Structure Preserving CCA Secure Encryption and Its Application to Oblivious Third Parties

Jan Camenisch[1], Kristiyan Haralambiev[2], Markulf Kohlweiss[3], Jorn Lapon[4], and Vincent Naessens[4]

[1] IBM Research
Zürich, Switzerland
jca@zurich.ibm.com
[2] New York University
New York, USA
haralambiev@cs.nyu.edu
[3] Microsoft Research
Cambridge, UK
markulf@microsoft.com
[4] Katholieke Hogeschool Sint-Lieven
Ghent, Belgium
jorn.lapon@kahosl.be
vincent.naessens@kahosl.be

Abstract. In this paper we present the first public key encryption scheme that is structure preserving, i.e., our encryption scheme uses only algebraic operations. In particular it does not use hash-functions or interpret group elements as bit-strings. This makes our scheme a perfect building block for cryptographic protocols where parties for instance want to prove, to each other, properties about ciphertexts or jointly compute ciphertexts. Our scheme is also very efficient and is secure against adaptive chosen ciphertext attacks. We also provide a few example protocols for our scheme. For instance, a joint computation of a ciphertext, generated from two secret plaintexts from each party respectively, where in the end, only one of the parties learns the ciphertext. This latter protocol serves as a building block for our second contribution which is a set of protocols that implement the concept of oblivious trusted third parties. This concept has been proposed before, but no concrete realization was known.

1 Introduction

Public key encryption and signature schemes have become indispensable building blocks for cryptographic protocols such as anonymous credential schemes, group signatures, anonymous voting schemes, and e-cash systems. To serve as building blocks, cryptographic primitives not only need to satisfy strong security requirements but also need to have additional properties that allow for the construction of more complex schemes. For instance, it is often necessary that one party be able to prove to another that it has correctly signed or encrypted a message without revealing the message. While in theory this is possible to do for any signature and encryption scheme, it is in general not efficient at all. Thus, suitable signature and encryption schemes need to allow for doing such proofs efficiently with either so-called generalized Schnorr protocols [1] or Groth-Sahai proofs [2]. Both these proof methods are efficient because they make use of the structure of algebraic groups. Thus, for the design of suitable signature and encryption schemes one should stay within the structure of algebraic groups and for instance not use hash-functions in an essential way.

When it comes to signature schemes, a designer can pick from a number of schemes that are suitable (e.g., [3–5]). Now, for adaptive chosen ciphertext attack (CCA) secure encryption schemes the situation is quite different. Two schemes that are somewhat suitable are the Camenisch-Shoup and the Cramer-Shoup encryption schemes [6, 7], allowing for the verifiable encryption (and decryption) of discrete logarithms and group elements, respectively. Both make use of some sort of cryptographic hash functions to achieve security against chosen ciphertext attacks. These, unfortunately, prevent one from efficiently proving certain relations between their input and output. This, however, is an important feature in more advanced protocols and is required, for instance, when two parties are to jointly compute an encryption of (a function of) their messages without revealing them or when a user is to prove knowledge of a ciphertext, e.g., as a part of a proof of knowledge of a leakage-resilient signature [8, 9] (proving knowledge of a signature is a central tool in privacy-preserving protocols which so far is not possible for leakage-resilient signatures).

In this paper we present the first efficient *structure preserving* CCA secure encryption scheme. The term "structure-preserving" is borrowed from the notion of structure-preserving digital signatures [5]. An encryption scheme is called structure-preserving if its public keys, messages, and ciphertexts are group elements and the encryption and decryption algorithm consists only of group and pairing operations. We achieve structure preserving encryption by a novel way to implement the consistency check that does not leave the realm of algebraic groups. More precisely, we make use of bilinear maps between algebraic groups and we embed a semantically secure encryption scheme in the base group. Our ciphertext consistency element(s) could be either one element in the target group or several pairs of group elements in the base group. The former gives better efficiency, whereas the latter can be used in more scenarios, in particular those making use of Groth-Sahai proofs [2]. We prove our encryption scheme secure under the decisional linear assumption [10].

Our new encryption scheme is well suited to build a variety of protocols. For instance, with our scheme the following protocol problems can be addressed which are common stumbling stones when designing advanced cryptographic protocols:

- Our scheme can be used in the construction of leakage-resilient signatures [9] which will then enable, for the first time, a user to prove knowledge of a leakage-resilient signature (we refer to the full paper for the details on this).
- A user who is given a ciphertext and a Groth-Sahai proof that the ciphertext was correctly computed, is able to prove to a third party that it is in possession of such a ciphertext without revealing it.
- Two users can jointly compute a ciphertext (of a function) of two plaintexts such that neither party learns the plain text of the other party and only one of the parties learns the ciphertext.

The last problem typically appears in protocols that do some kind of conflict resolution via a trusted third party. Examples include anonymity lifting (revocation) in group signatures and in anonymous credential systems [11] and optimistic fair exchange [12]. In these scenarios, there are typically two parties, say Alice and Bob, who run a protocol with each other and then provide each other with a ciphertext that can in case of a mishap (such as abuse of anonymity, conflict, unfair abortion of the protocol, etc.) be presented to a third party for resolution by decryption. Hereby, it is of course important that (1) the trusted third party be involved in case of mishap only and (2) the parties can convince each other that the ciphertext indeed contains the right information. So far, protocol designers have used verifiable encryption, which unfortunately has the disadvantage that both parties learn the ciphertexts of the other party. Hence, Alice could for instance take Bob's ciphertext and bribe the TTP so that it would act normally for all decryption requests except when Bob's ciphertext is presented in which case the TTP would just ignore the request.

To address this problem Camenisch, Gross, and Heydt-Benjamin [13] propose the concept of *oblivious trusted third parties (OTP)*: here, such conflict resolution protocols are designed in such a way that the trusted third party is kept oblivious of the concrete instance of the conflict resolution protocol. This means, in such a protocol, if Bob goes to such a TTP for resolution, the TTP cannot possibly discriminate Bob. Thus, if the TTP denies to co-operate too often it will be noted and otherwise there is no reason for Bob to believe that the TTP will not resolve the conflict for him if need be. Unfortunately, Camenisch et al. only provide a high-level construction for such a protocol but do not present a concrete instantiation. Their construction has a number of limitations, e.g., the TTP is required to be online during user enrollment, and in fact it is unclear whether a full realization of their ambitious program is possible along the lines they propose. In particular, our realization relies crucially on the CCA security of the encryption scheme, as the TTP essentially acts as a decryption oracle.

As our second contribution, we present the first concrete protocols that implement OTP based on our new encryption scheme. In our solution, the TTP becomes oblivious: Alice will not get any information about Bob's ciphertext nor about the plaintext contained in it and hence cannot provide the TTP with any information that it could use to distinguish the request of Bob from any other request. We achieve this by (1) a joint computation of the ciphertext such that Alice does not see Bob's ciphertext and (2) by ensuring that what gets encrypted is Alice's plaintext input, masked by a blinding factor only known to Bob. Thus, decryption will reveal only a random message that only Bob can unmask. Our encryption scheme and protocols also support so-called labels [6] which are public messages attached to a ciphertext and are important in these TTP scenarios to bind a decryption policy to the ciphertext.

We prove all our protocols secure in the so-called IITM simulation-based security model proposed by Küsters [14]. While being in the spirit of the UC framework [15] and the reactive simulatability model [16], it has certain (admittedly subjective) advantages over the UC model, as discussed in [14, 17, 18]. The results presented here would, however, also carry over to the UC model.

Related Work. There is of course a lot of related work on encryption schemes, but our scheme is the first one that is structure preserving. Considering our second contribution, the protocols for oblivious trusted parties, the only related work is by Camenisch, Gross, and Heydt-Benjamin [13]. They introduced the concept of oblivious trusted third parties but as we mentioned, do not provide any concrete protocol.

2 Preliminaries

2.1 Simulatability Model

We use strong simulation-based definitions that guarantee security under composition in the flavor of [15, 16, 14]. In particular we base our exposition on [14]. In [14] both ideal systems $\mathcal{I}$ and their realizations as cryptographic protocols $\mathcal{P}$ are configurations of multi-tape interactive Turing machines (ITMs). In particular, the framework of [14] guarantees that well-formed systems of polynomial time bounded ITMs can be simulated by a single ITM. This allows us to interpret an ideal system and a protocol either as an interconnected system that communicate via input/output tape pairs shared between component ITMs, or as a single ITM that manages all external tapes. We consider only such well-formed systems.

An ITM is triggered by another ITM if the latter writes a message on an output tape that corresponds to an input tape of the former. An ITM with the input tape named start for auxiliary input a, is called master ITM. It is the first ITM triggered, and is triggered if no other ITM was triggered in the last run. The master ITM can also write a binary decision on a $\mathsf{decision}$ tape at the end of a systems execution. Another special tape contains the security parameter k, is read only, and accessible by all ITMs. As a convention we bundle communication tapes into interfaces inf where an interface consists of named input/output tape pairs. An input/output tape pair is named $\mathsf{inf.R}$ after a combination of the interface name inf and a role name R. We refer to the set of all roles of an interface as $\mathsf{inf}.\mathcal{R}$. If a system of ITMs implementing an interface inf, is connected to another ITM M then as a convention, we refer to the swapped input/output tape pair of M connected to role R as $\overline{\mathsf{inf}}.\mathsf{R}$.

For simulation-based security definitions, the ideal system $\mathcal{I}$ and the protocol $\mathcal{P}$ that emulates this ideal system have to present the same application programming interface (API) inf towards their environment, i.e., they must be *API compatible*. We refer to an ideal system and a protocol that is API compatible with respect to interface inf as $\mathcal{I}_{\mathsf{inf}}$ and $\mathcal{P}_{\mathsf{inf}}$ respectively. In addition $\mathcal{I}_{\mathsf{inf}}$ and $\mathcal{P}_{\mathsf{inf}}$ must expose different network interfaces ninf_1 and ninf_2.

Strong simulatability. A proof that $\mathcal{P}_{\mathsf{inf}}$ emulates $\mathcal{I}_{\mathsf{inf}}$, short $\mathcal{P}_{\mathsf{inf}} \leq \mathcal{I}_{\mathsf{inf}}$ will need to prove existance of a simulator Sim that translates between the interfaces ninf_1 and ninf_2. This is formalized as *strong simulatability* which implies other simulatability notions such as dummy universal composability and blackbox simulatability.

We recall two definitions in [14]. The notion of negligible function is standard and follows [15, 14].

Definition 1. *Two systems $\mathcal{P}$ and $\mathcal{Q}$ are called indistinguishable ($\mathcal{P} \approx \mathcal{Q}$)* iff *the function*

$$f(1^k, a) = |\Pr[\mathcal{P}(1^k, a) = 1] - \Pr[\mathcal{Q}(1^k, a) = 1]| \qquad \textit{is negligible.}$$

Definition 2. *A protocol system $\mathcal{P}_{\mathsf{inf}}$ strongly emulates $\mathcal{I}_{\mathsf{inf}}$, short $\mathcal{P}_{\mathsf{inf}} \leq^{SS} \mathcal{I}_{\mathsf{inf}}$,* iff *there exists a simulator* Sim *connected to* Env *on interface* ninf_2 *and $\mathcal{I}_{\mathsf{inf}}$ on interface* ninf_1 *such that for all master ITMs* Env *(the environment) that connect to* inf *and* ninf_2*:* $\mathsf{Env}|\mathcal{P}_{\mathsf{inf}} \approx \mathsf{Env}|\mathsf{Sim}|\mathcal{I}_{\mathsf{inf}}$

Corruption. Like [17], we use a standard corruption model for ITMs. We consider only static corruption. A corrupted role forwards all inputs to $\mathsf{ninf}_i.\mathsf{R}$ and acts as a proxy that allows the environment to send messages to any of its other connected tapes by sending control messages to $\overline{\mathsf{ninf}_i}.\mathsf{R}$.

We consider ideal systems that are fully described by a virtual incorruptible party $\mathcal{F}_{\mathsf{inf}}$. As the functionality $\mathcal{F}_{\mathsf{inf}}$ implements the security critical parts of an ideal system, the ITM's representing the different roles of the interface only need to implement forwarding and corruption. We refer to such a dummy party for role R as $\mathcal{D}_{\mathsf{R}}$.

When operating over an adversarially controlled network, even an ideal cryptographic system cannot prevent denial of service attacks. We model delayed communication between an ideal functionality and a dummy by an ITM that sits between $\mathcal{F}_{\mathsf{inf}}$ and $\mathcal{D}_{\mathsf{R}}$ and only passes output to $\mathcal{D}_{\mathsf{R}}$ once it leaked the type and length of the message to ninf_1 and was approved by a continue message over ninf_1. As a convention, inputs to $\mathcal{F}_{\mathsf{inf}}$ do not leak and do not get delayed.

Secure Channels, multi-session versions, and joint resources. To abstract from communication details in the real world, we model communication as ideal systems. The ideal channel $\mathcal{I}_{\mathsf{sc}}$ supports only request/response communication and only a single message can be sent at a time.[5] We model corruption of sender and receiver through dummy users $\mathcal{D}_{\mathsf{S}_1}$ and $\mathcal{D}_{\mathsf{S}_2}$: $\mathcal{I}_{\mathsf{sc}} = \mathcal{D}_{\mathsf{S}_1}|\mathcal{F}_{\mathsf{sc}}|\mathcal{D}_{\mathsf{S}_2}$.

Küsters [14] describes how to turn every system $\mathcal{S}$ of ITM's into a multi session system $\underline{\mathcal{S}}$, by programming each ITM instance to accept only messages prefixed with a specific session id, and adding the same session id to all outputs produced by that instance. This is denoted by the session operator $\underline{\ \ }$. For polynomially many sessions the composition theorem guarantees that given $\mathcal{P}_{\mathsf{inf}} \leq^{SS} \mathcal{I}_{\mathsf{inf}}$, $!\underline{\mathcal{P}_{\mathsf{inf}}} \leq^{SS} !\underline{\mathcal{I}_{\mathsf{inf}}}$. Informally, the bang operator '!' denotes on demand creation of session specific instances.

The default way of obtaining a multi-session version of a protocol by the bang and session operator requires a fresh copy of all ITMs in a system for every session. However, the sessions of a protocol can often make use of joint resources. For an adequate joint-state realization $\mathsf{P}_1|\mathcal{I}_{\mathsf{sc}}|\mathsf{P}_2|\mathcal{I}_{\mathsf{crs}}$ of $\mathcal{P}_{\mathsf{inf}}$ that, for instance, makes use of a common reference string functionality[6], we can write $!\underline{\mathsf{P}_1|\mathcal{I}_{\mathsf{sc}}|\mathsf{P}_2}|\mathcal{I}_{\mathsf{crs}} \leq^{SS} !\underline{\mathcal{I}_{\mathsf{inf}}}$. For further information on the joint state theorem for the model of [14] we refer to [17].

Our construction of oblivious third parties operates in the $\mathcal{I}_{\mathsf{reg}}$-hybrid model [15], where parties register their public keys at a trusted registration entity. Below we depict the ideal system $\mathcal{I}_{\mathsf{reg}}$, which is parameterized with a set of participants $\mathcal{R}$ that in our case is restricted to contain U, SP, SA and RA only.[7]

[5] See Listing 17 in Appendix A for the details of the $\mathcal{I}_{\mathsf{sc}}$ functionality.
[6] See Listing 20 in Appendix A for the details of the $\mathcal{I}_{\mathsf{crs}}$ functionality.
[7] See Listing 21 in Appendix A for the details of the $\mathcal{I}_{\mathsf{reg}}$ functionality.

2.2 Practical Zero-Knowledge Proof of Knowledge Protocols

For the types of relations required in our protocols, there exist practical ZK protocols — indeed, our protocols were designed with such protocols specifically in mind. We refer to [19] for details.

We will be proving statements of the form

$$\text{⋊}\, w_1, \ldots, w_n : \phi(w_1, \ldots, w_n, \mathit{bases}). \tag{1}$$

Here, we use the symbol "⋊" instead of "$\exists$" to indicate that we are proving "knowledge" of a witness, rather than just its existence. ϕ is a predicate — we will presently place restrictions on the form of the domains and the predicate. A witness for a statement of the form (1) is a tuple $(w_1, \ldots, w_n)$ of integers such that $\phi(w_1, \ldots, w_n, \mathit{bases})$ holds. In cases where only the residue class of w_i modulo m is important, we may treat the domain of w_i as $\mathbb{Z}_m$.

The predicate $\phi(w_1, \ldots, w_n, \mathit{bases})$ is given by a formula that is built up from "atoms" using arbitrary combinations of ANDs and ORs. An atom may express several types of relations among the w_i's: *(i) integer relations*, such as $\mathcal{F} = 0$, $\mathcal{F} \geq 0$, $\mathcal{F} \equiv 0 \pmod{m}$, or $\gcd(\mathcal{F}, m) = 1$, where $\mathcal{F}$ is an integer polynomial in the variables $w_1, \ldots, w_n$, and m is a positive integer; *(ii) group relations*, such as $\prod_{j=1}^{k} g_j^{\mathcal{F}_j} = 1$, where the $g_j \in \mathit{bases}$ are elements of an abelian group, and the $\mathcal{F}_j$'s are integer polynomials in the variables $w_1, \ldots, w_n$.

We define the proof instance *inst* to consist of the set of *bases* and of a descriptions of the abelian groups. The proof relation $((w_1, \ldots, w_n), \mathit{inst}) \in \mathfrak{R}$ holds *iff* the predicate $\phi(w_1, \ldots, w_n, \mathit{bases})$ holds. We call a relation $\mathfrak{R}$ tractable, if such a predicate ϕ and consequently an efficient proof protocol for it, exists.

Camenisch et al. [19, 20] show how to construct efficient protocols for these types of statements that, under reasonable assumptions, multi-realize an ideal functionality with joint access to a common reference string. The computational complexity of these proof systems can be easily related to the arithmetic circuit complexity of the polynomials that appear in the description of ϕ: the number of exponentiations is proportional to the sum of the circuit complexities. We now describe an ideal functionality that fits into our definitional framework.

Listing 1 Functionality $\mathcal{F}_{\mathsf{zk}}(\mathfrak{R})$:

$\mathcal{F}_{\mathsf{zk}}$ receives input from $\mathcal{D}_{\mathsf{Pv}}$ over $\mathcal{F}_{\mathsf{zk}}.\mathsf{Pv}$ and provides output to $\mathcal{D}_{\mathsf{Vf}}$ through the delayed communication tape $\mathcal{F}_{\mathsf{zk}}.\mathsf{Vf}$. Variable *state* is initialized to "ready".

On $(\mathtt{Prove}, \mathit{inst}, \mathit{wit})$ from $\mathcal{F}_{\mathsf{zk}}.\mathsf{Pv}$ where *state* = "ready" and $(\mathit{inst}, \mathit{wit}) \in \mathfrak{R}$

– let *state* = "final"; send $(\mathtt{Prove}, \mathit{inst})$ to $\mathcal{F}_{\mathsf{zk}}.\mathsf{Vf}$

Listing 1 is, essentially, a simplification of the $\mathcal{F}_{ZK}^{R,R'}$ functionality of [19] for which we consider only static corruption, fixed size proofs for a particular relation, and languages with $R = R'$. This allows us to reuse their ZK protocol compiler to obtain efficient multi-session instantiations $\mathcal{P}_{\mathsf{zk}}$ of $\mathcal{I}_{\mathsf{zk}}(\mathfrak{R})$ in the hybrid $\mathcal{I}_{\mathsf{sc}}$ and joint-state $\mathcal{I}_{\mathsf{crs}}$model. More formally, $\mathcal{P}_{\mathsf{zk}} = \mathsf{Pv}|\mathcal{I}_{\mathsf{sc}}|\mathsf{Vf}|\mathcal{I}_{\mathsf{crs}}$, $\mathcal{I}_{\mathsf{zk}}(\mathfrak{R}) = \mathcal{D}_{\mathsf{Pv}}|\mathcal{F}_{\mathsf{zk}}(\mathfrak{R})|\mathcal{D}_{\mathsf{Vf}}$, and $\underline{!\mathsf{Pv}|\mathcal{I}_{\mathsf{sc}}|\mathsf{Vf}}|\mathcal{I}_{\mathsf{crs}} \leq^{SS} \underline{!\mathcal{I}_{\mathsf{zk}}(\mathfrak{R})}$.

3 Structure Preserving Encryption and Secure Joint Ciphertext Computation

In this section, we define structure preserving encryption and present the first instantiation of such a scheme. Next, we build a secure joint ciphertext computation based on this encryption scheme. We work in a group $\mathbb{G}$ of prime order q generated by g and equipped with a non-degenerate efficiently computable bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

3.1 Structure-Preserving Encryption

The term "structure-preserving" is borrowed from the notion of structure-preserving digital signatures [5]. It represents the idea that ciphertexts are constructed purely using group and (optionally) bilinear map operations in a generic way. This is important for protocols such as our joint ciphertext computation protocol that require not to reveal the ciphertext and yet need to provide evidence (using zero-knowledge proofs) of certain properties about the structure of the ciphertext or it's components.

Definition 3. *Structure Preserving Encryption.* A structure-preserving encryption scheme *has public keys, messages, and ciphertexts that consist entirely of group elements. Moreover, the encryption and decryption algorithm perform only group and bilinear map operations.*

Note that the well known Cramer-Shoup [21, 7] and Camenisch-Shoup [6] encryption schemes are not structure preserving, as they make use of a cryptographic hash function. Even the hash-free variant of Cramer-Shoup is not structure preserving, because its consistency check requires group elements to be interpreted as exponents, which is not a group operation. The details of a proof of knowledge of a hash-free ciphertext would depend on the group's internal structure, e.g., it might be based on so called double-discrete logarithm proofs [22], which are bit-wise and thus much less efficient than standard discrete logarithm representation proofs.

Construction. Recall the well-known DLIN assumption [10]:

Definition 4. *Decisional Linear Assumption (DLIN). Let $\mathbb{G}$ be a group of prime order q. For randomly chosen $g_1, g_2, g_3 \leftarrow \mathbb{G}$ and $r, s, t \leftarrow \mathbb{Z}_q$, the following two distributions are computationally indistinguishable:*

$$(\mathbb{G}, g_1, g_2, g_3, g_1^r, g_2^s, g_3^t) \approx (\mathbb{G}, g_1, g_2, g_3, g_1^r, g_2^s, g_3^{r+s}) \ .$$

We construct a structure-preserving CCA encryption scheme secure under DLIN. For simplicity, we describe the scheme when encrypting a message that is a single group element in $\mathbb{G}$, but it is easily extended to encrypt vectors of group elements. The scheme shares some similarities with the hash-free Cramer-Shoup encryption and with the Linear Cramer-Shoup encryption described by Shacham [23], the security of which is also based on DLIN, but relies crucially on the use of a hash function (hence, not structure-preserving).

Our scheme also supports labels. We consider the case when a label L is a single group element, but the scheme extends trivially for the case of a label which is a vector of group elements. Also, labels from the space $\{0, 1\}^*$ could be hashed to one or several group elements. Hence, labels could be any bit string.

- $\mathsf{KeyGen}(1^\lambda)$: Choose random group generators $g_1, g_2, g_3 \leftarrow \mathbb{G}^*$. For randomly chosen $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_q^3$, set $h_1 = g_1^{\alpha_1} g_3^{\alpha_3}$ and $h_2 = g_2^{\alpha_2} g_3^{\alpha_3}$. Then, select $\boldsymbol{\beta}_0, \ldots, \boldsymbol{\beta}_5 \leftarrow \mathbb{Z}_q^3$, and compute $f_{i,1} = g_1^{\beta_{i,1}} g_3^{\beta_{i,3}}$, $f_{i,2} = g_2^{\beta_{i,2}} g_3^{\beta_{i,3}}$, for $i = 0, \ldots, 5$. Output $\mathsf{pk} = (g_1, g_2, g_3, h_1, h_2, \{f_{i,1}, f_{i,2}\}_{i=0}^5)$ and $\mathsf{sk} = (\boldsymbol{\alpha}, \{\boldsymbol{\beta}\}_{i=0}^5)$.
- $\mathsf{Enc}(\mathsf{pk}, L, m)$: To encrypt a message m with a label L, choose random $r, s \leftarrow \mathbb{Z}_q$ and set

$$u_1 = g_1^r,\ u_2 = g_2^s,\ u_3 = g_3^{r+s}, \quad c = m \cdot h_1^r h_2^s, \quad v = \prod_{i=0}^{3} \hat{e}(f_{i,1}^r f_{i,2}^s, u_i) \cdot \hat{e}(f_{4,1}^r f_{4,2}^s, c) \cdot \hat{e}(f_{5,1}^r f_{5,2}^s, L),$$

  where $u_0 = g$. Output $\mathfrak{c} = (u_1, u_2, u_3, c, v)$.
- $\mathsf{Dec}(sk, L, \mathfrak{c})$: Parse $\mathfrak{c}$ as (u_1, u_2, u_3, c, v). Then check whether

$$v \stackrel{?}{=} \prod_{i=0}^{3} \hat{e}(u_1^{\beta_{i,1}} u_2^{\beta_{i,2}} u_3^{\beta_{i,3}}, u_i) \cdot \hat{e}(u_1^{\beta_{4,1}} u_2^{\beta_{4,2}} u_3^{\beta_{4,3}}, c) \cdot \hat{e}(u_1^{\beta_{5,1}} u_2^{\beta_{5,2}} u_3^{\beta_{5,3}}, L),$$

  where $u_0 = g$. If the latter is unsuccessful, reject the ciphertext as invalid.
  Otherwise, output $m = c \cdot (u_1^{\alpha_1} u_2^{\alpha_2} u_3^{\alpha_3})^{-1}$.

If needed, using the pairing randomization techniques of [24], $v \in \mathbb{G}_T$ can be replaced by six random group elements $v_0, \dots, v_5 \in \mathbb{G}$ for which the following equation holds: $v = \prod_{i=0}^{3} \hat{e}(v_i, u_i) \cdot \hat{e}(v_4, c) \cdot \hat{e}(v_5, L)$. This way, the ciphertext would consist only of elements in $\mathbb{G}$.

To observe the correctness of the decryption, note that

$$\begin{aligned} c \cdot (u_1^{\alpha_1} u_2^{\alpha_2} u_3^{\alpha_3})^{-1} &= m \cdot h_1^r h_2^s \cdot \big((g_1^r)^{\alpha_1} (g_2^s)^{\alpha_2} (g_3^{r+s})^{\alpha_3}\big)^{-1} \\ &= m \cdot (g_1^{\alpha_1} g_3^{\alpha_3})^r (g_2^{\alpha_2} g_3^{\alpha_3})^s \cdot \big((g_1^r)^{\alpha_1} (g_2^s)^{\alpha_2} (g_3^{r+s})^{\alpha_3}\big)^{-1} \;=\; m. \end{aligned}$$

The correctness of the validity element v can be verified similarly.

Next, we show the CCA security of the encryption scheme. Our security proof follows the high level idea of the Hash Proof System (HPS) paradigm [25]. Essentially, Lemma 1 says the "proof" π, which is used as a one-time pad for the encryption of the message, has a corresponding HPS which is 1-universal, whereas Lemma 2 shows that the "proof" φ, which constitutes the consistency check element, has a corresponding HPS that is 2-universal. To make the proof below more accessible to readers unfamiliar with the HPS paradigm, we opt for a self-contained proof which can be easily translated into the HPS framework.

Theorem 1. *If DLIN holds, the above public key encryption scheme is secure against chosen-ciphertext attacks (CCA).*

Proof sketch of Theorem 1: We proceed in a sequence of games. We start with a game where the challenger behaves like in the standard IND-CCA game (i.e., the challenge ciphertext is an encryption of m_b, for a randomly chosen bit b, where m_0, m_1 are messages given by the adversary), and end up with a game where the challenge ciphertext is an encryption of a message chosen uniformly at random from the message space. Then we show that all those games are computationally indistinguishable. Let W_i denote the event that the adversary $\mathcal{A}$ outputs b' such that $b = b'$ in Game i.

Game 0. This is the standard IND-CCA game. $Pr[W_0] = \frac{1}{2} + \mathsf{Adv}^{\mathcal{A}}(\lambda)$.

Game 1. For (m_0, m_1, L) chosen by the adversary, the challenge ciphertext $\mathfrak{c} = (\boldsymbol{u}, c, v)$ is computed using the "decryption procedure", i.e., $u_1 = g_1^r$, $u_2 = g_2^s$, $u_3 = g_3^{r+s}$, $c = m_b \cdot u_1^{\alpha_1} u_2^{\alpha_2} u_3^{\alpha_3}$ and $v = \prod_{i=0}^{3} \hat{e}(u_1^{\beta_{i,1}} u_2^{\beta_{i,2}} u_3^{\beta_{i,3}}, u_i) \cdot \hat{e}(u_1^{\beta_{4,1}} u_2^{\beta_{4,2}} u_3^{\beta_{4,3}}, c) \cdot \hat{e}(u_1^{\beta_{5,1}} u_2^{\beta_{5,2}} u_3^{\beta_{5,3}}, L)$. The change is only syntactical, so the two games produce the same distributions. $Pr[W_1] = Pr[W_0]$.

Game 2. The randomness vector $\boldsymbol{u} = (u_1, u_2, u_3)$ of the challenge ciphertext is computed as non-DLIN tuple, i.e., $u_1 = g_1^r$, $u_2 = g_2^s$, $u_3 = g_3^t$ where $r, s, t \leftarrow \mathbb{Z}_q$ and $r + s \neq t$. Game 1 and Game 2 are indistinguishable by DLIN. Therefore, $|\ Pr[W_2] - Pr[W_1]\ | = \text{negl}(\lambda)$.

Game 3. First note that in the previous game, as well as in this one, any decryption query with "correct" ciphertext, i.e., which has a randomness vector a DLIN tuple, yields a unique plaintext. That is, regardless of the concrete choice of sk which matches pk seen by the adversary, such queries do not reveal any information about the secret key.

In this game, unlike the previous one, any decryption query with "malformed" ciphertext, i.e, which has a non-DLIN randomness vector $\widehat{\boldsymbol{u}}$, is rejected. Let's consider two cases:

- $(\widehat{\boldsymbol{u}}, \widehat{c}, \widehat{L}) = (\boldsymbol{u}, c, L)$. Such decryption query is rejected because it is either the challenge ciphertext (when $\widehat{v} = v$) or the verification predicate fails trivially (when $\widehat{v} \neq v$). So, this case is the same in Game 2 and Game 3.
- $(\widehat{\boldsymbol{u}}, \widehat{c}, \widehat{L}) \neq (\boldsymbol{u}, c, L)$. By Lemma 2, such decryption query is rejected in Game 2 with overwhelming probability, whereas in Game 3 it is always rejected.

As the number of decryption queries is polynomial, $|\ Pr[W_3] - Pr[W_2]\ | = \text{negl}(\lambda)$.

Game 4. The challenge ciphertext encrypts a random message from the message space. Game 3 and Game 4 are (information theoretically) indistinguishable by Lemma 1. $Pr[W_4] = Pr[W_3]$.

In the last game, the challenger's choice b is independent from the ciphertext, so $Pr[W_4] = \frac{1}{2}$. Then, by the indistinguishability of the consecutive games $Pr[W_0] = \frac{1}{2} + \mathrm{negl}(\lambda)$, hence $\mathsf{Adv}^{\mathcal{A}}(\lambda) = \mathrm{negl}(\lambda)$. $\square$

Lemma 1 which we used in the above proof says that the one-time pad of the message, when computing the challenge ciphertext in Game 4, can be replaced by a random element. Whereas Lemma 2 shows that any decryption query with "malformed" ciphertext $\widehat{\mathfrak{c}}$ is rejected with overwhelming probability because the adversary $\mathcal{A}$ can hardly do better than guess the correct validity element.

For the formulation and proof of the lemmas, let $g_1, g_2, g_3 \leftarrow \mathbb{G}^*$ and $u_1 = g_1^r$, $u_2 = g_2^s$, $u_3 = g_3^t$, where r, s, t are randomly chosen from $\mathbb{Z}_q$ and $r + s \neq t$. And for convenience, denote $z_1 = \mathrm{dlog}_g(g_1)$, $z_2 = \mathrm{dlog}_g(g_2)$, and $z_3 = \mathrm{dlog}_g(g_3)$.

Lemma 1. *For randomly chosen $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_q^3$, let $h_1 = g_1^{\alpha_1} g_3^{\alpha_3}$, $h_2 = g_2^{\alpha_2} g_3^{\alpha_3}$, and $\pi = u_1^{\alpha_1} u_2^{\alpha_2} u_3^{\alpha_3}$. Then, for a randomly chosen $\psi \leftarrow \mathbb{G}$ it is true that the following distributions are equivalent: $(h_1, h_2, \pi) \equiv (h_1, h_2, \psi)$.*

Proof sketch of Lemma 1: Note that $h_1 = g^{\alpha_1 z_1 + \alpha_3 z_3}$ and $h_2 = g^{\alpha_2 z_2 + \alpha_3 z_3}$. Then, for the tuple (h_1, h_2, π) the following equation holds:

$$\begin{pmatrix} z_1 & 0 & z_3 \\ 0 & z_2 & z_3 \\ rz_1 & sz_2 & tz_3 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} \mathrm{dlog}_g(h_1) \\ \mathrm{dlog}_g(h_2) \\ \mathrm{dlog}_g(\pi) \end{pmatrix}$$

Denote the matrix with M. It has a determinant $\det(M) = z_1 z_2 z_3 (t - r - s)$ which is not equal to 0 due to the choice of the parameters. Therefore the matrix is invertible, and for any $\pi \in \mathbb{G}$, and fixed h_1, h_2, there exists a unique $\boldsymbol{x}$ which yields the tuple (h_1, h_2, π). $\square$

Lemma 2. *Let $\widehat{\boldsymbol{u}} = (\widehat{u}_1, \widehat{u}_2, \widehat{u}_3)$ be any tuple such that $\widehat{u}_1 = g_1^{\widehat{r}}$, $\widehat{u}_2 = g_2^{\widehat{s}}$, and $\widehat{u}_3 = g_3^{\widehat{t}}$, for $\widehat{r} + \widehat{s} \neq \widehat{t}$. And for randomly chosen $\boldsymbol{\beta}_0, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_5 \leftarrow \mathbb{Z}_q^3$, let $f_{i,1} = g_1^{\beta_{i,1}} g_3^{\beta_{i,3}}$, $f_{i,2} = g_2^{\beta_{i,2}} g_3^{\beta_{i,3}}$, for $i = 0, \ldots, 5$. For any $\boldsymbol{m}$ and $\widehat{\boldsymbol{m}}$ in $\mathbb{G}^{*\,5}$, let*

$$\varphi = \prod_{i=0}^{5} \hat{e}(u_1^{\beta_{i,1}} u_2^{\beta_{i,2}} u_3^{\beta_{i,3}}, m_i) \quad \textit{and} \quad \widehat{\varphi} = \prod_{i=0}^{5} \hat{e}((\widehat{u}_1)^{\beta_{i,1}} (\widehat{u}_2)^{\beta_{i,2}} (\widehat{u}_3)^{\beta_{i,3}}, \widehat{m}_i),$$

where $m_0 = \widehat{m}_0 = g$. Then, for any $\boldsymbol{m}$ and $\widehat{\boldsymbol{m}}$, $\boldsymbol{m} \neq \widehat{\boldsymbol{m}}$, it is true that the following two distributions are equivalent: $(\{f_{i,1} f_{i,2}\}_{i=0}^5, \varphi, \widehat{\varphi}) \equiv (\{f_{i,1} f_{i,2}\}_{i=0}^5, \varphi, \psi)$, where $\psi \leftarrow \mathbb{G}_T$ is randomly chosen.

Proof sketch of Lemma 2: Similarly to the proof of the previous lemma, let's define all variables which depend on $\{\boldsymbol{\beta}_i\}_{i=0}^5$ as the result of a constant matrix M multiplied by the vector $(\boldsymbol{\beta}_0^\top || \boldsymbol{\beta}_1^\top || \ldots || \boldsymbol{\beta}_5^\top)^\top$. For convenience, denote with $w_i = \mathrm{dlog}_g(m_i)$ and $\widehat{w}_i = \mathrm{dlog}_g(\widehat{m}_i)$, for $i = 1, \ldots, 5$. Then, we have:

$$\begin{pmatrix} z_1 & 0 & z_3 & - & - & - & \ldots & - & - & - \\ 0 & z_2 & z_3 & - & - & - & \ldots & - & - & - \\ - & - & - & z_1 & 0 & z_3 & \ldots & - & - & - \\ - & - & - & 0 & z_2 & z_3 & \ldots & - & - & - \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ - & - & - & - & - & - & \ldots & z_1 & 0 & z_3 \\ - & - & - & - & - & - & \ldots & 0 & z_2 & z_3 \\ rz_1 & sz_2 & tz_3 & w_1 rz_1 & w_1 sz_2 & w_1 tz_3 & \ldots & w_5 rz_1 & w_5 sz_2 & w_5 tz_3 \\ \widehat{r}z_1 & \widehat{s}z_2 & \widehat{t}z_3 & \widehat{w}_1\widehat{r}z_1 & \widehat{w}_1\widehat{s}z_2 & \widehat{w}_1\widehat{t}z_3 & \ldots & \widehat{w}_5\widehat{r}z_1 & \widehat{w}_5\widehat{s}z_2 & \widehat{w}_5\widehat{t}z_3 \end{pmatrix} \cdot \begin{pmatrix} | \\ \boldsymbol{\beta}_0 \\ | \\ \vdots \\ | \\ \boldsymbol{\beta}_5 \\ | \end{pmatrix} = \begin{pmatrix} \mathrm{dlog}_g(f_{0,1}) \\ \mathrm{dlog}_g(f_{0,2}) \\ \mathrm{dlog}_g(f_{1,1}) \\ \mathrm{dlog}_g(f_{2,2}) \\ \vdots \\ \mathrm{dlog}_g(f_{5,1}) \\ \mathrm{dlog}_g(f_{5,2}) \\ \mathrm{dlog}(\varphi) \\ \mathrm{dlog}(\widehat{\varphi}) \end{pmatrix}.$$

We would like to argue that the rows of the matrix M are linearly independent. As there exists i, $i \geq 1$, such that $m_i \neq \widehat{m}_i$, if we choose the sub-matrix M' consisting of the intersection of the last two rows and rows $1,\ 2,\ 2i+1,\ 2i+2$ with columns $1,\ 2,\ 3,\ 3i+1,\ 3i+2,\ 3i+3$, we get:

$$M' = \begin{pmatrix} z_1 & 0 & z_3 & 0 & 0 & 0 \\ 0 & z_2 & z_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & z_1 & 0 & z_3 \\ 0 & 0 & 0 & 0 & z_2 & z_3 \\ rz_1 & sz_2 & tz_3 & w_i r z_1 & w_i s z_2 & w_i t z_3 \\ \widehat{r}z_1 & \widehat{s}z_2 & \widehat{t}z_3 & \widehat{w}_i \widehat{r} z_1 & \widehat{w}_i \widehat{s} z_2 & \widehat{w}_i \widehat{t} z_3 \end{pmatrix}.$$

If the rows of M are not linearly independent, so are the rows of M'. However, M' has a determinant $\det(M') = \pm z_1^2 z_2^2 z_3^2 (w_i - \widehat{w}_i)(t - r - s)(\widehat{t} - \widehat{r} - \widehat{s})$ which is not equal to 0 due to choice of the parameters. Therefore, the rows of M are linearly independent. □

Our oblivious third party protocol makes use of a multi-message extension of the scheme. It is obtained in a straight-forward manner, and the details are described in Section 3.3.

3.2 Joint Computation of Ciphertext

We consider a two-party protocol for the joint computation of a ciphertext under a third-party public key pk. The encrypted value is a function of two secrets, each of which remains secret from the other protocol participant. We study the case where only the first party learns the ciphertext whereas the second one has no output. Listing 2 describes a general two-party computation for the joint computation of any function f on verifiable inputs inp_1 and inp_2. When performing such a two-party computation, party P_{1+i} is guaranteed that P_{2-i} knows a witness wit_{2-i} for its input inp_{2-i} such that $(inst, (wit_{2-i}, inp_{2-i})) \in \mathfrak{R}_{2-i}$. We restrict ourselves to tractable relations $\mathfrak{R}_i$ for which we can give efficient universally composable proofs of knowledge as described in Section 2.2. We are interested in a secure two party computation where $inp_i = (l_i, \boldsymbol{x_i})$, $pub = \mathsf{pk}$, and f is $f_{\mathsf{JC}}(\mathsf{pk},\ (l_1, \boldsymbol{x}_1),\ (l_2, \boldsymbol{x}_2)\ ) = \mathsf{Enc}(\mathsf{pk}, g^{l_1+l_2}, (g^{x_{1,1}+x_{1,2}}, \ldots, g^{x_{n,1}+x_{n,2}}))$.

Listing 2 Functionality $\mathcal{F}_{\mathsf{tpc}}(f, \mathfrak{R}_1, \mathfrak{R}_2)$

$\mathcal{F}_{\mathsf{tpc}}$ communicates with $\mathcal{D}_{\mathsf{P}_1}$ and $\mathcal{D}_{\mathsf{P}_2}$ through delayed communication tapes $\mathcal{F}_{\mathsf{tpc}}.\mathsf{P}_1$ and $\mathcal{F}_{\mathsf{tpc}}.\mathsf{P}_2$. Variables $inst$, pub, inp_1 store the input of the first party; variable $state$ is initialized to "ready".

On $(\mathtt{Input}_1, inst', pub', wit'_1, inp'_1)$ from $\mathcal{F}_{\mathsf{tpc}}.\mathsf{P}_1$ where $state$ = "ready" and $(inst', (wit'_1, inp'_1)) \in \mathfrak{R}_1$
- let $inp_1 = inp'_1$, $inst = inst'$, $pub = pub'$, and $state$ = "input1"; send $(\mathtt{Input}_1, inst, pub)$ to $\mathcal{F}_{\mathsf{tpc}}.\mathsf{P}_2$

On $(\mathtt{Input}_2, wit_2, inp_2)$ from $\mathcal{F}_{\mathsf{tpc}}.\mathsf{P}_2$ where $state$ = "input1" and $(inst, (wit_2, inp_2))) \in \mathfrak{R}_2$
- let $state$ = "final"; send $(\mathtt{Result}, f(pub, inp_1, inp_2))$ to $\mathcal{F}_{\mathsf{tpc}}.\mathsf{P}_1$

We model an ideal secure two-party computation system $\mathcal{I}_{\mathsf{tpc}}(f, \mathfrak{R}_1, \mathfrak{R}_2)$ with interface tpc as the combination of two dummy Parties $\mathcal{D}_{\mathsf{P}_1}$ and $\mathcal{D}_{\mathsf{P}_2}$ and an ideal two party computation functionality $\mathcal{F}_{\mathsf{tpc}}$: $\mathcal{I}_{\mathsf{tpc}}(f, \mathfrak{R}_1, \mathfrak{R}_2) = \mathcal{D}_{\mathsf{P}_1} | \mathcal{F}_{\mathsf{tpc}}(f, \mathfrak{R}_1, \mathfrak{R}_2) | \mathcal{D}_{\mathsf{P}_2}$.

Implementing $\mathcal{P}_{\mathsf{tpc}}$. We present the protocol for the special case where the jointly computed ciphertext encrypts a single message. This extends trivially in the multi-message case.

The idea of the protocol is as follows. The first party computes a partial and blinded encryption of her secret, she proves that the computation is carried out correctly, and sends the partial encryption to the other party. The second party takes the values from the first flow of the protocol and, using its secret and some randomness, computes a blinded full encryption of the agreed function of the two plaintext contributions. Then, the second party sends these values and proves that they are computed correctly. Finally, the first party unblinds the ciphertext and updates the consistency element to obtain a valid encryption of the function of the two secrets under jointly chosen randomness. The function could be any polynomial of the two secrets; for simplicity, we consider the function $g^{x_1+x_2}$ where g is a fixed group element and x_1, x_2 are the two secrets.

Listing 3 Protocol $\mathcal{P}_{\mathsf{jcc}}(\mathfrak{R}_1,\mathfrak{R}_2) = \mathsf{P}_1(\mathfrak{R}_1,\mathfrak{R}_2)|\mathcal{I}_{\mathsf{zk}_1}(\mathfrak{R}_{\mathsf{P}_1}(\mathfrak{R}_1))|\mathcal{I}_{\mathsf{zk}_2}(\mathfrak{R}_{\mathsf{P}_2}(\mathfrak{R}_2))|\mathsf{P}_2(\mathfrak{R}_1,\mathfrak{R}_2)$

Party P_1 and P_2 receive input from $\mathtt{tpc}.\mathsf{P}_1$ and $\mathtt{tpc}.\mathsf{P}_2$ respectively and communicate over $\mathcal{I}_{\mathsf{zk}_1}$ and $\mathcal{I}_{\mathsf{zk}_2}$.

On $(\mathtt{Input}_1, inst, \mathsf{pk}, wit_1, (l_1, x_1))$ from $\mathtt{tpc}.\mathsf{P}_1$
- if $(inst, (wit_1, l_1, x_1)) \notin \mathfrak{R}_1$ P_1 aborts
- P_1 computes $(msg_1, aux_1) \leftarrow \mathsf{BlindEnc_1}(\mathsf{pk}, l_1, x_1)$ and proves $((msg_1, \mathsf{pk}, inst), (wit_1, l_1, x_1, aux_1)) \in \mathfrak{R}_{\mathsf{P}_1}(\mathfrak{R}_1)$ to P_2 using $\mathcal{I}_{\mathsf{zk}_1}(\mathfrak{R}_{\mathsf{P}_1}(\mathfrak{R}_1))$
- P_2 learns $(msg_1, \mathsf{pk}, inst)$ from $\mathcal{I}_{\mathsf{zk}_1}$ and outputs $(\mathtt{Input}_1, inst, \mathsf{pk})$ to $\mathtt{tpc}.\mathsf{P}_2$

On $(\mathtt{Input}_2, wit_2, (l_2, x_2))$ from $\mathtt{tpc}.\mathsf{P}_2$
- if $(inst, (wit_2, l_2, x_2)) \notin \mathfrak{R}_2$ P_2 aborts
- P_2 runs $(msg_2, aux_2) \leftarrow \mathsf{BlindEnc_2}(\mathsf{pk}, l_2, x_2, msg_1)$
- P_2 proves $((msg_2, \mathsf{pk}, inst), (wit_2, l_2, x_2, aux_2)) \in \mathfrak{R}_{\mathsf{P}_2}(\mathfrak{R}_2)$ to P_1 using $\mathcal{I}_{\mathsf{zk}_2}(\mathfrak{R}_{\mathsf{P}_2}(\mathfrak{R}_2))$
- P_1 computes $\mathfrak{c} \leftarrow \mathsf{UnblindEnc}(\mathsf{pk}, msg_2, aux_1)$
- P_1 outputs $(\mathtt{Result}, \mathfrak{c})$ to $\mathtt{tpc}.\mathsf{P}_1$

We now give the details for the $\mathsf{BlindEnc_1}$, $\mathsf{BlindEnc_2}$, and $\mathsf{UnblindEnc}$ algorithms.

Listing 4 Algorithms of $\mathcal{P}_{\mathsf{tpc}}$

$(msg_1, aux_1) \leftarrow \mathsf{BlindEnc_1}(\mathsf{pk}, l_1, x_1)$
- parse pk as $(g_1, g_2, g_3, h_1, h_2, \{f_{i,1}, f_{i,2}\}_{i=0}^5)$.
- pick $\{\gamma_i\}_{i=1}^5$, $\{\delta_i\}_{i=1}^2$, r_1, and s_1 at random and compute

$$\bar{u}_1' = g^{\gamma_1} \cdot g_1^{r_1}, \quad \bar{u}_2' = g^{\gamma_2} \cdot g_2^{s_1}, \quad \bar{u}_3' = g^{\gamma_3} \cdot g_3^{r_1+s_1}, \quad \bar{u}_4' = g^{\gamma_4} \cdot g^{x_1} \cdot h_1^{r_1} h_2^{s_1}, \quad \bar{u}_5' = g^{\gamma_5} \cdot g^{l_1},$$
$$\bar{v}_1' = \hat{e}(g_1, g^{\delta_1}) \cdot \prod_{i=1} \hat{e}(f_{i,1}, g^{\gamma_i}), \qquad \bar{v}_2' = \hat{e}(g_2, g^{\delta_2}) \cdot \prod_{i=1} \hat{e}(f_{i,2}, g^{\gamma_i}).$$

- output $msg_1 = (\bar{u}_1', \bar{u}_2', \bar{u}_3', \bar{u}_4', \bar{u}_5', \bar{v}_1', \bar{v}_2')$ and $aux_1 = (\{\gamma_i\}_{i=1}^5, \{\delta_i\}_{i=1}^2, r_1, s_1)$.

$(msg_2, aux_2) \leftarrow \mathsf{BlindEnc_2}(\mathsf{pk}, l_2, x_2, msg_1)$
- parse pk as $(g_1, g_2, g_3, h_1, h_2, \{f_{i,1}, f_{i,2}\}_{i=0}^5)$ and msg_1 as $(\bar{u}_1', \bar{u}_2', \bar{u}_3', \bar{u}_4', \bar{u}_5', \bar{v}_1', \bar{v}_2')$.
- pick r_2 and s_2 at random and compute

$$\bar{u}_1 = \bar{u}_1' \cdot g_1^{r_2}, \quad \bar{u}_2 = \bar{u}_2' \cdot g_2^{s_2}, \quad \bar{u}_3 = \bar{u}_3' \cdot g_3^{r_2+s_2}, \quad \bar{u}_4 = \bar{u}_4' \cdot g^{x_2} \cdot h_1^{r_2} h_2^{s_2}, \quad \bar{u}_5 = \bar{u}_5' \cdot g^{l_2},$$
$$\bar{v} = \left(\prod_{i=0} \hat{e}(f_{i,1}, \bar{u}_i)/\bar{v}_1'\right)^{r_2} \cdot \left(\prod_{i=0} \hat{e}(f_{i,2}, \bar{u}_i)/\bar{v}_2'\right)^{s_2},$$

where $\bar{u}_0 = g$.
- output $msg_2 = (\bar{u}_1, \bar{u}_2, \bar{u}_3, \bar{u}_4, \bar{u}_5, \bar{v})$ and $aux_2 = (r_2, s_2)$.

$\mathfrak{c} \leftarrow \mathsf{UnblindEnc}(\mathsf{pk}, msg_2, aux_1)$
- parse pk as $(g_1, g_2, g_3, h_1, h_2, \{f_{i,1}, f_{i,2}\}_{i=0}^5)$, msg_2 as $(\bar{u}_1, \bar{u}_2, \bar{u}_3, \bar{u}_4, \bar{u}_5, \bar{v})$ and $aux_1 = (\{\gamma_i\}_{i=1}^5, \{\delta_i\}_{i=1}^2, r_1, s_1)$.
- compute

$$u_1 = \bar{u}_1/g^{\gamma_1} = g_1^r, \qquad u_2 = \bar{u}_2/g^{\gamma_2} = g_2^s, \qquad u_3 = \bar{u}_3/g^{\gamma_3} = g_3^{r+s},$$
$$u_4 = \bar{u}_4/g^{\gamma_4} = g^{x_1+x_2} \cdot h_1^r h_2^s, \qquad u_5 = \bar{u}_5/g^{\gamma_5} = g^{l_1+l_2},$$
$$v = \bar{v} \cdot \hat{e}(u_1 g_1^{-r_1}, g^{\delta_1}) \cdot \hat{e}(u_2 g_2^{-s_1}, g^{\delta_2}) \cdot \prod_{i=0} \hat{e}(f_{i,1}^{r_1} f_{i,2}^{s_1}, u_i),$$

where $u_0 = g$.
- output $\mathfrak{c} = (u_1, u_2, u_3, u_4, v)$ encrypted with label u_5.

We show how to efficiently prove the relations $\mathfrak{R}_{\mathsf{P}_1}(\mathfrak{R}_1))$ and $\mathfrak{R}_{\mathsf{P}_2}(\mathfrak{R}_2))$ by giving a ꓘ language statement in Listing 8 in Section 4.3.

Correctness. Recall the structure of the ciphertext of the public-key encryption scheme described in Section 3.1: for a public key $\mathsf{pk} = (g_1, g_2, g_3, h_1, h_2, \{f_{i,1}, f_{i,2}\}_{i=0})$, label u_5, and randomly chosen $r, s \leftarrow \mathbb{Z}_q$, the ciphertext is computed as

$$(u_1, u_2, u_3, u_4, v) = \left(g_1^r,\ g_2^s,\ g_3^{r+s},\quad m \cdot h_1^r h_2^s,\quad \prod_{i=0}^{5} \hat{e}(f_{i,1}^r f_{i,2}^s, u_i)\right), \text{where } u_0 = g.$$

Note that the protocol in Listing 3 computes a valid ciphertext because $u_1 = g_1^r$ for $r = r_1 + r_2$, $u_2 = g_2^s$ for $s = s_1 + s_2$, $u_3 = g_3^{r+s}$, $u_4 = m \cdot h_1^r h_2^s$ for $m = g^{x_1+x_2}$, and $v = \prod_{i=0} \hat{e}(f_{i,1}^r f_{i,2}^s, u_i)$. To see v is indeed computed this way, note that:

$$\bar{v} = \left(\prod_{i=0} \hat{e}(f_{i,1}, \bar{u}_i)/\bar{v}_1'\right)^{r_2} \cdot \left(\prod_{i=0} \hat{e}(f_{i,2}, \bar{u}_i)/\bar{v}_2'\right)^{s_2} = \frac{\prod_{i=0} \hat{e}(f_{i,1}^{r_2} f_{i,2}^{s_2}, u_i)}{\hat{e}(g_1, g^{\tau_1})^{r_2} \cdot \hat{e}(g_2, g^{\tau_2})^{s_2}}$$

and

$$\bar{v} \cdot \hat{e}\left(\frac{u_1}{g_1^{r_1}}, g^{\tau_1}\right) \cdot \hat{e}\left(\frac{u_2}{g_2^{s_1}}, g^{\tau_2}\right) = \bar{v} \cdot \hat{e}(g_1^{r_2}, g^{\tau_1}) \cdot \hat{e}(g_2^{s_2}, g^{\tau_2}) = \prod_{i=0} \hat{e}(f_{i,1}^{r_2} f_{i,2}^{s_2}, u_i).$$

Theorem 2. *The joint ciphertext computation protocol (Listing 3) strongly emulates the ideal two-party computation protocol (Listing 2) for function* f_{JC}*:* $\mathcal{P}_{\mathsf{jcc}}(\mathfrak{R}_1, \mathfrak{R}_2) \leq^{SS} \mathcal{I}_{\mathsf{tpc}}(f_{\mathsf{JC}}, \mathfrak{R}_1, \mathfrak{R}_2)$.

Proof sketch of Theorem 2: To prove security of Theorem 2 in Section 3.2, we show that there exists a simulator Sim connected to Env on interface $\mathtt{ntpc}_2$ and to $\mathcal{I}_{\mathsf{tpc}}$ on interface $\mathtt{ntpc}_1$ such that $\mathsf{Env}|\mathcal{P}_{\mathsf{jcc}} \approx \mathsf{Env}|\mathsf{Sim}|\mathcal{I}_{\mathsf{tpc}}$. The main cases to be considered for the security proof are when P_1 is corrupted and P_2 is honest, and vice versa.

For the case when P_1 is corrupted by Env, in the first step Sim receives $\bar{u}_1'$, $\bar{u}_2'$, $\bar{u}_3'$, $\bar{u}_4'$, $\bar{u}_5'$, $\bar{v}_1'$, $\bar{v}_2'$, pk as well as $x_1, l_1, r_1, s_1, \delta_1, \delta_2$ as a part of $(\mathtt{Prove}, (msg_1, \mathsf{pk}, inst), (wit_1, l_1, x_1, aux_1))$ being send to the simulated $\mathcal{I}_{\mathsf{zk}_1}$. Then, Sim sends $(\mathtt{Input}_1, inst, \mathsf{pk}, wit_1, (l_1, x_1))$ to $\mathcal{I}_{\mathsf{tpc}}$ and receives back $(\widehat{u}_1, \widehat{u}_2, \widehat{u}_3, \widehat{u}_4, \widehat{u}_5, \widehat{v})$ which is the ciphertext $(\widehat{u}_1, \widehat{u}_2, \widehat{u}_3, \widehat{u}_4, \widehat{v})$ to be computed at the end by P_1 with a label $\widehat{u}_5$. Using the values $r_1, s_1, r_1, s_1, \delta_1, \delta_2$ obtained earlier, Sim computes:

$$\begin{gathered}
\bar{u}_1 = \widehat{u}_1 \cdot \bar{u}_1'/g_1^{r_1}, \qquad \bar{u}_2 = \widehat{u}_2 \cdot \bar{u}_2'/g_2^{s_1}, \qquad \bar{u}_3 = \widehat{u}_3 \cdot \bar{u}_3'/g_3^{r_1+s_1}, \\
\bar{u}_4 = \widehat{u}_4 \cdot \bar{u}_4'/g_1^{x_1}, \qquad \bar{u}_5 = \widehat{u}_5 \cdot \bar{u}_5'/g^{l_1}, \\
\bar{v} = \widehat{v}\left(\hat{e}\left(u_1 g_1^{-r_1}, g^{\delta_1}\right)\ \hat{e}\left(u_2 g_2^{-s_1}, g^{\delta_2}\right) \textstyle\prod_{i=0} \hat{e}\left(f_{i,1}^{r_1} f_{i,2}^{s_1}, u_i\right)\right)^{-1},
\end{gathered}$$

and sends those to P_1 as part of the instance sent to $\mathcal{I}_{\mathsf{zk}_2}$. Thus, the jointly computed ciphertext obtained by P_1 is the one which was produced by the ideal functionary $\mathcal{I}_{\mathsf{tpc}}$.

In the case when P_2 is corrupt, Sim chooses random $\bar{u}_1', \bar{u}_2', \bar{u}_3', \bar{u}_4', \bar{u}_5', \leftarrow \mathbb{G}$ and $\bar{v}_1', \bar{v}_2' \leftarrow \mathbb{G}_T$, and delivers those to P_2 via $\mathcal{I}_{\mathsf{zk}_1}$. In the next step, Sim receives from P_2 the values $\bar{u}_1, \bar{u}_2, \bar{u}_3, \bar{u}_4, \bar{u}_5, \bar{v}$ as well as x_2, l_2 as a part of the message $(\mathtt{Prove}, (msg_2, \mathsf{pk}, inst), (wit_2, l_2, x_2, aux_2))$ sent to the simulated $\mathcal{I}_{\mathsf{zk}_2}$ by P_2. Finally, Sim submits $(\mathtt{Input}_2, wit_2, (l_2, x_2))$ to $\mathcal{I}_{\mathsf{tpc}}$ and P_1 obtains the correct ciphertext.

For the case when both P_1 and P_2 are honest, simulation is easy due to the use of $\mathcal{I}_{\mathsf{zk}_1}$ and $\mathcal{I}_{\mathsf{zk}_2}$, which only requires Env to receive certain notifications. No meaningful messages have to be exchanged between the two parties as the statements are not revealed to the environment over the network interfaces.

□

3.3 Multi-Message Version.

The scheme presented in Section 3.1 easily extends to encrypt multiple messages at the same time:

- $\mathsf{KeyGen}(1^\lambda)$: Choose random group generators $g_1, g_2, g_3 \leftarrow \mathbb{G}^*$. For randomly chosen $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n \leftarrow \mathbb{Z}_q^3$, set $h_{1,i} = g_1^{\alpha_{i,1}} g_3^{\alpha_{i,3}}$ and $h_{i,2} = g_2^{\alpha_{i,2}} g_3^{\alpha_{i,3}}$, where $i = 1, \ldots, n$. Then, select $\boldsymbol{\beta}_0, \ldots, \boldsymbol{\beta}_{n+4} \leftarrow \mathbb{Z}_q^3$, and compute $f_{i,1} = g_1^{\beta_{i,1}} g_3^{\beta_{i,3}}$ and $f_{i,2} = g_2^{\beta_{i,2}} g_3^{\beta_{i,3}}$, for $i = 0, \ldots, n+4$. Output $\mathsf{pk} = (g_1, g_2, g_3, \{h_{i,1}, h_{i,2}\}_{i=1}^n, \{f_{i,1}, f_{i,2}\}_{i=0}^{n+4})$ and $\mathsf{sk} = (\{\boldsymbol{\alpha}\}_{i=1}^n, \{\boldsymbol{\beta}\}_{i=0}^{n+4})$.

– $\mathsf{Enc}(\mathsf{pk}, L, \boldsymbol{m})$: To encrypt a message vector $\boldsymbol{m}$ with a label L, choose random $r, s \leftarrow \mathbb{Z}_q$ and set

$$u_1 = g_1^r,\ u_2 = g_2^s,\ u_3 = g_3^{r+s}, \qquad c_i = m_i \cdot h_{i,1}^r h_{i,2}^s, \text{ for } i = 1, \ldots, n,$$
$$v = \prod_{i=0}^{3} \hat{e}(f_{i,1}^r f_{i,2}^s, u_i) \cdot \prod_{i=4}^{n+3} \hat{e}(f_{i,1}^r f_{i,2}^s, c_{i-3}) \cdot \hat{e}(f_{(n+4),1}^r f_{(n+4),2}^s, L),$$

where $u_0 = g$. Output $\mathfrak{c} = (u_1, u_2, u_3, \boldsymbol{c}, v)$.

– $\mathsf{Dec}(sk, L, \mathfrak{c})$: Parse $\mathfrak{c}$ as $(u_1, u_2, u_3, \boldsymbol{c}, v)$. Then check whether

$$v \stackrel{?}{=} \prod_{i=0}^{3} \hat{e}(u_1^{\beta_{i,1}} u_2^{\beta_{i,2}} u_3^{\beta_{i,3}}, u_i) \cdot \prod_{i=4}^{n+3} \hat{e}(u_1^{\beta_{i,1}} u_2^{\beta_{i,2}} u_3^{\beta_{i,3}}, c_{i-3}) \cdot \hat{e}(u_1^{\beta_{(n+4),1}} u_2^{\beta_{(n+4),2}} u_3^{\beta_{(n+4),3}}, L),$$

where $u_0 = g$. If the latter is unsuccessful reject the ciphertext as invalid.
Otherwise, compute $m_i = c_i \cdot \left(u_1^{\alpha_{i,1}} u_2^{\alpha_{i,2}} u_3^{\alpha_{i,3}}\right)^{-1}$, for $i = 1, \ldots, n$, and output $\boldsymbol{m}$.

The security proof follows the same steps, with Lemma 1 modified to represent the n times more h-elements and one-time pads, and Lemma 2 adjusted to have $n+5$ rather than 6 pairs of f-elements and corresponding pairing products when computing φ.

4 Oblivious Third Parties

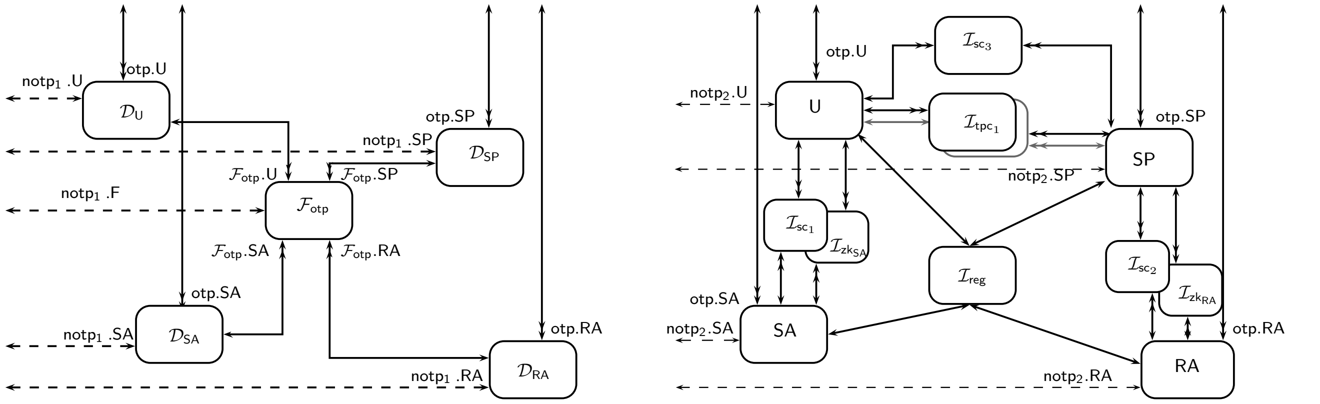
4.1 Modeling oblivious third parties

Transactions in the real world can be intricately related. They may depend on many conditions, of which the verification can be deferred to a number of (as oblivious as possible) third parties. For the sake of concreteness, we now formally model a system that involves two oblivious third parties: a satisfaction authority and a revocation authority. In our example scenario, after a service enrollment between a user U and a service provider SP, the user ought to make a payment for the service before t_{due}. Upon request, the satisfaction authority SA checks that the user indeed made the payment and provides the user with a blinded transaction token. The user unblinds the token and publishes it to prove the satisfaction of the payment. Finally, the revocation authority RA reveals the user's identity to the service provider if no payment has been made before the payment deadline (i.e. no token corresponding to the enrollment was published).

We model the security and privacy requirements of such a system with the help of an ideal functionality $\mathcal{F}_{\mathsf{otp}}$. As usual, corruption is modeled via dummies D_{U}, D_{SP}, D_{SA}, D_{RA} that allow to access the functionality both over the environment interface (before corruption) and the network interface (after corruption).

The Ideal System $\mathcal{I}_{\mathsf{otp}}$. The ideal system $\mathcal{I}_{\mathsf{otp}}$ is depicted in Figure 1(a) and consists of the ideal functionality connected to the dummy parties over delayed communication tapes. The system exports an environment interface named otp with roles $\mathcal{R} = \{\mathsf{U}, \mathsf{SP}, \mathsf{SA}, \mathsf{RA}\}$ and a network interface named $\mathsf{notp_1}$ with roles $\mathcal{R} \cup \{C_{\mathsf{R}}\}_{\mathsf{R} \in \mathcal{R}}$. Roles $\mathsf{C_R}$ are for the delays on the channel, while roles $\mathsf{U}, \mathsf{SP}, \mathsf{SA}, \mathsf{RA}$ allow to corrupt dummy parties and remotely control their behavior.
Listing 5 specifies the reactive behavior of $\mathcal{F}_{\mathsf{otp}}$. A user that can prove his identity with the help of a witness such that $(inst, (id, wit)) \in \mathfrak{R}$, is allowed to enroll. In particular, this interface supports the case where wit and $inst$ are the secrets and the public key of a CL-signature [3] on the user's identity, i.e. an anonymous credential [11, 4], or the opening and a commitment to the user's identity, i.e. a pseudonym [11]. For all these cases, the relation $\mathfrak{R}$ is tractable.

Enrollment consists of three rounds. The first round commits the user to her identity. The second round provides the user with a random satisfaction label with respect to which she can satisfy the condition, e.g. make the necessary payment. In this round the user is also made aware of the due date t_{due} for the payment. Note that the user has to check that t_{due} fulfills reasonable uniformity constraints to protect her



(a) $\mathcal{I}_{\mathsf{otp}} = \mathcal{D}_{\mathsf{U}}|\mathcal{F}_{\mathsf{otp}}|\mathcal{D}_{\mathsf{SP}}|\mathcal{D}_{\mathsf{SA}}|\mathcal{D}_{\mathsf{RA}}$ (b) $\mathcal{P}_{\mathsf{otp}} = \mathsf{SA}|\mathcal{I}_{\mathsf{sc}_1}|\mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}|\mathsf{U}|\mathcal{I}_{\mathsf{sc}_3}|\mathcal{I}_{\mathsf{tpc}_1}|\mathcal{I}_{\mathsf{tpc}_2}|\mathsf{SP}|\mathcal{I}_{\mathsf{sc}_2}|\mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}|\mathsf{RA}|\mathcal{I}_{\mathsf{reg}}$

Fig. 1. The ideal OTP system $\mathcal{I}_{\mathsf{otp}}$ and its realization as a protocol $\mathcal{P}_{\mathsf{otp}}$: The realization makes use of ideal resources $\mathcal{I}_{\mathsf{sc}_i}$, $\mathcal{I}_{\mathsf{zk}_{\mathsf{R}}}$, $\mathcal{I}_{\mathsf{reg}}$, $\mathcal{I}_{\mathsf{jcc}_i}$ for secure communication, proofs of knowledge, key registration, and joint ciphertext computation respectively.

privacy. The last round gives the service provider the possibility to ask the identity revocation authority for the user's identity. As a common limitation with other escrow mechanisms for anonymous credentials, we cannot extract the identity itself, but only the image of a bijection of it. We model this by giving the simulator the possibility to choose the bijection. As the identity space of realistic systems is small enough to allow for exhaustive search, this is not a serious limitation.

The client interface towards the ideal oblivious parties, i.e. the interface of the user and the service provider respectively, consists of two messages `ReqAction` and `TestAction`, with `Action` $\in$ {`Satisfy`, `Open`}. The obliviousness requirement guarantees that oblivious parties do not learn anything about the transactions of their clients. Indeed the decision of an oblivious party cannot be influenced in a transaction specific way, even if the other transaction participant colludes with the oblivious party. This is modeled with the help of test requests that are not related to any transaction. As these requests are indistinguishable from real requests, they allow the user to check whether the oblivious party indeed operates as required. [8]

Consequently, the decision of an oblivious party can only depend on explicit and relevant information. For *satisfaction*, this is the user known satisfaction label L with respect to which she makes her payment. For the *opening*, it is the transaction token T that is secret until after satisfaction, when it is learned by the user. We abstract from the way through which users make T available to the revocation authority, but envision some kind of anonymous publicly available bulletin board. It is in the responsibility of the user to make the token learned during satisfaction available to RA, and in the responsibility of RA to check it's existence. All the protocol guarantees is that RA learns the same T value during *opening* as the user learned during *satisfaction*.

Listing 5 Functionality $\mathcal{F}_{\mathsf{otp}}$

Upon initialization, let $state$ = "ready", $L = T = id = \widehat{T} = \widehat{id} = F = \mathbb{T} = \mathbb{L} = \epsilon$.

On (`SetF`, $F', \mathbb{T}', \mathbb{L}'$) from notp$_1$.F where $state$ = "ready":
- abort if F' is an efficient bijection and $\mathbb{T}'$ and $\mathbb{L}'$ are not of sufficient size;
  set $F = F'$, $\mathbb{T} = \mathbb{T}'$, and $\mathbb{L} = \mathbb{L}'$.

On (`EnrollU`, $inst, (id', wit')$) from $\mathcal{F}_{\mathsf{otp}}$.U where $state$ = "ready":
- if $(inst, (id', wit')) \notin \mathfrak{R}$) abort;
- set $state$ = "enrollu"; set $id = id'$; send (`EnrollU`, $inst$) to $\mathcal{F}_{\mathsf{T}}$.SP.

On (`DeliverEnrollU`, ${t_{due}}'$) from $\mathcal{F}_{\mathsf{T}}$.SP where $state$ = "enrollu":
- set $t_{due} = {t_{due}}'$; set T, L to random values from $\mathbb{T}$ and $\mathbb{L}$ respectively;

[8] An extension that allows not only the requester, but arbitrary external parties, e.g. an auditor, to make test requests is a useful and cryptographically straightforward extension to this interface.

– set $state$ = "deliverenrollu"; send ($\texttt{DeliverEnrollU}, L, t_{due}$) to $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$.

On ($\texttt{DeliverEnrollSP}$) from $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$ where $state$ = "deliverenrollu":
– set $state$ = "enrolled"; send ($\texttt{DeliverEnrollSP}$) to $\mathcal{F}_{\mathsf{otp}}.\mathsf{SP}$.

On ($\texttt{ReqSatisfy}$) from $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$ where $L \neq \epsilon$ and $\widehat{T} = \epsilon$:
– set $\widehat{T} = T$; send ($\texttt{ReqSatisfy}, L$) to $\mathcal{F}_{\mathsf{otp}}.\mathsf{SA}$.

On ($\texttt{TestSatisfy}, L', T'$) from $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$ where $\widehat{T} = \epsilon$:
– set $\widehat{T} = T'$; send ($\texttt{ReqSatisfy}, L'$) to $\mathcal{F}_{\mathsf{otp}}.\mathsf{SA}$.

On ($\texttt{Satisfy}, \mathit{satisfied}$) from $\mathcal{F}_{\mathsf{otp}}.\mathsf{SA}$ where $\widehat{T} \neq \epsilon$:
– if $\mathit{satisfied}$, set $m = (\texttt{Satisfy}, \widehat{T})$, otherwise set $m = (\texttt{Satisfy}, \perp)$; set $\widehat{T} = \epsilon$; send m to $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$.

On ($\texttt{ReqOpen}$) from $\mathcal{F}_{\mathsf{otp}}.\mathsf{SP}$ where $state$ = "enrolled" and $\widehat{id} = \epsilon$:
– set $\widehat{id} = id$; send ($\texttt{ReqOpen}, T, t_{due}$) to $\mathcal{F}_{\mathsf{otp}}.\mathsf{RA}$.

On ($\texttt{TestOpen}, T', id', t_{due}'$) from $\mathcal{F}_{\mathsf{otp}}.\mathsf{SP}$:
– set $\widehat{id} = id'$; send ($\texttt{ReqOpen}, T', t_{due}'$) to $\mathcal{F}_{\mathsf{otp}}.\mathsf{RA}$.

On ($\texttt{Open}, \mathit{open}$) from $\mathcal{F}_{\mathsf{otp}}.\mathsf{RA}$ where $\widehat{id} \neq \epsilon$:
– if open, set $m = (\texttt{Open}, F(\widehat{id}))$, otherwise set $m = (\texttt{Open}, \perp)$; set $\widehat{id} = \epsilon$; send m to $\mathcal{F}_{\mathsf{otp}}.\mathsf{SP}$.

4.2 Implementing oblivious third parties

To construct a protocol that securely emulates the above functionality we make essential use of (adaptive chosen-ciphertext attack secure) encryption. As depicted in Figure 1(b) the protocol makes use of several cryptographic building blocks. But at the core of the protocol are two joint-ciphertext computations that, as described in Section 3.2, can be efficiently realized thanks to structure preserving encryption.

The enrollment protocol has a few more communication rounds, because of the zero-knowledge proofs, but otherwise closely follows the three phases of the ideal system. In the first phase the user commits to and proves her identity. Both the user and the service provider commit to randomness that they will use to jointly compute the transaction token T. The user proves knowledge of the opening of her commitment as part of the joint computation of the satisfaction ciphertext $\mathfrak{c}_1$. In the second phase, the service provider transfers t_{due}, completes the joint ciphertext computation, and starts the computation of the revocation ciphertext $\mathfrak{c}_2$. In both cases, he proves knowledge of the opening to his commitment to guarantee that the transaction token is embedded correctly into both ciphertexts. The user outputs the label of $\mathfrak{c}_1$ as the random satisfaction label L. In the last phase the user again proves knowledge of openings for her commitments in the computation of $\mathfrak{c}_2$ to guarantee that it contains the transaction token T and a blinded user identity g^{id} under label $g^{t_{due}}$.

To satisfy her financial obligations, the user makes a payment with respect to label L and then asks the satisfaction authority to decrypt $\mathfrak{c}_1$. The user receives the blinded transaction token, that she unblinds using her locally stored randomness to learn T. She makes T available to the revocation authority, through some out-of-band anonymous bulletin board mechanism. Test satisfaction requests are just encryptions of blinded T' under label L'. To request the opening of a user identity, the service provider sends the ciphertext $\mathfrak{c}_2$ to the revocation authority, which checks the label t_{due}, decrypts the ciphertext to learn T and verifies whether T was posted by the user. Test opening requests are just encryptions of T' and blinded id' under label t_{due}'.

The Real System $\mathcal{P}_{\mathsf{otp}}$. The real protocol $\mathcal{P}_{\mathsf{otp}}$ implements the same API interface as $\mathcal{I}_{\mathsf{otp}}$ (see Figure 1(b)), but is realized as a distributed cryptographic protocol with parties U, SP, SA, and RA each with their corresponding pairs of API tapes otp.U, otp.SP, otp.SA, otp.RA, towards the environment.

The core security guarantees are achieved through the use of secure two-party computation, secure communication, and zero-knowledge proof protocols. We model secure communication through ideal systems $\mathcal{I}_{\mathsf{sc}_1}$, $\mathcal{I}_{\mathsf{sc}_2}$ and $\mathcal{I}_{\mathsf{sc}_3}$, zero-knowledge proofs through $\mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}$, $\mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}$, and two-party computation through

$\mathcal{I}_{\mathsf{tpc}_1}$, $\mathcal{I}_{\mathsf{tpc}_2}$, which are instances of respectively $\mathcal{I}_{\mathsf{sc}}$, $\mathcal{I}_{\mathsf{zk}}$, and $\mathcal{I}_{\mathsf{tpc}}$ with renamed tapes. Like in the ideal system, we model corruption via an adversarial interface to the protocol parties U, SP, SA, RA that allows to control corrupted parties. During initialization, protocol parties SA and RA register public keys pk_{SA} and pk_{RA} with a key registration authority $\mathcal{I}_{\mathsf{reg}}$.

The real protocol has a few more rounds but follows the same three phases as the ideal system. In the first phase the user commits to and proves her identity. Both the user and service provider commit to the randomness that they will use to compute the revocation token T in commitments $C_{x'_1}$ and Cx_2. The user proves knowledge of the opening of $C_{x'_1}$ as part of the joint computation of the satisfaction ciphertext $\mathfrak{c}_1$. In the second phase, the service provider transfers t_{due}, completes the joint ciphertext computation, and proves that his contribution to the blinded revocation token corresponds to the value in Cx_2. The user outputs the label of this ciphertext as her random satisfaction label. The last phase does a joint ciphertext computation of the revocation token T and the user's identity g^{id} under label $g^{t_{due}}$.

Listing 6 Protocol $\mathcal{P}_{\mathsf{otp}} = \mathsf{SA}|\mathcal{I}_{\mathsf{sc}_1}|\mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}|\mathsf{U}|\mathcal{I}_{\mathsf{sc}_3}|\mathcal{I}_{\mathsf{tpc}_1}|\mathcal{I}_{\mathsf{tpc}_2}|\mathsf{SP}|\mathcal{I}_{\mathsf{sc}_2}|\mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}|\mathsf{RA}|\mathcal{I}_{\mathsf{reg}}$

Upon initialization SA and RA generate keys $(sk_{\mathsf{SA}}, pk_{\mathsf{SA}})$ and $(sk_{\mathsf{RA}}, pk_{\mathsf{RA}})$ for the structure preserving encryption scheme and register pk_{SA} and pk_{RA} with $\mathcal{I}_{\mathsf{reg}}$. U and SP retrieve these keys on demand.

On $(\texttt{EnrollU}, (id, wit), inst)$ from otp.U:

- if $(inst, (id, wit)) \notin \mathfrak{R})$, U aborts, else U generates commitment parameters $params_{\mathsf{U}}$ and sends them to SP over $\mathcal{I}_{\mathsf{sc}_3}$.
- SP receives $params_{\mathsf{U}}$, generates a random $open_{x_2}$, computes $C_{x_2} \leftarrow \mathsf{Commit}(params_{\mathsf{U}}, x_2, open_{x_2})$, generates commitment parameters $params_{\mathsf{SP}}$, and sends C_{x_2}, $params_{\mathsf{SP}}$ to U over $\mathcal{I}_{\mathsf{sc}_3}$.
- U receives C_{x_2} and $params_{\mathsf{SP}}$, generates random $open_{id}$ and $open_{x'_1}$, computes $C_{id} \leftarrow \mathsf{Commit}(params_{\mathsf{SP}}, id, open_{id})$, $C_{x'_1} \leftarrow \mathsf{Commit}(params_{\mathsf{SP}}, x'_1, open_{x'_1})$ and sends C_{id}, $C_{x'_1}$ to SP over $\mathcal{I}_{\mathsf{sc}_3}$.
- SP receives C_{id}, $C_{x'_1}$ and sends an acknowledgement over $\mathcal{I}_{\mathsf{sc}_3}$.
- U sends $(\texttt{Input}_1, (inst, params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}), pk_{\mathsf{SA}}, (id, wit, open_{id}, x'_1, open_{x'_1}), (l_1, x_1))$ to $\mathcal{I}_{\mathsf{tpc}_1}.\mathsf{P}_1$.
- SP receives $(\texttt{Input}_1, (inst, params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}), pk_{\mathsf{SA}})$ on $\mathcal{I}_{\mathsf{tpc}_1}.\mathsf{P}_2$ and sends $(\texttt{EnrollU}, inst)$ to otp.SP.

On $(\texttt{DeliverEnrollU}, t_{due})$ from otp.SP:

- SP sends t_{due} over $\mathcal{I}_{\mathsf{sc}_3}$ and U replies with an acknowledgment.
- SP sends $(\texttt{Input}_2, (open_{x_2}, (l_2, x_2)))$ to $\mathcal{I}_{\mathsf{tpc}_1}.\mathsf{P}_2$.
- U receives $\mathfrak{c}_1 = \mathsf{Enc}(pk_{\mathsf{SA}}, g^{l_1+l_2}, (g^{x_1} \cdot g^{x_2}))$ and $L = g^{l_1+l_2}$ from $\mathcal{I}_{\mathsf{tpc}_1}.\mathsf{P}_1$ and sends $(\texttt{DeliverEnrollU}, L, t_{due})$ to otp.U.

On $(\texttt{DeliverEnrollSP})$ from otp.U:

- U sends an acknowledgment to SP over $\mathcal{I}_{\mathsf{sc}_3}$ and SP sends $(\texttt{Input}_1, (params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}, T_{due}), pk_{\mathsf{RA}}, open_{x_2}, (t_{due}, x_2, x'_2))$ to $\mathcal{I}_{\mathsf{tpc}_2}.\mathsf{P}_1$.
- U receives $(\texttt{Input}_1, (params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}), pk_{\mathsf{RA}})$ from $\mathcal{I}_{\mathsf{tpc}_2}.\mathsf{P}_2$ and sends $(\texttt{Input}_2, (open_{id}, open_{x'_1}), (0, x'_1, id))$ to $\mathcal{I}_{\mathsf{tpc}_2}.\mathsf{P}_2$.
- SP receives $\mathfrak{c}_2$ with $\mathfrak{c}_2 = \mathsf{Enc}(pk_{\mathsf{RA}}, g^{t_{due}}, (g^{x'_1+x_2}, g^{id+x'_2}))$ from $\mathcal{I}_{\mathsf{tpc}_2}.\mathsf{P}_1$ and sends $(\texttt{DeliverEnrollSP})$ to otp.SP.

On $(\texttt{ReqSatisfy})$ from otp.U where $L \neq \epsilon$:

- U sends $(\mathfrak{c}_1, L)$ to SA over $\mathcal{I}_{\mathsf{sc}_1}$.
- SA receives $\mathfrak{c}_1$ from $\mathcal{I}_{\mathsf{sc}_1}$ and if the ciphertext with label L validates correctly, SA sends $(\texttt{ReqSatisfy}, L)$ to otp.SA.

On $(\texttt{TestSatisfy}, \widehat{L}, \widehat{T})$ from otp.U:

- U generates a new ciphertext $\widehat{\mathfrak{c}}_1 = \mathsf{Enc}(pk_{\mathsf{SA}}, \widehat{L}, (\widehat{T} \cdot g^{x_1 - x'_1}))$ with random x_1 and x'_1 and sends $(\widehat{\mathfrak{c}}_1, \widehat{L})$ to SA over $\mathcal{I}_{\mathsf{sc}_1}$.
- SA receives $\widehat{\mathfrak{c}}_1$ from $\mathcal{I}_{\mathsf{sc}_1}$ and if the ciphertext with label $\widehat{L}$ validates correctly, SA sends $(\texttt{ReqSatisfy}, \widehat{L})$ to otp.SA.

On $(\texttt{Satisfy}, \mathit{satisfied})$ from otp.SA

- SA skips a communication round for $\mathcal{I}_{\mathsf{sc}_1}$.
- if *satisfied*, SA decrypts $\mathfrak{c}_1$ and proves correct decryption of the blinded token $m = \mathsf{Dec}(sk_{\mathsf{SA}}, L, \mathfrak{c}_1)$ to U using $\mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}$. otherwise, SA proves $m = \perp$ with an otherwise random instance and witness of correct size to U using $\mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}$.
- U receives m' as the instance of $\mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}$.
- if $m' \neq \perp$, U unblinds $T = m' \cdot g^{x'_1 - x_1} = g^{x'_1 + x_2}$ and sends $(\texttt{Satisfy}, T)$ to otp.U; otherwise U sends $(\texttt{Satisfy}, \perp)$ to otp.U.

On $(\texttt{ReqOpen})$ from otp.SP where $state =$ "enrolled":

- SP sends $(\mathfrak{c}_2, t_{due})$ to RA over $\mathcal{I}_{\mathsf{sc}_3}$.
- RA receives $(\mathfrak{c}_2, t_{due})$ from $\mathcal{I}_{\mathsf{sc}_3}$, decrypts $\mathfrak{c}_2$ under label $g^{t_{due}}$ into $(T, g^{id+x'_2})$, it sends $(\texttt{ReqOpen}, T)$ to otp.SA.

On $(\texttt{TestOpen}, \widehat{T}, \widehat{id}, \widehat{t_{due}})$ from otp.SP:

– SP generates ciphertext $\widehat{\mathfrak{c}}_2 = \mathsf{Enc}(pk_{\mathsf{RA}}, g^{t_{due}}, (\widehat{T}, g^{\widehat{id}+x'_2}))$ with random x'_2 and sends $(\widehat{\mathfrak{c}}_2, \widehat{t_{due}})$ to RA over $\mathcal{I}_{\mathsf{sc}_2}$.
– RA receives $(\widehat{\mathfrak{c}}_2, \widehat{t_{due}})$ from $\mathcal{I}_{\mathsf{sc}_2}$, decrypts the ciphertext under label $g^{\widehat{t_{due}}}$ into $(\widehat{T}, m)$ and sends $(\mathtt{ReqOpen}, \widehat{T})$ to otp.SP.

On $(\mathtt{Open}, open)$ from otp.RA:

– RA skips a communication round for $\mathcal{I}_{\mathsf{sc}_2}$.
– if $open$, RA proves correct decryption of the blinded identity m to SP using $\mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}$. otherwise, RA proves $m = \perp$ with an otherwise random instance and witness of correct size to SP using $\mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}$;
– SP receives m' as the instance of $\mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}$.
– if $m' \neq \perp$, SP unblinds $ID = g^{id+x'_2} \cdot g^{-x'_2} = g^{id}$ and sends $(\mathtt{Open}, ID)$ to otp.SP; otherwise it sends $(\mathtt{Open}, \perp)$ to otp.SP.

The two-party computation $\mathcal{I}_{\mathsf{tpc}_1} = \mathcal{I}_{\mathsf{tpc}}(f_{\mathsf{JC}_1}(pk_{\mathsf{SA}}, (l_1, x_1), (l_2, x_2)), \mathfrak{R}_{1,1}, \mathfrak{R}_{1,2})$ is parameterized by the function f_{JC_1} and two relations $\mathfrak{R}_{1,1}$ and $\mathfrak{R}_{1,2}$ for computing the satisfaction ciphertext $\mathfrak{c}_1$ that contains an encryption of $g^{x_1+x_2}$ under a jointly chosen label $L = g^{l_1+l_2}$:

$$\begin{aligned}\mathfrak{R}_{1,1} = \{((inst, params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}), (id, wit, open_{id}, x'_1, open_{x'_1}, l_1, x_1)) \mid (inst, (id, wit)) \in \mathfrak{R} \wedge \\ C_{id} = \mathsf{Commit}(params_{\mathsf{SP}}, id, open_{id}) \wedge C_{x'_1} = \mathsf{Commit}(params_{\mathsf{SP}}, x'_1, open_{x'_1})\} \text{ and} \\ \mathfrak{R}_{1,2} = \{((inst, params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}), (open_{x_2}, l_2, x_2)) \mid C_{x_2} = \mathsf{Commit}(params_{\mathsf{U}}, x_2, open_{x_2})\} \,.\end{aligned}$$

Similarly, the two-party computation $\mathcal{I}_{\mathsf{tpc}_2} = \mathcal{I}_{\mathsf{tpc}}(f_{\mathsf{JC}_2}(pk_{\mathsf{RA}}, (\epsilon, (x'_1, id)), (t_{due}, (x_2, x'_2)), \mathfrak{R}_{1,1}, \mathfrak{R}_{1,2})$ is parameterized by the function f_{JC_2} and relations $\mathfrak{R}_{1,1}, \mathfrak{R}_{1,2}$ for computing the identity ciphertext $\mathfrak{c}_2$ that contains an encryption of $(g^{x'_1+x_2}, g^{id+x'_2})$ under key pk_{RA} with public label $g^{t_{due}}$:

$$\begin{aligned}\mathfrak{R}_{2,1} = \{((params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}, T_{due}), (open_{x_2}, t_{due}, x_2, x'_2)) \mid \\ C_{x_2} = \mathsf{Commit}(params_{\mathsf{U}}, x_2, open_{x_2}) \wedge T_{due} = g^{t_{due}}\} \,, \\ \mathfrak{R}_{2,2} = \{((params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}, T_{due}), (open_{id}, open_{x'_1}, 0, x'_1, id)) \mid \\ \mathsf{C}_{id} = \mathsf{Commit}(params_{\mathsf{SP}}, id, open_{id}) \wedge C_{x'_1} = \mathsf{Commit}(params_{\mathsf{SP}}, x'_1, open_{x'_1})\} \,.\end{aligned}$$

The commitment scheme can be realized as a simple Pedersen commitment. Given a tractable relation $\mathfrak{R}$ the relations $\mathfrak{R}_{1,1}$, $\mathfrak{R}_{1,2}$, $\mathfrak{R}_{2,1}$, and $\mathfrak{R}_{2,2}$ are themselves tractable.

Satisfaction and opening make use of proofs of correct decryption. In case SA or RA rejects a request by U and SP respectively, we abuse the functionality $\mathcal{I}_{\mathsf{zk}_i}$ as a secure channel, by proving a statement with an arbitrary instance, and witness. We assume that the instance is of the correct size to thwart traffic analysis. The relations $\mathfrak{R}_{\mathsf{SA}}$ and $\mathfrak{R}_{\mathsf{RA}}$ for proving correct decryption are defined as follows:

$$\begin{aligned}\mathfrak{R}_{\mathsf{SA}} = \{((\mathfrak{c}_1, pk_{\mathsf{SA}}, L, m), sk_{\mathsf{SA}}) \mid (m = \mathsf{Dec}(sk_{\mathsf{SA}}, L, \mathfrak{c}_1) \wedge m \neq \perp) \vee m = \perp\} \,, \\ \mathfrak{R}_{\mathsf{RA}} = \{((\mathfrak{c}_2, pk_{\mathsf{RA}}, g^{t_{due}}, m), (sk_{\mathsf{RA}}, T) \mid ((m, T) = \mathsf{Dec}(sk_{\mathsf{RA}}, g^{t_{due}}, \mathfrak{c}_2) \wedge m \neq \perp) \vee m = \perp\} \,,\end{aligned}$$

Listing 7 Efficient realization of $\mathcal{P}_{\mathsf{zk}_{\mathsf{SA}}} \leq^{SS} \mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}(\mathfrak{R}_{\mathsf{SA}})$ and $\mathcal{P}_{\mathsf{zk}_{\mathsf{RA}}} \leq^{SS} \mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}(\mathfrak{R}_{\mathsf{RA}})$

Verifiable Decryption - a proof that $\mathfrak{c} = (u_1, u_2, u_3, \boldsymbol{c}, v)$ decrypts to $\boldsymbol{m}$ for a label L with a secret key $\mathsf{sk} = (\{\boldsymbol{\alpha}\}_{i=1}^{n}, \{\boldsymbol{\beta}\}_{i=0}^{(n+4)})$ corresponding to $\mathsf{pk} = (\{h_{i,1}, h_{i,2}\}_{i=1}^{n}, \{f_{i,1}, f_{i,2}\}_{i=0}^{(n+4)})$.

$$\begin{aligned}\pi = \text{⋊}\, \{\boldsymbol{\alpha}\}_{i=1}^{n}, \{\boldsymbol{\beta}\}_{i=0}^{n+4} \;:\; \{\, h_{i,1} = g_1^{\alpha_{i,1}} g_3^{\alpha_{i,3}} \,\}_{i=1}^{n} \wedge \{\, h_{i,2} = g_2^{\alpha_{i,2}} g_3^{\alpha_{i,3}} \,\}_{i=1}^{n} \wedge \{\, f_{i,1} = g_1^{\beta_{i,1}} g_3^{\beta_{i,3}} \,\}_{i=0}^{(n+4)} \wedge \\ \{\, f_{i,2} = g_2^{\beta_{i,2}} g_3^{\beta_{i,3}} \,\}_{i=0}^{(n+4)} \wedge \{\, m_i = c_i \cdot \prod_{j=1}^{3} u_j^{-\alpha_{i,j}} \,\}_{i=1}^{n} \\ \wedge\; v = \prod_{j=1}^{3} \left(\prod_{i=0}^{3} \hat{e}(u_j, u_i)^{\beta_{i,j}} \cdot \prod_{i=4}^{n+3} \hat{e}(u_j, c_{i-3})^{\beta_{i,j}} \cdot \hat{e}(u_j, L)^{\beta_{(n+4),j}} \right)\end{aligned}$$

Theorem 3. *Given the CCA security of the encryption scheme, our oblivious third party protocol (See Listing 6) strongly emulates the ideal oblivious third party system (See Listing 5):* $\mathcal{P}_{\mathsf{otp}}(\mathfrak{R}) \leq^{SS} \mathcal{I}_{\mathsf{otp}}(\mathfrak{R})$.

A note on using the same group setup. The proofs of Section 2.2 can efficiently deal with different abelian groups. This means that we can compose tractable relations that make use of different group setups and still obtain a tractable relation. This, however, comes with a cost on the performance of the proofs. To achieve optimal performance, parties should use common group parameters as much as possible. Such group parameters need to exist both in the real world and the ideal world, so they can be used by the identity certification system for implementing the relation $(inst, (wit, id)) \in \mathfrak{R}$. Two ways of achieving this are: 1) to describe a deterministic procedure for deriving adequate pairing parameters based on the security parameter alone. 2) use a global setup that exists both in the real world and the ideal world, i.e. we prove $\mathcal{P}_{\mathsf{otp}}(\mathfrak{R})|\mathcal{I}_{\mathsf{crs}} \leq^{SS} \mathcal{I}_{\mathsf{otp}}(\mathfrak{R})|\mathcal{I}_{\mathsf{crs}}$. Where $\mathcal{I}_{\mathsf{crs}}$ only provides a pairing setup. This can be seen as a variant of the GUC model [26]. We note, however, that this $\mathcal{I}_{\mathsf{crs}}$ does not allow us to overcome the impossibility results that have been shown for GUC. We still make use of UC common reference strings for the proofs of knowledge. We leave the construction of an OTP protocol based on an augmented common reference string as further work, but point to [27] as a starting point.

Multi-session version of the protocol. In a realistic deployment, a large number of users will be interacting with a slightly smaller number of service providers, the latter needing to accept multiple enrollment transactions in parallel. Moreover, to achieve real unlinkability between the different transactions of a user, secure channels need to be replaced with secure anonymous channels. The latter require a separation between network identifiers and session identifiers. However, the multi-session functionalities $\underline{\mathcal{I}_{\mathsf{zk}}}$ and $\underline{\mathcal{I}_{\mathsf{tpc}}}$ do not provide anonymity and cannot be realized without $\underline{\mathcal{I}_{\mathsf{sc}}}$ which outputs the same session id/address that it receives as input.

To see that a proof for the single session version of the OTP protocol is sufficient to guarantee the cryptographic property of the multi-session protocol with anonymous channels, we apply the split functionality theorem of [28, 29] that states that for every functionality realizable with authenticated/secure channels, there exists a corresponding split functionality that is realizable with split authenticated/secure channels. Intuitively in the split functionality it is the adversary that in a multi-session version controls which parties communicate together over which functionality. By applying the split functionality theorem and the composition theorem multiple times, a hybrid protocol with multiple split functionalities can be transformed into a protocol, that contains only split secure channels. After proving implicit session disjointness, one can achieve a multi-session version of the OTP protocol that has only local session ids [30].

4.3 Efficient Realization of Zero-Knowledge Proofs

We show how to efficiently prove the relations $\mathfrak{R}_{\mathsf{P}_1}(\mathfrak{R}_1))$ and $\mathfrak{R}_{\mathsf{P}_2}(\mathfrak{R}_2))$. Note that $aux_1 = (\{\gamma_i\}_{i=1}^5, \{\delta_i\}_{i=1}^2, r_1, s_1)$ and $aux_2 = (r_2, s_2)$. We write ϕ_1, ϕ_2, and *bases* to refer to the formulas of the tractable relations $\mathfrak{R}_1$, $\mathfrak{R}_2$ and the bases in *inst* respectively.

Listing 8 Efficient realization of $\mathcal{P}_{\mathsf{zk}_1} \leq^{SS} \mathcal{I}_{\mathsf{zk}_1}(\mathfrak{R}_{\mathsf{P}_1}(\mathfrak{R}_1))$ and $\mathcal{P}_{\mathsf{zk}_2} \leq^{SS} \mathcal{I}_{\mathsf{zk}_2}(\mathfrak{R}_{\mathsf{P}_2}(\mathfrak{R}_2))$

The proofs of correctness are as follows:

$$\pi_1 = \text{\reflectbox{K}}\, wit_1, l_1, x_1, \gamma_1, \ldots, \gamma_5, r_1, s_1, \delta_1, \delta_2 : \phi_1(wit_1, l_1, x_1, bases) \;\wedge\; \bar{u}_1' = g^{\gamma_1} \cdot g_1^{r_1} \;\wedge\; \bar{u}_2' = g^{\gamma_2} \cdot g_2^{s_1} \;\wedge$$
$$\bar{u}_3' = g^{\gamma_3} \cdot g_3^{r_1+s_1} \;\wedge\; \bar{u}_4' = g^{\gamma_4} \cdot g^{x_1} \cdot h_1^{r_1} h_2^{s_1} \;\wedge \bar{u}_5' = g^{\gamma_5} \cdot g^{l_1} \;\wedge\; \bar{v}_1' = \hat{e}(g_1, g^{\delta_1}) \cdot \prod_{i=1}^{5} \hat{e}(f_{i,1}, g^{\gamma_i}) \;\wedge$$
$$\bar{v}_2' = \hat{e}(g_2, g^{\delta_2}) \cdot \prod_{i=1}^{5} \hat{e}(f_{i,2}, g^{\gamma_i})$$

and

$$\pi_2 = \text{⅄}\, wit_2, l_2, x_2, r_2, s_2 : \phi_2(wit_2, l_2, x_2, bases) \ \wedge\ \bar{u}_1 = \bar{u}'_1 \cdot g_1^{r_2} \ \wedge\ \bar{u}_2 = \bar{u}'_2 \cdot g_2^{s_2} \ \wedge\ \bar{u}_3 = \bar{u}'_3 \cdot g_3^{r_2+s_2} \ \wedge$$
$$\bar{u}_4 = \bar{u}'_4 \cdot g^{x_2} \cdot h_1^{r_2} h_2^{s_2} \ \wedge\ \bar{u}_5 = \bar{u}'_5 \cdot g^{l_2} \ \wedge\ \bar{v} = \left(\prod_{i=0}^{5} \hat{e}(f_{i,1}, \bar{u}_i)/\bar{v}'_1\right)^{r_2} \left(\prod_{i=0}^{5} \hat{e}(f_{i,2}, \bar{u}_i)/\bar{v}'_2\right)^{s_2},$$

where $\bar{u}_0 = g$.

Listing 9 Efficient realization of $\mathcal{P}_{\mathsf{zk}_{\mathsf{SA}}} \leq^{SS} \mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}(\mathfrak{R}_{\mathsf{SA}})$ and $\mathcal{P}_{\mathsf{zk}_{\mathsf{RA}}} \leq^{SS} \mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}(\mathfrak{R}_{\mathsf{RA}})$

Verifiable Decryption - a proof that $\mathfrak{c} = (u_1, u_2, u_3, \boldsymbol{c}, v)$ decrypts to $\boldsymbol{m}$ for a label L with a secret key $\mathsf{sk} = (\{\boldsymbol{\alpha}\}_{i=1}^{n}, \{\boldsymbol{\beta}\}_{i=0}^{(n+4)})$ corresponding to $\mathsf{pk} = (\{h_{i,1}, h_{i,2}\}_{i=1}^{n}, \{f_{i,1}, f_{i,2}\}_{i=0}^{(n+4)})$.

$$\pi = \text{⅄}\, \{\boldsymbol{\alpha}\}_{i=1}^{n}, \{\boldsymbol{\beta}\}_{i=0}^{n+4} \ :\ \{\ h_{i,1} = g_1^{\alpha_{i,1}} g_3^{\alpha_{i,3}}\ \}_{i=1}^{n} \ \wedge\ \{\ h_{i,2} = g_2^{\alpha_{i,2}} g_3^{\alpha_{i,3}}\ \}_{i=1}^{n} \ \wedge\ \{\ f_{i,1} = g_1^{\beta_{i,1}} g_3^{\beta_{i,3}}\ \}_{i=0}^{(n+4)} \ \wedge$$
$$\{\ f_{i,2} = g_2^{\beta_{i,2}} g_3^{\beta_{i,3}}\ \}_{i=0}^{(n+4)} \ \wedge\ v = \prod_{j=1}^{3} \left(\prod_{i=0}^{3} \hat{e}(u_j, u_i)^{\beta_{i,j}} \cdot \prod_{i=4}^{n+3} \hat{e}(u_j, c_{i-3})^{\beta_{i,j}} \cdot \hat{e}(u_j, L)^{\beta_{(n+4),j}} \right) \wedge\ \{\ m_i = c_i \cdot \prod_{j=1}^{3} u_j^{-\alpha_{i,j}}\ \}_{i=1}^{n}$$

4.4 Proof of Oblivious Third Party Protocol

The simulator needs to do some trivial forwarding for every corrupted role R: it forwards all messages from the environment leaked through $\mathsf{notp_1.R}$ to $\mathsf{notp_2.R}$; all messages from $\mathsf{notp_2.R}$, addressed to the environment are forwarded to the corrupted party on $\mathsf{notp_1.R}$. The simulator internally simulates most of the real world ideal functionalities to simulate delays and corruption of submodules. All messages addressed to another corrupted real world entity are forwarded to an internal simulation of that entity.

For ideal communication between honest roles, the simulator simply simulates the delays of the real communication internally based on the delays in the ideal communication. The simulator creates and registers the keys of honest SA and RA. After the keys of RA are registered, Sim sends $(\texttt{SetF}, F, \mathbb{G}, \mathbb{G})$ to $\mathcal{F}_{\mathsf{otp}}$ to set $F(id) = g^{id}$.

As we will see, the two most interesting cases of the simulation are when either the user or the service provider, but not both are corrupted. We cover the other corner cases first.

Listing 10 Sim if both user and service provider are corrupted

We only need to simulate for an honest SA or RA.

- Upon receiving $(\mathfrak{c}_1, L)$ from $\mathcal{I}_{\mathsf{sc}_1}$, the simulator checks whether the ciphertext correctly decrypts under label L to some value m, picks a random T and sends $(\texttt{TestSatisfy}, L, T)$ to $\mathsf{notp_1.U}$.
- Upon receiving $(\texttt{Satisfy}, \perp)$ or $(\texttt{Satisfy}, T)$, it skips a communication round for $\mathcal{I}_{\mathsf{sc}_1}$ and either proves $m = \perp$ with an otherwise random instance and witness of correct size or $((\mathfrak{c}_1, pk_{\mathsf{SA}}, L, m), sk_{\mathsf{SA}}) \in \mathfrak{R}_{\mathsf{SA}}$ respectively.
- Upon receiving $(\mathfrak{c}_2, t_{due})$ from $\mathcal{I}_{\mathsf{sc}_2}$, the simulator decrypts the ciphertext under label $g^{t_{due}}$ into (T, m), picks a random id, and sends $(\texttt{TestOpen}, T, id, t_{due})$ to $\mathsf{notp_1.SP}$.
- Upon receiving $(\texttt{Open}, \perp)$ or $(\texttt{Open}, id)$, it skips a communication round for $\mathcal{I}_{\mathsf{sc}_2}$ and either proves $m = \perp$ with an otherwise random instance and witness of correct size or $((\mathfrak{c}_2, pk_{\mathsf{RA}}, g^{t_{due}}, m), (sk_{\mathsf{RA}}, T)) \in \mathfrak{R}_{\mathsf{RA}}$ respectively.

Listing 11 Sim if both user and service provider are honest

We only need to simulate for a corrupted SA or RA.

- Upon receiving $(\texttt{ReqSatisfy}, L)$, the simulator picks a random message m and sends $(\mathsf{Enc}(pk_{\mathsf{SA}}, L, m), L)$ to $\mathcal{I}_{\mathsf{sc}_1}$.
- When receiving $(\texttt{Prove}, \perp)$, it sends $(\texttt{Satisfy}, \mathit{false})$ to $\mathsf{notp_1.SA}$.
- When receiving $(\texttt{Prove}, \mathfrak{c}_1, pk_{\mathsf{SA}}, L, m)$, it sends $(\texttt{Satisfy}, \mathit{true})$.
- Upon receiving $(\texttt{ReqOpen}, T, t_{due})$, the simulator picks a random message m and send $\mathsf{Enc}(pk_{\mathsf{RA}}; g^{t_{due}}; T, m)$ to $\mathcal{I}_{\mathsf{sc}_2}$.
- When receiving $(\texttt{Prove}, \perp)$, it sends $(\texttt{Open}, \mathit{false})$ to $\mathsf{notp_1.RA}$. When receiving $(\texttt{Prove}, \mathfrak{c}_2, pk_{\mathsf{RA}}, g^{t_{due}}, m)$ it sends $(\texttt{Open}, \mathit{true})$.

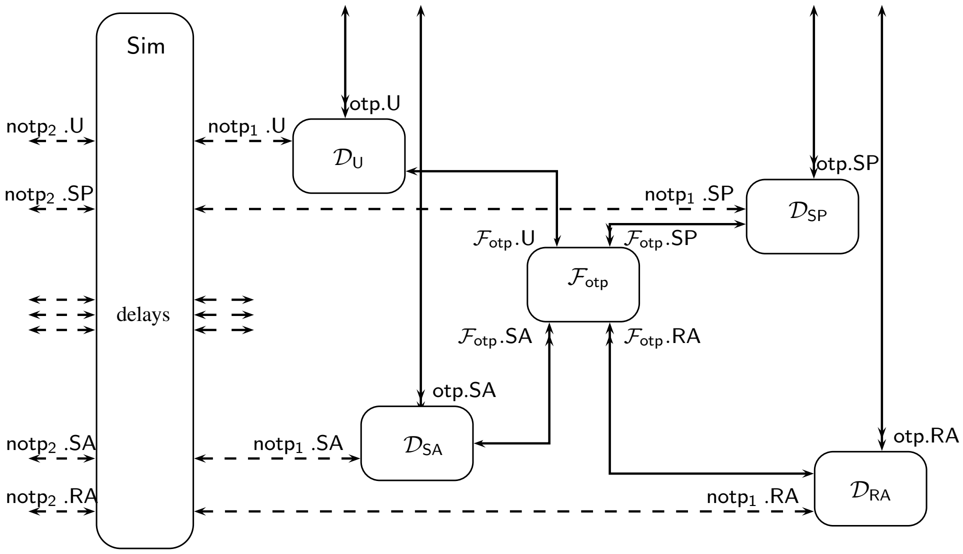


Fig. 2. OTP simulator

Because of the use of ideal functionalities, the simulation for all of these cases is perfect. We now consider a corrupted user and an honest service provider.

Listing 12 Sim if the user is corrupted, the service provider is honest

The simulator Sim sets $state$ = "ready" and follows the instructions for SP of the real world protocol.

On input $params_{\mathsf{U}}$ on $\mathcal{I}_{\mathsf{sc}_3}$ with $state$ = "ready";

- generate commitment parameters $params_{\mathsf{SP}}$ and $C_{x_2} \leftarrow \mathsf{Commit}(params_{\mathsf{U}}, x_2, open_{x_2})$ on random $open_{x_2}$, and return both to $\mathcal{I}_{\mathsf{sc}_3}$.
- wait for C_{id} and $C_{x'_1}$ over $\mathcal{I}_{\mathsf{sc}_3}$ and reply with an acknowledgment.
- wait for $(\mathtt{Input}_1, (inst, params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}), pk_{\mathsf{SA}}, (id, wit, open_{id}, x'_1, open_{x'_1}), (l_1, x_1))$ on $\mathcal{I}_{\mathsf{tpc}_1}.\mathsf{P}_1$, store id, wit, x'_1, x_1 and forward the message to the simulated $\mathcal{F}_{\mathsf{tpc}_1}.\mathsf{P}_1$.
- wait for $(\mathtt{Input}_1, (inst, params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}), pk_{\mathsf{SA}})$ on $\mathcal{F}_{\mathsf{tpc}_1}.\mathsf{P}_2$, send $(\mathtt{EnrollU}, inst, (id, wit))$ to $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$, and continue the delay on $\mathcal{F}_{\mathsf{otp}}.\mathsf{SP}$.
- let $state$ = "enrollu".

On a delay on $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$ with $state$ = "enrollu":

- confirm the delay and wait for $(\mathtt{DeliverEnrollU}, L, t_{due})$ from $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$.
- send t_{due} over $\mathcal{I}_{\mathsf{sc}_3}$ to $\mathsf{notp}_2.\mathsf{U}$ and wait for an acknowledgment.
- if SA corrupted, send $(\mathtt{ReqSatisfy}, L)$ to $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$, confirm satisfaction, learn $(\mathtt{Satisfy}, T)$ and set $m = T \cdot g^{x_1 - x'_1}$.
- otherwise set $m = 1$.
- send $(\mathtt{Input}_2, open_{x_2}, (l_2, x_2))$ to the simulated $\mathcal{I}_{\mathsf{tpc}_2}$, in which we set $\mathfrak{c}_1 = \mathsf{Enc}(pk_{\mathsf{SA}}, L, m)$ and store $\mathfrak{c}_1$. Finally, set $state$ = "deliverenrollu".

On the acknowledgment over $\mathcal{I}_{\mathsf{sc}_3}$ with $state$ = "deliverenrollu"

- send $(\mathtt{Input}_1, (params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x'_1}, C_{x_2}, T_{due}), pk_{\mathsf{RA}}, open_{x_2}, (t_{due}, x_2, x'_2))$ to $\mathcal{I}_{\mathsf{tpc}_2}.\mathsf{P}_1$.
- wait for $(\mathtt{Input}_2, (\widehat{open}_{id}, \widehat{open}_{x'_1}), (0, x'_1, id))$ on $\mathsf{notp}_2.\mathsf{U}$, simulate $\mathcal{I}_{\mathsf{tpc}_2}$ resulting in the ciphertext $\mathfrak{c}_2$.
- finally, set $state$ = "enrolled" and send $(\mathtt{DeliverEnrollSP})$ to $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$ with a confirmation on the delay of $\mathcal{F}_{\mathsf{otp}}.\mathsf{SP}$.[9]

As the user is corrupted and the service provider is honest, no extra simulation is needed for a corrupted SA, or a honest RA. If SA is honest, we have to handle satisfaction requests.[10]

On $(\widehat{\mathfrak{c}}_1, \widehat{L})$ on $\mathsf{notp}_2.\mathsf{U}$ with $state$ = "enrolled";

[9] If $\widehat{open}_{id}$, and $\widehat{open}_{x'_1}$ correspond to the $open_{id}$, and $open_{x'_1}$ sent by the user during the EnrollU phase, this simulation step is perfect. We will show in Lemma 3 that given the binding property of the commitment scheme this is the case except with negligible probability.

[10] Note that in this case, as we did not know the value of T yet, we used a fake encryption of 1 and rely on the CCA security of the ciphertext for the indistinguishability of the simulation. We describe the reduction in Lemma 4.

– simulate $\mathcal{I}_{\mathsf{sc}_1}$ and if $\mathfrak{c}_1 = \widehat{\mathfrak{c}}_1$, send $(\mathtt{ReqSatisfy})$ to the corrupted $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$, otherwise if the ciphertext validates with label L, pick a random T and send $(\mathtt{TestSatisfy}, L, T)$. Finally, confirm the delay on $\mathcal{F}_{\mathsf{otp}}.\mathsf{SA}$.

On a delay on $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$

– confirm the delay and wait for $(\mathtt{Satisfy}, \widehat{T})$ on $\mathcal{F}_{\mathsf{otp}}.\mathsf{U}$;
– if $\widehat{T} = \epsilon$ prove $m = \perp$ with an otherwise random instance and witness of correct size using a simulated $\mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}$.
– if $\widehat{T} \neq \epsilon$, if this is in reply to a $\mathtt{TestSatisfy}$ request, prove correct decryption of the blinded token $m = \mathsf{Dec}(sk_{\mathsf{SA}}, \widehat{L}, \widehat{\mathfrak{c}}_1)$ using $\mathcal{I}_{\mathsf{zk}_{\mathsf{SA}}}$ to $\mathsf{notp}_2.\mathsf{U}$, otherwise (this is in reply to a $\mathtt{ReqSatisfy}$) prove correct decryption of the blinded token $m = \mathsf{Dec}(sk_{\mathsf{SA}}, L, \mathfrak{c}_1) = T \cdot g^{x_1 - x'_1})$.

If RA is corrupted, we have to simulate opening requests towards it. This is done in the same way as for the case of an honest U and an honest SP.[11]

Lemma 3. *Given the DLIN assumption, if* U *is corrupted,* SP *is honest, and* SA *and* RA *are either honest or corrupted,* $\mathcal{P}_{\mathsf{otp}}(\mathfrak{R}) \leq^{SS} \mathcal{I}_{\mathsf{otp}}(\mathfrak{R})$.

Proof sketch of Lemma 3: We proceed in a sequence of games. We start with a game where the environment interacts with the real protocol, and end up with a game where the environment interacts with the simulator and the ideal system. Then we show that all those games are computationally indistinguishable. Let W_i denote the event that the environment Env outputs 1 in Game i.

Game 0. This is the real protocol run.

Game 1. This game is the same as Game 0, except that the game aborts if the environment controlling the corrupted user sends two different openings for one of the commitments C_{id} or $C_{x'_1}$ as part of its input to $\mathcal{I}_{\mathsf{tpc}_1}$ and $\mathcal{I}_{\mathsf{tpc}_2}$. An environment that distinguishes between Game 0 and Game 1 breaks the binding property of the commitment scheme which for Pedersen commitments would contradict the Discrete Logarithm assumption. Therefore $|\Pr[W_1] - \Pr[W_0]| = \mathrm{negl}(k)$.

Game 2. This game is the same as Game 1, except that the checks of relations in the zero-knowledge functionality and the two party computation are turned off for honest parties, and that the real commitment of the service provider is replaced by a random commitment. Honest users never do proofs that wouldn't verify, and commitments are perfectly hiding. Therefore $\Pr[W_2] = \Pr[W_1]$.

Game 3. This game is the same as Game 2, except that, if SA is honest, the ciphertext $\mathfrak{c}_1$ is replaced with an encryption of 1. By Lemma 4, $|\Pr[W_3] - \Pr[W_2]| = \mathrm{negl}(k)$.

Game 4. Game 4 replaces the control logic of the real protocol with the control logic of the simulator and the ideal functionality. No further cryptographic values need to be changed. Therefore $\Pr[W_4] = \Pr[W_3]$. □

Lemma 4. *If* U *is corrupted,* SP *and* SA *are honest, and* RA *is either honest or corrupted* $|\Pr[W_3] - \Pr[W_2]| = \mathrm{negl}(k)$*, if* KeyGen*,* Enc*,* Dec *is a CCA secure encryption scheme.*

Proof. The proof is by contradiction, by showing a reduction from a distinguishing environment Env to a successful CCA adversary $\mathcal{A}$. $\mathcal{A}$ receives the public key pk as input and playing the role of the honest SA, registers it with $\mathcal{I}_{\mathsf{reg}}$. Depending on the bit b of the CCA challenger, the adversary will (without knowing it himself) either simulate Game 2 or Game 3 towards Env.

$\mathcal{A}$ follows the instructions of the games but uses the decryption oracle to decrypt messages. When the ciphertext $\mathfrak{c}_1$ needs to be created, the CCA adversary $\mathcal{A}$ asks for a challenge ciphertext $\mathfrak{c}$ by sending $m_0 = g^{x_1+x_2}$ and $m_1 = 1$ to the CCA challenge oracle and uses the result as $\mathfrak{c}_1$. For the rest of the interactions with Env, $\mathcal{A}$ follows the joint instructions of the games and forwards the output of Env as its guess.

[11] This aspect of the simulation is perfect.

If the bit b chosen by the CCA challenge game is 0 the behavior of the CCA adversary perfectly follows the behavior of Game 2, otherwise it corresponds to Game 3. Consequently, $\mathcal{A}$ has the same advantage as Env. $\square$

Listing 13 Sim when the service provider is corrupted, the user is honest

The simulator sets *state* = "ready" and follows the instructions for U of the real world protocol.

On a delay on $\mathcal{F}_T$.SP with *state* = "ready":

- confirm the delay and wait for ($\texttt{EnrollU}$, *inst*) from the corrupted $\mathcal{F}_{\mathsf{otp}}$.SP, store *inst*.
- generate $params_{\mathsf{U}}$ and send them over $\mathcal{I}_{\mathsf{sc}_3}$.
- wait for C_{x_2} and $params_{\mathsf{SP}}$ on $\mathcal{I}_{\mathsf{sc}_3}$, generate random $l_1, id, x_1, x_1', open_{id}$ and $open_{x_1'}$, compute $C_{id} \leftarrow \mathsf{Commit}(params_{\mathsf{SP}}, id, open_{id})$, $C_{x_1'} \leftarrow \mathsf{Commit}(params_{\mathsf{SP}}, x_1', open_{x_1'})$ and send C_{id} and C_{x_1} over $\mathcal{I}_{\mathsf{sc}_3}$.
- upon receiving an acknowledgement over $\mathcal{I}_{\mathsf{sc}_3}$, set *state* = "enrollu", and send $(\texttt{Input}_1, (inst, params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x_1'}, C_{x_2}), pk_{\mathsf{SA}}, (id, wit, open_{id}, x_1', open_{x_1'}), (l_1, x_1))$ to $\mathcal{I}_{\mathsf{tpc}_1}$.

On t_{due} on $\mathcal{I}_{\mathsf{sc}_3}$ where *state* = "enrollu":

- reply with an acknowledgment.
- receive $(\texttt{Input}_2, (open_{x_2}, (l_2, x_2, x_2')))$ on notp_2.SP and forward it to the simulated $\mathcal{I}_{\mathsf{tpc}_1}$ resulting in $\mathfrak{c}_1$ and $l = g^{l_1+l_2}$.
- set *state* = "deliverenrollu", and send ($\texttt{DeliverEnrollU}, t_{due}$) to $\mathcal{F}_{\mathsf{otp}}$.SP with a confirmation on the delay of $\mathcal{F}_{\mathsf{otp}}$.U.

On a delay on $\mathcal{F}_{\mathsf{otp}}$.SP where *state* = "deliverenrollu":

- confirm the delay, wait for ($\texttt{DeliverEnrollSP}$) from $\mathcal{F}_{\mathsf{otp}}$.SP and send an acknowledgement over $\mathcal{I}_{\mathsf{sc}_3}$
- wait for $(\texttt{Input}_1, (params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x_1'}, C_{x_2}, T_{due}), pk_{\mathsf{RA}}, \widehat{open}_{x_2}, (t_{due}, x_2, x_2'))$, simulate $\mathcal{I}_{\mathsf{tpc}_2}$ and receive message $(\texttt{Input}_1, (params_{\mathsf{U}}, params_{\mathsf{SP}}, C_{id}, C_{x_1'}, C_{x_2}), pk_{\mathsf{RA}})$. [12]
- if RA is corrupted, send ($\texttt{ReqOpen}$) to $\mathcal{F}_{\mathsf{otp}}$.SP, learn T, confirm the opening, learn ($\texttt{Open}$,ID) and set $m = (T, ID \cdot g^{x_2'})$.
- otherwise set $m = (1, 1)$.
- send $(\texttt{Input}_2, (open_{id}, open_{x_1'}), (0, x_1', id))$ to the simulated $\mathcal{I}_{\mathsf{tpc}_2}$ in which we set $\mathfrak{c}_2 = \mathsf{Enc}(pk_{\mathsf{SA}}, t_{due}, m)$.
- set *state* = "enrolled".

As the service provider is corrupted and the user is honest, no extra simulation is needed for an honest SA, or a corrupted RA. If RA is honest, we have to handle opening requests.[13]

On $(\widehat{\mathfrak{c}}_2, t_{due})$ on notp_2.SP where *state* = "enrolled";

- simulate $\mathcal{I}_{\mathsf{sc}_2}$ and if $\mathfrak{c}_2 = \widehat{\mathfrak{c}}_2$, send ($\texttt{ReqOpen}$) to the corrupted $\mathcal{F}_{\mathsf{otp}}$.SP, otherwise if the ciphertext validates with label $g^{t_{due}}$, pick a random T, id and send ($\texttt{TestSatisfy}, T, id, t_{due}$). Finally, confirm the delay on $\mathcal{F}_{\mathsf{otp}}$.RA.

On a delay on $\mathcal{F}_{\mathsf{otp}}$.SP where *state* = "enrolled":

- confirm the delay and wait for $(\texttt{Open}, \widehat{ID})$ on $\mathcal{F}_{\mathsf{otp}}$.SP.
- if $\widehat{ID} = \epsilon$ prove $m = \perp$ with an otherwise random instance and witness of correct size using a simulated $\mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}$.
- if $\widehat{ID} \neq \epsilon$, if this is in reply to a $\texttt{TestOpen}$ request, prove correct decryption of the blinded identity $m = \mathsf{Dec}(sk_{\mathsf{RA}}, t_{due}, \widehat{\mathfrak{c}}_2)$ using $\mathcal{I}_{\mathsf{zk}_{\mathsf{RA}}}$ to notp_2.SP, otherwise (this is in reply to a $\texttt{ReqOpen}$) prove correct decryption of the blinded identity $m = \mathsf{Dec}(sk_{\mathsf{RA}}, t_{due}, \mathfrak{c}_2) = ID \cdot g^{x_2'}$.

If SA is corrupted, we have to simulate satisfaction requests towards it. This is done in the same way as for the case of an honest U and an honest SP.[14]

Lemma 5. *Given the DLIN assumption, if* SP *is corrupted,* U *is honest, and* SA *and* RA *are either honest or corrupted,* $\mathcal{P}_{\mathsf{otp}}(\mathfrak{R}) \leq^{SS} \mathcal{I}_{\mathsf{otp}}(\mathfrak{R})$.

The proof follows the proof of Lemma 3.

[12] If $\widehat{open}_{x_2}$ correspond to the $open_{x_2}$ sent by the service provider during the $\texttt{EnrollU}$ phase, this simulation step is perfect. We will show in Lemma 5 that given the binding property of the commitment scheme this is the case except with negligible probability.

[13] Note that in this case, as we did not know the value of T and id yet, we used a fake encryption of $(1, 1)$ and rely on the CCA security of the ciphertext for the indistinguishability of the simulation. We describe the reduction in Lemma 5.

[14] This aspect of the simulation is perfect.

5 Conclusion

We propose the first public key encryption scheme that is structure preserving and secure against adaptive chosen ciphertext attacks. We demonstrate the usefulness of this new primitive by the joint ciphertext computation protocol and our proposal for instantiating oblivious third parties. We conjecture, however, that the combination of the structure preserving encryption scheme and efficient zero-knowledge proofs facilitate a much larger set of efficient protocol constructions. All protocols are proven secure in the IITM model. The results carry over to the universal composability model.

References

1. J. Camenisch, A. Kiayias, and M. Yung, "On the portability of generalized schnorr proofs," in *EUROCRYPT*, pp. 425–442, 2009.
2. J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *EUROCRYPT*, pp. 415–432, 2008.
3. J. Camenisch and A. Lysyanskaya, "A signature scheme with efficient protocols," in *SCN*, pp. 268–289, 2002.
4. M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya, "P-signatures and noninteractive anonymous credentials," in *TCC*, pp. 356–374, 2008.
5. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo, "Structure-preserving signatures and commitments to group elements," in *Advances in Cryptology - CRYPTO 2010*, LNCS, pp. 209–237, Springer-Verlag, 2010.
6. J. Camenisch and V. Shoup, "Practical verifiable encryption and decryption of discrete logarithms," *Advances in Cryptology-CRYPTO 2003*, pp. 126–144, 2003.
7. R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," *SIAM Journal on Computing*, vol. 33, pp. 167–226, 2001.
8. J. Katz and V. Vaikuntanathan, "Signature schemes with bounded leakage resilience," in *ASIACRYPT*, pp. 703–720, 2009.
9. Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs, "Efficient public-key cryptography in the presence of key leakage," in *ASIACRYPT*, pp. 613–631, 2010.
10. D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Advances in Cryptology — CRYPTO '04* (M. Franklin, ed.), vol. 3152 of *LNCS*, pp. 41–55, Springer-Verlag, 2004.
11. J. Camenisch and A. Lysyanskaya, "Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation," in *Advances in Cryptology — EUROCRYPT 2001* (B. Pfitzmann, ed.), vol. 2045 of *LNCS*, pp. 93–118, Springer Verlag, 2001.
12. N. Asokan, V. Shoup, and M. Waidner, "Optimistic fair exchange of digital signatures," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 591–610, Apr. 2000.
13. J. Camenisch, T. Groß, and T. S. Heydt-Benjamin, "Rethinking accountable privacy supporting services: extended abstract," in *ACM DIM – Digital Identity Management*, pp. 1–8, 2008.
14. R. Kusters, "Simulation-based security with inexhaustible interactive turing machines," in *Computer Security Foundations Workshop, 2006. 19th IEEE*, pp. 12–320, IEEE, 2006.
15. R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS*, pp. 136–145, 2001.
16. M. Backes, B. Pfitzmann, and M. Waidner, "The reactive simulatability (rsim) framework for asynchronous systems," *Inf. Comput.*, vol. 205, no. 12, pp. 1685–1720, 2007.
17. R. Küsters and M. Tuengerthal, "Joint state theorems for public-key encryption and digital signature functionalities with local computation," in *CSF*, pp. 270–284, IEEE Computer Society, 2008.
18. R. Küsters and M. Tuengerthal, "Universally composable symmetric encryption," in *CSF*, pp. 293–307, IEEE Computer Society, 2009.
19. J. Camenisch, N. Casati, T. Groß, and V. Shoup, "Credential authenticated identification and key exchange," in *CRYPTO*, pp. 255–276, 2010.
20. J. Camenisch, S. Krenn, and V. Shoup, "A framework for practical universally composable zero-knowledge protocols." Cryptology ePrint Archive, Report 2011/228, 2011. `http://eprint.iacr.org/`.
21. R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *CRYPTO '98*, pp. 13–25, Springer-Verlag, 1998.
22. J. Camenisch, *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem.* PhD thesis, ETH Zürich, 1998.
23. H. Shacham, "A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants," 2007. Cryptology ePrint Archive, Report 2007/074.
24. M. Abe, K. Haralambiev, and M. Ohkubo, "Signing on group elements for modular protocol designs." Cryptology ePrint Archive, Report 2010/133, 2010. `http://eprint.iacr.org`.

25. R. Cramer and V. Shoup, "Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption," in *EUROCRYPT '02*, pp. 45–64, Springer-Verlag, 2002.
26. R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *TCC* (S. P. Vadhan, ed.), vol. 4392 of *Lecture Notes in Computer Science*, pp. 61–85, Springer, 2007.
27. Y. Dodis, V. Shoup, and S. Walfish, "Efficient constructions of composable commitments and zero-knowledge proofs," in *CRYPTO* (D. Wagner, ed.), vol. 5157 of *Lecture Notes in Computer Science*, pp. 515–535, Springer, 2008.
28. B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin, "Secure computation without authentication," in *CRYPTO*, pp. 361–377, 2005.
29. J. Camenisch, N. Casati, T. Groß, and V. Shoup, "Credential authenticated identification and key exchange," in *CRYPTO*, pp. 255–276, 2010.
30. R. Küsters and M. Tuengerthal, "Composition Theorems Without Pre-Established Session Identifiers," tech. rep., University of Trier, 2011.

A Details on the IITM Model

The inexhaustible interactive Turing machines (IITM) model [14] is a refinement of the universal composability and reactive simulatability model. In the IITM model both ideal functionalities $\mathcal{F}$ and protocols $\mathcal{P}$ are configurations of IITMs. The inexhaustibility, guarantees that a well-formed system of polynomial time bounded IITMs can be simulated by a single ITM. This allows us to interpret ideal functionalities and protocols either as interconnected systems that communicate via input-output tape pairs shared between component IITMs, or as a single ITM.

An IITM with an (enriching) input tape named start, is called a master IITM. It is triggered if no other IITM was triggered. An IITM is triggered by another IITM if the latter writes a message on an output tape that corresponds to an input tape of the former. Note that on each activation of an IITM, it can write to at most one output tape. Each IITM M has an associated polynomial q which is used to bound the computation time per activation where the polynomial is not only in the security parameter, but also in the length of the current input and the size of the current configuration (i.e. the length of the content written on the working tapes). As a result, inputs of arbitrary size (e.g. stream ciphers) may be processed. Input tapes can be either *consuming* ($\rightarrow$) or *enriching* ($\twoheadrightarrow$), of which the length of the inputs on the latter is a bounding factor for the size of the current configuration and the output produced by that IITM. In order to ensure that such systems run in polynomial time, *well-formed* systems require a graph defined by the enriching tapes to be acyclic. In addition, systems of IITMs may contain an unbound number of copies of IITMs as indicated by the bang operator ('!'). Küsters therefore proposes a flexible and generic mechanism for addressing those copies of IITMs. An IITM may run in 2 modes: in the $\mathsf{CheckAddress}$ mode the IITM runs a deterministic algorithm to verify that a message is in fact addressed to it; in the $\mathsf{Compute}$ mode the IITM will do the actual processing of the input and possibly writes output to one of its output tapes. If no current instance of the banged IITM accepts the input, and the default instance accepts in the $\mathsf{CheckAddress}$ mode, a new instance is created.

Tapes. As a convention we bundle external[15] tapes of systems into interfaces. An interface inf consists of named input-output tape pairs. An input-output tape pair is named $\mathsf{inf.R}$ after a combination of the interface name inf and a role name R. We refer to the set of all roles of an interface as $\mathsf{inf}.\mathcal{R}$. If a system of IITMs implementing an interface inf, is connected to another ITM M, then as a convention we refer to the input-output tape pair of party M that is connected to role R of the interface as $\overline{\mathsf{inf}}.\mathsf{R}$.

For each system $\mathcal{S}_{\mathsf{inf}}$ implementing a functionality inf, we distinguish between the *API interface* inf (called IO interface in Küsters terminology), defining the environmental/trusted connections of the system and *network interface* ninf, defining the adversarial/untrusted connections of the system.

For example, if an IITM M wants to send a message to role R of a system of IITMs $\mathcal{S}_{\mathsf{inf}}$ implementing inf, M would write the request to the output tape of $\overline{\mathsf{inf}}.\mathsf{R}$ and $\mathcal{S}_{\mathsf{inf}}$ would read it on the input tape of $\mathsf{inf.R}$. To answer the request $\mathcal{S}$ would write the response on the output tape of $\mathsf{inf.R}$ and M would read the

[15] As apposed to internal tapes connecting the internal IITMs of a system.

request on the input tape of $\overline{\mathsf{inf}}$.R. Similarly, an adversary $\mathcal{A}$ would send messages to the network output tape of $\overline{\mathsf{ninf}}$.R and $\mathcal{S}_{\mathsf{inf}}$ would read it on the input tape of ninf.R.

For simulation-based security definitions the ideal protocol $\mathcal{I}$ and the real protocol $\mathcal{P}$ that emulates this ideal system have to present the same API interface inf towards their environment, i.e., they must be *API compatible.* We refer to an ideal system and a protocol that is API compatible with respect to interface inf as $\mathcal{I}_{\mathsf{inf}}$ and $\mathcal{P}_{\mathsf{inf}}$ respectively. In addition $\mathcal{I}_{\mathsf{inf}}$ and $\mathcal{P}_{\mathsf{inf}}$ must expose different network interfaces ninf_1 and ninf_2.

Simulatability. A proof that $\mathcal{P}_{\mathsf{inf}}$ emulates $\mathcal{I}_{\mathsf{inf}}$, short $\mathcal{P}_{\mathsf{inf}} \leq^{SS} \mathcal{I}_{\mathsf{inf}}$ will need to prove existance of a simulator Sim that translates between the interfaces ninf_1 and ninf_2. This is formalized as *strong simulatability* which implies other simulatability notions such as dummy universal composability and blackbox simulatability.

Definition 5 (Strong Simulatability (SS). $\mathcal{P}_{\mathsf{inf}} \leq^{SS} \mathcal{I}_{\mathsf{inf}}$). *A protocol system $\mathcal{P}_{\mathsf{inf}}$ strongly emulates $\mathcal{I}_{\mathsf{inf}}$ if there exists a simulator* Sim *connected to environment* Env *on interface* ninf_2 *and $\mathcal{I}_{\mathsf{inf}}$ on interface* ninf_1 *such that for all master IITMs* Env *that connect to* inf *and* ninf_2*:* $\mathsf{Env}|\mathcal{P}_{\mathsf{inf}} \equiv \mathsf{Env}|\mathsf{Sim}|\mathcal{I}_{\mathsf{inf}}$

Corruption. Küsters [17] presents a standard corruption model for ITMs listed below. In our exposition, we consider only static corruption. A corrupted role forwards all inputs on tapes $\mathcal{T}_{\mathsf{U}}$ to ninf_i.R and acts as a proxy that allows the environment to send messages to any of its tapes in $\mathcal{T}_{\mathsf{U}}$, by sending control messages to $\overline{\mathsf{ninf}}_i$.R.

Listing 14 Macro $\mathsf{Corr}(\mathit{corrupted} \in \mathit{Bool}, \mathit{corruptible} \in \mathit{Bool}, \mathit{initialized} \in \mathit{Bool}, \mathit{msg}, \mathsf{inf}.R, \mathsf{ninf}.R, \mathcal{T}_{\mathsf{U}})$

Initialization: res =0
Compute:

On (`Corrupted`?) from inf $.R$ where *initialized*:
- send (*corrupted*) to inf $.R$. (**Corruption Request**)

On (`Corrupt`) received from ninf.R where *corruptable*, *initialized* and not *corrupted* (**Corruption**)
- let $\mathit{corrupted} = \mathit{true}$; send (`Corrupted`, msg) to ninf.R

On m received from $T \in \mathcal{T}_{\mathsf{U}}$ where *corrupted* (**Forward to* ninf.R *(Rule takes precedence over all other rules)**)
- let $\mathit{res} \leftarrow 0$; send (`LeakRecv`, m, T) to ninf.R

On (`Send`, m, T) received from ninf.R, $T \in \mathcal{T}_{\mathsf{U}}$, *corrupted*, $0 < |m| \leq \mathit{res}$ (**Forward to tape**)
- let $\mathit{res} \leftarrow 0$; send m to T.

On (`Resources`, r) received from inf.R where *corrupted* (**Resources**)
- let $\mathit{res} \leftarrow |r|$ and send (`Resources`, r) to ninf.R.

As mentioned before, cryptography has a particular interest in ideal systems that model a virtual incorruptible party F_T. As the functionality F_T implements the security critical parts of an ideal system, the parties representing the different roles of the interface only need to implement forwarding and corruption. We describe this in the following macro:

Listing 15 Dummy functionality: $\mathsf{Dummy}(\mathsf{inf}.R, \mathsf{ninf}.R, F_{\mathsf{T}}.R)$:

Tapes $\mathsf{inf}.R \longleftrightarrow\!\!\!\!\rightarrow \overline{\mathsf{inf}}.R$, $\mathsf{ninf}.R \longleftrightarrow \overline{\mathsf{ninf}}.R$, $\overline{F}_{\mathsf{T}}.R \not\twoheadleftarrow\!\!\!\rightarrow F_{\mathsf{T}}.R$
Initialization: $\mathit{state} = \epsilon, \mathit{corrupted} = \mathit{corruptible} = \mathit{false}.$
Compute:

On (`Ready`) from inf.R where $\mathit{state} = \epsilon$:
let state = "ready"; let $\mathit{corruptible} = \mathit{true}$
send (`Ready`) to ninf.R

On m from inf.R where state = "ready"
- let $\mathit{corruptible} = \mathit{false}$; send m to $\overline{F}_{\mathsf{T}}.R$

On m from $\overline{F}_{\mathsf{T}}.R$ where state = "ready"
- let $\mathit{corruptible} = \mathit{false}$; send m to inf.R

Corruption: Corr $(\mathit{corrupted}, \mathit{corruptible}, \mathit{state} \neq \epsilon, \epsilon, \mathsf{inf}.R, \mathsf{ninf}.R, \{\mathsf{inf}.R, \overline{F}_{\mathsf{T}}.R\})$

A.1 Modeling communication channels

Both ideal and real protocols have to model communication. Ideal protocols model both ideal cryptography, as well as ideal communication. A common situation is when the adversary is ideally only able to arbitrarily delay the delivery of results. This models the restriction that cryptography cannot prevent denial of service attacks against an adversary that is in control of communication resources. We model this commonly recurring pattern as an IITM that, on the (`Continue`) command on adversarial channel C, copies messages from one tape to another.

Delayed communication. We model enriching delayed communication $T \overset{C}{\twoheadleftarrow\!\!\not\to} \overline{T}$ which leaks to network tape C by the following ideal system. In short, we delay all messages in the non-enriching direction.

Listing 16 Functionality $\mathsf{Delay}(T, \overline{T}, C) \equiv T \overset{C}{\twoheadleftarrow\!\!\not\to} \overline{T}$

Tapes: $T_{aux} \longleftrightarrow\!\!\!\!\!\to T$, $\overline{T}_{aux} \twoheadleftarrow\!\!\longrightarrow \overline{T}$, $C \longleftrightarrow \overline{C}$
Initialization: $buffer = \epsilon$.
Compute:

On $m = (\dots, \langle\texttt{MsgName}\rangle, \dots)$ or $m = (\langle\texttt{MsgName}\rangle, \dots)$ from T_{aux}:
- let n_{C} fresh $\mathbb{C}$ nonce
- store (n_{C}, m) in $buffer$ and leak $(n_{\mathsf{C}}, \texttt{Leak}\langle\texttt{MsgName}\rangle, |m|)$ to C

On $(n_{\mathsf{C}}, \texttt{Continue})$ from C:
- if $(n_{\mathsf{C}}, m) \notin buffer$ `abort`
- remove (n_{C}, m) from $buffer$ and send m to $\overline{T}_{aux}$

On m from $\overline{T}_{aux}$: forward m to T_{aux}.

To abstract from communication details, we model communication as functionalities. One important mechanism is TLS like end to end authenticated secure communication. Key exchange protocols and public-key infrastructures allow for the construction of such secure channels. For simplicity we model secure channels through an ideal uncorruptible functionality.

Secure Channel $\mathcal{I}_{\mathsf{sc}}$. We model an ideal secure channel, as a channel in which both the receiver and sender is authenticated. The ideal channel functionality $\mathcal{I}_{\mathsf{sc}}$ supports only request/response communication and only a single message can be sent at a time. We model corruption of sender and receiver through dummy users $\mathcal{D}_{\mathsf{S}_1} = \mathsf{Dummy}(\mathsf{sc}.\mathsf{S}_1, \mathsf{nsc}.\mathsf{S}_1, \overline{\mathcal{F}_{\mathsf{sc}}}.\mathsf{S}_1)$ and $\mathcal{D}_{\mathsf{S}_2} = \mathsf{Dummy}(\mathsf{sc}.\mathsf{S}_2, \mathsf{nsc}.\mathsf{S}_2, \overline{\mathcal{F}}_{\mathsf{sc}}.\mathsf{S}_2)$: $\mathcal{I}_{\mathsf{sc}} = \mathcal{D}_{\mathsf{S}_1} | \mathcal{F}_{\mathsf{sc}} | \mathcal{D}_{\mathsf{S}_2}$.

Listing 17 Functionality $\mathcal{F}_{\mathsf{sc}}$

Tapes: $\mathsf{sc}.\mathsf{S}_1 \twoheadleftarrow\!\!\not\to \overline{\mathsf{sc}}.\mathsf{S}_1$, $\mathsf{sc}.\mathsf{S}_2 \twoheadleftarrow\!\!\not\to \overline{\mathsf{sc}}.\mathsf{S}_2$
Initialization: $active = 1$.
Compute:

On $(\texttt{Send}, m)$ on $\mathcal{F}_{\mathsf{sc}}.\mathsf{S}_{2-i}$ where $active = 2 - i$
- set $active = 1 + i$; send $(\texttt{Send}, m)$ to $\mathcal{F}_{\mathsf{sc}}.\mathsf{S}_{1+i}$

On $(\texttt{Skip})$ on $\mathcal{F}_{\mathsf{sc}}.\mathsf{S}_{2-i}$ where $active = 2 - i$
- set $active = 1 + i$; send $(\texttt{Skipped})$ to $\mathcal{F}_{\mathsf{sc}}.\mathsf{S}_{2-i}$

Anonymous Channel $\mathcal{I}_{\mathsf{ac}}$. We model an ideal anonymous channel, as a channel in which the receiver is authenticated, but the sender is anonymous. Once a channel is established, the receiver is, however, guaranteed that he is talking to the same party.

The ideal system for realizing such a channel consists of multiple sender ITM instances $\mathcal{F}_{\mathsf{acS}}$ and one anonymizing receiver ITM instance $\mathcal{F}_{\mathsf{acR}}$ which corresponds to a trusted anonymization proxy. More formally $\mathcal{I}_{\mathsf{ac}} = !\mathcal{F}_{\mathsf{acS}} | \mathcal{F}_{\mathsf{acR}}$.

Listing 18 Functionality $\mathcal{F}_{\mathsf{acS}}$

Tapes: $\mathsf{ac.S} \twoheadleftarrow\!\!\!\longrightarrow \overline{\mathsf{ac}}.\mathsf{S}$, $\mathsf{nac.S} \longleftrightarrow \overline{\mathsf{nac}}.\mathsf{S}$, $\mathsf{nac.C} \longleftrightarrow \overline{\mathsf{nac}}.\mathsf{C}$, $\overline{\mathcal{F}}_{\mathsf{acR}} \longleftrightarrow\!\!\!\!\twoheadrightarrow \mathcal{F}_{\mathsf{acR}}$
Initialization: $state = sid = n_{\mathsf{C}} = n_{\mathsf{R}} = corrupted = corruptible = \epsilon$.
Compute:

On (sid', `Ready`) from $\mathsf{ac.S}$ where $state = \epsilon$:
- let $state$ = "ready"; let $corruptible = true$; let $sid = sid'$; send (sid, `Ready`) to $\mathsf{nac.S}$

On (sid, `Send`, m) on $\mathsf{ac.S}$ where $state$ = "ready" and $ssid' = ssid$
let $corruptible = false$; let n_{C} fresh $\mathbb{C}$ nonce
send (n_{C}, `LeakSend`) to $\mathsf{nac.C}$; rcv (n_{C}, `InfSend`) from $\mathsf{nac.C}$
let $state$ = "sent"; let n_{R} to fresh $\mathbb{S}$ nonce
send (n_{R}, `Send`, m) to $\mathcal{F}_{\mathsf{R}}.\mathsf{acS}$

On (n_{R}, `Reply`, m) from $\mathcal{F}_{\mathsf{R}}.\mathsf{acS}$ where $state$ = "sent"
let n_{C} fresh $\mathbb{C}$ nonce
send (n_{C}, `LeakReply`) to $\mathsf{nac.C}$; rcv (n_{C}, `InfReply`) from $\mathsf{nac.C}$
let $state$ = "ready"
send (sid, `Reply`, m) to $\mathsf{ac.S}$

Corruption: Corr ($corrupted, corruptible, state \neq \epsilon, \epsilon, \mathsf{ac.S}, \mathsf{nac.S}, \{\mathsf{ac.S}, \overline{F}_{\mathsf{acR}}\}$)
CheckAddress: Once sid is set check for it in all messages from $\mathsf{ac.S}$ Check for n_{C} respectively $ssid$, n_{R} in all messages from $\mathsf{nac.C}_1$ and $\mathcal{F}_{\mathsf{R}}.\mathsf{S}$ respectively.

Listing 19 Functionality $\mathcal{F}_{\mathsf{acR}}$

Tapes: $\mathsf{ac.R} \twoheadleftarrow\!\!\!\longrightarrow \overline{\mathsf{ac}}.\mathsf{R}$, $\mathsf{nac.R} \longleftrightarrow \overline{\mathsf{nac}}.\mathsf{R}$, $\mathcal{F}_{\mathsf{acR}} \longleftrightarrow\!\!\!\!\twoheadrightarrow \overline{\mathcal{F}}_{\mathsf{acR}}$
Initialization: $state = corrupted = corruptible = \epsilon$.
Compute:

On (`Ready`) from $\mathsf{ac.R}$ where $state = \epsilon$:
- let $state$ = "ready"; let $corruptible = true$; send (`Ready`) to $\mathsf{nac.R}$

On (n_{R}, `Send`, m) from $\mathcal{F}_{\mathsf{U}}.\mathsf{R}$ where $state$ = "ready":
- let $corruptible = false$; send (n_{R}, `Send`, m) to $\mathsf{ac.R}$

On (n_{R}, `Reply`, m) from $\mathsf{ac.R}$ where $state$ = "ready"
- send (n_{R}, `Reply`, m) to $\mathcal{F}_{\mathsf{acR}}$

A.2 Other Functionalities

We describe a common reference string and a key registration functionality for the IITM model.

Listing 20 Functionality $\mathcal{I}_{\mathsf{crs}}(\mathcal{R}, \{D_k\}_{k\in\mathbb{N}})$

Tapes: $\{\mathsf{crs}.R \twoheadleftarrow\!\!\!\longrightarrow \overline{\mathsf{crs}}.R\}_{R\in\mathcal{R}}$
Initialization: $params = \epsilon$.
Compute:

On (`GetParams`) on $\mathcal{I}_{\mathsf{crs}}.R$, $R \in \mathcal{R}$
- if $params = \epsilon$ sample $params \leftarrow D_k$.
- send (`Params`, $params$) to $\mathcal{I}_{\mathsf{crs}}.R$

Listing 21 Functionality $\mathcal{I}_{\mathsf{reg}}(\mathcal{R})$

Tapes: $\{\mathsf{reg.R} \twoheadleftarrow\!\!\!\longrightarrow \overline{\mathsf{reg}}.\mathsf{R}\}_{\in\mathcal{R}}$
Initialization: $state = \emptyset$. On (`Register`, v) from $\mathsf{reg.R} \in \mathcal{R}$
Compute:

- Records the value (R, v).

On (`Retrieve`, R) from $\mathsf{reg.R}' \in \mathcal{R}$
- If (R, v) is recorded then return (`Retrieve`, v) to $\mathsf{reg.R}'$.
- Otherwise send (`Retrieve`, $\perp$) to $\mathsf{reg.R}'$