{S}_j}^{(j,x_i')}\}_{i=1}^{2}$ such that
   $\alpha^{(j)} = \alpha_{\mathsf{E}_{\mathrm{otp}}}^{(j)} + \alpha_{\mathsf{S}_j}^{(j)} \bmod p$, $\mathcal{Q}_{\mathrm{cor}} \leftarrow \mathcal{Q}_{\mathrm{cor}} \cup \{(j, \{r_{i,j}', x_{i,j}'\}_{i=1}^{2})\}$,
   $\alpha^{(j)} r_{i,j}' = m_{\mathsf{E}_{\mathrm{otp}}}^{(j,r_i')} + m_{\mathsf{S}_j}^{(j,r_i')} \bmod p$, $\alpha^{(j)} x_{i,j}' = m_{\mathsf{E}_{\mathrm{otp}}}^{(j,x_i')} + m_{\mathsf{S}_j}^{(j,x_i')} \bmod p$ for $i = 1, 2$.
   $\mathcal{S}$ sends S_j, ($sid, \mathsf{Enc}(pk_{\mathsf{S}_j}, h_j), \mathsf{Enc}(pk_{\mathsf{S}_j}, \langle \alpha_{\mathsf{S}_j}^{(j)}, \{r_{i,j}', x_{i,j}', m_{\mathsf{S}_j}^{(j,r_i')}, m_{\mathsf{S}_j}^{(j,x_i')}\}_{i=1}^{2}\rangle)$).
6. $\mathcal{S}$ initializes $\mathcal{H}$ to be a set of indexes of honest S_j's and $\mathcal{Q}_{\mathrm{hon}} \leftarrow \emptyset$.

// $\mathcal{A}$ retrievs the shares from honest S_j's to run an OTP.
// We note that the simulation between corrupted $\mathsf{E}_{\mathrm{otp}}$ and S_j is not necessary.

1. $\mathcal{S}$ runs $\mathcal{A}$ with ($\mathsf{OTP\text{-}RUN}$, *sid*, $\{qid_j\}_{j=1}^{m}$, *pwd'*, c'). Each time $\mathcal{A}$ sends (*sid*, qid_j, $\mathsf{Enc}(pk_{\mathsf{S}_j}, pk_{\mathsf{E}_{\mathrm{otp}}}, c_j', ath_j)$) to honest S_j, $\mathcal{S}$ does the following as S_j:
   if $j \notin \mathcal{H}$ or $ath_j \neq H(sid \parallel qid_j \parallel h_j)$
      // share was already retrieved or password is wrong
      send (*sid*, qid_j, $\mathsf{Enc}(pk_{\mathsf{E}_{\mathrm{otp}}}, \perp)$) to $\mathcal{A}$ and return.
   if $|\mathcal{H}| > 1${
      choose random $\{r_{i,j}', x_{i,j}'\}_{i=1}^{2}$, and compute, for $i = 1, 2$,
      $z_{i,j}' \leftarrow r_{i,j}' - c_j' \cdot x_{i,j}' \bmod p$, $\mathcal{Q}_{\mathrm{hon}} \leftarrow \mathcal{Q}_{\mathrm{hon}} \cup \{(j, c_j', \{r_{i,j}', x_{i,j}'\}_{i=1}^{2})\}$
   } else { // this is the last query to honest S_j (i.e., $\mathcal{A}$ collected all the shares)
      If $\mathcal{A}$ used the same value c' for all c_j''s in $\mathcal{Q}_{\mathrm{hon}}${
         call ($\mathsf{OTP\text{-}RUN}$, *sid*, *qid*, "1", c') and, for $i = 1, 2$, obtain
         $z_i = r_i - c' x_i \bmod p$ from $\mathcal{F}_{\mathrm{otp}}^{\mathrm{BOS}}$.
         compute $z_{i,j}' \leftarrow z_i - (\sum_{\substack{(j', r_{i,j'}', x_{i,j'}') \in \mathcal{Q}_{\mathrm{cor}} \\ (j', r_{i,j'}', x_{i,j'}', *) \in \mathcal{Q}_{\mathrm{hon}}}} (r_{i,j'}' - c' x_{i,j'}')) - (r_i'' - c' x_i'') \bmod p$,
            for $i = 1, 2$ // $z_{i,j}'$ corresponds to $r_{i,j}' - c' x_{i,j}' \bmod p$
      } else { // $\mathcal{S}$ does not need to call ($\mathsf{OTP\text{-}RUN}, \ldots$)
         // This case happens only when at least two honest clouds exist.
         // In this case, $\mathsf{E}_{\mathrm{otp}}$ cannot obtain the output.
         just choose random value for $z_{1,j}', z_{2,j}' \in \mathbb{Z}_p$[29]
      }
   }
   remove j from $\mathcal{H}$.
   $v_{i,\mathsf{S}_j} \leftarrow -(\alpha_{\mathsf{E}_{\mathrm{otp}}}^{(j)} z_{i,j}' - (m_{\mathsf{E}_{\mathrm{otp}}}^{(j,r_i')} - c_j' \cdot m_{\mathsf{E}_{\mathrm{otp}}}^{(j,x_i')})) \bmod p$ for $i = 1, 2$
   send (*sid*, qid_j, $\mathsf{Enc}(pk_{\mathsf{E}_{\mathrm{otp}}}, \{z_{i,j}', v_{i,\mathsf{S}_j}\}_{i=1}^{2})$) to $\mathcal{A}$.

Fig. 14. Simulator $\mathcal{S}$ for Proof of Theorem 1 (case for corrupted $\mathsf{E}_{\mathrm{otp}}$)

2. *(**Queries**) $\mathcal{A}$ adaptively requests signatures on at most q messages $m_1, \ldots, m_q$. $\mathcal{C}$ answers each query by returning $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m_i)$.*
3. *(**Output**) $\mathcal{A}$ eventually outputs a message-signature pair (m^*, σ^*) and wins the game if $\mathsf{Sig.Vrfy}(\mathsf{pk}, m^*, \sigma^*) = 1$ and $\forall i \in [1, q], m^* \neq m_i$.*

Further we give the definition of one-more unforgeability for the sequential composition case adapted from [9].

Definition 2 (Sequential One-More Unforgeability for Blind Signature Scheme [9] Adapted to sOTP Setting). *A blind signature scheme (used in the anonymous credential scheme) is one-more unforgeable if for every PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins in the following game is negligible. The game is defined between a challenger $\mathcal{C}$ and adversary $\mathcal{A}$.*

1. *$\mathcal{C}$ runs $\mathsf{params} \leftarrow \mathsf{OAC.Setup}_{\mathrm{otp}}(1^\lambda)$, $(\mathsf{pk}_\mathcal{I}, \mathsf{sk}_\mathcal{I}) \leftarrow \mathsf{OAC.IKeyGen}_{\mathrm{otp}}(\mathsf{params}, w)$.*
2. *$\mathcal{A}$ with $(\mathsf{pk}_\mathcal{I}, \mathsf{params})$ engages in polynomially many adaptive, sequential interactive protocols $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$ with $\mathcal{C}$ (signer).*
   Let ℓ be the number of successful executions of $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$.
3. *$\mathcal{A}$ engages in sequential protocols $\mathsf{OAC.CredShow}_{\mathrm{otp}}$ with $\mathcal{C}$. We assume that $\mathcal{A}$ does not break the security of sOTPs.*
   Let ℓ' be the number of successful executions of $\mathsf{OAC.CredShow}_{\mathrm{otp}}$.
   Then we say that $\mathcal{A}$ wins the game if $\ell' > \ell$.

Now we prove Theorem 2.

Proof. One-timeness follows immediately because $\mathcal{U}$ is forced to run the sOTP P to prove the knowledge of $a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}$ in $\mathsf{OAC.CredShow}_{\mathrm{otp}}$. Blindness of $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$ also follows immediately because, as mentioned in [78], the multiple attributes (messages) are information-theoretically hidden from $\mathcal{I}$ in the Pedersen commitment. Regarding unforgeability of $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$, we prove that if $\mathcal{A}$ exists which wins the game in Definition 2, we can construct a simulator $\mathcal{S}$ which uses $\mathcal{A}$ and breaks the security of the underlying non-blind PS multi-message signature scheme [78, Sect. 4.2]. Here $\mathcal{S}$ acts as an adversary in the EUF-CMA game with an external challenger $\mathcal{C}_e$, and acts as a challenger in the one-more unforgeability game with $\mathcal{A}$. The reduction proceeds as follows:

1. $\mathcal{S}$ is given $\mathsf{pk}_\mathcal{I}$ from $\mathcal{C}_e$ in the EUF-CMA game, and sends it to $\mathcal{A}$ in the one-more unforgeability game.
2. When $\mathcal{A}$ engages in $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$ with $\mathcal{S}$, $\mathcal{S}$ extracts $(r, \mathsf{att}_\mathcal{U})$ from the commitment $C = g^r \cdot \prod_{i=1}^{w} Y_i^{a_i}$ by using the knowledge extractor $\mathcal{E}_\Sigma$ [30]. Then $\mathcal{S}$ sends the message $(\mathsf{att}_\mathcal{U}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}})$ to $\mathcal{C}_e$ to obtain a signature σ on $(\mathsf{att}_\mathcal{U}, a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}})$ where $\mathcal{S}$ chooses random $a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}$. From (r, σ), $\mathcal{S}$ can generate a signature σ' on C and returns σ' and the sOTP P including $a_{1,\mathrm{otp}}, a_{2,\mathrm{otp}}$ to $\mathcal{A}$.

[30] This involves rewinding of $\mathcal{A}$ as in many of the existing anonymous credential schemes including [29, 9, 64].

3. Let ℓ and ℓ' be the numbers of $\mathcal{A}$'s successful executions of $\mathsf{OAC.CredIssue}_{\mathrm{otp}}$ and $\mathsf{OAC.CredShow}_{\mathrm{otp}}$ respectively. If $\ell' > \ell$, at least one of ℓ' executions of $\mathsf{OAC.CredShow}_{\mathrm{otp}}$ results from the forgery, so $\mathcal{S}$ guesses such $i\,(1 \leq i \leq \ell')$ at random, and extracts $(t', \mathsf{att}'_{\mathcal{U}}, a'_{1,\mathrm{otp}}, a'_{2,\mathrm{otp}})$ from the i-th proof of knowledge of a signature by using $\mathcal{E}_{\Sigma}$ again [31].
4. Because of witness indistinguishability of OS signatures[32], the extracted $(t', \mathsf{att}'_{\mathcal{U}}, a'_{1,\mathrm{otp}}, a'_{2,\mathrm{otp}})$ is a new message (attributes) with non-negligible probability (i.e., $\mathcal{S}$ did not obtain the signature on the message from $\mathcal{C}_e$ in the EUF-CMA game), and it means that $\mathcal{S}$ could forge a new pair of a message and signature, and wins the EUF-CMA game. This contradicts the fact that the underlying non-blind PS multi-message signature scheme is secure. □

C Proof of E-Cash Scheme

We prove the following thereom.

Theorem 3. *The e-cash scheme in Fig. 8, 9 realizes the following properties in the RO model:* **balance** [33], **anonymity** [34], **identification of double-spenders**, **strong exculpability**, **clearing**.

We mainly follow the descriptions of security properties in [64, 19]. First we give the brief explanations about each security property.

balance: This property ensures that no coalition of users and merchants can spend more e-coins than they withdrew.

anonymity: No coalition of $\mathcal{B}$ and merchants $\{\mathcal{M}_j\}_j$ should be able to distinguish a real execution of Spend protocol from a simulated one where the simulator $\mathcal{S}$ is restrained from accessing the users' secret keys.

identification of double-spenders: This property requires that, given two valid double-spent e-coins, $\mathcal{B}$ should be able to identify the double-spender.

exculpability: This property captures that no coalition of $\mathcal{B}$ and $\{\mathcal{M}_j\}_j$ should be able to frame an honest user $\mathcal{U}$ by producing two valid deposits $\Pi_{\mathsf{ds}} = (\mathsf{dpst}, \mathsf{dpst}')$ such that $\mathsf{EC.Identify}(\mathsf{params}, \mathsf{pk}_{\mathcal{B}}, \Pi_{\mathsf{ds}}) \to \mathsf{pk}'_{\mathcal{U}}$ although $\mathcal{U}$ did not double-spend.

clearing: This property captures that no coalition of $\mathcal{B}$ and $\{\mathcal{U}_i\}_i$ should be able to deposit the e-coin instead of an honest merchant who received the e-coin.

Definition 3 (balance). *An e-cash scheme realizes* **balance** *if, for every PPT $\mathcal{A}$, the probability that $\mathcal{A}$ wins in the following game is negligible: (here we assume all the users and merchants are corrupted)*

[31] $\mathcal{S}$ also controls clouds for sOTPs.

[32] This is why we need "Okamoto-Schnorr" instead of plain "Schnorr".

[33] This corresponds to the unforgeability property of e-coins.

[34] If $\mathcal{B}$ corrupts part of clouds, a coalition of $\mathcal{B}$ and merchants may be able to violate anonymity by knowing when a user accessed the clouds and ran an OTP, and this seems inevitable, so we will need to assume that clouds do not collude with $\mathcal{B}$.

1. *The challenger* $\mathcal{C}$ *runs* $\mathsf{params} \leftarrow \mathsf{EC.Setup}(1^\lambda)$, $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B}) \leftarrow \mathsf{EC.BKeyGen}(\mathsf{params})$. $\mathcal{C}$ *gives* $(\mathsf{params}, \mathsf{pk}_\mathcal{B})$ *to* $\mathcal{A}$ *and initializes the deposit DB and the withdrawal DB* $\mathsf{DB}_{\mathrm{dpst}}, \mathsf{DB}_{\mathrm{wtdr}} \leftarrow \emptyset$.
2. $\mathcal{A}$ *interacts with the following oracles:*
   - $\mathcal{Q}_{\mathsf{wtdr}}(\mathsf{sk}_\mathcal{B})$: $\mathcal{Q}_{\mathsf{wtdr}}$ *acts as* $\mathcal{B}$ *in the* Withdraw *protocol with* $\mathcal{A}$ *having a public key* $\mathsf{pk}_{\mathcal{U}_i}$ [35]*, and invokes* $\mathcal{E}_\Sigma$ *(with rewinding of* $\mathcal{A}$*) to extract the serial number* sn_j*. After each query,* $\mathcal{Q}_{\mathsf{wtdr}}$ *stores* $(\mathsf{pk}_{\mathcal{U}_i}, \mathsf{sn}_j)$ *in* $\mathsf{DB}_{\mathrm{wtdr}}$.
   - $\mathcal{Q}_{\mathsf{dpst}}(\mathsf{pk}_\mathcal{B})$: $\mathcal{Q}_{\mathsf{dpst}}$ *acts as* $\mathcal{B}$ *in the* Deposit *protocol with* $\mathcal{A}$ *acting as a merchant* [36]*. If the protocol was successful,* $\mathcal{Q}_{\mathsf{dpst}}$ *stores the serial number in* $\mathsf{DB}_{\mathrm{dpst}}$.
3. *After polynomially many queries,* $\mathcal{A}$ *wins the game if there exists a serial number* sn *such that* $\mathsf{sn} \in \mathsf{DB}_{\mathrm{dpst}} \wedge \mathsf{sn} \notin \mathsf{DB}_{\mathrm{wtdr}}$.

Definition 4 (anonymity). *An e-cash scheme realizes* **anonymity** *if there exists an efficient simulator* $\mathcal{S} = (\mathsf{EC.SimSetup}, \mathsf{EC.SimSpend})$ *such that every PPT* $\mathcal{A}$ *has negligible advantage in the following game* [37]*. (here we assume* $\mathcal{B}$ *and all the merchants are corrupted)*

1. $\mathcal{C}$ *flips a random coin* $b \leftarrow \{0,1\}$*. If* $b = 0$*,* $\mathcal{C}$ *runs* $\mathsf{params} \leftarrow \mathsf{EC.Setup}(1^\lambda)$ *whereas, if* $b = 1$*, it runs* $\mathsf{params} \leftarrow \mathsf{EC.SimSetup}(1^\lambda)$*.* $\mathcal{C}$ *gives* params *to* $\mathcal{A}$.
2. $\mathcal{A}$ *runs* $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B}) \leftarrow \mathsf{EC.BKeyGen}(\mathsf{params})$*, hands* $\mathsf{pk}_\mathcal{B}$ *to* $\mathcal{C}$*, and starts adaptively invoking the following oracles:*
   - $\mathcal{Q}_{\mathsf{GetUKey}}(i)$: *If no public key has been created for the user* $\mathcal{U}_i$*,* $\mathcal{Q}_{\mathsf{GetUKey}}$ *generates* $(\mathsf{pk}_{\mathcal{U}_i}, \mathsf{pk}'_{\mathcal{U}_i}, \mathsf{sk}_{\mathcal{U}_i}) \leftarrow \mathsf{EC.UKeyGen}(\mathsf{params}, \mathsf{pk}_\mathcal{B})$*, and returns* $(\mathsf{pk}_{\mathcal{U}_i}, \mathsf{pk}'_{\mathcal{U}_i}, \mathsf{PK}\{(\mathsf{sk}_{\mathcal{U}_i}) : \mathsf{pk}_{\mathcal{U}_i} = g_\mathsf{u}^{\mathsf{sk}_{\mathcal{U}_i}} \wedge \mathsf{pk}'_{\mathcal{U}_i} = g_{\mathsf{u}'}^{\mathsf{sk}_{\mathcal{U}_i}}\})$.
   - $\mathcal{Q}_{\mathsf{wtdr}}(\mathsf{pk}_\mathcal{B}, i)$: $\mathcal{Q}_{\mathsf{wtdr}}$ *acts as* $\mathcal{U}_i$ *with* $\mathsf{sk}_{\mathcal{U}_i}$ *in the* Withdraw *protocol with* $\mathcal{A}$ *acting as* $\mathcal{B}$*. We denote by* $\mathsf{co}_{i,j}$ *the* j*-th successful output of* $\mathcal{U}_i$.
   - $\mathcal{Q}_{\mathsf{spnd}}(\mathsf{pk}_\mathcal{B}, i, j, \mathsf{pk}_{\mathcal{M}_k}, \mathsf{info})$: $\mathcal{Q}_{\mathsf{spnd}}$ *checks if* $\mathsf{co}_{i,j}$ *has been issued to* $\mathcal{U}_i$ *by* $\mathcal{B}$ *(i.e.,* $\mathcal{A}$*) via* $\mathcal{Q}_{\mathsf{wtdr}}(\mathsf{pk}_\mathcal{B}, i)$*. If not,* $\mathcal{Q}_{\mathsf{spnd}}$ *outputs* $\perp$*. Otherwise* $\mathcal{Q}_{\mathsf{spnd}}$ *checks if* $\mathsf{co}_{i,j}$ *is already spent. If so,* $\mathcal{Q}_{\mathsf{spnd}}$ *outputs* $\perp$*, and otherwise* $\mathcal{Q}_{\mathsf{spnd}}$ *responds as follows where* $\mathcal{M}_k$, info *are specified by* $\mathcal{A}$*:*
     - *If* $b = 0$*,* $\mathcal{Q}_{\mathsf{spnd}}$ *runs* $\mathsf{EC.Spend}(\mathcal{A}(\mathsf{pk}_\mathcal{B}, \sigma_{i,j}, v, \mathsf{sn}), \mathcal{U}_i(\mathsf{pk}_\mathcal{B}, \mathsf{co}_{i,j}))$ *with* $\mathcal{A}$ *acting as a merchant.*
     - *If* $b = 1$*,* $\mathcal{Q}_{\mathsf{spnd}}$ *runs* $\mathsf{EC.SimSpend}(\mathsf{params}, \mathsf{pk}_\mathcal{B}, v, \mathsf{pk}_{\mathcal{M}_k}, \mathsf{info})$ *without using* $\mathsf{co}_{i,j}$ *except* v *which is already known to* $\mathcal{A}$ *during the* Withdraw *protocol.*
3. *When* $\mathcal{A}$ *halts, it outputs a bit* $b' \in \{0,1\}$ *and wins if* $b' = b$*. Here* $\mathcal{A}$*'s advantage is defined to be* $\mathsf{Adv}^{\mathrm{anon}}_\mathcal{A}(\lambda) = 2|\Pr[b' = b] - 1/2|$.

[35] Here we assume that $\mathcal{A}$ can register its public keys to $\mathcal{B}$ freely.

[36] $\mathcal{A}$ can spend its e-coin with itself and after that, $\mathcal{A}$ can interact with $\mathcal{Q}_{\mathsf{dpst}}$ as a merchant.

[37] In our proof, EC.SimSetup is the same as EC.Setup.

Definition 5 (identification of double-spenders). *An e-cash scheme ensures* **identification of double-spenders** *if, for every PPT $\mathcal{A}$, the following experiment outputs 1 with only negligible probability: (here we assume all the users and merchants are corrupted)*

1. *$\mathcal{C}$ runs* $\mathsf{params} \leftarrow \mathsf{EC.Setup}(1^\lambda)$, $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B}) \leftarrow \mathsf{EC.BKeyGen}(\mathsf{params})$. *$\mathcal{C}$ gives* $(\mathsf{params}, \mathsf{pk}_\mathcal{B})$ *to $\mathcal{A}$, and initializes the withdrawal DB* $\mathsf{DB}_{\mathrm{wtdr}} \leftarrow \emptyset$.
2. *$\mathcal{A}$ is given access to $\mathcal{Q}_{\mathsf{wtdr}}$ as in Definition 3. At each invocation, $\mathcal{Q}_{\mathsf{wtdr}}$ stores the user's public key of the issued e-coin in* $\mathsf{DB}_{\mathrm{wtdr}}$.
3. *After polynomially many queries, $\mathcal{A}$ outputs two valid deposits* $\Pi_{\mathsf{ds}} = (\mathsf{dpst}_1, \mathsf{dpst}_2)$ *having the same serial number for which no cheating of the merchant exists (i.e., two different double-spending challenges are used). The experiment outputs 1 if and only if* $\mathsf{EC.Identify}(\mathsf{params}, \mathsf{pk}_\mathcal{B}, \Pi_{\mathsf{ds}}) \notin \mathsf{DB}_{\mathrm{wtdr}}$.

Definition 6 (weak exculpability). *An e-cash scheme realizes* **weak exculpability** *if every PPT $\mathcal{A}$ has only negligible advantage in the following game: (here we assume $\mathcal{B}$ and all the merchants are corrupted)*

1. *$\mathcal{C}$ runs* $\mathsf{params} \leftarrow \mathsf{EC.Setup}(1^\lambda)$, *gives* params *to $\mathcal{A}$, and initializes the DB of honest users* $\mathsf{DB}_{\mathrm{usr}} \leftarrow \emptyset$.
2. *$\mathcal{A}$ runs* $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B}) \leftarrow \mathsf{EC.BKeyGen}(\mathsf{params})$ *as $\mathcal{B}$, and interacts with the following oracles.*
   - $\underline{\mathcal{Q}_{\mathsf{GetUKey}}(i)}$: *If no public key has been created for the user $\mathcal{U}_i$, $\mathcal{Q}_{\mathsf{GetUKey}}$ generates* $(\mathsf{pk}_{\mathcal{U}_i}, \mathsf{pk}'_{\mathcal{U}_i}, \mathsf{sk}_{\mathcal{U}_i}) \leftarrow \mathsf{EC.UKeyGen}(\mathsf{params}, \mathsf{pk}_\mathcal{B})$, *and returns* $(\mathsf{pk}_{\mathcal{U}_i}, \mathsf{pk}'_{\mathcal{U}_i}, \mathsf{PK}\{(\mathsf{sk}_{\mathcal{U}_i}) : \mathsf{pk}_{\mathcal{U}_i} = g_{\mathsf{u}}^{\mathsf{sk}_{\mathcal{U}_i}} \wedge \mathsf{pk}'_{\mathcal{U}_i} = g_{\mathsf{u}'}^{\mathsf{sk}_{\mathcal{U}_i}}\})$, *which is added to* $\mathsf{DB}_{\mathrm{usr}}$.
   - $\underline{\mathcal{Q}_{\mathsf{cor}}(i)}$: $\mathcal{Q}_{\mathsf{cor}}$ *returns* $\mathsf{sk}_{\mathcal{U}_i}$, *and removes* $\mathsf{pk}_{\mathcal{U}_i}$ *from* $\mathsf{DB}_{\mathrm{usr}}$.
   - $\underline{\mathcal{Q}_{\mathsf{wtdr}}(\mathsf{pk}_\mathcal{B}, i)}$: $\mathcal{Q}_{\mathsf{wtdr}}$ *acts as $\mathcal{U}_i$ with* $\mathsf{sk}_{\mathcal{U}_i}$ *in the* Withdraw *protocol with $\mathcal{A}$ acting as $\mathcal{B}$. We denote by* $\mathsf{co}_{i,j}$ *the j-th successful output of $\mathcal{U}_i$, which is kept secret from $\mathcal{A}$.*
   - $\underline{\mathcal{Q}_{\mathsf{spnd}}(\mathsf{pk}_\mathcal{B}, i, j, \mathsf{pk}_{\mathcal{M}_k}, \mathsf{info})}$: $\mathcal{Q}_{\mathsf{spnd}}$ *checks if* $\mathsf{co}_{i,j}$ *has been issued to* $\mathcal{U}_i$ *by $\mathcal{B}$ (i.e., $\mathcal{A}$) via* $\mathcal{Q}_{\mathsf{wtdr}}(\mathsf{pk}_\mathcal{B}, i)$. *If not, $\mathcal{Q}_{\mathsf{spnd}}$ outputs $\perp$. Otherwise $\mathcal{Q}_{\mathsf{spnd}}$ checks if* $\mathsf{co}_{i,j}$ *is already spent. If so, $\mathcal{Q}_{\mathsf{spnd}}$ outputs $\perp$, and otherwise $\mathcal{Q}_{\mathsf{spnd}}$ runs* $\mathsf{EC.Spend}(\mathcal{A}(\mathsf{pk}_\mathcal{B}, \sigma_{i,j}, v, \mathsf{sn}), \mathcal{U}_i(\mathsf{pk}_\mathcal{B}, \mathsf{co}_{i,j}))$ *with $\mathcal{A}$ acting as a merchant. If the protocol fails, $\mathcal{Q}_{\mathsf{spnd}}$ outputs $\perp$.*
3. *When $\mathcal{A}$ halts, it outputs Π_{ds} consisting of two valid deposits. $\mathcal{A}$ wins the game if* $\mathsf{EC.Identify}(\mathsf{params}, \mathsf{pk}_\mathcal{B}, \Pi_{\mathsf{ds}}) \in \mathsf{DB}_{\mathrm{usr}}$. *Here $\mathcal{A}$'s advantage is defined to be its success probability.*

Definition 7 (clearing, [19]). *An e-cash scheme realizes* **clearing** *if every PPT $\mathcal{A}$ has only negligible advantage in the following game: (here we assume $\mathcal{B}$ and all the users are corrupted)*

1. *$\mathcal{C}$ runs* $\mathsf{params} \leftarrow \mathsf{EC.Setup}(1^\lambda)$, *gives* params *to $\mathcal{A}$, and initializes the DB of honest merchants* $\mathsf{DB}_{\mathrm{mct}} \leftarrow \emptyset$.
2. *$\mathcal{A}$ runs* $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B}) \leftarrow \mathsf{EC.BKeyGen}(\mathsf{params})$ *as $\mathcal{B}$, and interacts with the following oracles.*

- $\underline{\mathcal{Q}_{\mathsf{GetMKey}}(k)}$: *If no signing key pair has been created for the merchant* $\mathcal{M}_k$, $\underline{\mathcal{Q}_{\mathsf{GetMKey}}}$ *generates* $(\mathsf{pk}_{\mathcal{M}_k}, \mathsf{sk}_{\mathcal{M}_k})$, *and returns* $\mathsf{pk}_{\mathcal{M}_k}$, *which is added to* $\mathsf{DB}_{\mathrm{mct}}$.
- $\underline{\mathcal{Q}_{\mathsf{cor}}(k)}$: $\mathcal{Q}_{\mathsf{cor}}$ *returns* $\mathsf{sk}_{\mathcal{M}_k}$, *and removes* $\mathsf{pk}_{\mathcal{M}_k}$ *from* $\mathsf{DB}_{\mathrm{mct}}$.
- $\underline{\mathcal{Q}_{\mathsf{rcv}}(\mathsf{pk}_{\mathcal{B}}, k, i, j)}$: $\mathcal{Q}_{\mathsf{rcv}}$ *acts as* $\mathcal{M}_k$ *with* $\mathsf{sk}_{\mathcal{M}_k}$ *in the* Spend *protocol with* $\mathcal{A}$ *acting as* $\mathcal{U}_i$ *spending the e-coin* $\mathsf{co}_{i,j}$.
- $\underline{\mathcal{Q}_{\mathsf{dpst}}(\mathsf{pk}_{\mathcal{B}}, k, i, j)}$: $\mathcal{Q}_{\mathsf{dpst}}$ *checks if* $\mathsf{co}_{i,j}$ *has been spent via* $\mathcal{Q}_{\mathsf{rcv}}(\mathsf{pk}_{\mathcal{B}}, k, i, j)$. *If not,* $\mathcal{Q}_{\mathsf{dpst}}$ *outputs* $\perp$. *Otherwise* $\mathcal{Q}_{\mathsf{dpst}}$ *checks if* $\mathsf{co}_{i,j}$ *is already deposited. If so,* $\mathcal{Q}_{\mathsf{spnd}}$ *outputs* $\perp$, *and otherwise* $\mathcal{Q}_{\mathsf{dpst}}$ *acts as* $\mathcal{M}_k$ *depositing* $\mathsf{co}_{i,j}$ *with* $\mathsf{sk}_{\mathcal{M}_k}$ *in the* Deposit *protocol with* $\mathcal{A}$ *acting as* $\mathcal{B}$.

3. *When* $\mathcal{A}$ *halts, it outputs a deposit* $(\mathsf{dpst}, \sigma_{\mathcal{M}_k}(\mathsf{dpst}))$. $\mathcal{A}$ *wins the game if the deposit is valid,* $\mathsf{pk}_{\mathcal{M}_k} \in \mathsf{DB}_{\mathrm{mct}}$, *and the deposit is not the output of* $\mathcal{Q}_{\mathsf{dpst}}$. *Here* $\mathcal{A}$*'s advantage is defined to be its success probability.*

Remark 1. Definition 6 protects only honest users (who never double-spend) from being falsely accused. Camenisch et al. [29] define a stronger property called **strong exculpability** where $\mathcal{A}$ can let $\mathcal{Q}_{\mathsf{spnd}}$ double-spend (or over-spend) e-coins. In this case, $\mathcal{A}$ wins the game if it can produce a valid $\Pi_{\mathsf{ds}} = (\mathsf{dpst}_1, \mathsf{dpst}_2)$ whose serial number is not used more than once by $\mathcal{Q}_{\mathsf{spnd}}$.

Now we prove Theorem 3.

Proof (of **balance** *(Definition 3)).* This proof is similar to the unforgeability proof of Theorem 2 where the adversary is used to break the EUF-CMA game. Suppose that there exists $\mathcal{A}$ that wins the **balance** security game with non-negligible probability, then it means that $\mathcal{A}$ can forge a signature against the underlying non-blind PS multi-message signature scheme because it submitted a proof of knowledge of the signature to $\mathcal{Q}_{\mathsf{dpst}}$. We can show there exists a simulator $\mathcal{S}$ (acting as $\mathcal{C}$ in the **balance** security game) using $\mathcal{A}$ that breaks the unforgeability of the underlying signature scheme. I.e., for a forged serial number $\mathsf{sn}_{\mathrm{fg}} \in \mathsf{DB}_{\mathrm{dpst}}$, $\mathcal{S}$ invokes $\mathcal{E}_{\Sigma}$ and extracts the witnesses from $\mathcal{A}$ by controlling the RO and using the standard rewinding technique from [73, 79]. However, this contradicts the security of the underlying PS signature scheme. □

Proof (of **anonymity** *(Definition 4)).* What we need to show here is that there exists a simulator $\mathcal{S}$ that can simulate the SPK in Fig. 10 (which is generated during the Spend protocol) by controlling the RO and using $\mathcal{S}_{\Sigma}$ (Sect. 2) without using the e-coin's secrets except v which is already learned by $\mathcal{A}$ acting as $\mathcal{B}$ during the Withdraw protocol. The SPK is based on the Σ protocol, so this is quite straightforward. $\mathcal{S}$ works with $\mathcal{S}_{\Sigma}$ as follows:

1. Given v, c_{ds}, $\mathcal{S}_{\Sigma}$ chooses random $\sigma_{\mathsf{co}} = (\sigma_1, \sigma_2) \in \mathbb{G}_1^2$ and $\mathsf{sn} \in \mathbb{Z}_p$.
2. $\mathcal{S}_{\Sigma}$ computes

$$\mathsf{Com}_{\mathsf{co}} \leftarrow \frac{e(\sigma_2, \widetilde{g})}{e(\sigma_1,\ \widetilde{X} \cdot \widetilde{g}_{\mathsf{v}}^{v} \cdot \widetilde{g}_{\mathsf{s}}^{\mathsf{sn}})}$$

and chooses random $\mathsf{Com}_{\mathsf{ds}} \in \mathbb{G}_T$.

3. $\mathcal{S}_{\Sigma}$ chooses random c, z_1, z_2, z_3, z_4, $z_5 \in \mathbb{Z}_p$.
4. $\mathcal{S}$ programs the RO as follows.

$$c = H(e(\sigma_1, \widetilde{g})^{z_1} \cdot e(\sigma_1, \widetilde{g}_{\mathsf{u}})^{z_2} \cdot e(\sigma_1, \widetilde{f}_{\mathsf{u}})^{z_3} \cdot e(\sigma_1, \widetilde{g}_{1,\mathrm{otp}})^{z_4} \cdot e(\sigma_1, \widetilde{g}_{2,\mathrm{otp}})^{z_5} \cdot \mathsf{Com}^{c}_{\mathsf{co}} \parallel g_{\mathsf{u}'}^{z_2} \cdot (g_{\mathsf{u}'}^{c_{\mathsf{ds}}})^{z_3} \cdot \mathsf{Com}^{c}_{\mathsf{ds}} \parallel \sigma_{\mathsf{co}} \parallel v \parallel \mathsf{sn} \parallel \mathsf{Com}_{\mathsf{ds}} \parallel \mathsf{pk}_{\mathcal{M}_j} \parallel \mathsf{info}_j).$$

5. $\mathcal{S}$ accesses clouds to run the OTP P although $\mathcal{S}$ actually does not need the output.

The above outputs of $\mathcal{S}$ are indistinguishable from the real ones because σ_1, σ_2 are randomized after they are obtained from $\mathcal{B}$, and further sn, ω_{u} can be chosen at random because they are information-theoretically hidden from $\mathcal{B}$ during the $\mathsf{Withdraw}$ protocol, so $\mathsf{Com}_{\mathsf{ds}}$ can also be random. □

Proof (of **identification of double-spenders** *(Definition 5)).* From the proof of $\mathsf{balance}$, no $\mathcal{A}$ can deposit forged e-coins, so we can assume that both dpst_1 and dpst_2 are generated from legitimately issued e-coins. Further the serial numbers are the sums of random numbers chosen by $\mathcal{B}$ and $\mathcal{A}$, so we can ignore the probability that two different legitimate e-coins have the same serial number. Thus we can assume that dpst_1 and dpst_2 are generated from the same e-coin. Because $\mathcal{A}$ is forced to use different double-spending challenges for $\mathsf{EC.Spend}$ by the RO and the underlying Σ protocol is sound, $\mathsf{EC.Identify}$ can compute the public key of the double-spender from the two double-spender tags with overwhelming probability. □

Proof (of **strong exculpability** *(Definition 6)).* We show if $\mathcal{A}$'s advantage is non-negligible, then we can construct $\mathcal{S}$ that, given g', $g'^{\rho} \in \mathbb{G}_1$, can compute ρ by using $\mathcal{A}$ and controlling the RO. If $\mathcal{A}$ wins the game, there are two cases:

- (Case 1) $\mathcal{A}$ generates one fake deposit which has the same serial number as the deposit generated by an honest user.
- (Case 2) $\mathcal{A}$ generates two fake deposits framing an honest user.

$\mathcal{S}$ works as follows:

1. When $\mathcal{A}$ acting as $\mathcal{B}$ generates $g_{\mathsf{u}'}$ by a hash function, $\mathcal{S}$ returns $g'^{u'}$ with random $u' \in \mathbb{Z}_p$.
2. Let q_{u} be an upper bound on the number of $\mathcal{Q}_{\mathsf{GetUKey}}$ queries. $\mathcal{S}$ chooses $1 \leq i^* \leq q_{\mathsf{u}}$ at random, and answers oracle queries as follows:
   - $\mathcal{Q}_{\mathsf{GetUKey}}(i)$: If $i \neq i^*$, $\mathcal{S}$ responds normally, and otherwise $\mathcal{S}$ chooses $(\mathsf{pk}_{\mathcal{U}_i} = g_{\mathsf{u}}^{\rho'}, \mathsf{pk}'_{\mathcal{U}_i} = (g'^{\rho})^{u'} = g_{\mathsf{u}'}^{\rho})$ where ρ' is random, and simulates a fake PK although $\rho' \neq \rho$, which is indistinguishable from the real one because DDH holds in $\mathbb{G}_1$.
   - $\mathcal{Q}_{\mathsf{cor}}(i)$: If $i \neq i^*$, $\mathcal{S}$ responds normally, and otherwise $\mathcal{S}$ aborts.
   - $\mathcal{Q}_{\mathsf{wtdr}}(\mathsf{pk}_{\mathcal{B}}, i)$: If $i \neq i^*$, $\mathcal{S}$ responds normally, and otherwise $\mathcal{S}$ simulates a fake PK without knowing ρ.

– $\mathcal{Q}_{\mathsf{spnd}}(\mathsf{pk}_{\mathcal{B}}, i, j, \mathsf{pk}_{\mathcal{M}_k}, \mathsf{info})$: If $i \neq i^*$, $\mathcal{S}$ responds normally, and otherwise $\mathcal{S}$ simulates a fake $\overline{\mathsf{SPK}}$ as in the proof of **anonymity** without knowing ρ.

3. $\mathcal{A}$ outputs the SPK with Π_{ds} including the double-spending tags $(\mathsf{Com}_{\mathsf{ds}}, c_{\mathsf{ds}})$, $(\mathsf{Com}'_{\mathsf{ds}}, c'_{\mathsf{ds}})$. If Π_{ds} does not frame $\mathcal{U}_{i^*}$, $\mathcal{S}$ aborts. Otherwise, we have two cases as mentioned earlier:
   – (Case 1) We can assume that

$$\begin{aligned}
\mathsf{Com}_{\mathsf{ds}} &= g_{\mathsf{u}'}^{\rho} \cdot (g_{\mathsf{u}'}^{c_{\mathsf{ds}}})^{\omega_{\mathsf{u}}} && \text{(where } \mathcal{S} \text{ knows only } \omega_{\mathsf{u}}\text{)},\\
\mathsf{Com}'_{\mathsf{ds}} &= g_{\mathsf{u}'}^{\beta'} \cdot (g_{\mathsf{u}'}^{c'_{\mathsf{ds}}})^{\gamma'} && \text{(where } \mathcal{A} \text{ knows } \beta', \gamma'\text{)},\\
\rho &= \beta' + c'_{\mathsf{ds}} \cdot \gamma' - c'_{\mathsf{ds}} \cdot \omega_{\mathsf{u}} && (\because \mathsf{EC.Identify} \text{ outputs } g_{\mathsf{u}'}^{\rho}).
\end{aligned}$$

   Thus by extracting β', γ' from $\mathcal{A}$ with the standard rewinding technique, $\mathcal{S}$ can compute ρ with non-negligible probability.
   – (Case 2) Similarly we can assume that

$$\begin{aligned}
\mathsf{Com}_{\mathsf{ds}} &= g_{\mathsf{u}'}^{\beta} \cdot (g_{\mathsf{u}'}^{c_{\mathsf{ds}}})^{\gamma} && \text{(where } \mathcal{A} \text{ knows } \beta, \gamma\text{)},\\
\mathsf{Com}'_{\mathsf{ds}} &= g_{\mathsf{u}'}^{\beta'} \cdot (g_{\mathsf{u}'}^{c'_{\mathsf{ds}}})^{\gamma'} && \text{(where } \mathcal{A} \text{ knows } \beta', \gamma'\text{)},\\
\rho &= \frac{1}{c'_{\mathsf{ds}} - c_{\mathsf{ds}}}(c'_{\mathsf{ds}} \cdot (\beta + c_{\mathsf{ds}} \cdot \gamma) - c_{\mathsf{ds}} \cdot (\beta' + c'_{\mathsf{ds}} \cdot \gamma')) \\
& && (\because \mathsf{EC.Identify} \text{ outputs } g_{\mathsf{u}'}^{\rho}).
\end{aligned}$$

   Thus by extracting β, γ, β', γ' from $\mathcal{A}$ with the standard rewinding technique, $\mathcal{S}$ can compute ρ with non-negligible probability. □

Proof (of **clearing** *(Definition 7)).* If $\mathcal{A}$'s advantage is non-negligible $\mathsf{non\text{-}neg}$, we show that we can construct a forger $\mathcal{F}$ against the signature scheme used by merchants. Here $\mathcal{F}$, given pk^*, plays the EUF-CMA game with the challenger $\mathcal{C}$, and plays the clearing security game with $\mathcal{A}$.

1. $\mathcal{F}$ chooses $k^* \in [1, q_{\mathsf{m}}]$ at random where q_{m} is an upper bound on the number of $\mathcal{Q}_{\mathsf{GetMKey}}$ queries made by $\mathcal{A}$.
2. To answer a $\mathcal{Q}_{\mathsf{GetMKey}}$ query made by $\mathcal{A}$, $\mathcal{F}$ returns a random signing key pair to $\mathcal{A}$ except $\mathcal{F}$ returns pk^* in response to the k^*-th $\mathcal{Q}_{\mathsf{GetMKey}}$ query.
3. To answer a $\mathcal{Q}_{\mathsf{dpst}}$ query made by $\mathcal{A}$, $\mathcal{F}$ sends a signing query to $\mathcal{C}$ to obtain the corresponding signature under pk^*.
4. At the end of the game, $\mathcal{A}$ outputs a valid $(\mathsf{dpst}, \sigma_{\mathcal{M}_k}(\mathsf{dpst}))$ with probability $\mathsf{non\text{-}neg}$. If $k \neq k^*$, $\mathcal{F}$ aborts. Otherwise, $\mathcal{F}$ forwards $(\mathsf{dpst}, \sigma_{\mathcal{M}_k}(\mathsf{dpst}))$ to $\mathcal{C}$ to win the EUF-CMA game, and the success probability of $\mathcal{F}$ is $\frac{\mathsf{non\text{-}neg}}{q_{\mathsf{m}}}$. □