Algebraic Zero Knowledge Contingent Payment

✉Javier Gomez-Martinez[1,2] [0009−0009−9495−2339] javier.gomez@imdea.org,
Dimitrios Vasilopoulos*[5] [0000−0003−2237−9202] vasilopo@eurecom.fr,
Pedro Moreno-Sanchez[2,3,4] [0000−0003−2315−7839] pedro.moreno@imdea.org, and
Dario Fiore[2] [0000−0001−7274−6600] dario.fiore@imdea.org

[1] Universidad Politécnica de Madrid
[2] IMDEA Software Institute
[3] VISA Research
[4] Max Planck Institute for Security and Privacy
[5] Independent Researcher

Abstract. In this work, we introduce Modular Algebraic Proof Contingent Payment (MAPCP), a novel zero-knowledge contingent payment (ZKCP) construction. Unlike previous approaches, MAPCP is the first that simultaneously avoids using zk-SNARKs as the tool for zero-knowledge proofs and HTLC contracts to exchange a secret for a payment atomically. As a result, MAPCP sidesteps the common reference string (*crs*) creation problem and is compatible with virtually any cryptocurrency, even those with limited or no smart contract support. Moreover, MAPCP contributes to fungibility, as its payment transactions seamlessly blend with standard cryptocurrency payments.
We analyze the security of MAPCP and demonstrate its atomicity, meaning that, (i) the buyer gets the digital product after the payment is published in the blockchain (buyer security); and (ii) the seller receives the payment if the buyer gets access to the digital product (seller security). Moreover, we present a construction of MAPCP in a use case where a customer pays a notary in exchange for a document signature. Moreover, we implement MAPCP and evaluate its performance in a use case where a customer pays a notary in exchange for a document signature. Our results show that MAPCP imposes a small computation and communication overhead even on commodity hardware, proving its practicality.

Keywords: Zero Knowledge Contingent Payment · Blockchain · Zero Knowledge Proof

1 Introduction

Cryptocurrencies are now increasingly accepted for purchasing digital products such as music, software, e-books, authentication tokens for websites, or mobile phone plans [29, 2, 13, 22, 30, 3, 6]. At first sight, this use case resembles the fair exchange problem [10] in which two parties want to swap digital goods such that

* The work was carried out while in employment of IMDEA Software Institute

neither can cheat the other. As an illustrative example, consider a buyer, who holds α funds on a blockchain and wants to transfer them to a seller in exchange for a digital product s (e.g., a solution to a Sudoku puzzle). Here, the buyer wants the seller to first provide the Sudoku's solution so that they can verify its correctness before transferring the funds, whereas the seller insists on not sending the solution to the buyer until they have received the funds.

The fair exchange problem is provably unsolvable without a trusted party [10]. However, existing blockchains enable a weaker version of fair exchange, where the blockchain assumes the role of the trusted third party to execute a *smart contract* faithfully. A smart contract permits a user to lock funds so that they are only released if the contract's logic is satisfied. Henceforth, in our running example, the buyer can lock funds into a smart contract that (i) encodes the Sudoku puzzle and its rules; and (ii) establishes that the seller can retrieve the funds by providing a valid solution to the Sudoku puzzle. While working in theory, the issue in practice with existing blockchains is that they either offer complex smart contract logic with high transaction fees (e.g., Ethereum) or limit the supported logic due to restrictions in the scripting language (e.g., Bitcoin) or the absence thereof of such scripting language (e.g., Monero).

As a first attempt to overcome this practical challenge, Maxwell [24] introduced the *zero-knowledge contingent payment* ZKCP protocol. Maxwell's ZKCP relies on a smart contract called *hash time-lock contract* (HTLC) [33] that is supported by some of the existing blockchains such as Ethereum or Bitcoin. An HTLC contract is defined w.r.t. two users, namely, the buyer and the seller, a hash value $h := \mathsf{H}(k)$ (where k is chosen uniformly at random), the amount of funds (i.e., α) and a timeout t, as follows: (i) If the *seller* produces a value k^* such that $\mathsf{H}(k^*) = h$ before t expires, the seller can withdraw α; (ii) after t expires, the buyer can get α refunded.

With HTLC in place, Maxwell's ZKCP runs in two phases, as illustrated in Fig. 1. First, during the *setup phase*, ① the seller encrypts the solution s using a fresh encryption key k into $ct \leftarrow \mathsf{Enc}(\mathsf{k}, s)$. The seller also computes $h \leftarrow \mathsf{H}(\mathsf{k})$ together with a zero-knowledge proof π certifying that (i) the ciphertext ct is the encryption of s under key k, (ii) h is the hash of k, and (iii) s is the product the buyer wants (e.g., a valid solution to the Sudoku puzzle). Upon receiving ct, h, and π, ② the buyer checks the validity of π .

During the execution phase, ③ the buyer creates an HTLC contract locking α funds. The HTLC is defined w.r.t. the buyer and seller as users, and h and time t as parameters. After the seller checks that the HTLC is included in the blockchain ④, the seller can claim the funds locked in the HTLC contract by submitting k to the contract before time t ⑤, thereby making k available on the blockchain. Then, the buyer reads the blockchain to learn k and uses it to decrypt ct and obtain the product s ⑥. On the other hand, if the seller does not claim the funds in the HTLC contract before t, the buyer can reclaim the funds held in the HTLC contract.

The HTLC contract ensures the *atomicity* of the exchange, meaning that payment to the seller is only made if the seller discloses the otherwise secret key k.

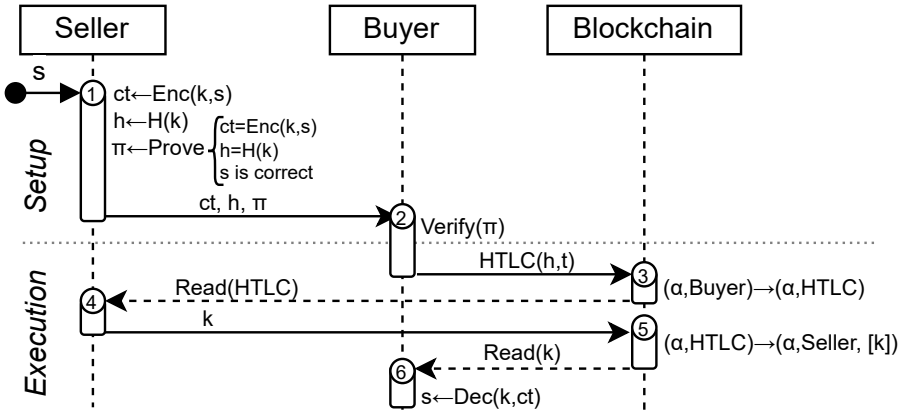

Fig. 1: Maxwell's ZKCP protocol. Solid arrows denote communication between parties and/or the blockchain whereas dashed arrows denote reads from the blockchain. Boxes represent local functionality executed by each user or the blockchain. In the blockchain, a tuple $(\alpha, \textit{user/contract})$ denotes that *user/contract* owns α funds, whereas square bracket denotes data stored in the blockchain. The dotted line denotes the separation between ZKCP phases.

Moreover, the protocol ensures that the seller cannot cheat by requiring the proof system to achieve *soundness*, i.e., the seller cannot create a valid proof if any of the statements (i), (ii) or (iii) is not true. Additionally, the proof system must be *zero-knowledge* to prevent the buyer from obtaining any sensitive information, that would reveal the key k or the product s, from the proof π itself.

1.1 Challenges in Maxwell's ZKCP

– *Trusted setup*: The necessary proofs are implemented using zk-SNARKS, a generic zero-knowledge proof system that permits one to prove statements of any NP relation, hence those needed in the first step of ZKCP. Practically efficient zk-SNARKS require running a setup algorithm that outputs a common reference string *crs* (i.e., a public parameter required to compute and verify proofs) along with a trapdoor. It is essential that a malicious party does not know the trapdoor to ensure both the soundness and zero-knowledge properties. This raises the question of who should initially run the setup algorithm. This problem was labeled as the *crs problem*. For instance, the introduction of a trusted third party to generate the *crs* is undesirable since the purpose of ZKCP is precisely to overcome the fair exchange problem in a trustless manner (or at least to overcome it while requiring the weakest trust assumption possible). Allowing the buyer or the seller to do the setup introduces subtle yet crucial security issues [8, 18]. A decentralized ceremony introduces computation and communication overhead, scaling linearly with the number of participants [27].

– *Interoperability*: The required HTLC contract is not available in cryptocurrencies that lack any kind of scripting functionality such as Monero. Nevertheless, compatibility with these cryptocurrencies remains of interest due to their strong privacy guarantees for transactions.
– *Fungibility*: The required HTLC contract leaves a trace on the blockchain, revealing details about the parties involved in the transaction and the amount of the contingent payment —sensitive information that the seller would prefer to keep confidential from competitors. To ensure fungibility, the transactions involved in ZKCP must blend seamlessly with standard transfers between users.

1.2 Contributions

In this work, we present Modular Algebraic Proof Contingent Payment (MAPCP), a novel protocol for ZKCP that builds upon the following key insights. First, we observe that the seller can replace the operation $h \leftarrow \mathsf{H}(\mathsf{k})$ by $h \leftarrow g^{\mathsf{k}}$, where g is the generator of a cyclic group. Second, using the commit-and-prove technique in [9] one can modularize the statement required in ZKCP into two simple statements: (i) s is a valid product, and (ii) ct is an encryption of s under key k and $h = g^{\mathsf{k}}$. Moreover, both statements share the clause "the commitment com commits to the value s" to ensure that the same value s is used in both statements. In this modular setting, we observe that the statement (ii) can consist solely of algebraic conditions with the appropriate constructions of the corresponding cryptographic schemes, enabling the use of zero-knowledge proofs without the *crs* problem (e.g., Sigma protocols). To support any arbitrary value s, we can still rely on zk-SNARKS to prove the statement (i). We observe that statement (i) can also be done algebraically for certain digital products (e.g., a digital signature), in which case we can use Sigma protocols (cf. Appendix A.5) to prove it and merge (i) and (ii) into a single proof using their composability properties.

Third, we observe that the HTLC contract leverages the blockchain to facilitate the fair exchange of a payment for the key k. Instead, in MAPCP we use a two-party protocol called PayForWitness, which differs from HTLC but still allows for a fair exchange between a payment and the key k when encoded as $h = g^{\mathsf{k}}$ (cf. Section 5). Such a two-party protocol only requires that the scripting language of the blockchain supports the authorization of transactions based on digital signatures, thereby increasing the interoperability and fungibility.

Fourth, we analyze the atomicity of MAPCP and show with cryptographic proofs that MAPCP achieves the notion of atomicity in [8]. This security notion intuitively encodes that (i) if the buyer pays the seller, then the buyer gets the product (i.e., security for the buyer); and (ii) if the seller delivers the product to the buyer, then the seller gets the payment (i.e., security for the seller).

Finally, we present a proof of concept of MAPCP for the use case where the buyer, who holds a digital document, wants to pay a notary (i.e., the seller) for its signature on the document. We observe that our implementation imposes a computation overhead in the order of milliseconds for both buyer and seller whereas the communication overhead is less than 90 KB, showing that our approach is practical even on commodity hardware.

1.3 Related Work

Security Analysis of Existing ZKCP. The initial ZKCP presented by Maxwell [24] was analyzed and shown insecure in consecutive academic works [8, 18], motivating thereby the design of alternative ZKCP protocols.

ZKCP *from Subversion Zero-knowledge.* To overcome the security issue in the ZKCP proposed by Maxwell, the notion of *subversion zero-knowledge* was proposed in [5]. This notion asserts that the security of the proofs can be maintained even if the adversary generates the *crs*. Adapting SNARKs to this notion requires conducting certain sanity checks on the *crs*. However, these checks could result in execution times of up to an hour [8], which hinders its practicality. Moreover, the protocol still relies on HTLC, hindering its interoperability and fungibility.

ZKCP *from Zero-knowledge Service Payments.* An alternative construction for ZKCP was proposed by Campanelli et al. [8] based on the notion of zero-knowledge service payment (ZKCSP). A ZKCSP differs from a ZKCP in the statement of the proof. In particular, the seller in ZKCSP proves a statement of the form "either I have product s and $h = \mathsf{H}(\mathsf{k})$, or I do not have product s and $h \neq \mathsf{H}(\mathsf{k})$". The security of this protocol was first studied under the notion of *witness indistinguishability* [17]. This notion asserts that even though the proof is not zero-knowledge, the leakage of information is common to all possible witnesses: no *meaningful* information is leaked. However, it was later shown that even with this notion, both the ZKCSP protocol and the ZKCP from ZKCSP constructions were still insecure [18]. A stronger notion called trapdoor subversion witness indistinguishability was introduced in [26] and used to prove the ZKCP-from-ZKCSP construction secure. Nevertheless, the ZKCP-from-ZKCSP construction still relies on HTLC contracts and zk-SNARKS, suffering from the *crs* creation problem and lacking practical properties such as interoperability and fungibility.

Smart Contract-based Generic Fair Exchange. In FairSwap [12], Dziembowski et al. leverage the Turing complete language only available in certain cryptocurrencies (e.g., Ethereum) to design a smart contract fairly exchanges a witness of an arbitrary NP statement for α funds. A subsequent work, OptiSwap [14] improves the efficiency of FairSwap by allowing the parties to conduct the exchange optimistically and using an interactive smart contract to resolve disputes if they arise. This line of work faces challenges regarding interoperability and fungibility since smart contract relies on logic that is only supported by ledgers with a Turing-complete language. Moreover, such a contract reveals the complete logic of the exchange in the publicly available ledger maintained by the blockchain.

ZKCP *Tailored for Data Exchange.* Another line of work has focused on designing ZKCP that lets the buyer pay the seller in exchange for a (possibly long) piece of data. The works in [20, 21] extend the HTLC contract to encode the release of several chunks of data atomically. Still, they rely on zk-SNARKS to prove the validity of the statement required by the buyer before funding the HTLC contract. Therefore, this line of work also faces challenges related to the *crs* creation problem, interoperability, and fungibility.

Similar to our work, a recent work [31] also moves away from HTLC contracts providing a construction that only requires digital signatures from the underlying blockchain. However, the proof system (which is based on KZG commitments) requires a trusted setup, hence encountering similar challenges with the *crs* problem as existing works on ZKCP. Moreover, this work relies on the assumption that the buyer has access to a commitment to the correct data (e.g., their own files outsourced to a cloud storage provider). While this assumption is valid for their use case (i.e., a user downloading data they stored on a paid server), it does not necessarily hold for every use case of ZKCP. For example, when buying a new book, the buyer does not have a commitment to the product beforehand.

In summary, our work proposes the first construction for ZKCP that permits to *simultaneously* depart from zk-SNARKS and the HTLC contract.

2 Key Ideas and Solution Overview

Maxwell's ZKCP permits a buyer to pay a seller in exchange for a product s such that $f(s) = 1$, where f: $\mathbf{S} \longrightarrow \{0,1\}$ is an efficiently computable predicate that encodes whether a product s is valid (e.g., s is a valid solution for a given Sudoku puzzle).

The first practical challenge in Maxwell's ZKCP is that it relies on zk-SNARKS to generate a zero-knowledge proof for the relation

$$\mathsf{R}_{\mathsf{H}}((f, h, ct); (s, \mathsf{k})) \Leftrightarrow f(s) = 1 \wedge h = \mathsf{H}(\mathsf{k}) \wedge ct = \mathsf{Enc}(\mathsf{k}, s)$$

In the following, we present the rationale behind our decision to move away from zk-SNARKs in our Modular Algebraic Proof Contingent Payment (MAPCP) protocol.

Modularizing the Proof. We observe that we can divide the aforementioned relation into two parts. First, a relation regarding the correctness of the product can be defined as

$$\mathsf{R}^1(f; s) \Leftrightarrow f(s) = 1$$

Second, a relation concerning the ability to retrieve the product following a successful payment in the HTLC contract:

$$\mathsf{R}^2_{\mathsf{H}}((h, ct); (s, \mathsf{k})) \Leftrightarrow h = \mathsf{H}(\mathsf{k}) \wedge ct = \mathsf{Enc}(\mathsf{k}, s)$$

This approach enables modular handling of the relations, as it allows the seller to create separate proofs for R^1 and $\mathsf{R}^2_{\mathsf{H}}$, rather than proving a statement in R_{H}. The buyer can then independently verify the validity of both proofs.

A technical subtlety with this approach is that proving that both R^1 and $\mathsf{R}^2_{\mathsf{H}}$ hold is not the same as proving that R_{H} holds. A malicious seller can use a value s to generate a proof for R^1 and a different value s' to generate a proof for $\mathsf{R}^2_{\mathsf{H}}$. Consequently, while both proofs may verify correctly, the product that the buyer receives (i.e., s') is not the desired one, as specified in R^1.

To overcome this, we leverage the commit and prove technique in [7]. We augment both R^1 and $\mathsf{R}^2_{\mathsf{H}}$ by including a commitment to the product s as follows:

$$\mathsf{R}^1((f, com_s); (s)) \Leftrightarrow f(s) = 1 \wedge com_s = \mathsf{Commit}(s)$$

$$\mathsf{R}^2_{\mathsf{H}}((h, ct, com_s); (s, \mathsf{k})) \Leftrightarrow h = \mathsf{H}(\mathsf{k}) \wedge ct = \mathsf{Enc}(\mathsf{k}, s) \wedge com_s = \mathsf{Commit}(s)$$

With access to the committed value com_s, the buyer can be assured that if both relations hold, they must be referring to the same product s.

With this modular approach to the proof established, we now explore how to prove each statement using a proof system that does not require a trusted setup (e.g., Sigma protocols).

Making the Proof Relation Amenable to Sigma Protocols. A concrete implementation of a proof for R^1 is application-dependent. On the one hand, if the product s has an algebraic structure (e.g., a Schnorr signature), then one could prove the complete R^1 as a Sigma protocol. On the other hand, for applications where the product has a more complex structure (e.g., a solution to a Sudoku puzzle), it might still be necessary to use a zk-SNARK for R^1.

As regards to $\mathsf{R}^2_{\mathsf{H}}$, we observe that we can slightly modify it such that it is amenable to be proven with Sigma protocols. Specifically, the challenge in using Sigma protocols lies in the substatement $h = \mathsf{H}(\mathsf{k})$. Our observation is that there exist Sigma protocols to prove knowledge of the plaintext of an encryption scheme (e.g., ElGamal [15]) and a commitment scheme (e.g., Pedersen Commitment [28]). Hence, instead of using a non-algebraic hash function like SHA-256 or Keccak, as in current implementations of the HTLC contract, we propose using an algebraic instance of the hash function, such as $h = g^{\mathsf{k}}$, where g is a generator of a cyclic group. In this manner, the proof for the thereby modified $\mathsf{R}^2_{\mathsf{H}}$ can always be instantiated using a Sigma protocol.

In summary, we have decomposed the proof statement into two substatements: the (modified) $\mathsf{R}^2_{\mathsf{H}}$ can always be proven using a Sigma protocol, while the concrete proof system for R^1 depends on the specific application and the concrete definition of f. When R^1 can be proven using a Sigma protocol, we completely move away from zk-SNARKs, eliminating the need for a trusted setup and improving the overall efficiency of the protocol. When R^1 requires a zk-SNARK, hence a trusted setup is still necessary, our approach offers improved efficiency compared to prior work since we have simplified the statement that needs to be proven with the zk-SNARK. This contributes to our goal to depart from a trusted setup and address the *crs* problem.

The reader might have noticed that modifying the hash function in $\mathsf{R}^2_{\mathsf{H}}$ makes it incompatible with HTLC contracts that are deployed in some of the current cryptocurrencies since they instantiate the hash function to a concrete function (i.e., SHA-256 or Keccak) other than the one we require (i.e., $h = g^{\mathsf{k}}$). Hence, our next step is to overcome this challenge.

Replace HTLC-based Payment by Adaptor Signature-based Payment. Incorporating the modified relation $\mathsf{R}^2_{\mathsf{H}}$ into a ZKCP protocol would require a cryptocurrency that supports a smart contract similar to HTLC, but with the condition

that the seller receives the payment after disclosing the discrete logarithm of h. One option is to design and deploy such a smart contract on cryptocurrencies that support an expressive scripting language (e.g., Ethereum). However, this approach hinders the interoperability and fungibility objectives of our work.

Instead, we propose to leverage adaptor signatures [4]. Adaptor signatures extend standard digital signatures by linking the creation of a signature with the leakage of a secret value. Intuitively, one can first compute a pre-signature $\hat{\sigma}$ with respect to some public statement (e.g., $h = g^{\mathsf{k}}$). The pre-signature can be adapted into a standard digital signature by anyone who knows the secret value related to the public statement (e.g., k in our case). Finally, with the pre-signature and the signature, one can extract the secret value.

Note that the public statement and the secret value cannot be arbitrary, but rather, there must exist a certain relation between public statements and secret values. We will refer to this relation as R_{AS}. An example of such a relation is the discrete logarithm, defined as $\mathsf{R}_{\mathsf{AS}}(h; \mathsf{k}) \Leftrightarrow h = g^{\mathsf{k}}$.

There exist adaptor signature schemes that extend the widely used ECDSA and Schnorr digital signatures [4], which are employed in nearly all cryptocurrencies to authorize transactions. Notably, some cryptocurrencies, like Monero, do not support hash-based payments but do support payments based on linkable ring signatures for which a construction of adaptor signatures exists [32, 25]. Hence, a ZKCP protocol that replaces HTLC-based payments with adaptor signature-based payments would be compatible with most cryptocurrencies, including e.g., Monero. This contributes to our goal of improving interoperability.

Furthermore, payments using adaptor signatures blend with those made with standard digital signatures, meaning both payments appear identical when published in the blockchain. As a consequence, it becomes harder for an observer of the blockchain to determine whether a payment is the result of the ZKCP protocol or a standard transaction between two users. This contributes to our goal of improving fungibility.

In a bit more technical detail, by leveraging adaptor signatures it is possible to design a protocol in which the buyer pays the seller in exchange for the discrete logarithm of the value $h = g^{\mathsf{k}}$. We will call this protocol PayForWitness and the high-level idea of how this protocol works is as follows:

1. The buyer transfers coins into an account shared with the seller so that the funds can be transferred further when both buyer and seller agree to it.
2. The buyer uses their signing key to create a pre-signature $\hat{\sigma}$ of a transaction with respect to the public statement h. This transaction transfers funds from the shared account to the seller and it is required to be signed by the buyer to be valid. Then, the buyer sends this pre-signature to the seller.
3. The seller uses the secret k to adapt the pre-signature $\hat{\sigma}$ into a signature σ. The latter is a valid signature of the transaction under the buyer's verification key. The seller submits the transaction and the signature to the blockchain.
4. The blockchain validates that σ is a correct signature of the transaction and transfers the funds from the shared account to the seller.
5. Thereafter, the buyer can read the signature σ from the blockchain. Using both the signature and the pre-signature, buyer can extract the secret k.

Therefore, at the end of this protocol, the seller gets the funds and the buyer gets the secret k in exchange.

Putting Everything Together. In summary, we have a protocol that allows the buyer to pay in exchange for the discrete logarithm of a group element $h = g^{\mathsf{k}}$. In addition, we have a modular approach to proving, in zero-knowledge, that this discrete logarithm can be used to decrypt a ciphertext into the correct product. Fig. 2 depicts the steps of how a ZKCP protocol using these modifications would work at a high level. We will describe these steps where the index numbers refer to those of the figure.

During the setup phase, ① the seller samples a key k, encrypts the product s into a ciphertext ct and computes $h := g^{\mathsf{k}}$. They also compute a commitment to the product com_s and generate proofs π_1 (regarding the correctness of the product) and π_2 (regarding the payment and encryption methods). Upon receiving π_1, π_2, ct, com_s and h, ② the buyer verifies both proofs and sends to the seller a pre-signature σ' of the payment transaction. Next, ③ the buyer submits to the blockchain a transaction that transfers the α funds to an account shared between buyer and seller.

During the execution phase, ④ the seller adapts the pre-signature into a signature and submits this signature of the payment to the blockchain. ⑤ the blockchain verifies the signature and transfers the funds from the shared account to the seller. Finally, ⑥ the buyer retrieves the signature from the ledger and extracts the key k and decrypts the product s.

3 Preliminaries

Notation. We denote the security parameter by $n \in \mathbb{N}$, by which each cryptographic scheme and adversary is parameterized. We denote by $\mathsf{negl}(n)$ a *negligible* function. A function $f : \mathbb{N} \to \mathbb{R}$ is negligible if its absolute value is smaller than the inverse of any polynomial (i.e., if $\forall d\ \exists k_0\ \forall n \geq k_0 : |\mathsf{negl}(n)| \leq 1/n^d$). We denote by $x \leftarrow_{\$} \mathcal{X}$ the uniform sampling of the variable x from the set $\mathcal{X}$. We write $x \leftarrow \mathcal{A}(y)$ to denote that a probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ on input y outputs x. If $\mathcal{A}$ is a deterministic polynomial time (DPT) algorithm, we use the notation $x := \mathcal{A}(y)$. We use the notation $s \leftarrow s_1 + s_2$ for the assignment of computation results. We use the notation $\sigma := (\sigma_1, \sigma_2)$ for parsing a tuple σ composed of two elements σ_1 and σ_2. We use the dot notation to access the elements of a tuple (e.g., we denote by $\sigma.\sigma_1$ the element σ_1 of σ). We denote by $[0, \ldots, n]$ the range of numbers from 0 to n. We denote by $|\sigma|$ the number of bits of an element σ.

Hard Languages. Let R be a relation with the associated language $L_{\mathsf{R}} := \{x | \exists w \text{ s.t. } (x, w) \in \mathsf{R}\}$. We say R is *hard* if it is efficiently decidable, there exists an efficient instant sampling function GenR and it is one-way (i.e., it is computationally infeasible for an efficient adversary to output a witness w for a given statement x such that $(x, w) \in \mathsf{R}$).

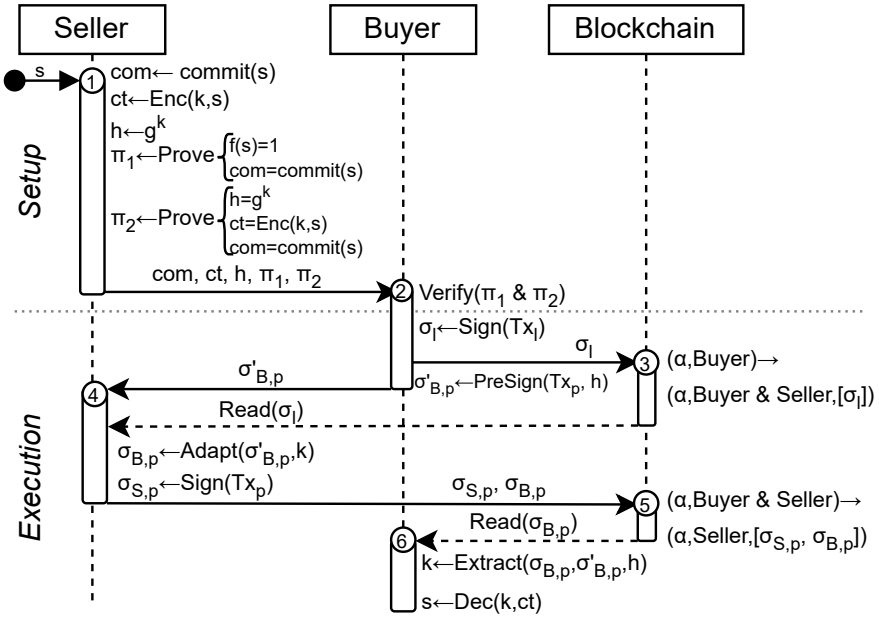

Fig. 2: Illustrative example of our ZKCP protocol. We use the same notation as in Fig. 1. Here, tx_p is the transaction that pays α coins from buyer to seller.

Hash Functions. A hash scheme H is an algorithm H, where $h \leftarrow \mathsf{H}(m)$ computes a hash value on input a message m. We require that the hash scheme satisfies the properties of *collision resistance*, *preimage resistance*, and *second-preimage resistance*.

Non-Interactive Zero Knowledge Argument Systems. A non-interactive zero-knowledge argument system NIZK consists of three algorithms $\mathsf{NIZK} := (\mathsf{CrsGen}, \mathsf{Prove}, \mathsf{Verify})$, where CrsGen is the public parameter (i.e., common reference string) generation algorithm; $\pi \leftarrow \mathsf{Prove}(crs, x, w)$ is the prover algorithm for a statement x and a witness w; and $\{0,1\} \leftarrow \mathsf{Verify}(crs, x, \pi)$ is the verification algorithm. A NIZK argument system allows a prover to convince a verifier, using a proof π, about the existence of a witness w for a statement x without revealing any information apart from the fact that it knows the witness w. We require the NIZK argument system to satisfy the usual properties of *completeness*, *computational soundness* and *computational zero-knowledge*.

Commitment Schemes. A commitment scheme COM is a tuple of algorithms $\mathsf{COM} := (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$, where Setup is the public parameters generation algorithm; $(com, open) \leftarrow \mathsf{Commit}(p, x)$ is the commitment creation algorithm for a value x; and $\{0,1\} \leftarrow \mathsf{Open}(p, x, com, open)$ is the verification algorithm. A commitment scheme should satisfy the properties of *binding* and *hiding*.

Public Key Encryption. A public key encryption scheme PKE is a tuple of algorithms $\mathsf{PKE} := (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^n)$ is the key generation algorithm; $ct \leftarrow \mathsf{Enc}(\mathsf{ek}, m)$ is the encryption algorithm on input a public key ek and a message m; and $\{m, \perp\} \leftarrow \mathsf{Dec}(\mathsf{dk}, ct)$ is the decryption algorithm. A public key encryption scheme should satisfy indistinguishability under chosen-plaintext attacks.

Digital Signatures. A digital signature DS is a tuple of algorithms $\mathsf{DS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$, where $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$ is the key generation algorithm; $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ is the signing algorithm on input the signing key sk and a message m; and $\{0, 1\} \leftarrow \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$ is the verification algorithm. A digital signature scheme should satisfy *existential unforgeability under an adaptative chosen-message attack*.

Adaptor Signatures. An *adaptor signature scheme* AS is a tuple of algorithms $\mathsf{AS} := (\mathsf{KeyGen}, \mathsf{PreSign}, \mathsf{PreVrfy}, \mathsf{Adapt}, \mathsf{Extract})$ defined with respect to a hard relation R and a digital signature scheme DS. For every statement/witness pair $(x, w) \in \mathsf{R}$, key pair $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$ and a message m, we have that $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, m, x)$ is a pre-signature; and $\sigma \leftarrow \mathsf{Adapt}(\hat{\sigma}, w)$ is a valid signature; and (pre-)verification holds under vk and m for $\hat{\sigma}$ and σ, respectively. Furthermore, it holds that $w \leftarrow \mathsf{Extract}(\hat{\sigma}, \sigma, x)$. An adaptor signature scheme should satisfy the notions of *existential unforgeability under a chosen-message attack*, *pre-signature adaptability* and *witness extractability*.

UTXO Model. Following the notation in [4], in the UTXO model, coins are held in *outputs*. An output θ is a tuple (α, φ) where α denotes the number of coins associated with θ and φ defines the conditions (also known as scripts) that need to be satisfied to spend the output.

A *transaction* transfers coins across outputs, meaning it maps (possibly multiple) existing outputs to a list of new outputs. The existing outputs included in the transaction are called *transaction inputs*. In other words, transaction inputs are those tied with previously unspent transaction outputs of older transactions. Formally, a transaction tx is a tuple $(txid, ins, outs, wits)$ where $txid \in \{0,1\}^*$ is the unique identifier of tx and is calculated as $txid := \mathsf{H}(\llbracket tx \rrbracket)$, where H is a hash function modeled as a random oracle and $\llbracket tx \rrbracket$ is the *body of the transaction* defined as $\llbracket tx \rrbracket := (ins, outs)$; ins is a vector of strings identifying all transaction inputs; $outs := (\theta_1, \ldots, \theta_n)$ is a vector of new outputs; and $wits \in \{0,1\}^*$ contains the witness allowing to spend the transaction inputs. For readability, we write $\llbracket tx \rrbracket := (ins, outs)$ as $\llbracket tx \rrbracket := ins \rightarrow outs$.

The conditions of transaction outputs might be complex. Hence, we describe our notation for some of them. We denote by $\varphi := \mathsf{vk}$ the condition of signature verification w.r.t. vk on the body of the spending transaction. We denote by $\varphi := h$ the condition of providing a value $m \in \mathcal{M}$ such that $h = \mathsf{H}(m)$. We denote by $\varphi := +t$ the condition that at least t rounds passed since the publication of the transaction containing this output. Finally, an output can require a conjunction or disjunction of several conditions. We denote the conjunction of conditions

by $\varphi_1 \wedge \ldots \wedge \varphi_n$ for some $n \in \mathbb{N}$. We denote the disjunction of conditions by $\varphi_1 \vee \ldots \vee \varphi_n$ for some $n \in \mathbb{N}$.

We say that a transaction tx is *valid* if the sum of the coins in the transaction inputs is at least the sum of the coins in the transaction outputs, i.e., $\sum_{\theta_i \in tx.ins} \theta_i.\alpha \geq \sum_{\theta_j \in tx.outs} \theta_j.\alpha$.

We say that a transaction tx is *authorized* if, for each input $\theta_i \in tx.ins$, the transaction contains a witness in $tx.wits$ such that the condition $\theta_i.\varphi$ is satisfied.

A *ledger* $\mathcal{L}$ maintains the set of all transactions. We assume users can interact with such a ledger with the following operations:

$\mathsf{Submit}(tx)$. This operation takes as input a transaction tx. If the transaction is valid, authorized, and none of its inputs appear in a different transaction $tx' \in \mathcal{L}$, then tx is appended to $\mathcal{L}$. Otherwise, tx is discarded.

$\mathsf{LedgerRead}(\lfloor tx \rfloor)$. This operation takes as input the body of a transaction $\lfloor tx \rfloor$. It returns $tx \in \mathcal{L}$ such that $tx.txid = \mathsf{H}(\lfloor tx \rfloor)$ if it exists, or $\perp$ otherwise.

4 Problem Description: Contingent Payment

Environment. A zero-knowledge contingent payment (ZKCP) involves a digital product s and two parties: buyer B and seller S. The buyer owns a key pair $(\mathsf{vk}_B, \mathsf{sk}_B)$ whereas the seller owns a key pair $(\mathsf{vk}_S, \mathsf{sk}_S)$. There exists a blockchain where the buyer owns α funds. This means that there exists an *unspent* output $\theta := (\alpha, \varphi)$ where $\varphi := \mathsf{vk}_B$. Moreover, to ease readability, we assume that the blockchain supports transaction authorization verification based on a single digital signature (i.e., $\varphi := \mathsf{vk}$), multi-signatures (i.e., $\varphi := \mathsf{vk}_0 \wedge \ldots \wedge \mathsf{vk}_n$), and timelocks (i.e., $\varphi := +t$).

Remark 1. Our approach is compatible with blockchains that only support transaction authorization verification based on a single digital signature (i.e., $\varphi := \mathsf{vk}$) since multi-signatures (i.e., $\varphi := \mathsf{vk}_0 \wedge \ldots \wedge \mathsf{vk}_n$) can be handled in a two-party protocol and encoded as a single public key [16] and timelock smart contracts can be replaced by cryptographic timelock puzzles [23].

Notably, we do not assume that the blockchain supports transaction authorization verification based on hash pre-images (i.e., $\varphi := h$), hence we have the same minimal assumptions on the blockchain as prior works on protocols with interoperability across ledgers [34]. Finally, we assume that the predicate $f\colon \mathbf{S} \longrightarrow \{0,1\}$ does not leak the product s. More formally, the predicate f satisfies the following one-way property:

$$\forall\ \mathsf{PPT}\ \mathcal{A}, \Pr[f(s) = 1 | s \leftarrow \mathcal{A}(f)] < \mathsf{negl}(n)$$

Threat Model. The two parties carrying out a zero-knowledge contingent payment, namely the buyer and the seller, are mutually distrustful. Moreover, we assume that the blockchain accepts a transaction only if it is valid, correctly

authorized, and none of its inputs appear already in a previous transaction (i.e., they are not already spent).

We next define the zero-knowledge contingent payment protocol as an instance of a fair exchange protocol, as defined in [8].

Definition 1 (Zero-knowledge contingent payment). *A zero-knowledge contingent payment protocol is a two-party communication protocol* $\mathsf{ZKCP} := \langle S(s, f, (\mathsf{vk}_S, \mathsf{sk}_S), \mathsf{vk}_B), B(f, \mathsf{vk}_S, (\mathsf{vk}_B, \mathsf{sk}_B)\rangle$*, where:*

- *seller runs algorithm S on input the product s, the predicate f: $\mathbf{S} \longrightarrow \{0,1\}$, the seller key pair $(\mathsf{vk}_S, \mathsf{sk}_S)$ and the buyer public key vk_B,*
- *buyer runs algorithm B on input the predicate f, the buyer key pair $(\mathsf{vk}_B, \mathsf{sk}_B)$ and the seller public key vk_B.*

A ZKCP *is* complete *if the execution of the protocol by honest parties results in (i) the seller getting α from the buyer, and (ii) the buyer obtaining product s.*

4.1 Security Definitions

Here we recall the security notions of a zero-knowledge contingent payment, namely, *Extractability* and *Zero-knowledge*, as defined in [8].

Let $View_B$ be the view of the buyer on an execution of the ZKCP protocol, defined as $View_B(s, f) := Coins_B \parallel Messages(\langle S(\ldots), B(\ldots)\rangle) \parallel Out(\langle S(\ldots), B(\ldots)\rangle)$, where $Coins_B$ denotes the randomness used by the buyer, *Messages* denotes the messages exchanged between the parties executing the ZKCP protocol, and *Out* denotes the outputs of all parties after executing the ZKCP protocol.

Definition 2 (Extractability). *A* ZKCP *is* extractable *if at the end of the protocol the balance of any possibly malicious and efficient seller algorithm $\widehat{S}$ increases with non-negligible probability, then there exists an efficient extractor algorithm* Ext *that on input the predicate f, and the view of the buyer $View_B$, outputs $\widehat{s}$ such that $f(\widehat{s}) = 1$.*

Intuitively, this property ensures that, at the end of the protocol, if the seller gets the α from the buyer, then the buyer must be able to extract product s.

Definition 3 (Zero-knowledge). *A* ZKCP *is* zero-knowledge *if for any possibly malicious and efficient $\widehat{B}$, there exists an efficient simulator algorithm $\mathsf{Sim}^{\widehat{B}}$ which, on input the predicate f, outputs a distribution which is computationally indistinguishable from $View_{\widehat{B}}(s, f)$.*

Intuitively, this property guarantees that any potentially malicious $\widehat{B}$ does not inadvertently gain information about the secret input s during the protocol execution unless $\widehat{B}$ transfers α to S.

5 Our Protocol for Modular ZKCP (MAPCP)

MAPCP is divided into two phases, namely, *setup* and *execution*. During the *setup* phase (cf. Section 5.1), the seller runs ZKCPProve to encrypt the product and prove its correctness. The buyer then verifies the validity of the proof using ZKCPVerify. During the *execution* phase (cf. Section 5.2), the buyer and the seller engage in the PayForWitness protocol where the buyer pays the seller in exchange for the decryption key, thereby obtaining the product s.

5.1 Setup Phase: Product Setup and Proofs

Building Blocks. We require a public key encryption scheme PKE, a commitment scheme COM, and a non-interactive zero knowledge proof system NIZK with the properties described in Section 3.

Overview of our Construction. Recall that during the product setup phase, the seller encrypts the product s into a ciphertext ct and must convince the buyer of two key points: (i) the ct indeed encrypts the product s; and (ii) the product s satisfies the predicate f. Furthermore, as we will discuss in the *execution* phase (cf. Section 5.2), the seller must also convince the buyer of a third key point: (iii) the encryption and decryption key pair has been generated according to the key generation algorithm of the public key encryption scheme.

Technically, these three key points can be encoded in the following three relations:

- $\mathsf{R}_{\mathsf{PKE}}((ct, \mathsf{ek}); (s, r)) \Leftrightarrow (ct := \mathsf{Enc}(\mathsf{ek}, s; r))$.
- $\mathsf{R}_{prod}(f\,; s) \Leftrightarrow (f(s) = 1)$.
- $\mathsf{R}_{\mathrm{Key}}(\mathsf{ek}; (\mathsf{dk}, r')) \Leftrightarrow ((\mathsf{dk}, \mathsf{ek}) := \mathsf{KeyGen}(1^n; r'))$.

The seller can leverage a NIZK scheme to convince the buyer that these three relations hold. While prior works rely on a single, general purpose NIZK instance to prove the three relations together (e.g., using zk-SNARKS), our approach relies on two different NIZK instances, one for the R_{prod} and one for $\mathsf{R}_{\mathsf{PKE}}$ and $\mathsf{R}_{\mathrm{Key}}$. As discussed in Section 2, this is interesting because while the NIZK instance to prove R_{prod} may still need to be general-purpose if the predicate f is complex, the NIZK instance to prove $\mathsf{R}_{\mathsf{PKE}}$ and $\mathsf{R}_{\mathrm{Key}}$ can be simpler, e.g., a simple Sigma protocol when PKE is instantiated as e.g., ElGamal with keys as group elements in a cyclic group. However, running the two NIZK instances in parallel does not ensure that the product s used in R_{prod} is the same as the one used in $\mathsf{R}_{\mathsf{PKE}}$. To account for that, we consider a fourth relation defined as $\mathsf{R}_{\mathsf{COM}}((com_s, p); (open_s, s, r'')) \Leftrightarrow ((com_s, open_s) := \mathsf{Commit}(p, s; r''))$ to prove that a given commitment com_s commits to the product s. By using the same commitment in both NIZK instances, the buyer is assured that both instances are proven with regard to the same product s.

More precisely, our first construction for ZKCPProve and ZKCPVerify (cf. Fig. 5) relies on two NIZK instances, namely, $\mathsf{NIZK}_{\mathsf{R}_\Sigma}$ and $\mathsf{NIZK}_{\mathsf{R}_\Omega}$, for the relations:

$\mathsf{ZKCPProve}(s, f, \mathbf{pp})$	$\mathsf{ZKCPVerify}(f, \mathbf{pp}, \boldsymbol{\pi}, \mathsf{ek}, ct)$
$\mathbf{pp} := (p, crs_{\Omega}, crs_{\Sigma})$	$\mathbf{pp} := (p, crs_{\Omega}, crs_{\Sigma})$
$(com_s, open_s) \leftarrow \mathsf{COM.Commit}(p, s)$	$\boldsymbol{\pi} := (\pi_{\Omega}, \pi_{\Sigma})$
$(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{PKE.KeyGen}(1^n)$	$ct := (ct, com_s)$
$ct \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}, s)$	$b_0 \leftarrow \mathsf{NIZK}_{\mathsf{R}_{\Omega}}.\mathsf{Verify}(crs_{\Omega}, \pi_{\Omega},$
$\pi_{\Omega} \leftarrow \mathsf{NIZK}_{\mathsf{R}_{\Omega}}.\mathsf{Prove}(crs_{\Omega}, (f, com_s, p),$	$\quad (f, com_s, p))$
$\quad (s, open_s, r''))$	$b_1 \leftarrow \mathsf{NIZK}_{\mathsf{R}_{\Sigma}}.\mathsf{Verify}(crs_{\Sigma}, \pi_{\Sigma},$
$\pi_{\Sigma} \leftarrow \mathsf{NIZK}_{\mathsf{R}_{\Sigma}}.\mathsf{Prove}(crs_{\Sigma}, (ct, \mathsf{ek}, com_s, p),$	$\quad (ct, \mathsf{ek}, com_s, p))$
$\quad (r, \mathsf{dk}, r', open_s, s, r''))$	$\mathbf{return}\ b_0 \wedge b_1$
$\mathbf{return}\ ((\pi_{\Omega}, \pi_{\Sigma}), \mathsf{ek}, (ct, com_s))$	

Fig. 3: Algorithms ZKCPProve and ZKCPVerify.

- $\mathsf{R}_{\Sigma}((ct, \mathsf{ek}, com_s, p); (r, \mathsf{dk}, r', open_s, s, r'')) \Leftrightarrow (((ct, \mathsf{ek}), (s, r)) \in \mathsf{R}_{\mathsf{PKE}} \wedge (\mathsf{ek}, (\mathsf{dk}, r')) \in \mathsf{R}_{\mathrm{Key}} \wedge ((com_s, p), (open_s, s, r'')) \in \mathsf{R}_{\mathsf{COM}})$
- $\mathsf{R}_{\Omega}((f, com_s, p); (s, open_s, r'')) \Leftrightarrow (f(s) = 1 \wedge ((com_s, p), (open_s, s, r'')) \in \mathsf{R}_{\mathsf{COM}})$

Furthermore, we observe that if the relation R_{prod} can be proven with a simple Sigma protocol, we can resort to a single NIZK instance to prove the following relation:

$$\mathsf{R}_{\mathsf{merged}}((f, ct, \mathsf{ek}); (r, \mathsf{dk}, r', s)) \Leftrightarrow (((ct, \mathsf{ek}), (s, r)) \in \mathsf{R}_{\mathsf{PKE}} \wedge (\mathsf{ek}, (\mathsf{dk}, r')) \in \mathsf{R}_{\mathrm{Key}} \wedge (f, s) \in \mathsf{R}_{prod})$$

In Section 6 we show a concrete example where R_{prod} can be proven as a Sigma protocol. It is important to note that this construction differs from previous approaches that rely on a single NIZK instance, as those methods require the instance to be a general-purpose one (e.g., zk-SNARKs).

5.2 Execution Phase: PayForWitness

Building Blocks. We require an adaptor signature scheme AS. We assume that the time that spans between the submission and the publication of a transaction is bounded by Δ_t.

Notation. Henceforth, we use the following notation: sk, vk, and tx denote signing keys, verification keys, and transactions, respectively. These symbols may include up to two subscripts: The first subscript indicates the party who owns the key (B for *Buyer* and S for *Seller*). The second subscript, if present, denotes the associated transaction, using l for the lock transaction, p for the pay transaction, and r for the recover transaction. For instance, $\mathsf{vk}_{B,l}$ refers to the buyer's verification key controlling the input of the lock transaction; and $\sigma_{B,p}$ refers to the signature that authorizes the pay transaction with respect to the buyer's verification key.

Transactions. To ease the readability we introduce the transactions that we use in the execution phase. These are:

- The lock transaction tx_l: $(\alpha, \mathsf{vk}_{B,l}) \rightarrow (\alpha, (\mathsf{vk}_{B,p} \wedge \mathsf{vk}_{S,p}) \vee (\mathsf{vk}_{B,r} + t))$ moves α funds from an output controlled by B to a shared output that is jointly controlled by B and S.
- The pay transaction tx_p: $(\alpha, (\mathsf{vk}_{B,p} \wedge \mathsf{vk}_{S,p}) \vee (\mathsf{vk}_{B,r} + t)) \rightarrow (\alpha, \mathsf{vk}_S)$ moves α funds from the output that is jointly controlled by B and S to an output controlled by S.
- The refund transaction tx_r: $(\alpha, (\mathsf{vk}_{B,p} \wedge \mathsf{vk}_{S,p}) \vee (\mathsf{vk}_{B,r} + t)) \rightarrow (\alpha, \mathsf{vk}_B)$. If time t has passed, it moves α funds from the output that is jointly controlled by B and S to an output controlled by the B.

We explain the need to lock the funds using the tx_l transaction in Appendix B .

Overview of our Construction. The goal of the PayForWitness is to let B pay S α funds in exchange for a witness w. To achieve this, they engage in a protocol where B has as input the private keys $\mathsf{sk}_{B,l}$, $\mathsf{sk}_{B,p}$, and $\mathsf{sk}_{B,r}$ which are the signing keys to be used to sign transactions tx_l, tx_p, and tx_r, respectively; whereas S has as input the witness w and the private key $\mathsf{sk}_{S,p}$ used to sign transaction tx_p. Hereby, we assume that the corresponding verification keys and the aforementioned transactions are available to both parties.

In this setting, the PayForWitness protocol works as illustrated in Fig. 4 and described next. We require that $(x\ ; w) \in \mathsf{R}_{\mathsf{AS}}$ and that the adaptor signature scheme extends a digital signature scheme which can be used to authorize transactions in the blockchain. Moreover, we assume that in the blockchain there exists an output holding α funds. This output is controlled by the buyer and it is associated with the verification key $\mathsf{vk}_{B,l}$. During the protocol execution, the blockchain verifies the authorization of the transactions submitted by B and S. If valid, the blockchain adds them to the publicly available ledger.

1. B executes the LockPayment algorithm to authorize and submit tx_l. α funds are then locked in an output that is shared between B and S. To spend these funds, both signatures from both B ($\sigma_{B,p}$) and S ($\sigma_{S,p}$) are required, or after time t, only B's signature ($\sigma_{B,r}$) is needed. B then computes a pre-signature $\hat{\sigma}_{B,p}$ for the transaction tx_p w.r.t. statement x and forwards it to S.
2. S waits Δ_t (i.e., the upper bound on the time it takes for a transaction to be included in the blockchain) and checks that B previously included tx_l in the blockchain. If so, S adapts the pre-signature into a valid signature $\sigma_{B,p}$ by executing the Adapt algorithm of the adaptor signature scheme using the witness w. Thereafter, S generates their own signature $\sigma_{S,p}$ and submits tx_p along with both signatures.
3. If after waiting $2\Delta_t$, B sees that tx_p is published on the blockchain, B executes the Extract algorithm of the adaptor signature scheme using the pre-signature $\hat{\sigma}_{B,p}$, the signature $\sigma_{B,p}$, and the statement x to extract the witness w. Otherwise, B generates a signature $\sigma_{B,r}$ for the tx_r transaction and submits it to reclaim their funds.

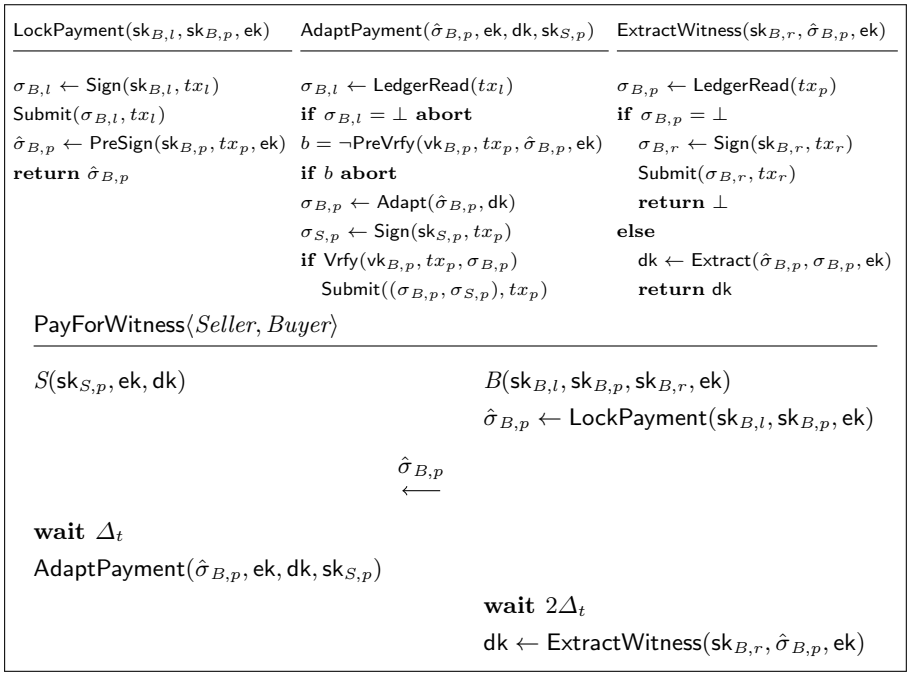

Fig. 4: On the bottom, the PayForWitness protocol. On the top: the algorithms required by the protocol. For readability, we assume that all transactions and public keys are available to both B and S.

Property of the Adaptor Signature Hard Relation. The idea of the protocol is that the *Buyer* buys the witness w which can later be used to decrypt the ciphertext and get the secret s. The *Buyer* learns this witness by extracting it from a pre-signature $\hat{\sigma}$ and a signature σ using the $\mathsf{Extract}$ algorithm of the adaptor signature scheme. Even if the pre-signature is correctly pre-signed with respect to a statement x such that, $(x, w) \in \mathsf{R}_{\mathsf{AS}}$, we need to enforce that the $\mathsf{Extract}$ algorithm returns exactly that witness w which will allow *Buyer* to decrypt the secret. In other words, we need to avoid the event that $\widehat{w} \leftarrow \mathsf{Extract}(\hat{\sigma}, \sigma, x)$ such that $(x, \widehat{w}) \in \mathsf{R}_{\mathsf{AS}}$ but $w \neq \widehat{w}$. If such event takes place, then the buyer will not be able to decrypt the correct product s. To address this, we require that the hard relation R_{AS} satisfies the following property that ensures that there is only one witness for a given statement:

Definition 4 (Injectiveness). *A hard relation* R *is* injective *if for all statements x, it holds that:* $(x, w) \in \mathsf{R} \wedge (x, \widehat{w}) \in \mathsf{R} \implies w = \widehat{w}$.

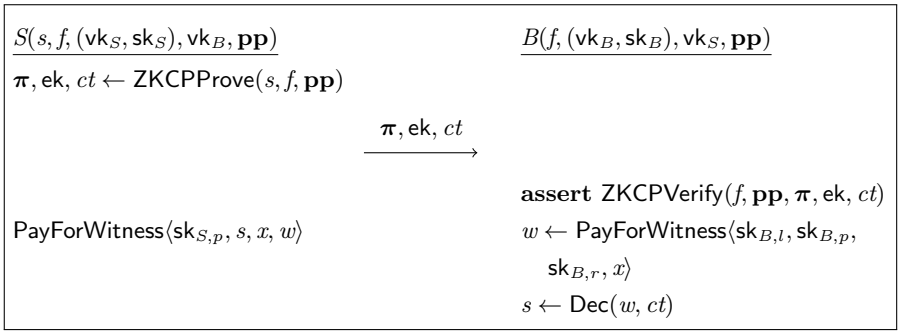

Fig. 5: The ZKCP-Protocol between parties *Seller* and *Buyer*. Here, we assume that **pp** is a common input to both parties.

5.3 Everything Together: Our Modular Algebraic Proof Contingent Payment

Our Modular Algebraic Proof Contingent Payment protocol is depicted in Fig. 5. The seller begins by generating proofs using the ZKCPProve algorithm, resulting in a vector $\boldsymbol{\pi}$, which includes π_Σ and π_Ω; the encryption key ek, that is the statement of the R_{AS} relation; and a vector of ciphertexts ct, containing both the product s and its commitment. The seller sends these to the buyer, who verifies the proofs and is now confident that the ciphertext contains the desired product. The buyer then only needs to purchase the decryption key dk corresponding to ek, and thus, the seller and buyer proceed with the PayForWitness protocol. Upon completion, the seller receives the α funds, and the buyer obtains the decryption key, which they use to decrypt ct and retrieve s.

Security Analysis Next, we state our security theorems and provide a sketch of our security analysis. We defer the formal security proofs to Appendix D .

Theorem 1. *If the predicate f is one-way; the adaptor signature scheme has existential unforgeability and witness extractability; the hard relation of the adaptor signature scheme* R_{AS} *is injective; the commitment scheme is binding and hiding; the encryption scheme has IND-CPA; and the NIZKs are knowledge-sound and zero-knowledge, then the* ZKCP-Protocol *defined in Fig. 5 is secure as defined in Section 4.1.*

Proof sketch 1 (Extraction for ZKCP-Protocol) *It follows from the security of the adaptor signature scheme and the knowledge-soundness of the non-interactive zero-knowledge proofs:*

Assume that a malicious Seller $\mathcal{A}$ *can get the* α *funds. We want to prove that the honest Buyer can then extract the secret s such that* $f(s) = 1$.

$\mathcal{A}$ *getting the* α *funds requires that* $\mathcal{A}$ *published* tx_p *to the blockchain. Thus,* $\mathcal{A}$ *presented the witness authorizing* tx_p. *This witness is the pair of signatures*

$\sigma_{B,p}$ and $\sigma_{S,p}$. $\mathcal{A}$ knows the signing key $\mathsf{sk}_{S,p}$ therefore it is easy for $\mathcal{A}$ to compute $\sigma_{S,p}$. $\mathcal{A}$ computed $\sigma_{B,p}$ by either adapting the pre-signature $\hat{\sigma}_{B,p}$ or some other way. If $\mathcal{A}$ computed $\hat{\sigma}_{B,p}$ in some other way, then this would contradict the unforgeability of the adaptor signature scheme. Thus, $\mathcal{A}$ likely computed $\sigma_{B,p}$ by adapting $\hat{\sigma}_{B,p}$ using w. $\mathcal{A}$ only gets the pre-signature $\hat{\sigma}_{B,p}$ if the buyer sends it. The honest buyer computes $\hat{\sigma}_{B,p}$ by pre-signing tx_p with respect to the signing key $\mathsf{sk}_{B,p}$ and the statement x. The buyer only sends the pre-signature $\hat{\sigma}_{B,p}$ if the proofs π_Σ and π_Ω verify. The knowledge-soundness property of π_Ω implies that $f(s) = 1$. The knowledge-soundness property of π_Σ implies that (i) the ciphertext ct encrypts the product s, and (ii) w is the decryption key for the ciphertext. The binding property of the commitment scheme implies that the product s used in π_Ω is the same used in π_Σ. The buyer can extract a witness w' from the pre-signature $\hat{\sigma}_{B,p}$ and the signature $\sigma_{B,p}$. This witness w' must be a valid witness for the statement x. Otherwise $\mathcal{A}$ would break the witness extractability property of the adaptor signature scheme. The injectiveness property of the hard relation implies that the extracted witness w' is exactly the witness w. Therefore the buyer decrypts the ciphertext ct with w. From such a decryption, the buyer obtains s.

Proof sketch 2 (Zero-knowledge for ZKCP-Protocol) *The security proof consists in describing a simulator $\mathsf{Sim}^{\widehat{B}}$ that on input f outputs a distribution indistinguishable from $View_{\widehat{B}}(s, f)$. This simulator is formally defined in Fig. 28 . Essentially, it sets the commitment com_s to 0 and the ciphertext ct of 0. Then generates proofs π_Σ and π_Ω using their respective simulators and proceeds as prescribed by the protocol. Finally, when the Seller would need to adapt the pre-signature, the simulator would abort. We construct this simulator step by step by changing one line at a time, showing in each step that the distribution computed is indistinguishable from the one before. When changing the proofs into simulated ones, we use their zero-knowledge property. When changing the commitment into a commitment to 0, we use the hiding property. When changing the ciphertext to encrypt 0, we use the IND-CPA property. When changing aborting instead of adapting the pre-signature, we use the one-wayness of the predicate f. Finally, we arrive at a simulator that does not use the secret s to compute the view of the protocol, thereby showing that the protocol does not leak sensitive information about the secret s.*

6 Application

As an illustrative use case for our MAPCP protocol, we consider a notary who attests to digital documents using a digital signature scheme. A customer wishing to get their document attested can engage with the notary to receive the attestation (e.g., a digital signature on the document) in exchange for paying the notary α units of a pre-agreed cryptocurrency. If the customer and notary wanted to use the MAPCP, the different building blocks of the construction we presented should be instantiated according to their needs, as we describe next.

Table 1: Benchmarking results. Network usage denotes the amount of data that both parties send to each other while interacting.

	Secp256k1	NistP256
Seller setup:	70.38 ms	179.01 ms
Buyer lock payment:	91.03 ms	245.11 ms
Seller adapt signature:	< 0.01 ms	< 0.01 ms
Buyer extracting witness:	10.61 ms	31.73 ms
network usage:	83746 bytes	83746 bytes
blockchain storage:	128 bytes	128 bytes

Application Setting. A customer, who holds a document m, wishes to pay a notary for the Schnorr signature on m using the notary's signing key sk. We recall the Schnorr signature scheme in Fig. 7. Technically, given a Schnorr signature $\sigma = (e, s)$, the customer wants to pay in exchange for the scalar s. For that, they execute the MAPCP (c.f. Section 5), where the different required relations are set as described next. All of them are proven using a Sigma protocol.

First, the product-dependent relation R_{prod}[6] is defined as

$$\mathsf{R}_{prod}((\mathsf{vk}, e, r); (s, \mathsf{sk})) \Leftrightarrow g^s \mathsf{vk}^e = r \wedge \mathsf{vk} = g^{\mathsf{sk}}$$

To see why is defined like that, observe that a signature is verified if e matches a hash value e'. This hash value depends on a group element r, as well as public information like the verification key vk and the message m. The group element r is defined as $g^e \mathsf{vk}^s$, so the notary must prove knowledge of both e and s such that this equality is satisfied. Hence, r must be provided as the statement. Moreover, by knowing r, the buyer can compute the hash e' using the verification algorithm and expect it to be equal to e since a mismatch would cause the verification to fail. This implies that r being a public statement reveals e, but this is not an issue because knowing e does enable the buyer to compute the value s; doing so would violate the pre-image resistance of the hash and the discrete logarithm problem. For simplicity of notation, we assume that e is also given as a public statement.

Second, naturally, we instantiate the relation required in adaptor signatures as the discrete logarithm relation.

$$\mathsf{R}_{\mathsf{AS}}(x; w) \Leftrightarrow x = g^w$$

Third, we instantiate the public-key encryption scheme. The goal here is to encrypt the element s of the Schnorr signature. We can do this with exponential ElGamal where the key generation algorithm KeyGen is the same as the GenR algorithm of R_{AS}. However, this presents a challenge. With exponential ElGamal, the decryption of the ciphertext would return g^s and one must break the discrete

[6] In the relation, we include a statement $\mathsf{vk} = g^{\mathsf{sk}}$ to state that the notary knows the signing key. An alternative formulation would remove this statement. This implies that the seller may not be the holder of the signing key.

logarithm assumption to extract s. The problem is that s is random, therefore this cannot be directly and efficiently achieved. One solution is to split s in bits s_i for $i \in [0 \dots |s|-1]$ such that $s = \sum_{i=0}^{|s|-1} s_i 2^i$. The notary would encrypt each one of these bits as $ct_i \leftarrow \mathsf{Enc}(x, s_i)$. These ciphertexts are of the form $(g^{t_i}, \mathsf{ek}^{t_i} g^{s_i})$ where t_i is the randomness chosen in the encryption algorithm. Due to the homomorphic properties of ElGamal encryption, the ciphertext ct that encrypts the value s is the product $\prod_{i=0}^{|s|-1}(ct_i)^{2^i} = (g^t, \mathsf{ek}^t g^{\sum_{i=0}^{|s|-1} s_i 2^i})$ where $t = \sum_{i=0}^{|s|-1} t_i 2^i$. When the buyer needs to decrypt, they must decrypt each ciphertext ct_i to get g^{s_i} and then break this discrete logarithm. However, in this case, s_i is either 0 or 1, therefore it is feasible to do.

For the sake of simplicity of notation, let us say that a ciphertext ct is of the form (A, B). With this instantiation of the public-key encryption scheme, the notary must prove that each ciphertext ct_i encrypts a bit s_i that can be either 1 or 0. These facts can be represented as hard relations $\mathsf{R}_i((A, B, \mathsf{ek}); (t_i, s_i)) \Leftrightarrow (A = g^{t_i}) \wedge (B = \mathsf{ek}^{t_i} \vee B = \mathsf{ek}^{t_i} g)$. Moreover, the notary must prove that the product of these ciphertexts, ct, is an ElGamal encryption of s. In conclusion, the hard relation $\mathsf{R}_{\mathsf{PKE}}$ is defined as follows. Let $A = \prod_{i=0}^{|s|-1} A_i^{2^i}$ and $B = \prod_{i=0}^{|s|-1} B_i^{2^i}$.

$$\begin{gathered}\mathsf{R}_{\mathsf{PKE}}((\{(A_i, B_i)\}_{i\in[0\dots|s|-1]}, A, B, \mathsf{ek}, g^s); (\{(t_i, s_i)\}_{i\in[0\dots|s|-1]}, s)) \Leftrightarrow \\ (\forall i \in [0 \dots |s|-1] : (A_i = g^{t_i}) \wedge (B_i = \mathsf{ek}^{t_i} \vee B_i = \mathsf{ek}^{t_i} g)) \wedge \\ (A = g^{\sum_{i=0}^{|s|-1} t_i 2^i} \wedge B = \mathsf{ek}^{\sum_{i=0}^{|s|-1} t_i 2^i} g^s)\end{gathered}$$

Finally, since R_{prod} can be proven using a Sigma protocol, we define the final relation R_Σ as

$$\begin{gathered}\mathsf{R}_\Sigma((\mathsf{vk}, e, r, g^s, x, \{(A_i, B_i)\}_{i\in[0\dots|s|-1]}); (s, \mathsf{sk}, w, \{(t_i, s_i)\}_{i\in[0\dots|s|-1]})) \Leftrightarrow \\ g^s \mathsf{vk}^e = r \wedge \mathsf{vk} = g^{\mathsf{sk}} \wedge x = g^w \wedge \\ (\forall i \in [0 \dots |s|-1] : (A_i = g^{t_i}) \wedge (B_i = x^{t_i} \vee B_i = x^{t_i} g)) \wedge \\ (A = g^{\sum_{i=0}^{|s|-1} t_i 2^i} \wedge B = x^{\sum_{i=0}^{|s|-1} t_i 2^i} g^s)\end{gathered}$$

Testbed and Evaluation Results. We have developed our proof of concept [19] in Rust and the elliptic-curve library [11]. For benchmarking, we ran the cryptographic operations for the complete protocol 1000 times and calculated the average time. The experiments have been executed in a machine with the following specifications: (i) 10-core (2-mt/8-st) 13th Gen Intel Core i5-1335U, with a maximum speed of 4600 MHz and a minimum speed of 400 MHz; (ii) 16 GiB of RAM of 6400 MHz of speed; and (iii) Linux kernel: 6.5.0-1020-oem x86_64 as the operating system.

We show the results of the benchmarking in Table 1. Here, we see that the computation overhead for all the operations is below 1 second whereas the communication overhead is around 80 KB, making the approach affordable even in commodity hardware. Additionally, to study the impact of different curves, we repeated the experiments with curves secp256k1 and nistp256, observing that the former (currently used e.g., in Bitcoin) yields more efficient computation. Nevertheless, both curves impose a small computation overhead, whereas the communication overhead remains the same.

7 Conclusion

In this work, we presented MAPCP, a protocol for obtaining digital goods in exchange for a cryptocurrency payment. Our protocol differs from previous works in that it is the first one that simultaneously departs from using zk-SNARKS for the zero-knowledge proofs and HTLC for the atomic exchange of coins and a secret. Given that, MAPCP does not suffer from the *crs* creation problem and provides interoperability and fungibility, important properties for the privacy of the users. We analyze the security of MAPCP and provide a proof of concept implementation where a customer pays a notary in exchange for the signature on a document. Our evaluation shows that MAPCP is practical even on commodity hardware.

As future work, we find it interesting to study whether our modular treatment for ZKCP also applies to the problem of ZKCSP. Our preliminary work in this direction is in Appendix C.

Acknowledgments We would like to thank the reviewers for their helpful feedback. This work has been supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001283), by the ESPADA project (grant PID2022-142290OB-I00), MCIN/AEI/10.13039/501100011033/ FEDER, UE, and by the PRODIGY project (grant ED2021-132464B-I00), MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU/ PRTR.

References

1. Abadi, A., Murdoch, S.J., Zacharias, T.: Recurring contingent payment for proofs of retrievability. Cryptology ePrint Archive, Report 2021/1145 (2021), https://eprint.iacr.org/2021/1145
2. Adachi, M., Born, A., Gschossmann, I., van der Kraaij, A.: The expanding functions and uses of stablecoins. ECB website (2021), https://www.ecb.europa.eu/pub/financial-stability/fsr/focus/2021/html/ecb.fsrbox202111_04 45293c08fc.en.html
3. AT&T: At&t now accepts bitpay. AT&T webpage (2019), https://about.att.com/story/2019/att_bitpay.html
4. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized channels from limited blockchain scripts and adaptor signatures. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 635–664. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92075-3_22
5. Bellare, M., Fuchsbauer, G., Scafuro, A.: NIZKs with an untrusted CRS: Security in the face of parameter subversion. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 777–804. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53890-6_26
6. Bitpay: Buy from microsoft. Bitpay merchan directory (2023), https://bitpay.com/directory/microsoft/

7. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2075–2092. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3339820
8. Campanelli, M., Gennaro, R., Goldfeder, S., Nizzardo, L.: Zero-knowledge contingent payments revisited: Attacks and payments for services. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 229–243. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134060
9. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th ACM STOC. pp. 494–503. ACM Press (May 2002). https://doi.org/10.1145/509907.509980
10. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: 18th ACM STOC. pp. 364–369. ACM Press (May 1986). https://doi.org/10.1145/12130.12168
11. Doc.rs team: Rust elliptic curve library, https://docs.rs/elliptic-curve/latest/elliptic_curve/index.html
12. Dziembowski, S., Eckey, L., Faust, S.: FairSwap: How to fairly exchange digital goods. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 967–984. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243857
13. ECB Crypto-Assets Task Force: Stablecoins: Implications for monetary policy, financial stability, market infrastructure and payments, and banking supervision in the euro area. European Central Bank Occasional Paper Series (2020), https://www.ecb.europa.eu/pub/pdf/scpops/ecb.op247 fe3df92991.en.pdf
14. Eckey, L., Faust, S., Schlosser, B.: OptiSwap: Fast optimistic fair exchange. In: Sun, H.M., Shieh, S.P., Gu, G., Ateniese, G. (eds.) ASIACCS 20. pp. 543–557. ACM Press (Oct 2020). https://doi.org/10.1145/3320269.3384749
15. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
16. Erwig, A., Faust, S., Hostáková, K., Maitra, M., Riahi, S.: Two-party adaptor signatures from identification schemes. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 451–480. Springer, Cham (May 2021). https://doi.org/10.1007/978-3-030-75245-3_17
17. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: 22nd ACM STOC. pp. 416–426. ACM Press (May 1990). https://doi.org/10.1145/100216.100272
18. Fuchsbauer, G.: WI is not enough: Zero-knowledge contingent (service) payments revisited. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 49–62. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3354234
19. Gomez-Martinez, J.: Repository for mapcp implementation, https://github.com/jgomezmartinez/mapcp_rust
20. He, S., Lu, Y., Tang, Q., Wang, G., Wu, C.Q.: Fair peer-to-peer content delivery via blockchain. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021, Part I. LNCS, vol. 12972, pp. 348–369. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-88418-5_17
21. Janin, S., Qin, K., Mamageishvili, A., Gervais, A.: Filebounty: Fair data exchange. In: IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, Genoa, Italy, September 7-11, 2020. pp. 357–366. IEEE (2020). https://doi.org/10.1109/EUROSPW51379.2020.00056, https://doi.org/10.1109/EuroSPW51379.2020.00056

22. KPMG: Frontiers in finance: Innovating through platforms and ecosystems. KPMG (2022), https://assets.kpmg.com/content/dam/kpmg/xx/pdf/2022/05/frontiers-in-finance.pdf
23. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 620–649. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_22
24. Maxwell, G.: Zero knowledge contingent payment. Bitcoin Wiki (2015)
25. Moreno-Sanchez, P., Blue, A., Le, D.V., Noether, S., Goodell, B., Kate, A.: DL-SAG: Non-interactive refund transactions for interoperable payment channels in monero. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 325–345. Springer, Cham (Feb 2020). https://doi.org/10.1007/978-3-030-51280-4_18
26. Nguyen, K., Ambrona, M., Abe, M.: WI is almost enough: Contingent payment all over again. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 641–656. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417888
27. Nikolaenko, V., Ragsdale, S., Bonneau, J., Boneh, D.: Powers-of-tau to the people: Decentralizing setup ceremonies. In: Pöpper, C., Batina, L. (eds.) ACNS 24International Conference on Applied Cryptography and Network Security, Part III. LNCS, vol. 14585, pp. 105–134. Springer, Cham (Mar 2024). https://doi.org/10.1007/978-3-031-54776-8_5
28. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Berlin, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_9
29. PriceWaterhouseCoopers: El salvador's law: a meaningful test for bitcoin. PWC webpage (2021), https://www.pwc.com/gx/en/financial-services/pdf/el-salvadors-law-a-meaningful-test-for-bitcoin.pdf
30. Shopify: Shopify help center: cryptocurrencies. Shopify webpage (2023), https://help.shopify.com/en/manual/payments/additional-payment-methods/cryptocurrency
31. Tas, E.N., Seres, I.A., Zhang, Y., Melczer, M., Kelkar, M., Bonneau, J., Nikolaenko, V.: Atomic and fair data exchange via blockchain. Cryptology ePrint Archive, Paper 2024/418 (2024), https://eprint.iacr.org/2024/418
32. Thyagarajan, S.A.K., Malavolta, G., Schmidt, F., Schröder, D.: PayMo: Payment channels for monero. Cryptology ePrint Archive, Report 2020/1441 (2020), https://eprint.iacr.org/2020/1441
33. Wiki, T.B.: (2019), https://en.bitcoin.it/wiki/Hashlock
34. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W.J.: SoK: Communication across distributed ledgers. In: Borisov, N., Díaz, C. (eds.) FC 2021, Part II. LNCS, vol. 12675, pp. 3–36. Springer, Berlin, Heidelberg (Mar 2021). https://doi.org/10.1007/978-3-662-64331-0_1

A Extended Preliminaries

A.1 Hard Languages

Definition 5 (Binary relation). *A binary relation* R *is a function* $\mathsf{R} : \mathcal{X} \times \mathcal{W} \to \{0,1\}$. *The language associated to* R *is* $L_{\mathsf{R}} := \{x \in \mathcal{X} \mid \exists w \in \mathcal{W} : \mathsf{R}(x,w) = 1\}$

Definition 6 (Hard relation). *A relation* R *is hard if (i) there exists a* PPT *sampling algorithm* $\mathsf{GenR}(1^n)$ *that on input the security parameter* 1^n*, it outputs* $(x, w) \in \mathsf{R}$*; (ii) there exists a* PPT *algorithm* $\mathcal{D}$ *such that* $\mathcal{D}(x, w) = 1 \Leftrightarrow (x, w) \in \mathsf{R}$*; (iii) for all* PPT *algorithms* $\mathcal{A}$ *it holds that:*

$$\Pr\left[(x, w^*) \in \mathsf{R} \middle| \begin{array}{c}(x, w) \leftarrow \mathsf{GenR}(1^n) \\ w^* \leftarrow \mathcal{A}(x)\end{array}\right] \leq \mathsf{negl}(n)$$

A.2 Hash Functions

Definition 7 (Hash function). *A* hash *function* H *is defined as follows:*

$h \leftarrow \mathsf{H}(m)$*. A* DPT *algorithm that takes as input a message* m*. It outputs a hash value* h*.*

We now review the notions of security for a hash function.

Definition 8 (Colllision resistance). *A hash function* H *is* collision resistant *if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\Pr[\mathsf{CollRes}_{\mathcal{A},\mathsf{H}}(n) = 1] \leq \mathsf{negl}(n)$$

where $\mathsf{CollRes}_{\mathcal{A},\mathsf{H}}(n)$ *is defined as follows:*

$\mathsf{CollRes}_{\mathcal{A},\mathsf{H}}(n)$

1: $(m, m') \leftarrow \mathcal{A}^{\mathsf{H}}$
2: $b_0 := (m \neq m')$
3: $b_1 := (\mathsf{H}(m) = \mathsf{H}(m'))$
4: **return** $b_0 \wedge b_1$

Definition 9 (Pre-image resistance). *A hash function* H *is* preimage resistant *if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\Pr[\mathsf{PreImageRes}_{\mathcal{A},\mathsf{H}}(n) = 1] \leq \mathsf{negl}(n)$$

where $\mathsf{PreImageRes}_{\mathcal{A},\mathsf{H}}(n)$ *is defined as follows:*

$\mathsf{PreImageRes}_{\mathcal{A},\mathsf{H}}(n)$

1: $h \leftarrow_{\$} \mathcal{H}$
2: $m \leftarrow \mathcal{A}(h)$
3: **return** $h = \mathsf{H}(m)$

Definition 10 (Second Pre-image resistance). *A hash function* H *is* second preimage resistant *if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\Pr[\mathsf{SecondPreImageRes}_{\mathcal{A},\mathsf{H}}(n) = 1] \leq \mathsf{negl}(n)$$

where $\mathsf{SecondPreImageRes}_{\mathcal{A},\mathsf{H}}(n)$ *is defined as follows:*

$\mathsf{SecondPreImageRes}_{\mathcal{A},\mathsf{H}}(n)$

1: $m \leftarrow_{\$} \mathcal{M}$
2: $m' \leftarrow \mathcal{A}(m)$
3: $b_0 := (m \neq m')$
4: $b_1 := (\mathsf{H}(m) = \mathsf{H}(m'))$
5: **return** $b_0 \wedge b_1$

A.3 Non-Interactive Zero Knowledge Proofs

Definition 11 (Non-Interactive Zero Knowledge Proof). *A* non-interactive zero knowledge proof *for a relation* R *is a tuple of algorithms* $\mathsf{NIZK} := (\mathsf{CrsGen}, \mathsf{Prove}, \mathsf{Verify})$ *defined as follows:*

$crs \leftarrow \mathsf{CrsGen}(1^n)$. *A* PPT *algorithm that takes as input the security parameter* 1^n*. It outputs a common reference string crs.*
$\pi \leftarrow \mathsf{Prove}(crs, x, w)$. *A* PPT *algorithm that takes as input the common reference string crs, a statement* x *and a witness* w*. It outputs a proof* π
$\{0,1\} \leftarrow \mathsf{Verify}(crs, x, \pi)$. *A* DPT *algorithm that takes as input the common reference string, a statement* x *and a proof* π*. It outputs a bit* $b \in \{0,1\}$*.*

A NIZK *is* complete *if for all* $(x, w) \in \mathsf{R}$ *and all* $crs \leftarrow \mathsf{CrsGen}(1^n)$ *it holds that:*

$$\Pr\left[\mathsf{Verify}(crs, x, \pi) = 1 \middle| \begin{matrix} crs \leftarrow \mathsf{CrsGen}(1^n) \\ \pi \leftarrow \mathsf{Prove}(crs, x, w) \end{matrix}\right] \geq 1 - \mathsf{negl}(n)$$

We now review the notion of security for a non-interactive zero-knowledge proof.

Definition 12 (Zero-knowledge). *A non-interactive zero knowledge proof* $\mathsf{NIZK} := (\mathsf{CrsGen}, \mathsf{Prove}, \mathsf{Verify})$ *is* zero-knowledge *if there exist two* PPT *simulators* $(\mathsf{CrsSim}, \mathsf{ProveSim})$ *such that for every* PPT *adversary* $\mathcal{A}$*, then the following probability is negligible in* n*:*

$$\left| \Pr\left[\begin{matrix} (x,w) \in \mathsf{R} \wedge \\ 1 \leftarrow \mathcal{A}(\pi) \end{matrix} \middle| \begin{matrix} (crs) \leftarrow \mathsf{CrsGen}(1^n) \\ (x,w) \leftarrow \mathcal{A}(crs) \\ \pi \leftarrow \mathsf{Prove}(crs, w, x) \end{matrix} \right] - \Pr\left[\begin{matrix} (x,w) \in \mathsf{R} \wedge \\ 1 \leftarrow \mathcal{A}(\pi) \end{matrix} \middle| \begin{matrix} (crs, \tau) \leftarrow \mathsf{CrsSim}(1^n) \\ (x,w) \leftarrow \mathcal{A}(crs) \\ \pi \leftarrow \mathsf{ProveSim}(crs, \tau, x) \end{matrix} \right] \right|$$

Definition 13 (Knowledge-soundness). *A non-interactive zero knowledge proof* $\mathsf{NIZK} := (\mathsf{CrsGen}, \mathsf{Prove}, \mathsf{Verify})$ *is* knowledge-sound *if there exists a* PPT *algorithm* Ext *such that for all adversaries* $\mathcal{A}$*, it holds that:*

$$\Pr\left[\begin{array}{c} 1 \leftarrow \mathsf{Verify}(crs, x, \pi) \\ (x, w) \notin \mathsf{R} \end{array} \middle| \begin{array}{c} crs \leftarrow \mathsf{CrsGen}(1^n) \\ (x, \pi) \leftarrow \mathcal{A}(crs) \\ w \leftarrow \mathsf{Ext}^{\mathcal{A}}(crs, x, \pi) \end{array}\right] \leq \mathsf{negl}(n)$$

A.4 Commitment Schemes

Definition 14 (Commitment Scheme). *A* commitment scheme $\mathsf{COM} := (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *is defined as follows:*

- $p \leftarrow \mathsf{Setup}(1^n)$*. A* PPT *algorithm that takes as input the security parameter* 1^n *and outputs the commitment parameters* p*.*
- $(com, open) \leftarrow \mathsf{Commit}(p, x)$*. A* PPT *algorithm that takes as input the commitment parameters* p *and a message* x*. It outputs the commitment* com *and the opening information* $open$*.*
- $\{0, 1\} \leftarrow \mathsf{Open}(p, x, com, open)$*. A* PPT *algorithm tha takes as input the commitment parameters* p*, a commited value* x*, a commitment* com *and the opening information* $open$*. It outputs a bit* $b \in \{0, 1\}$*.*

A commitment scheme is correct *if for every message* $x \in \mathcal{X}$ *it holds that:*

$$\Pr\left[\mathsf{Open}(p, x, com, open) \middle| \begin{array}{c} p \leftarrow \mathsf{Setup}(1^n) \\ (com, open) \leftarrow \mathsf{Commit}(p, x) \end{array}\right] \geq 1 - \mathsf{negl}(n)$$

We now review the notions of security for a commitment scheme.

Definition 15 (Binding). *A commitment scheme* $\mathsf{COM} := (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *is* binding *if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\Pr[\mathsf{Binding}_{\mathcal{A},\mathsf{COM}}(n) = 1] \leq \mathsf{negl}(n)$$

where the experiment $\mathsf{Binding}_{\mathcal{A},\mathsf{COM}}(n)$ *is defined as follows:*

$\mathsf{Binding}_{\mathcal{A},\mathsf{COM}}(n)$

1: $p \leftarrow \mathsf{Setup}(1^n)$
2: $(com, open, x, open', x') \leftarrow \mathcal{A}(p)$
3: $b_0 := \mathsf{Open}(p, x, com, open)$
4: $b_1 := \mathsf{Open}(p, x', com, open')$
5: $b_2 := x \neq x'$
6: **return** $b_0 \wedge b_1 \wedge b_2$

Definition 16 (Hiding). *A commitment scheme* $\mathsf{COM} := (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *is* hiding *if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that*

$$\Pr[\mathsf{Hiding}_{\mathcal{A},\mathsf{COM}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$$

where the experiment $\mathsf{Hiding}_{\mathcal{A},\mathsf{COM}}(n)$ *is defined as follows:*

$\mathsf{Hiding}_{\mathcal{A},\mathsf{COM}}(n)$

1: $p \leftarrow \mathsf{Setup}(1^n)$
2: $(x_0, x_1) \leftarrow \mathcal{A}(p)$
3: $b \leftarrow_{\$} \{0,1\}$
4: $(com_b, open_b) \leftarrow \mathsf{Commit}(p, x_b)$
5: $b^* \leftarrow \mathcal{A}(com_b)$
6: **return** $b = b^*$

A.5 Sigma Protocols

Definition 17 (Sigma Protocol for Relation R**).** *A Sigma protocol for relation* R *is a protocol between a prover* $\mathcal{P}$ *and a verifier* $\mathcal{V}$ *of the form given in Fig. 6 satisfying the following three properties.*

Completeness. If $\mathcal{P}$ *and* $\mathcal{V}$ *follow the protocol, then* V *always accepts.*
Special soundness. There exists a PPT *algorithm* E *(extractor) which given any* $v \in V$ *and any pair of accepting conversations* $(a; c; r)$ *and* $(a; c'; r')$ *with* $c \neq c'$ *always computes a witness* w *satisfying* $(v; w) \in \mathsf{R}$.
Special honest-verifier zero-knowledgeness. There exists a PPT *algorithm* S *(simulator) which given any* $v \in L_R$ *and any challenge* $c \in C$ *produces conversations* $(a; c; r)$ *with the same probability distribution as conversations between honest* $\mathcal{P}$ *and* $\mathcal{V}$ *on common input* v *and challenge* c*, where* $\mathcal{P}$ *uses any witness* w *satisfying* $(v; w) \in R$*. Furthermore, given any* $v \in V \backslash L_R$*, simulator* S *is just required to produce arbitrary accepting conversations* $(a; c; r)$*, for any given challenge* $c \in C$*.*

If C *consists of a single element, the Sigma protocol is said to be trivial.*

A.6 Public Key Encryption

Definition 18 (Public key encryption scheme). *A* public key encryption scheme $\mathsf{PKE} := (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is defined as follows:*

$(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^n)$*. A* PPT *algorithm that takes as input the security parameter* 1^n*. It outputs a pair of encryption and decryption keys* $(\mathsf{dk}, \mathsf{ek})$*.*

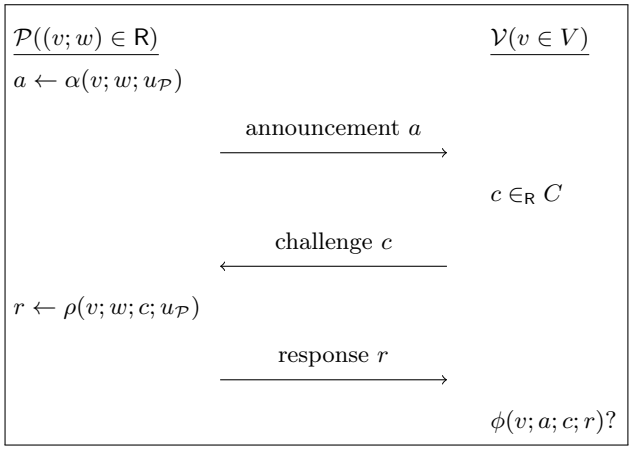

Fig. 6: Sigma Protocol for Relation R. Conversation $(a; c; r)$ accepting if $\phi(v; a; c; r)$ holds. Polynomial time predicate ϕ, finite set $C \neq \emptyset$, random tape $u_{\mathcal{P}}$, PPT algorithms α and ρ.

$ct \leftarrow \mathsf{Enc}(\mathsf{ek}, m)$. *A* PPT *algorithm that takes as input the encryption key* ek *and a message* m*. It outputs a ciphertext* ct*.*

$\{m, \perp\} \leftarrow \mathsf{Dec}(\mathsf{dk}, ct)$. *A* DPT *algorithm that takes as input the decryption key* dk *and a ciphertext* ct*. It outputs a message* m *or a special symbol* $\perp$ *denoting failure.*

A public key encryption scheme is correct *if for every message* $m \in \mathcal{M}$ *it holds that:*

$$\Pr\left[m = m' \middle| \begin{array}{c}(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^n) \\ m' := \mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{ek}, m))\end{array}\right] = 1$$

We now review the notion of security for a public key encryption scheme.

Definition 19 (Indistinguishable encryptions under chosen-plaintext attacks).

A public key encryption scheme $\mathsf{PKE} := (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions under chosen-plaintext attacks *(or is* CPA-secure*) if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\Pr\left[\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\mathsf{PKE}}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

where the experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\mathsf{PKE}}(n)$ *is defined as follows:*

$\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\mathsf{PKE}}(n)$

1: $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^n)$
2: $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{ek})$
3: $b \leftarrow_{\$} \{0,1\}$
4: $ct_b \leftarrow \mathsf{Enc}(\mathsf{ek}, m_b)$
5: $b^* \leftarrow \mathcal{A}(ct_b)$
6: $b_0 := (|m_0| = |m_1|)$
7: $b_1 := (b = b^*)$
8: **return** $b_0 \wedge b_1$

A.7 Digital Signatures

Definition 20 (Digital signature scheme). *A* digital signature scheme $\mathsf{DS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is defined as follows:*

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$. *A* PPT *algorithm that takes as input the security parameter* 1^n*. It outputs a pair of signing and verification keys* $(\mathsf{sk}, \mathsf{vk})$.

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$. *A* PPT *algorithm that takes as input the signing key* sk *and a message* m*. It outputs a signature* σ.

$\{0,1\} \leftarrow \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$. *A* DPT *algorithm that takes as input the verification key* vk*, a message* m*, and a signature* σ*. It outputs a bit* $b \in \{0,1\}$.

A digital signature scheme is correct *if for every* $m \in \mathcal{M}$ *it holds that:*

$$\Pr\left[\mathsf{Vrfy}(\mathsf{vk}, m, \sigma) \middle| \begin{array}{c}(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)\end{array}\right] \geq 1 - \mathsf{negl}(n)$$

We now review the notion of security for a digital signature scheme.

Definition 21 (Existential unforgeability under an adaptative chosen-message attack). *A digital signature scheme* $\mathsf{DS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* existentially unforgeable under an adaptative chosen-message attack *(or just* secure*) if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\Pr[\mathsf{Sig\text{-}Forge}_{\mathcal{A},\mathsf{DS}}(n) = 1] \leq \mathsf{negl}(n)$$

where the experiment $\mathsf{Sig\text{-}Forge}_{\mathcal{A},\mathsf{DS}}(n)$ *is defined as follows:*

$\mathsf{Sig\text{-}Forge}_{\mathcal{A},\mathsf{DS}}(n)$

1: $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$
2: $(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{OSign}}(\mathsf{vk})$
3: $b_0 := m \notin \mathcal{Q}$
4: $b_1 := \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$
5: **return** $b_0 \wedge b_1$

$\mathsf{OSign}(m)$

1: $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$
2: $\mathcal{Q} := \mathcal{Q} \cup m$
3: **return** σ

$\mathsf{KeyGen}(1^n)$	$\mathsf{Sign}(\mathsf{sk}, m)$	$\mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$
1 : $\mathsf{sk} \leftarrow_\$ Z_p$	1 : $z \leftarrow_\$ \{0,1\}^\lambda$	1 : $(e, s) := \sigma$
2 : $\mathsf{vk} := g^{\mathsf{sk}}$	2 : $r := g^z$	2 : $r := g^e * \mathsf{vk}^s$
3 : **return** $(\mathsf{sk}, \mathsf{vk})$	3 : $e \leftarrow \mathsf{H}(\mathsf{vk}, r, m)$	3 : $e' := \mathsf{H}(\mathsf{vk}, r, m)$
	4 : $s := z + e \cdot \mathsf{sk}$	4 : **return** $e = e'$
	5 : **return** (e, s)	

Fig. 7: Algorithms of Schnorr signatures.

Schnorr Signatures For completeness, we revise here the implementation of Schnorr Signatures, since they are used as building block in our tested and evaluation.

A.8 Adaptor Signatures

Definition 22 (Adaptor signature scheme). *An* adaptor signature scheme *is a tuple of algorithms* $\mathsf{AS} := (\mathsf{KeyGen}, \mathsf{PreSign}, \mathsf{PreVrfy}, \mathsf{Adapt}, \mathsf{Extract})$ *with respect to a digital signature scheme* DS *and a hard relation* R *is defined as follows:*

- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$. *A* PPT *algorithm that takes as input the security parameter* 1^n. *It outputs a pair of signing and verification keys* $(\mathsf{sk}, \mathsf{vk})$.
- $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, m, x)$. *A* PPT *algorithm that takes as input the signing key* sk, *a message* m *and a statement* $x \in \mathcal{X}$. *It outputs a pre-signature* $\hat{\sigma}$.
- $\{0,1\} \leftarrow \mathsf{PreVrfy}(\mathsf{vk}, m, \hat{\sigma}, x)$. *A* DPT *algorithm that takes as input the verification key* vk, *a message* m, *a pre-signature* σ *and a statement* $x \in \mathcal{X}$. *It outputs a bit* $b \in \{0,1\}$.
- $\sigma \leftarrow \mathsf{Adapt}(\hat{\sigma}, w)$. *A* PPT *algorithm that takes as input a pre-signature* $\hat{\sigma}$ *and a witness* $w \in \mathcal{W}$. *It outputs a signature* σ.
- $w \leftarrow \mathsf{Extract}(\hat{\sigma}, \sigma, x)$. *A* DPT *algorithm that takes as input a pre-signature* $\hat{\sigma}$, *a signature* σ *and a statement* $x \in \mathcal{X}$. *It outputs a witness* $w \in \mathcal{W}$.

An adaptor signature scheme is pre-signature correct *if for every* $m \in \mathcal{M}$ *it holds that:*

$$\Pr\left[\mathsf{PreVrfy}(\mathsf{vk}, m, x, \hat{\sigma}) \;\middle|\; \begin{array}{c}(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n) \\ (x, w) \leftarrow \mathsf{GenR}(1^n) \\ \hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, m, x)\end{array}\right] \geq 1 - \mathsf{negl}(n)$$

We now review the notion of security for an adaptor signature scheme.

Definition 23 (Existential unforgeability under a chosen-message attack). *An adaptor signature scheme* $\mathsf{AS} := (\mathsf{KeyGen}, \mathsf{PreSign}, \mathsf{PreVrfy}, \mathsf{Adapt},$

Extract) *is* existentially unforgeable under a chosen-message attack *if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\Pr[\mathsf{ASig\text{-}EUF\text{-}CMA}_{\mathcal{A},\mathsf{AS}}(n) = 1] \leq \mathsf{negl}(n)$$

where the experiment $\mathsf{ASig\text{-}EUF\text{-}CMA}_{\mathcal{A},\mathsf{AS}}(n)$ *is defined as follows:*

$\mathsf{ASig\text{-}EUF\text{-}CMA}_{\mathcal{A},\mathsf{AS}}(n)$

1: $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$
2: $(x, w) \leftarrow \mathsf{GenR}(1^n)$
3: $m \leftarrow \mathcal{A}^{\mathsf{OSign},\mathsf{OPreSign}}(\mathsf{vk})$
4: $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, m, x)$
5: $\sigma \leftarrow \mathcal{A}^{\mathsf{OSign},\mathsf{OPreSign}}(\hat{\sigma}, x)$
6: $b_0 := m \notin \mathcal{Q}$
7: $b_1 := \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$
8: **return** $b_0 \wedge b_1$

$\mathsf{OSign}(m)$

1: $\sigma \leftarrow \mathsf{Sign}(\sigma, m)$
2: $\mathcal{Q} := \mathcal{Q} \cup m$
3: **return** σ

$\mathsf{OPreSign}(m, x)$

1: $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, m, x)$
2: $\mathcal{Q} := \mathcal{Q} \cup m$
3: **return** $\hat{\sigma}$

Definition 24 (Pre-signature adaptability). *An adaptor signature scheme* $\mathsf{AS} := (\mathsf{KeyGen}, \mathsf{PreSign}, \mathsf{PreVrfy}, \mathsf{Adapt}, \mathsf{Extract})$ *is* pre-signature adaptable *if for any message* $m \in \mathcal{M}$*, any statement/witness pair* $(x, w) \in \mathsf{R}$*, any key pair* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$ *and any pre-signature* $\hat{\sigma}$ *such that* $\mathsf{PreVrfy}(\mathsf{vk}, m, x, \hat{\sigma}) = 1$ *it holds that:*

$$\Pr\left[\mathsf{Vrfy}(\mathsf{vk}, m, \sigma) \middle| \sigma \leftarrow \mathsf{Adapt}(\hat{\sigma}, w)\right] \geq 1 - \mathsf{negl}(n)$$

Definition 25 (Witness extractability). *An adaptor signature scheme* $\mathsf{AS} := (\mathsf{KeyGen}, \mathsf{PreSign}, \mathsf{PreVrfy}, \mathsf{Adapt}, \mathsf{Extract})$ *is* witness extractable *if for all* PPT *adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\Pr[\mathsf{ASig\text{-}WE}_{\mathcal{A},\mathsf{AS}}(n) = 1] \leq \mathsf{negl}(n)$$

where the experiment $\mathsf{ASig\text{-}WE}_{\mathcal{A},\mathsf{AS}}(n)$ *is defined as follows:*

$\mathsf{ASig\text{-}WE}_{\mathcal{A},\mathsf{AS}}(n)$

1: $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$
2: $(m, x) \leftarrow \mathcal{A}^{\mathsf{OSign},\mathsf{OPreSign}}(\mathsf{vk})$
3: $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, m, x)$
4: $\sigma \leftarrow \mathcal{A}^{\mathsf{OSign},\mathsf{OPreSign}}(\hat{\sigma}, x)$
5: $w' \leftarrow \mathsf{Extract}(\hat{\sigma}, \sigma, x)$
6: $b_0 := m \notin \mathcal{Q}$
7: $b_1 := \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$
8: $b_2 := (x, w') \notin \mathsf{R}$
9: **return** $b_0 \wedge b_1$

$\mathsf{OSign}(m)$

1: $\sigma \leftarrow \mathsf{Sign}(\sigma, m)$
2: $\mathcal{Q} := \mathcal{Q} \cup m$
3: **return** σ

$\mathsf{OPreSign}(m, x)$

1: $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, m, x)$
2: $\mathcal{Q} := \mathcal{Q} \cup m$
3: **return** $\hat{\sigma}$

B The Logic Behind Locking Funds

Recall that the construction in Section 5.1 requires a lock transaction. This transaction, tx_l, transfers funds of the buyer to an output shared by both the seller and the buyer. To be more specific, the lock transaction is defined as $(\alpha, \mathsf{vk}_{B,l}) \rightarrow (\alpha, (\mathsf{vk}_{B,p} \wedge \mathsf{vk}_{S,p}) \vee (\mathsf{vk}_{B,r} + t))$. In this notation the subscripts B and S on the verification keys imply that buyer and seller respectively own the corresponding signing keys. Authorizations to spend the output of the lock transaction require either (i) a signature from each party or (ii) a signature from the buyer after some timeout. Later, there will be transactions tx_p and tx_r that spend the locked funds and move them to outputs controlled by either the seller or the buyer respectively.

This is a common technique that prevents *double spending* attacks and *griefing* attacks.

B.1 Double Spending Attacks

For illustration purposes, assume that the buyer directly moves the funds they initially held to the seller. A bit more technically, the funds are moved by a transaction tx_p defined as $(\alpha, \mathsf{vk}_B) \rightarrow (\alpha, \mathsf{vk}_S)$. Recall that, in the MAPCP protocol, it is the seller who authorizes the transaction tx_p with the signature σ_B from the buyer. Then, the buyer reads this authorizing signature from the ledger and uses it to extract the witness they wanted to buy. However, note that in a blockchain setting, the seller submits the transaction tx_p together with the authorizing signature by broadcasting it to the different validators of the blockchain. Note that if the buyer is one of those validators, the buyer can extract the witness they wanted to buy *before the transaction tx_p is actually published*. The time window between the submission and the publication of the transaction tx_p enables the buyer to perform a *double spending attack*.

In a *double spending attack*, the buyer submits a different transaction tx_p'. This transaction spends the same output (α, vk_B) that is an input for the transaction tx_p. The goal is that tx_p' moves these funds to an output controlled by the buyer. To be more specific, tx_p' is defined as $(\alpha, \mathsf{vk}_B) \rightarrow (\alpha, \mathsf{vk}'_B)$. Both tx_p and tx_p' are submitted roughly at the same time. However, the order in which these transactions are published is non-deterministic. It could be the case that tx_p' is published before tx_p. If this is the case, tx_p would no longer be a valid transaction, because the output it tries to spend has already been spent by tx_p. In this situation, the buyer would succeed on his attack: (i) the buyer has seen the signature σ_B and can extract the witness they wanted to buy; and (ii) the buyer still controls the funds.

To prevent this attack the seller asks the buyer to lock their funds in an output controlled by both the seller and the buyer. In other words, the seller should not broadcast the signature σ_B until the see published a transaction tx_l defined as $(\alpha, \mathsf{vk}_{B,l}) \rightarrow (\alpha, (\mathsf{vk}_{B,p} \wedge \mathsf{vk}_{S,p}))$. This way, the buyer cannot spend the output created by this transaction as they would need a signature σ_S from the seller. This technique prevents *double spending attacks* from a malicious seller.

B.2 Griefing Attacks

The technique preventing double spending attacks is not sufficient. The buyer locking their funds as described before enables the seller to perform a *griefing attack*. In this attack, the seller waits until the buyer locks their funds by publishing the transaction tx_l defined as $(\alpha, \mathsf{vk}_{B,l}) \rightarrow (\alpha, (\mathsf{vk}_{B,p} \wedge \mathsf{vk}_{S,p}))$. Then, the seller simply aborts or stops collaborating. This achieves that, even though the seller will not earn the funds, the buyer will not learn the witness they wanted to buy either. Moreover, the buyer cannot recover the funds they locked.

To prevent this attack, we need to modify the transaction tx_l. The buyer should be able to spend the locked funds after enough time has passed. This way, they are able to recover their funds in the case that the seller stops collaborating. Thus, the transaction tx_l must be defined as $(\alpha, \mathsf{vk}_{B,l}) \rightarrow (\alpha, (\mathsf{vk}_{B,p} \wedge \mathsf{vk}_{S,p}) \vee (\mathsf{vk}_{B,r} + t))$.

C Our Protocol for Modular ZKCSP

C.1 Problem Description

Campanelli et al. [8] observed that a slight variation of the ZKCP can serve as a protocol to pay for a *service* instead of a product, contributing the so-called *zero-knowledge contingent service payment* (ZKCSP). As an illustrative example, consider that a cloud user (i.e., the buyer) is interested in paying a cloud storage service provider (i.e., the seller) for proof of the integrity of their outsourced files [1]. Similarly to the contingent payment scenario, the seller must provide a proof that convinces the buyer that their files are correctly stored and can be fully retrieved. However, as soon as the buyer receives such proof, they no longer have an incentive to pay since it is the proof itself what the buyer is interested in.

To address this challenge, the fundamental concept of ZKCSP is to retain the same protocol as ZKCP, with the only difference being what is certified by the zero-knowledge proof. Concretely, the proof in ZKCSP is used by the seller to prove to the buyer that "either the files are in my storage and k is a preimage of h (i.e., $h := H(k)$) or I neither keep the files nor have a preimage of h". A bit more formally, the seller needs to prove the following relation:

$$\mathsf{R}_{f,\mathsf{H}}(h; s, \mathsf{k}) \Leftrightarrow (f(s) \wedge h = \mathsf{H}(\mathsf{k})) \vee (\neg f(s) \wedge h = \widehat{\mathsf{H}}(\mathsf{k}))$$

where $\widehat{\mathsf{H}}$ is a hash function different from H. More precisely it must hold that $\widehat{\mathsf{H}}$ and H are *claw-free* and *indistinguishable*. *Claw-freeness* means that it is computationally hard to find a pair of values k_0, k_1 such that $\mathsf{H}(k_0) = \widehat{\mathsf{H}}(k_1)$. *Indistinguishability* means that their outputs must have the same distribution.

A ZKCSP has to provide the following guarantees. First, if the buyer pays the seller, then the buyer learns that the seller knows s such that $f(s) = 1$ but no additional information beyond that. Second, the buyer does not learn any information without first paying to the HTLC contract.

For the first guarantee, recall that the HTLC contract is configured with $h = \mathsf{H}(k)$ as its parameter. Hence, if the seller gets paid, they must have provided a value k in satisfying the HTLC: $h = \mathsf{H}(k)$. This ensures that the clause $(f(s) \wedge h = \mathsf{H}(k))$ is satisfied. Note that the *claw-freeness* property is required in this context. If the adversary can find a value that produces the same output for both hash functions, the buyer would be unable to determine which branch of the relation is true.

For the second guarantee, the buyer must not be able to tell which branch of the relation —either $f(s) \wedge h = \mathsf{H}(\mathsf{k})$ or $\neg f(s) \wedge h = \widehat{\mathsf{H}}(\mathsf{k})$— is true until they pay and learn the value k. Note that the *indistinguishability* property is required here. The buyer gets h from the seller to set up the HTLC contract. If the outputs of H and $\widehat{\mathsf{H}}$ are easy to distinguish, then the buyer could tell which branch of the proof is true by looking at h. Therefore, the buyer could easily learn whether it is true that the service is being provided or not before paying.

In this section, we present Modular Algebraic Proof Contingent Service Payment (MAPCSP), a protocol that implements ZKCSP, analyze its security and present a proof of concept.

Similarly to MAPCP, MAPCSP is divided into *setup* phase and *execution* phase. Considering the setup phase, the proofs need to be adjusted to align with the service payment scenario. The execution phase is identical to that of MAPCP (cf. Section 5.2).

C.2 Setup Phase: Product Setup and Proofs

Building blocks. We require a commitment scheme COM and a non-interactive zero knowledge proof system NIZK with the properties described in Section 3.

Overview Of Our Construction. In MAPCSP, the product setup phase conceals the result of evaluating $f(s)$ within a commitment com_{bit} and must convince the buyer of two key points: (i) the commitment com_{bit} effectively conceals the bit obtained from evaluating $f(s)$; and (ii) if com_{bit} represents a commitment to 1, the payment will be authorized. We observe that, in our implementation of the $\mathsf{PayForWitness}$ protocol, the payment is only authorized if S knows a witness w and a statement x such that $(x, w) \in \mathsf{R}_{\mathsf{AS}}$. We use this fact to encode the previous points in the following relations:

- $\mathsf{R}_{\Sigma}((com_{bit}, x); (open_{bit}, w)) \Leftrightarrow (com_{bit}, open_{bit}) := \mathsf{Commit}(1) \wedge (x, w) \in \mathsf{R}_{\mathsf{AS}}) \vee (x, w) \in \widehat{\mathsf{R}_{\mathsf{AS}}}$
- $\mathsf{R}_{\Omega}((f, com_{bit}); (s, open_{bit})) \Leftrightarrow (com_{bit}, open_{bit}) := \mathsf{Commit}(f(s))$

Furthermore, we observe that if the relation R_{prod} can be proven with a Sigma protocol, we can resort to our second construction where the seller uses a single NIZK instance to prove the following relation:

$$\mathsf{R}_{\Sigma}((f, x); (s, w)) \Leftrightarrow (f(s) = 1 \wedge (x, w) \in \mathsf{R}_{\mathsf{AS}}) \vee (x, w) \in \widehat{\mathsf{R}_{\mathsf{AS}}}$$

$\mathsf{ZKCSPProve}(s, f, \mathbf{pp})$	$\mathsf{ZKCSPVerify}(f, \mathbf{pp}, \boldsymbol{\pi}, x, ct)$
$\mathbf{pp} := (p, crs_\Omega, crs_\Sigma)$	$\mathbf{pp} := (p, crs_\Omega, crs_\Sigma)$
$bit := f(s)$	$\boldsymbol{\pi} := (\pi_\Omega, \pi_\Sigma)$
$(com_{bit}, open_{bit}) \leftarrow \mathsf{COM.Commit}(p, bit)$	$ct := (com_{bit})$
$(x, w) \leftarrow \mathsf{R_{AS}.GenR}(1^n)$	$b_0 \leftarrow \mathsf{NIZK}_{\mathsf{R}_\Omega}.\mathsf{Verify}(crs_\Omega, \pi_\Omega,$
$\pi_\Omega \leftarrow \mathsf{NIZK}_{\mathsf{R}_\Omega}.\mathsf{Prove}(crs_\Omega, (f, com_{bit}, p),$	$\quad (f, com_{bit}, p))$
$\quad (s, open_{bit}))$	$b_1 \leftarrow \mathsf{NIZK}_{\mathsf{R}_\Sigma}.\mathsf{Verify}(crs_\Sigma, \pi_\Sigma,$
$\pi_\Sigma \leftarrow \mathsf{NIZK}_{\mathsf{R}_\Sigma}.\mathsf{Prove}(crs_\Sigma, (x, com_s),$	$\quad (x, com_{bit}))$
$\quad (w, open_s))$	$\mathbf{return}\ b_0 \wedge b_1$
$\mathbf{return}\ ((\pi_\Omega, \pi_\Sigma), x, (com_{bit}))$	

Fig. 8: Algorithms ZKCSPProve and ZKCSPVerify.

In Appendix C.6 we showcase a proof of concept where R_{prod} can be proven as a Sigma protocol. Hence, we can utilize the same API defined for the ZKCPProve and ZKCPVerify algorithms presented in Fig. 8.

Properties Of The Adaptor Signature Hard Relation.

The protocol requires relations $\mathsf{R_{AS}}$ and $\widehat{\mathsf{R_{AS}}}$ to be mutually exclusive. This means that a *Seller* cannot know two witnesses w and $\widehat{w}$ for the same statement x such that $(x, w) \in \mathsf{R_{AS}}$ and $(x, \widehat{w}) \in \widehat{\mathsf{R_{AS}}}$. Otherwise, a malicious seller could compute a proof for R_Σ using $(x, \widehat{w}) \in \widehat{\mathsf{R_{AS}}}$ and later use $(x, w) \in \mathsf{R_{AS}}$ in the execution phase. This would result in the seller getting the α from the buyer without really knowing s such that $f(s) = 1$, in other words, without really providing the service. To capture this, we define the *claw-free* property:

Definition 26 (Claw free). *Two hard relations* R *and* $\widehat{\mathsf{R}}$ *are* claw free *if for all* PPT $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(n)$ *such that*

$$\Pr\left[(x, w) \in \mathsf{R} \wedge (x, \widehat{w}) \in \widehat{\mathsf{R}} \,\middle|\, x, w, \widehat{w} \leftarrow \mathcal{A}(1^n)\right] \leq \mathsf{negl}(n)$$

Another requirement of $\mathsf{R_{AS}}$ and $\widehat{\mathsf{R_{AS}}}$ is that their statements should be indistinguishable. This helps us hide which branch of the R_Σ relation is true. We formalize this property as follows:

Definition 27 (Indistinguishability). *Two hard relations* R *and* $\widehat{\mathsf{R}}$ *are* indistinguishable *if for all* PPT $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(n)$ *such that*

$$\left| \Pr\left[1 \leftarrow \mathcal{A}(x) | (x, w) \leftarrow \mathsf{GenR}(1^n)\right] - \Pr\left[1 \leftarrow \mathcal{A}(x) | (x, w) \leftarrow \widehat{\mathsf{GenR}}(1^n)\right] \right| \leq \mathsf{negl}(n)$$

Observation. The claw-freeness property implies that

$$\Pr\left[(x, w) \in \widehat{\mathsf{R}} | (x, w) \leftarrow \mathsf{GenR}(1^n)\right] \leq \mathsf{negl}(n) \wedge \Pr\left[(x, w) \in \mathsf{R} | (x, w) \leftarrow \widehat{\mathsf{GenR}}(1^n)\right] \leq \mathsf{negl}(n)$$

Therefore, proving $(x, w) \in \mathsf{R}$ implies that $(x, w) \notin \widehat{\mathsf{R}}$ with overwhelming probability.

Theorem 2. *Let* $\mathsf{R_{AS}}$ *be the discrete logarithm relation with respect to a group generator* g*. i.e.* $\mathsf{R_{AS}}(x; w) \Leftrightarrow x = g^w$*. Let* $\widehat{\mathsf{R_{AS}}}$ *be the discrete logarithm relation with respect to a different generator of the same group* h*. i.e.* $\widehat{\mathsf{R_{AS}}}(x; w) \Leftrightarrow x = h^w$*. Assume that the discrete logarithm between* g *and* h *is unknown. i.e. the secret value* β *such that* $h = g^\beta$ *is unknown. Then* $\mathsf{R_{AS}}$ *and* $\widehat{\mathsf{R_{AS}}}$ *are claw-free.*

Proof sketch 3 *It follows from the fact that an adversary that outputs* $\widehat{w}$ *on input* (x, w) *such that* $(x, \widehat{w}) \in \widehat{\mathsf{R}}$ *could be used to find the discrete logarithm between* g *and* h*:* $\beta = w/\widehat{w}$*.*

Theorem 3. *Let* $\mathsf{R_{AS}}$ *and* $\widehat{\mathsf{R_{AS}}}$ *be the hard relations defined in Theorem 2 with respect to the same generators* g *and* h*. Then* $\mathsf{R_{AS}}$ *and* $\widehat{\mathsf{R_{AS}}}$ *are indistinguishable.*

Proof sketch 4 *It follows from the fact that the set of witnesses and statements are the same for both hard relations and the* GenR *algorithms of both relations yield the same distributions on both witnesses and statements.*

C.3 Everything Together: Our Modular Algebraic Proof Contingent Service Payment

We leverage Fig. 5 to illustrate our Modular Algebraic Proof Contingent Service Payment. The MAPCSP follows the same design as MAPCP, except that instead of using the ZKCPProve and ZKCPVerify algorithms, we use the algorithms ZKCSPProve and ZKCSPVerify defined in Fig. 8. The seller begins by generating proofs using the ZKCSPProve algorithm, resulting in $\boldsymbol{\pi}$ (which includes π_Σ and π_Ω); the statement x of the relation $\mathsf{R_{AS}}$ (or the relation $\widehat{\mathsf{R_{AS}}}$) and the commitment to the value *bit* (that is the result of evaluating $f(s)$).

The seller sends these elements to the buyer. Then the buyer verifies the proofs. At this point, the buyer knows that the seller knows a witness w that

either $(x, w) \in \mathsf{R}_{\mathsf{AS}}$ or $(x, w) \in \widehat{\mathsf{R}_{\mathsf{AS}}}$. If the former is the case, then that would mean that the seller is providing the service.

The buyer then only needs to purchase the witness w to learn which branch of the relation R_{Σ} is true. By learning this, the buyer learns the value of the knowledge bit *bit*.

Thus, the seller and buyer proceed with the PayForWitness protocol. Upon completion, the seller receives the α funds, and the buyer learns that knowledge bit *bit* is indeed 1, which implies that the service is provided.

As a remark, if *bit* is 0, the seller will not get the α funds upon completing the PayForWitness protocol. The reason is that the adaptor signature scheme used in the PayForWitness protocol should be instantiated w.r.t. the relation R_{AS}. This implies that the seller gets the α funds only if they know a witness w such that $(x, w) \in \mathsf{R}_{\mathsf{AS}}$.

C.4 Security Definitions

Here we recall the security notions of a zero-knowledge contingent service payment, namely, *Extractability* and *Zero-knowledge*, as defined in [8].

Let $View_B$ be the buyer's view on an execution of the ZKCSP protocol, defined as: $View_B(s, f) := Coins_B \parallel Messages(\langle S(\ldots), B(\ldots)\rangle) \parallel Out(\langle S(\ldots), B(\ldots)\rangle)$

Where $Coins_B$ denotes the randomness used by the buyer, *Messages* denotes the messages exchanged between the parties executing the ZKCSP protocol, and *Out* denotes the outputs of all parties after executing the ZKCSP protocol.

Definition 28 (Extractability).

A ZKCSP *is* extractable *if at the end of the protocol the balance of any possibly malicious and efficient seller algorithm $\widehat{S}$ increases with non-negligible probability, then there exists an efficient extractor algorithm* Ext *that on input the predicate f, and the view of the buyer $View_B$, outputs bit such that $f(\widehat{s}) = bit = 1$.*

Intuitively, this property ensures that, at the end of the protocol, if the seller gets the α from the buyer, then the buyer knows that seller is providing the service.

Definition 29 (Zero-knowledge).

A ZKCSP *is* zero-knowledge *if for any possibly malicious and efficient $\widehat{B}$, there exists an efficient simulator algorithm $\mathsf{Sim}^{\widehat{B}}$ which, on input the predicate f, outputs a distribution which is computationally indistinguishable from $View_{\widehat{B}}(s, f)$*

Intuitively, this property guarantees that any potentially malicious $\widehat{B}$ does not inadvertently gain information about the secret input s during the protocol execution, unless $\widehat{B}$ transfers α to S.

C.5 Security Analyis

Theorem 4. *If the predicate f is one-way; the adaptor signature scheme has existential unforgeability and witness extractability; the hard relation of the adaptor signature scheme* $\mathsf{R_{AS}}$ *is injective; the hard relations* $\mathsf{R_{AS}}$ *and* $\widehat{\mathsf{R_{AS}}}$ *are claw-free and indistinguishable; the commitment scheme is binding and hiding; and the NIZKs used for* R_Σ *and* R_Ω *are knowledge-sound and zero-knowledge, then the* MAPCSP *protocol is secure.*

Proof sketch 5 (Extraction for ZKCSP-Protocol) *The proof follows the same steps as the proof for extraction of* MAPCP *with a difference at the end. The extractor retrieves the witness w and instead of then recovering the s, the extractor returns whether* $(x, w) \in \mathsf{R_{AS}}$ *or not. The claw-freeness property of* $\mathsf{R_{AS}}$ *and* $\widehat{\mathsf{R_{AS}}}$ *ensures that this check will be true with overwhelming probability.*

Proof sketch 6 (Zero-knowledge for ZKCSP-Protocol) *The proof is the same as the one of zero-knowledge for* ZKCP-Protocol *with the difference that: (i) Instead of hiding the commit to the secret, we use the hiding of the commit to the knowledge bit. (ii) The encryption scheme is not required. (iii) the indistinguishability property of the hard relations* $\mathsf{R_{AS}}$ *and* $\widehat{\mathsf{R_{AS}}}$ *needs to be used.*

C.6 Application: Selling the Service of Knowing a Schnorr Signature

To illustrate an application of the MAPCSP, we developed a proof of concept. We use the same building blocks as in Section 6: Schnorr adaptor signatures, the discrete logarithm relation for $\mathsf{R_{AS}}$, and the relation R_{prod}.

The relation R_Σ that we will need to prove is:

$$\mathsf{R}_\Sigma((x, \mathsf{vk}, e, r); (w, s, \mathsf{sk})) \Leftrightarrow x = h^w \vee \begin{pmatrix} g^s \mathsf{vk}^e = r \wedge \\ \mathsf{vk} = g^{\mathsf{sk}} \wedge \\ x = g^w \end{pmatrix}$$

Note that, for this particular implementation, we need two distinct generators from the same group, with an unknown discrete logarithm between them. To derive these generators, the buyer and seller can either engage in a Diffie-Hellman exchange or compute the second generator by hashing the first one onto the curve.

With all the building blocks set, the protocol follows the steps described in Section 5.

Testbed and Evaluation Results. We have developed our proof of concept [19] in Rust using the libraries and benchmarking that we did with MAPCP. For benchmarking, we ran the cryptographic operations for the complete protocol 1000 times and calculated the average times in microseconds. The experiments have been executed in the same machine.

We show the results of the benchmarking in Table 2. Here, we see that the computation overhead for most of the operations is below 1 millisecond whereas

Table 2: Benchmarking results. Network usage denotes the amount of data that both parties send to each other while interacting. Signature size denotes the size of the signatures to be stored in the blockchain.

	Secp256k1	NistP256
Seller setup:	0.46 μs	1.3 μs
Buyer lock payment:	0.56 μs	1.59 μs
Seller adapt signature:	0.16 μs	0.46 μs
Buyer extracting witness:	< 0.01 μs	< 1 μs
network usage:	576 bytes	576 bytes
blockchain storage:	128 bytes	128 bytes

the communication overhead is below 600 bytes, making the approach practical even in commodity hardware. Additionally, to study the impact of different curves, we repeated the experiments with curves secp256k1 and nistp256, observing that the former (currently used e.g., in Bitcoin) yields more efficient computation. Nevertheless, both curves impose a small computation overhead, whereas the communication overhead remains the same.

D ZKCP Security Proofs

D.1 Extraction

Proof (Proof of extraction for MAPCP*).* We want to prove extraction 2 for the protocol described in Fig. 5. To increase the balance of $\widehat{Seller}$ at the end of the protocol, it is necessary that $\widehat{Seller}$ has published transaction tx_p together with its valid signatures $\sigma_{B,p}$ and $\sigma_{S,p}$. This requires that both signatures for tx_p verify. Thus, let W be the event:

$$W = \text{“}\mathsf{Vrfy}(\mathsf{vk}_{B,p}, \sigma_{B,p}, tx_p) = 1 \wedge \mathsf{Vrfy}(\mathsf{vk}_{S,p}, \sigma_{S,p}, tx_p) = 1\text{”}$$

Therefore, we formalize the extraction property as:

$$\Pr[W] > negl \Rightarrow \exists \mathsf{ExtPPT} :$$
$$\Pr\left[f(\widehat{s}) = 0 | \widehat{s} \leftarrow \mathsf{Ext}(f, View_B)\right] < negl$$

If the balance of $\widehat{Seller}$ has increased, then that means that the pay transaction tx_p has been published. Therefore all the preconditions are satisfied and such algorithm must exist. This algorithm is defined in Fig. 9.

We want to prove that, if the condition on the left hand side is satisfied, then the extraction algorithm we presented returns s' such that $f(s') = 1$ with overwhelming probability.

Event W only happens when transaction tx_p is submitted along with the signatures $\sigma_{B,p}$ and $\sigma_{S,p}$. Hence the malicious $\widehat{Seller}$ must have found a way to

$\mathsf{Ext}(f, View_B)$

1 : $\sigma_{B,p}, \hat{\sigma}_{B,p}, x, ct := View_B$
2 : $w' \leftarrow \mathsf{Extract}(\sigma_{B,p}, \hat{\sigma}_{B,p}, x)$
3 : $s' \leftarrow \mathsf{Dec}(w', ct)$
4 : **return** s'

Fig. 9: Extractor algorithm.

compute them. $\widehat{Seller}$ knows the signing key used in $\sigma_{S,p}$, so they can compute it normally.

$\widehat{Seller}$ requires $\hat{\sigma}_{B,p}$ in order to compute $\sigma_{B,p}$. Otherwise $\widehat{Seller}$ would have been able to forge a valid signature breaking the signature unforgeability property of the underlying digital signature scheme as shown in Fig. 10.

The pre-signature $\hat{\sigma}_{B,p}$ is computed by an honest *Buyer* using the $\mathsf{PreSign}$ algorithm. However, following the protocol, *Buyer* only computes the pre-signature if both proofs π_Σ and π_Ω verify. Therefore, we say that

$$\Pr[W] > \mathsf{negl}(n) \Rightarrow$$
$$\mathsf{Verify}_\Sigma(crs_\Sigma, (ct, x, com_s, p), \pi_\Sigma) = 1 \wedge \mathsf{Verify}_\Omega(crs_\Omega, (f, com_s, p), \pi_\Omega) = 1$$

In the extraction algorithm, we compute w' and s'. The witness w' is computed by extracting from the signature and pre-signature. It must be the case that $(x, w') \in \mathsf{R_{AS}}$. Otherwise, the malicious $\widehat{Seller}$ could be used to break witness extractability of the adaptor signature scheme as shown in Fig. 11.

Since the relation $\mathsf{R_{AS}}$ is injective, this w' must be the only witness such that $(x, w') \in \mathsf{R_{AS}}$.

The extractor algorithm of π_Σ, denoted Ext_Σ, on input the statements com_s, x, ct, returns $\widehat{open_s}$, $\widehat{w}$ and $\widehat{s}$. These witnesses satisfy the hard relation of the NIZK:

$$(x, \widehat{w}) \in \mathsf{R_{AS}} \wedge (com_s, open_s) = \mathsf{Commit}(p, \widehat{s}) \wedge ct = \mathsf{Enc}(\widehat{w}, \widehat{s})$$

Otherwise, $\widehat{Seller}$ could be used to break the knowledge-soundness of the NIZK as shown in Fig. 12.

Again, since $\mathsf{R_{AS}}$ is injective, for each x, there is only one w satisfying $(x, w) \in \mathsf{R_{AS}}$, therefore $w' = \widehat{w}$. Thus, because of the correctness of the encryption scheme, we know that $\widehat{s} = \mathsf{Dec}(w', ct)$, which is precisely what the second line of the extractor algorithm does. We can conclude that $\widehat{s} = s'$.

The extractor algorithm of π_Ω, denoted Ext_Ω, returns $\widetilde{s}$ and $\widetilde{open_s}$ such that

$$f(\widetilde{s}) = 1 \wedge (com_s, \widetilde{open_s}) = \mathsf{Commit}(p, \widetilde{s})$$

Again, this must be the case, as otherwise, $\widehat{Seller}$ could be used to break the knowledge-soundness of the NIZK as shown in Fig. 13

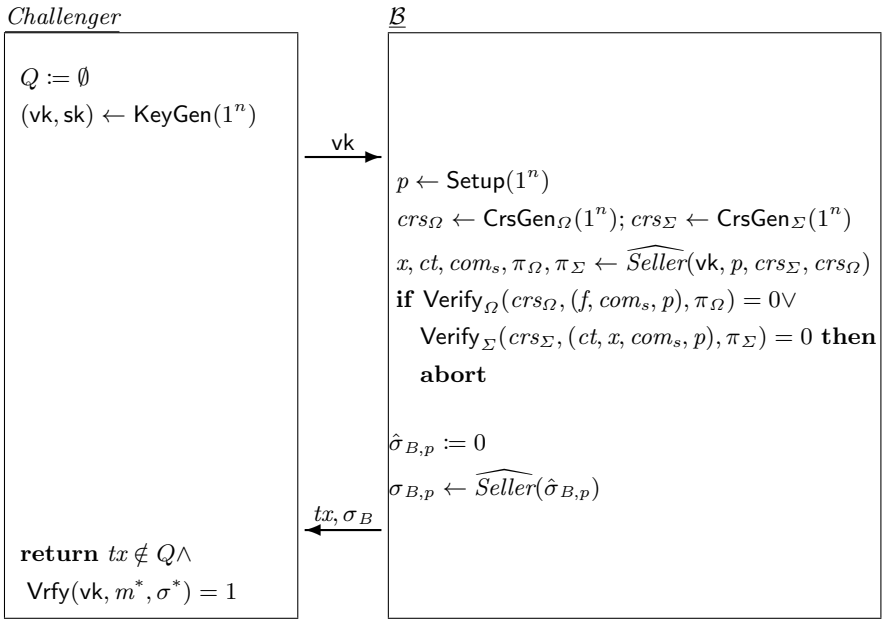

Fig. 10: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the signature unforgeability of the digital signature scheme.

Since com_s is part of the statement of both proofs it must be the case that $\widetilde{s} = \widehat{s} = s'$. Otherwise, the extractors of both proofs could be used to break the binding of the commitment scheme as shown in Fig. 14.

Therefore, $s' = \widetilde{s}$ and it is true that $f(s') = 1$.

D.2 Zero-knowledge

Proof (Proof of zero-knowledge for MAPCP*).*

In order to prove zero-knowledge 3 for the protocol described in Fig. 5, we need to find a simulator that on input f generates a view that is indistinguishable from the one generated by the actual protocol. In order to construct such simulator we modify the intended pseudocode of the *Buyer* line by line creating computationally indistinguishable simulators in each step until we arrive at the simulator that does not require the secret s as input.

For reference, the pseudocode of the protocol is shown in Fig. 15.

The code of the simulator, named Sim_5, will be in Fig. 25.

Note that both the seller protocol and the simulator can have two different possible views: $(com_s, x, ct, \pi_\Omega, \pi_\Sigma)$ and $(com_s, x, ct, \pi_\Omega, \pi_\Sigma, \sigma)$ depending on whether the buyer oracle returns a valid $\hat{\sigma}$ or not.

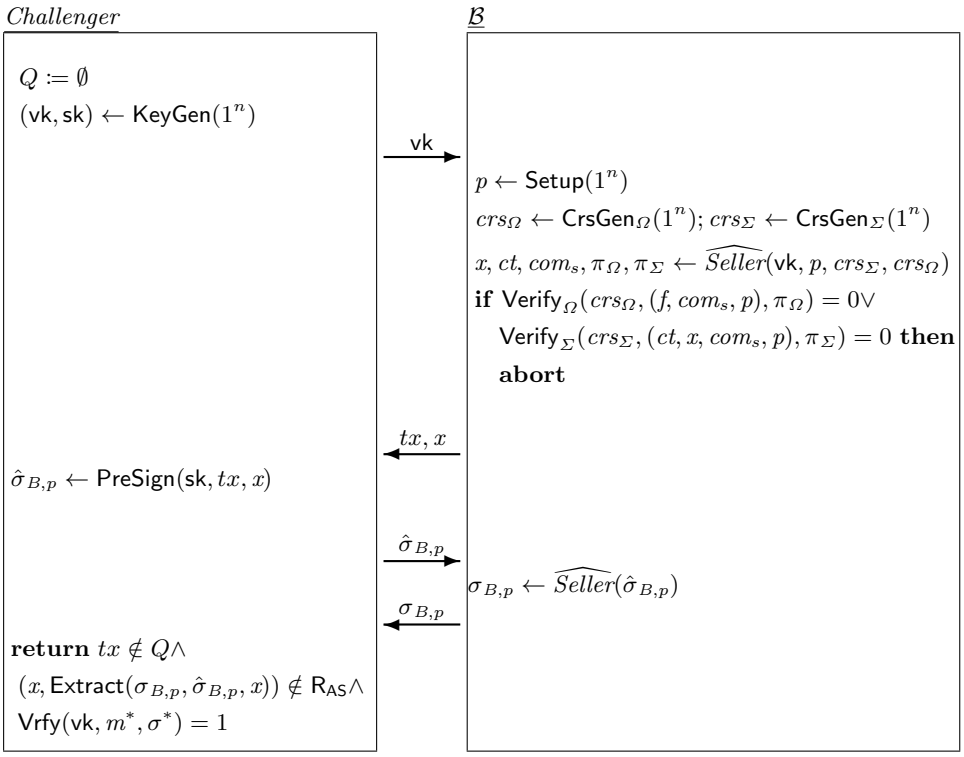

Fig. 11: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the witness extractability of the adaptor signature scheme.

We claim that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \overset{c}{\equiv} \{Sim_5(f, crs_\Omega, crs_\Sigma, p)\}_f$$

We present the code of the first simulator in Fig. 16.
We claim that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \overset{c}{\equiv} \{Sim_1(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Assume that this is not the case. Then, there would exist a distinguisher $\mathcal{A}$ between Sim_1 and the seller. This distinguisher wins in the game that we show in Fig. 17.

If such distinguisher exists, we can build a distinguisher $\mathcal{B}$ that breaks the zero-knowledge property of the NIZK of π_Ω as we show in Fig. 18.

Arriving at a contradiction. Therefore, it must hold that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \overset{c}{\equiv} \{Sim_1(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

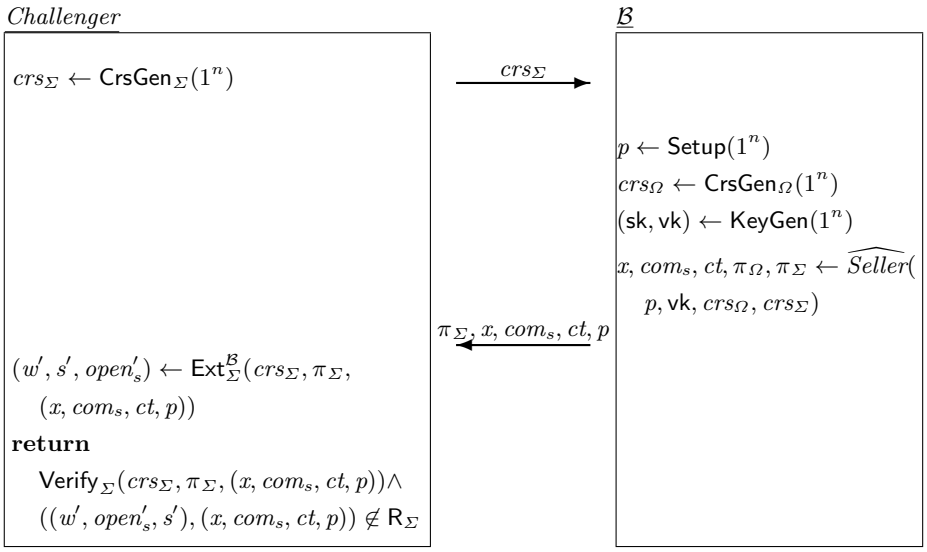

Fig. 12: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the knowledge-soundness of proof π_Σ.

We present the code of the second simulator in Fig. 19.
We claim that:

$$\{Sim_1(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

An therefore it would hold that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Assume that this is not the case. Then, there would exist a distinguisher $\mathcal{A}$ that wins the game shown in Fig. 20, with non-negligible probability.

Then, we could create a distinguisher $\mathcal{B}$ that breaks the zero knowledge property of the NIZK proof system for π_Σ as shown in Fig. 21.

Arriving at a contradiction. Therefore, it must hold that:

$$\{Sim_1(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

We present the code of the third simulator in Fig. 22.
We claim that:

$$\{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

An therefore it would hold that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

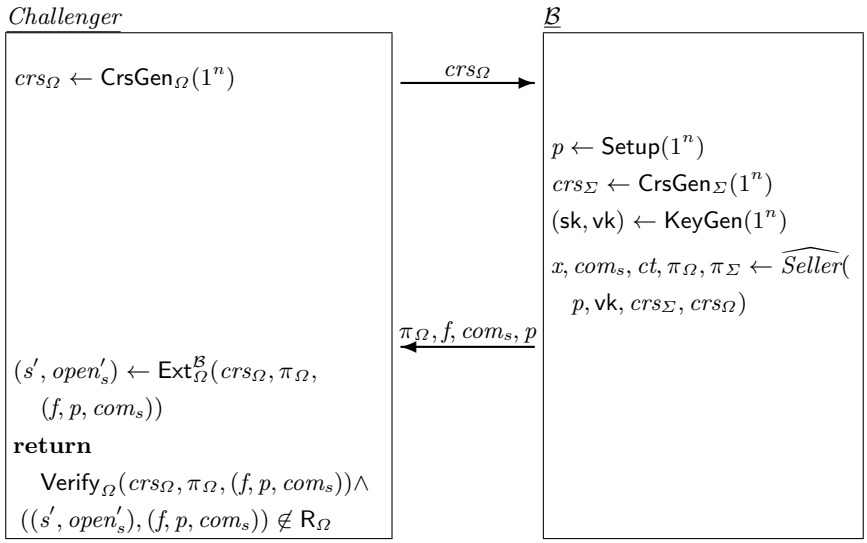

Fig. 13: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the knowledge-soundness of proof π_Ω.

Assume that this is not the case. Then, there would exist a distinguisher $\mathcal{A}$ that is able to distinguish between algorithms Sim_2 and Sim_3 as modeled in Fig. 23.

Then, we could create a distinguisher $\mathcal{B}$ that breaks the hiding property of the commitment scheme as shown in Fig. 24.

Arriving at a contradiction. Therefore, it must hold that:

$$\{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \overset{c}{\equiv} \{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

We present the code of the fourth simulator in Fig. 25.

We claim that:

$$\{Sim_4(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \overset{c}{\equiv} \{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

An therefore it would hold that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \overset{c}{\equiv} \{Sim_4(f, s, crs_\Omega, crs_\Sigma, p)\}_f$$

Assume for the sake of contradiction that there exists a distinguisher $\mathcal{A}$ such that given game of Fig. 26 is able to win with non-negligible probability.

Note that, due to the definition of the AdaptPayment protocol, Sim_4 is a faithful computation of Sim_3 in the cases where $f(s) = 0$. Therefore, the adversary $\mathcal{A}$ can distinguish in the cases where they input a secret s such that

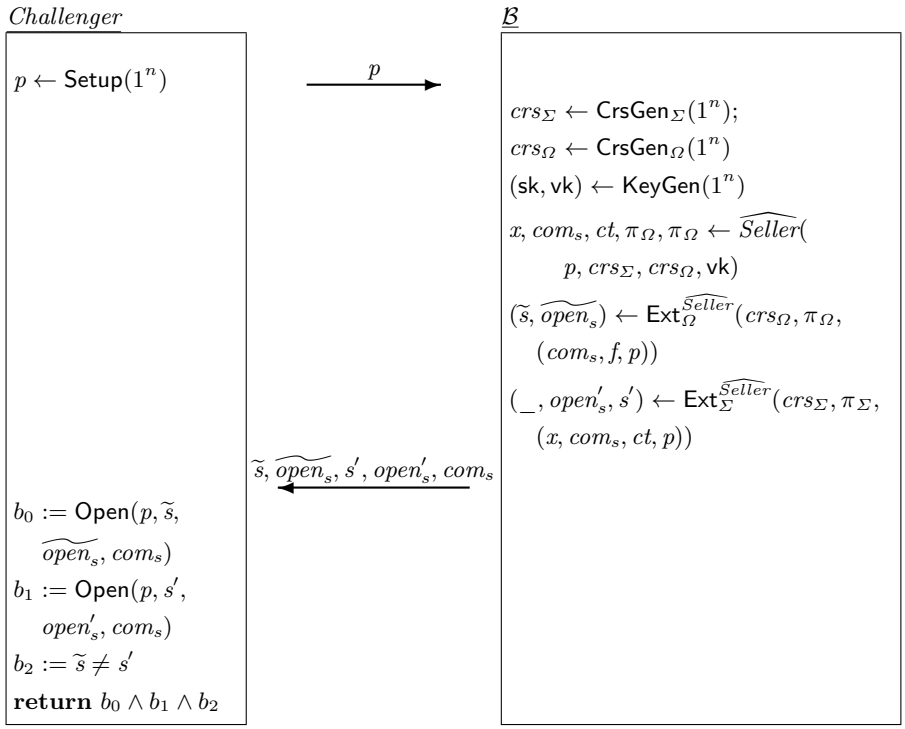

Fig. 14: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the binding property of the commitment scheme.

$f(s) = 1$. But $\mathcal{A}$ being able to distinguish between Sim_4 and Sim_3 with non-negligible probability would imply that $\mathcal{A}$ can find such s with non-negligible probability.

Therefore, we could create a distinguisher $\mathcal{B}$ that breaks the one-wayness property of the predicate f in Fig. 27.

We present the fifth and final simulator in Fig. 28

We claim that:

$$\{Sim_5(f, crs_\Omega, crs_\Sigma, p)\}_f \stackrel{c}{\equiv} \{Sim_4(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

An therefore it would hold that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_5(f, crs_\Omega, crs_\Sigma, p)\}_f$$

Assume for the sake of contradiction that there exists a distinguisher $\mathcal{A}$ such that given game of Fig. 29 is able to win with non-negligible probability.

We need to show that, if $\mathcal{A}$ wins with non-negligible probability, then we can use it to create an adversary $\mathcal{B}$ that breaks the IND-CPA of the encryption scheme as shown in Fig. 30.

$Seller(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetup}(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProve}(f, s, x, w, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_s, x, ct)$
4 : **wait** Δ_t
5 : $\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$

Fig. 15: Pseudocode of the seller.

$Sim_1(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetup}(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_1(f, s, x, w, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_s, x, ct)$
4 : **wait** Δ_t
5 : $\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$

$\mathsf{ZKCPProveSim}_1(f, s, x, w, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$

1 : $\pi_\Omega \leftarrow \mathsf{ProveSim}_\Omega(crs_\Omega, (f, com_s, p))$
2 : $\pi_\Sigma \leftarrow \mathsf{Prove}(crs_\Sigma, (x, ct, com_s, p), (w, s, open_s))$
3 : **return** π_Ω, π_Σ

Fig. 16: Pseudocode of the first simulator.

Therefore, there does not exist such distinguisher and we have:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \overset{\mathrm{c}}{\equiv} \{Sim_5(f, crs_\Omega, crs_\Sigma, p)\}_f$$

E ZKCSP Security Proofs

E.1 Extraction

Proof (Proof of extraction for MAPCSP*).* We want to prove extraction Definition 2 for the MAPCSP protocol To increase the balance of $\widehat{Seller}$ at the end of the protocol, it is necessary that $\widehat{Seller}$ has published transaction tx_p together with its valid signatures $\sigma_{B,p}$ and $\sigma_{S,p}$. This requires that both signatures for tx_p verify. Thus, let W be the event:

$$W = \text{“}\mathsf{Vrfy}(\mathsf{vk}_{B,p}, \sigma_{B,p}, tx_p) = 1 \wedge \mathsf{Vrfy}(\mathsf{vk}_{S,p}, \sigma_{S,p}, tx_p) = 1\text{”}$$

Therefore, we formalize the extraction property as:

$$\Pr[W] > negl \Rightarrow \exists \mathsf{ExtPPT} :$$
$$\Pr\left[f(\widehat{s}) = 0 | \widehat{s} \leftarrow \mathsf{Ext}(f, View_B)\right] < negl$$

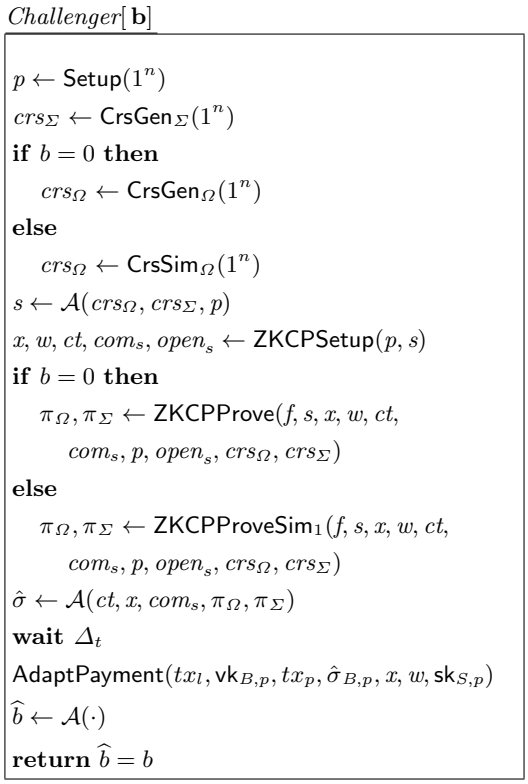

Fig. 17: Game where adversary $\mathcal{A}$ tries to distinguish between algorithms *Seller* and Sim_1.

If the balance of $\widehat{Seller}$ has increased, then that means that the pay transaction tx_p has been published. Therefore the precondition is satisfied and such algorithm must exist. This algorithm is defined in Fig. 31.

We want to prove that, if the condition on the left hand side is satisfied, then the extraction algorithm we presented returns 1 with overwhelming probability.

Event W only happens when transaction tx_p is submitted along with the signatures $\sigma_{B,p}$ and $\sigma_{S,p}$. Hence the malicious $\widehat{Seller}$ must have found a way to compute them. $\widehat{Seller}$ knows the signing key used in $\sigma_{S,p}$, so they can compute it normally.

In order to compute $\sigma_{B,p}$, $\widehat{Seller}$ requires $\hat{\sigma}_{B,p}$. Otherwise $\widehat{Seller}$ would be able to forge a valid signature breaking the signature unforgeability property of the underlying digital signature scheme as shown in Fig. 32.

The pre-signature $\hat{\sigma}_{B,p}$ is computed by an honest *Buyer* using the PreSign algorithm. However, following the protocol, *Buyer* only computes the pre-signature if both proofs π_Σ and π_Ω verify. Therefore, we say that

$$\Pr[W] > negl \Rightarrow \mathsf{Verify}_{\Sigma}(crs_{\Sigma}, (com_{bit}, p, x), \pi_{\Sigma}) = 1 \wedge \mathsf{Verify}_{\Omega}(crs_{\Omega}, (com_{bit}, p), \pi_{\Omega}) = 1$$

Now, in the extractor algorithm, we use the extract algorithm of the adaptor signature scheme in order to get w' from $\hat{\sigma}_{B,p}$ and $\sigma_{B,p}$. We argue that this witness satisfies $(x, w) \in \mathsf{R}_{\mathsf{AS}}$. Otherwise, $\widehat{Seller}$ could be used to break witness extractability of the adaptor signature scheme as shown in Fig. 33.

Next, we see that π_{Σ} is a NIZK satisfying the knowledge-soundness property. Therefore its extractor algorithm $\mathsf{Ext}_{\Sigma}^{\widehat{Seller}}$, on input (x, com_{bit}) returns $(\widehat{w}, \widehat{open_{bit}})$ such that:

$$\begin{pmatrix} (x, \widehat{w}) \in \mathsf{R}_{\mathsf{AS}} \wedge \\ (com_{bit}, open_{bit}) = \mathsf{Commit}(p, 1) \end{pmatrix} \vee \begin{pmatrix} (x, \widehat{w}) \in \widehat{\mathsf{R}_{\mathsf{AS}}} \wedge \\ (com_{bit}, open_{bit}) = \mathsf{Commit}(p, 0) \end{pmatrix}$$

Otherwise, $\widehat{Seller}$ could be used to break the knowledge-soundness property of the NIZK as shown in Fig. 34.

Now, we have w' and $\widehat{w}$ such that $(x, w') \in \mathsf{R}_{\mathsf{AS}}$ and $(x, \widehat{w}) \in \mathsf{R}_{\mathsf{AS}} \vee (x, \widehat{w}) \in \widehat{\mathsf{R}_{\mathsf{AS}}}$.

Assume that $w' \neq \widehat{w}$. R_{AS} is injective, therefore necessarily $(x, \widehat{w}) \in \widehat{\mathsf{R}_{\mathsf{AS}}}$. Then we would have found an adversary $\mathcal{B}$ that using both $\widehat{Seller}$ and $\mathsf{Ext}_{\Sigma}^{\widehat{Seller}}$ breaks the claw-freeness of R_{AS} and $\widehat{\mathsf{R}_{\mathsf{AS}}}$ as shown in Fig. 35.

We conclude that $w' = \widehat{w}$. Since $(x, \widehat{w}) \in \mathsf{R}_{\mathsf{AS}}$, it must be the case that $(com_{bit}, \widehat{open_{bit}}) = \mathsf{Commit}(p, 1)$.

Since π_{Ω} is also a NIZK that is knowledge-sound, then we know that there exists an extractor $\mathsf{Ext}_{\Omega}^{\widehat{Seller}}$ that on input (f, com_{bit}), outputs $(\widetilde{open_{bit}}, \widetilde{s})$ such that:

$$bit = f(s) \wedge (com_{bit}, \widetilde{open_{bit}}) = \mathsf{Commit}(p, bit)$$

As otherwise, *Seller* could be used to break knowledge-soundness of the NIZK as shown in Fig. 36

We just need to show that $(com_{bit}, \widetilde{open_{bit}}) = \mathsf{Commit}(p, 1)$. To do this, we need to note that $\widetilde{open_{bit}} = \widehat{open_{bit}}$, as otherwise, an adversary would be able to use both extractors Ext_{Σ} and Ext_{Ω} to break the binding property of the commitment scheme as shown in Fig. 37.

Therefore, it must be the case that $\widetilde{open_{bit}} = \widehat{open_{bit}}$. Since we knew that $(com_{bit}, \widehat{open_{bit}}) = \mathsf{Commit}(p, 1)$, we can conclude that $(com_{bit}, \widetilde{open_{bit}}) = \mathsf{Commit}(p, 1)$. Hence, $bit = 1$ and, since $f(s) = bit$, it must be the case that $f(s) = 1$.

E.2 Zero-knowledge

Proof (Proof of zero-knowledge for MAPCSP*).*

In order to prove zero-knowledge 3 for the protocol described in Fig. 5, we need to find a simulator that on input f generates a view that is indistinguishable from the one generated by the actual protocol. In order to construct such

simulator, we modify the intended pseudocode of the *Buyer* line by line creating computationally indistinguishable simulators until we arrive at the final one that we need, showing in every step that these simulators generate a view that is indistinguishable from $View_{\widehat{B}}(s, f)$.

For reference, the pseudocode of the protocol is presented in Fig. 38.

We present the code for the first simulator in Fig. 39.

We claim that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_1(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Assume that this is not the case. Then, there would exist a distinguisher $\mathcal{A}$ between Sim_1 and the seller. This distinguisher wins the game that we show in Fig. 40.

If such distinguisher exists, we can build a distinguisher $\mathcal{B}$ that breaks the zero-knowledge property of the NIZK of π_Ω as we show in Fig. 41

Arriving at a contradiction. Therefore, it must hold that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_1(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

We present the code for the second simulator in Fig. 42.

We claim that:

$$\{Sim_1(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Therefore:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Assume that this is not the case. Then, there would exist a distinguisher $\mathcal{A}$ that wins the game shown in Fig. 43, with non-negligible probability.

Then, we could create a distinguisher $\mathcal{B}$ that breaks the zero knowledge property of the NIZK proof system for π_Ω as shown in Fig. 44.

Arriving at a contradiction. Therefore, it must hold that:

$$\{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

We present the code of the third simulator in Fig. 45.

We claim that:

$$\{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Therefore:

$$\{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Assume that this is not the case. Then, there would exist a distinguisher $\mathcal{A}$ that wins the game shown in Fig. 46.

Note that, due to the definition of the AdaptPayment protocol, Sim_3 is a faithful computation of Sim_2 (and therefore, a faithful computation of *Seller*) in

the cases where $f(s) = 0$. Therefore, the adversary $\mathcal{A}$ can distinguish in the cases where they input a secret s such that $f(s) = 1$. But $\mathcal{A}$ being able to distinguish between Sim_3 and Sim_2 with non-negligible probability would imply that $\mathcal{A}$ can find such s with non-negligible probability.

Therefore, we could create a distinguisher $\mathcal{B}$ that breaks the one-wayness property of the predicate f in Fig. 47.

Arriving at a contradiction. Therefore, it must hold that:

$$\{Sim_2(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

We present the code for the fourth simulator in Fig. 48.

We claim that:

$$\{Sim_4(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Therefore:

$$\{Sim_4(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{seller(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Assume that this is not the case. Then, there would exist a distinguisher $\mathcal{A}$ that wins the game shown in Fig. 49.

Then, we could create a distinguisher $\mathcal{B}$ that breaks the indistinguishability between $\mathsf{R_{AS}}$ and $\widehat{\mathsf{R_{AS}}}$ as shown in Fig. 50.

Arriving at a contradiction. Therefore, it must hold that:

$$\{Sim_3(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f} \stackrel{c}{\equiv} \{Sim_4(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

We present the code for the fifth simulator in Fig. 51.

We claim that:

$$\{Sim_5(f, crs_\Omega, crs_\Sigma, p)\}_{f} \stackrel{c}{\equiv} \{Sim_4(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

Therefore:

$$\{Sim_5(f, crs_\Omega, crs_\Sigma, p)\}_{f} \stackrel{c}{\equiv} \{seller(s, f)\}_{s,f}$$

Assume that this is not the case. Then, there would exist a distinguisher $\mathcal{A}$ that wins the game shown in Fig. 52.

Then, we could create a distinguisher $\mathcal{B}$ that breaks the hiding property of the commitment scheme with respect to com_{bit} as shown in Fig. 53.

Arriving at a contradiction. Therefore, it must hold that:

$$\{Sim_5(f, crs_\Omega, crs_\Sigma, p)\}_{f} \stackrel{c}{\equiv} \{Sim_4(s, f, crs_\Omega, crs_\Sigma, p)\}_{s,f}$$

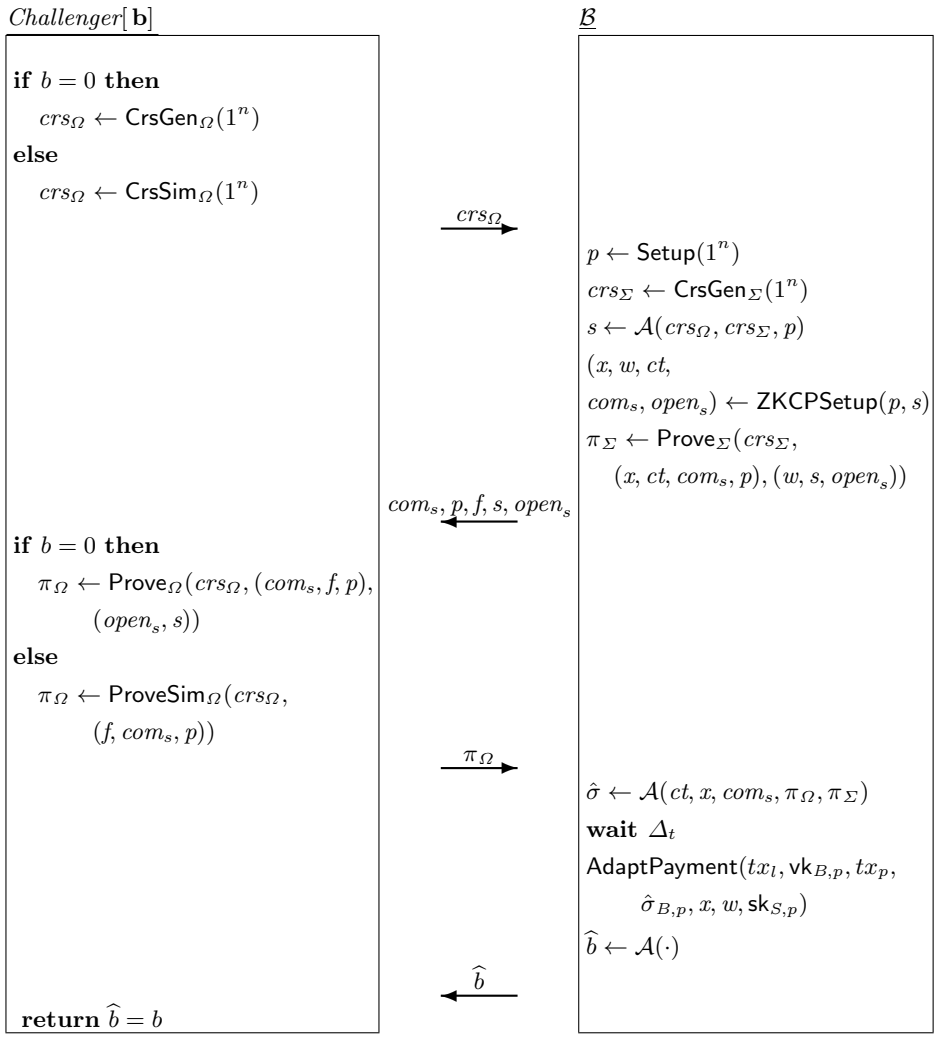

Fig. 18: Game showing how adversary $\mathcal{B}$ would use adversary $\mathcal{A}$ in order to break the zero knowledge of proof π_Ω.

$Sim_2(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetup}(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_2(f, s, x, w, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_s, x, ct)$
4 : **wait** Δ_t
5 : $\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
6 :

$\mathsf{ZKCPProveSim}_2(f, x, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$

1 : $\pi_\Omega \leftarrow \mathsf{ProveSim}_\Omega(crs_\Omega, (f, com_s, p))$
2 : $\pi_\Sigma \leftarrow \mathsf{ProveSim}_\Sigma(crs_\Sigma, (com_s, x, ct, p))$
3 : **return** π_Ω, π_Σ

Fig. 19: Pseudocode of the second simulator.

$Challenger[\mathbf{b}]$

$p \leftarrow \mathsf{Setup}(1^n)$
$crs_\Omega \leftarrow \mathsf{CrsSim}_\Omega(1^n)$
if $b = 0$ **then**
  $crs_\Sigma \leftarrow \mathsf{CrsGen}_\Sigma(1^n)$
else
  $crs_\Sigma \leftarrow \mathsf{CrsSim}_\Sigma(1^n)$
$s \leftarrow \mathcal{A}(crs_\Omega, crs_\Sigma, p)$
$x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetup}(p, s)$
if $b = 0$ **then**
  $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_1(f, s, x, w, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
else
  $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_2(f, x, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
$\hat{\sigma} \leftarrow \mathcal{A}(ct, x, com_s, \pi_\Omega, \pi_\Sigma)$
wait Δ_t
$\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
$\widehat{b} \leftarrow \mathcal{A}(\cdot)$
return $\widehat{b} = b$

Fig. 20: Game where adversary $\mathcal{A}$ tries to distinguish between algorithms Sim_1 and Sim_2.

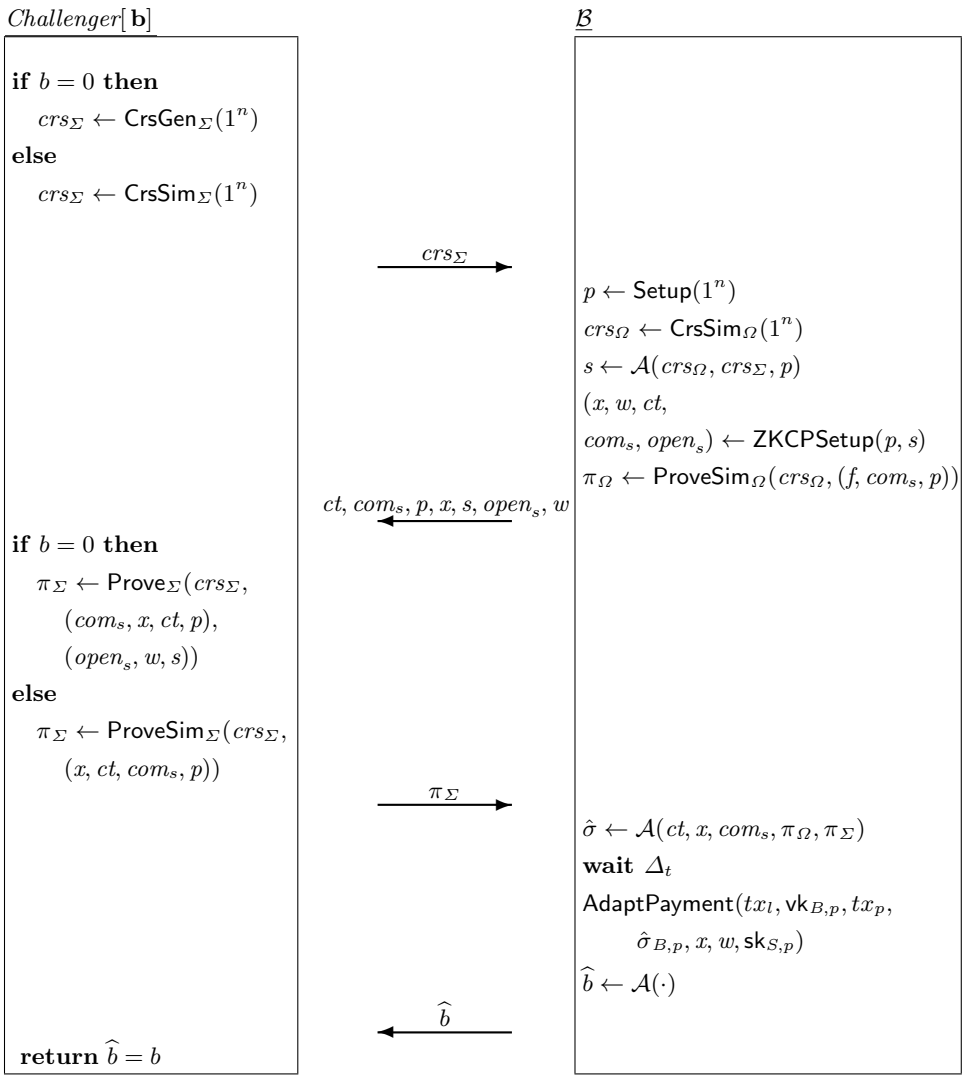

Fig. 21: Game showing how adversary $\mathcal{B}$ would use adversary $\mathcal{A}$ in order to break the zero-knowledge or proof π_Σ.

$Sim_3(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetupSim}_3(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_2(f, x, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_s, x, ct)$
4 : **wait** Δ_t
5 : $\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
6 :

$\mathsf{ZKCPSetupSim}_3(p, s)$

1 : $(com_s, open_s) \leftarrow \mathsf{Commit}(0)$
2 : $(x, w) \leftarrow \mathsf{GenR}(1^n)$
3 : $ct \leftarrow \mathsf{Enc}(w, s)$
4 : **return** $x, w, ct, com_s, open_s$

Fig. 22: Pseudocode of the third simulator.

$Challenger[\mathbf{b}]$

$p \leftarrow \mathsf{Setup}(1^n)$
$crs_\Omega \leftarrow \mathsf{CrsGen}_\Omega(1^n)$
$crs_\Sigma \leftarrow \mathsf{CrsSim}_\Sigma(1^n)$
$s \leftarrow \mathcal{A}(crs_\Omega, crs_\Sigma, p)$
if $b = 0$ **then**
  $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetup}(p, s)$
else
  $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetupSim}_1(p, s)$
$\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_2(f, x, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
$\hat{\sigma} \leftarrow \mathcal{A}(ct, x, com_s, \pi_\Omega, \pi_\Sigma)$
wait Δ_t
$\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
$\widehat{b} \leftarrow \mathcal{A}(\cdot)$
return $\widehat{b} = b$

Fig. 23: Game where adversary $\mathcal{A}$ tries to distinguish between algorithms Sim_2 and Sim_3.

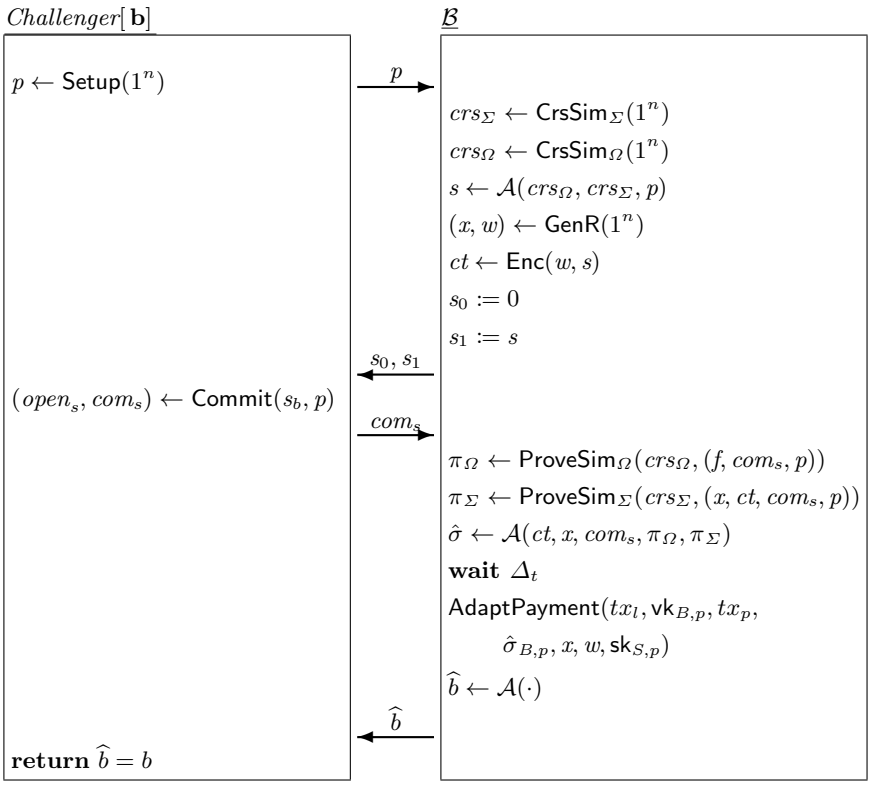

Fig. 24: Game showing how adversary $\mathcal{B}$ would use $\mathcal{A}$ in order to break the hiding property of the commitment scheme.

$Sim_4(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetupSim}_3(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_2(f, x, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_s, x, ct)$
4 : **wait** Δ_t
5 : **abort**

Fig. 25: Pseudocode of the fourth simulator.

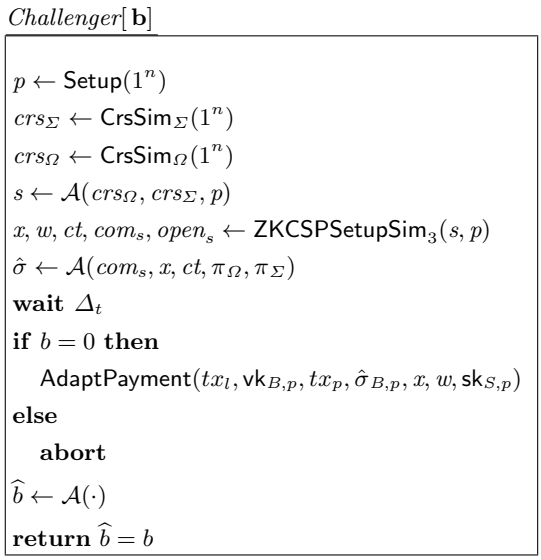

Fig. 26: Game where adversary $\mathcal{A}$ tries to distinguish between Sim_3 and Sim_4.

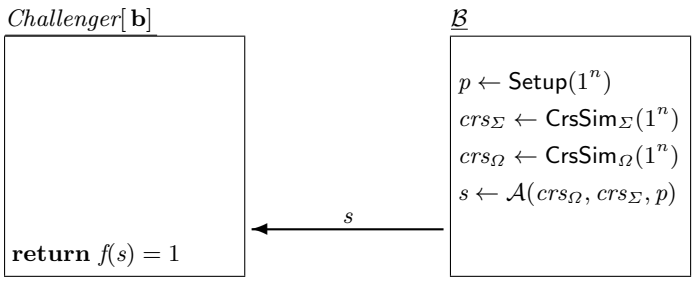

Fig. 27: Game showing how adversary $\mathcal{B}$ would use $\mathcal{A}$ in order to break the one-wayness of predicate f.

$Sim_5(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetupSim}_5(p)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_2(f, x, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_s, x, ct)$
4 : **wait** Δ_t
5 : **abort**

$\mathsf{ZKCPSetupSim}_5(p)$

1 : $(com_s, open_s) \leftarrow \mathsf{Commit}(0)$
2 : $(x, w) \leftarrow \mathsf{GenR}(1^n)$
3 : $ct \leftarrow \mathsf{Enc}(w, 0)$
4 : **return** $x, w, ct, com_s, open_s$

Fig. 28: Pseudocode of the fifth simulator.

$Challenger[\mathbf{b}]$

$p \leftarrow \mathsf{Setup}(1^n)$
$crs_\Sigma \leftarrow \mathsf{CrsSim}_\Sigma(1^n)$
$crs_\Omega \leftarrow \mathsf{CrsSim}_\Omega(1^n)$
$s \leftarrow \mathcal{A}(crs_\Omega, crs_\Sigma, p)$
if $b = 0$ **then**
  $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetupSim}_4(p, s)$
else
  $x, w, ct, com_s, open_s \leftarrow \mathsf{ZKCPSetupSim}_5(p)$
endif
$\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCPProveSim}_2(f, x, ct, com_s, p, open_s, crs_\Omega, crs_\Sigma)$
$\hat{\sigma} \leftarrow \mathcal{A}(ct, x, \pi_\Omega, \pi_\Sigma)$
wait Δ_t
abort
$\widehat{b} \leftarrow \mathcal{A}(\cdot)$
return $\widehat{b} = b$

Fig. 29: Game where adversary $\mathcal{A}$ tries to distinguish between algorithms Sim_4 and Sim_5.

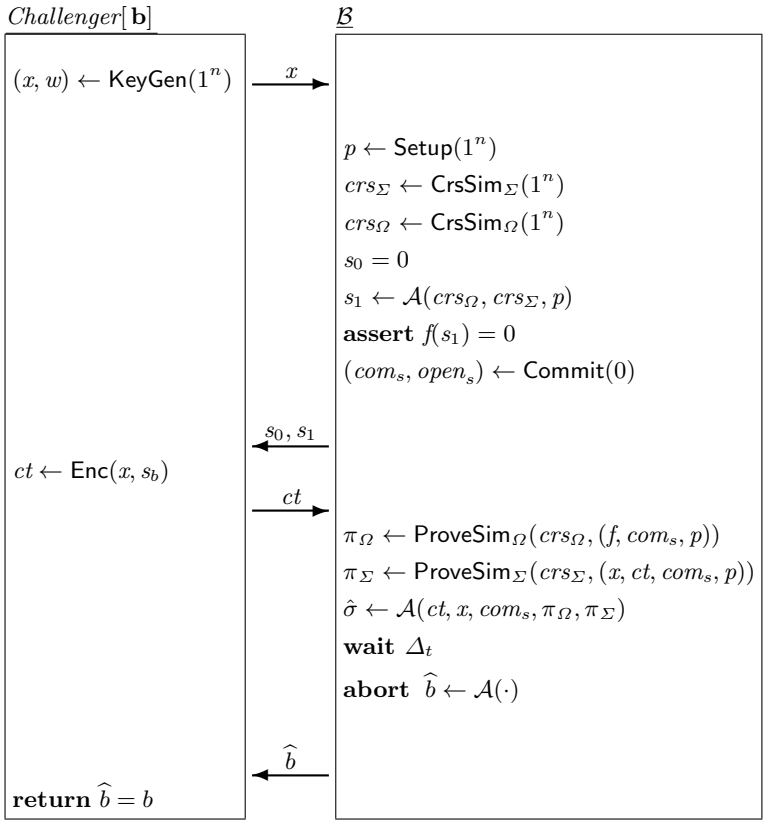

Fig. 30: Game showing how adversary $\mathcal{B}$ would use $\mathcal{A}$ in order to break the IND-CPA security property of the encryption scheme.

$\mathsf{Ext}(f, View_B)$

1 : $\sigma_{B,p}, \hat{\sigma}_{B,p}, x := View_B$
2 : $w' \leftarrow \mathsf{Extract}(\sigma_{B,p}, \hat{\sigma}_{B,p}, x)$
3 : **return** $(x, w') \in \mathsf{R_{AS}}$

Fig. 31: Extractor algorithm.

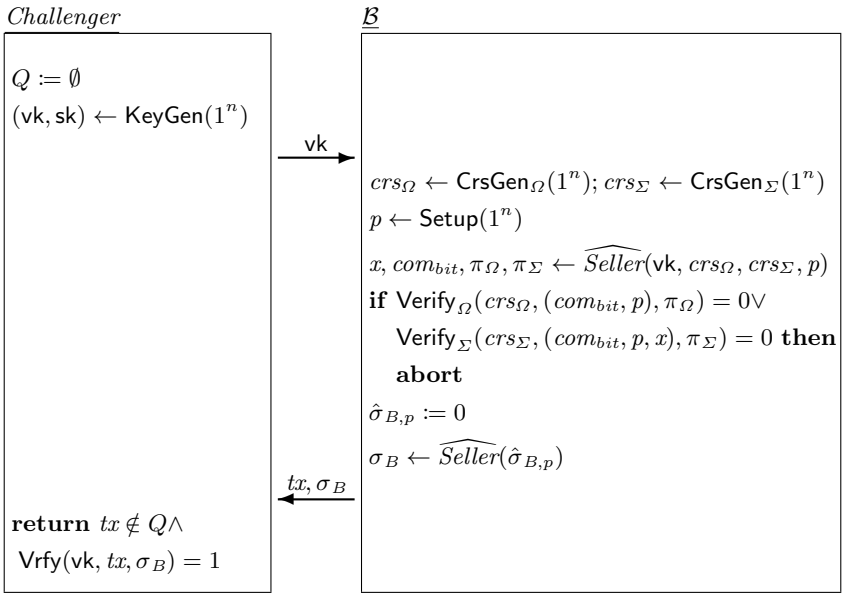

Fig. 32: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the signature unforgeability of the digital signature scheme.

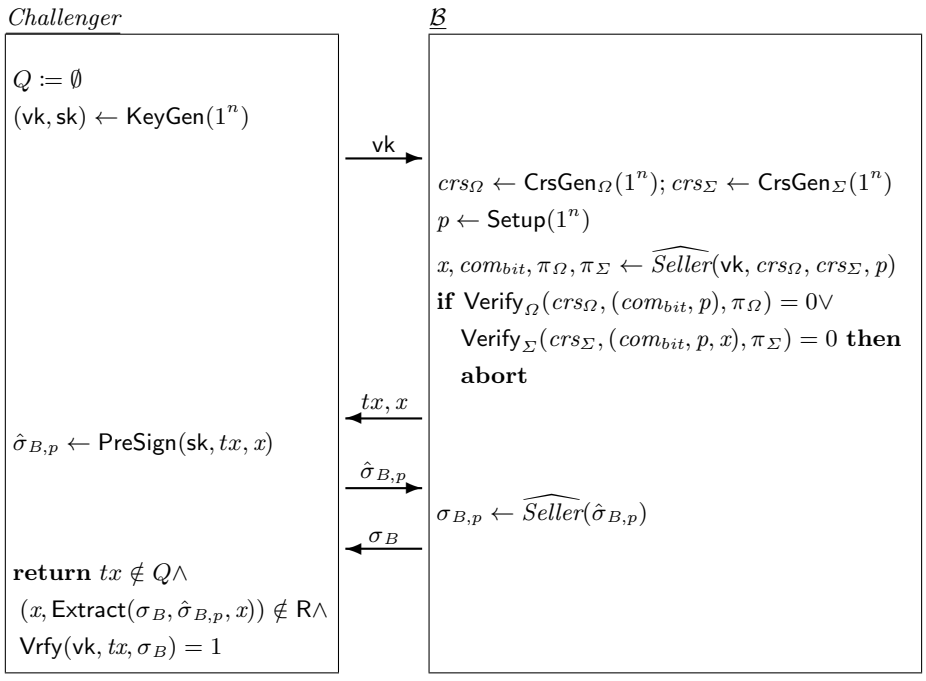

Fig. 33: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the witness extractability of the adaptor signature scheme.

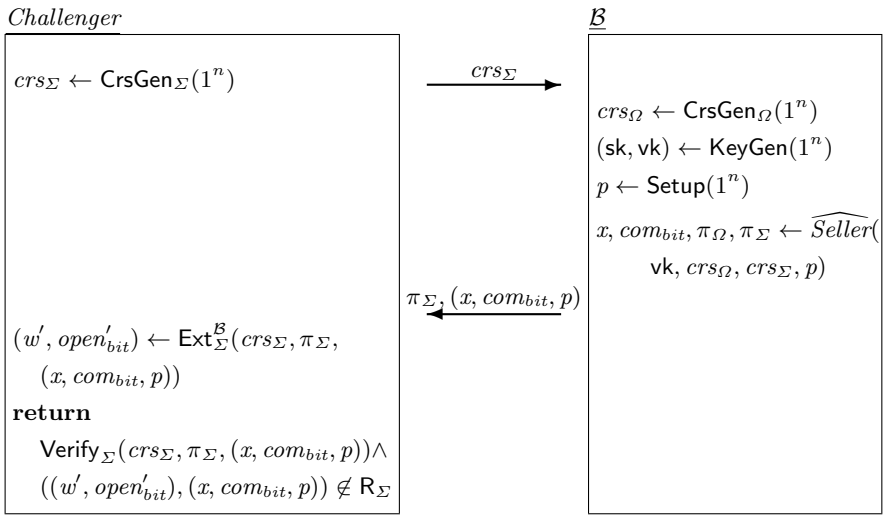

Fig. 34: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the knowledge-soundness of proof π_Σ.

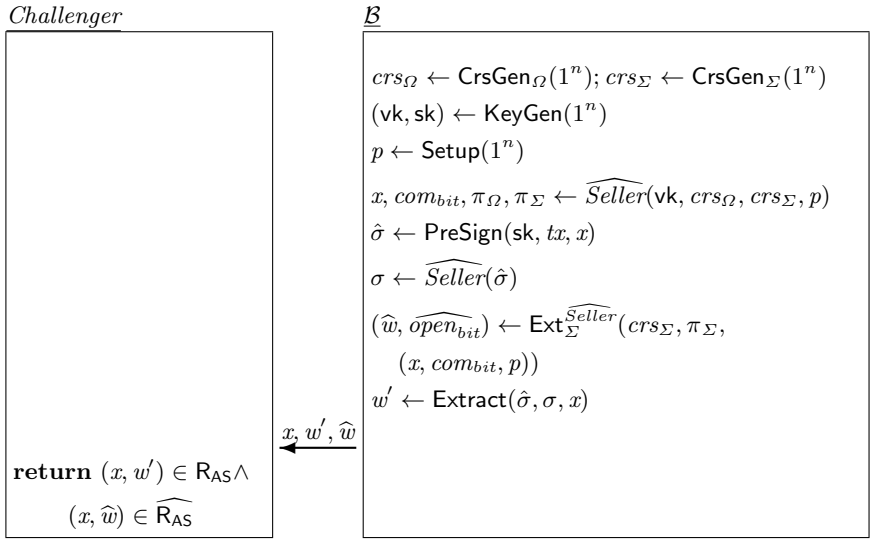

Fig. 35: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ and $\mathsf{Ext}_\Sigma^{\widehat{Seller}}$ in order to break the claw-freeness property of $\mathsf{R_{AS}}$ and $\widehat{\mathsf{R_{AS}}}$.

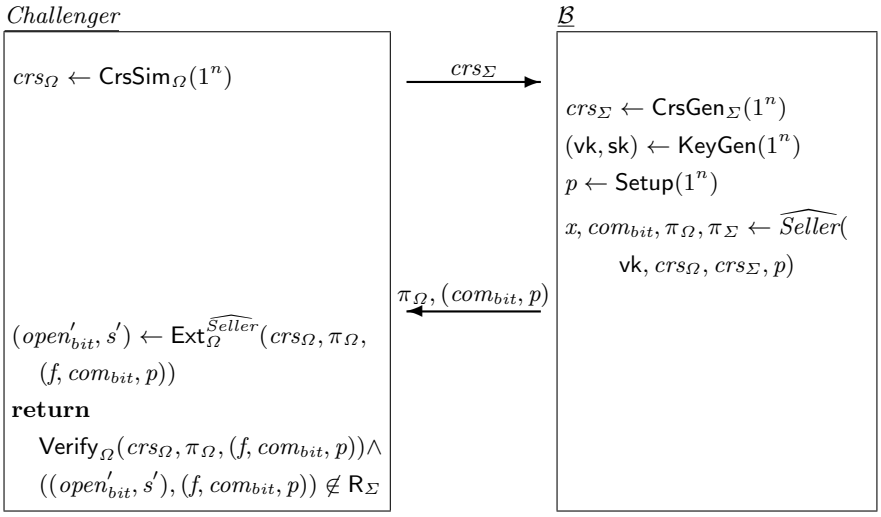

Fig. 36: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the knowledge-soundness property of the proof π_Ω.

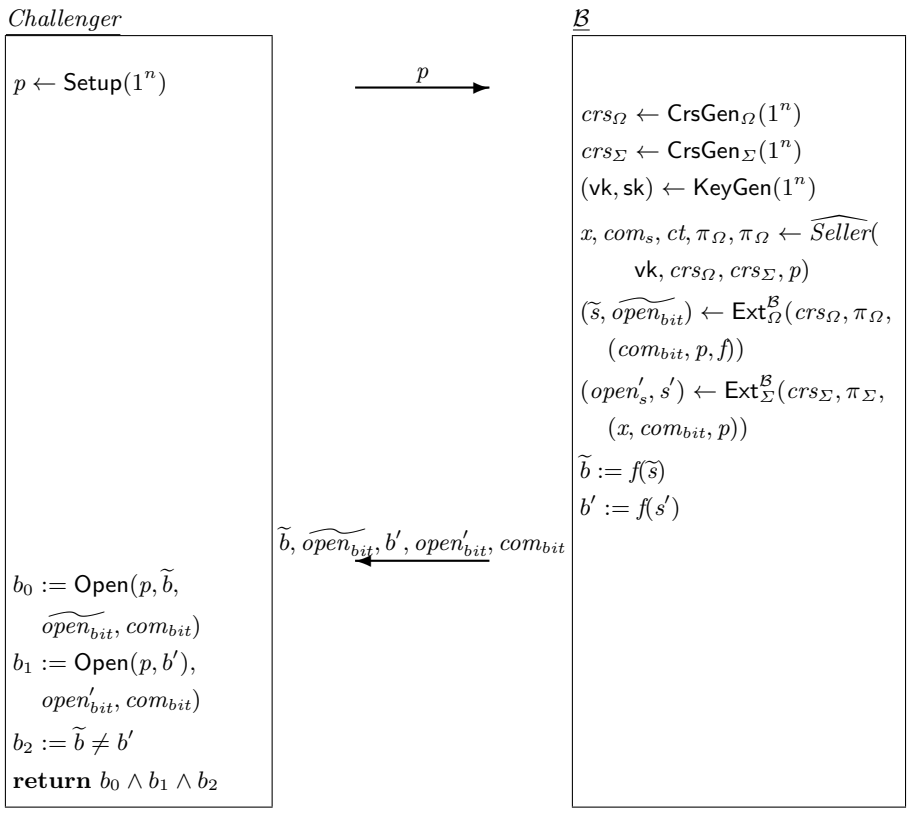

Fig. 37: Game where adversary $\mathcal{B}$ would use adversary $\widehat{Seller}$ in order to break the binding property of the commitment scheme.

$Seller(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetup}(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProve}(f, s, x, w, com_{bit}, p, open_{bit}, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_{bit}, p, x)$
4 : **wait** Δ_t
5 : $\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$

Fig. 38: Pseudocode of the seller.

$Sim_1(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetup}(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProveSim}_1(f, s, x, w, com_{bit}, p, open_{bit}, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_{bit}, p, x)$
4 : **wait** Δ_t
5 : $\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
6 :

$\mathsf{ZKCSPProveSim}_1(f, s, x, w, com_{bit}, p, open_{bit}, crs_\Omega, crs_\Sigma)$

1 : $\pi_\Omega \leftarrow \mathsf{ProveSim}_\Omega(crs_\Omega, (f, com_{bit}, p))$
2 : $\pi_\Sigma \leftarrow \mathsf{Prove}(crs_\Sigma, (com_{bit}, p, x), (open_{bit}, w))$
3 : **return** π_Ω, π_Σ

Fig. 39: Pseudocode of the first simulator.

Challenger[**b**]

$p \leftarrow \mathsf{Setup}(1^n)$
$crs_\Sigma \leftarrow \mathsf{CrsGen}_\Sigma(1^n)$
if $b = 0$ **then**
  $crs_\Omega \leftarrow \mathsf{CrsGen}_\Omega(1^n)$
else
  $crs_\Omega \leftarrow \mathsf{CrsSim}_\Omega(1^n)$
$s \leftarrow \mathcal{A}(crs_\Sigma, crs_\Omega, p)$
$x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetup}(p, s)$
if $b = 0$ **then**
  $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProve}(f, s, x, w, com_{bit}, p, open_{bit}, crs_\Omega, crs_\Sigma)$
else
  $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProveSim}_1(f, s, x, w, com_{bit}, p, open_{bit}, crs_\Omega, crs_\Sigma)$
$\hat{\sigma} \leftarrow \mathcal{A}(x, com_{bit}, p, \pi_\Omega, \pi_\Sigma)$
wait Δ_t
$\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
$\widehat{b} \leftarrow \mathcal{A}(\cdot)$
return $\widehat{b} = b$

Fig. 40: Game where adversary $\mathcal{A}$ tries to distinguish between algorithms *Seller* and Sim_1.

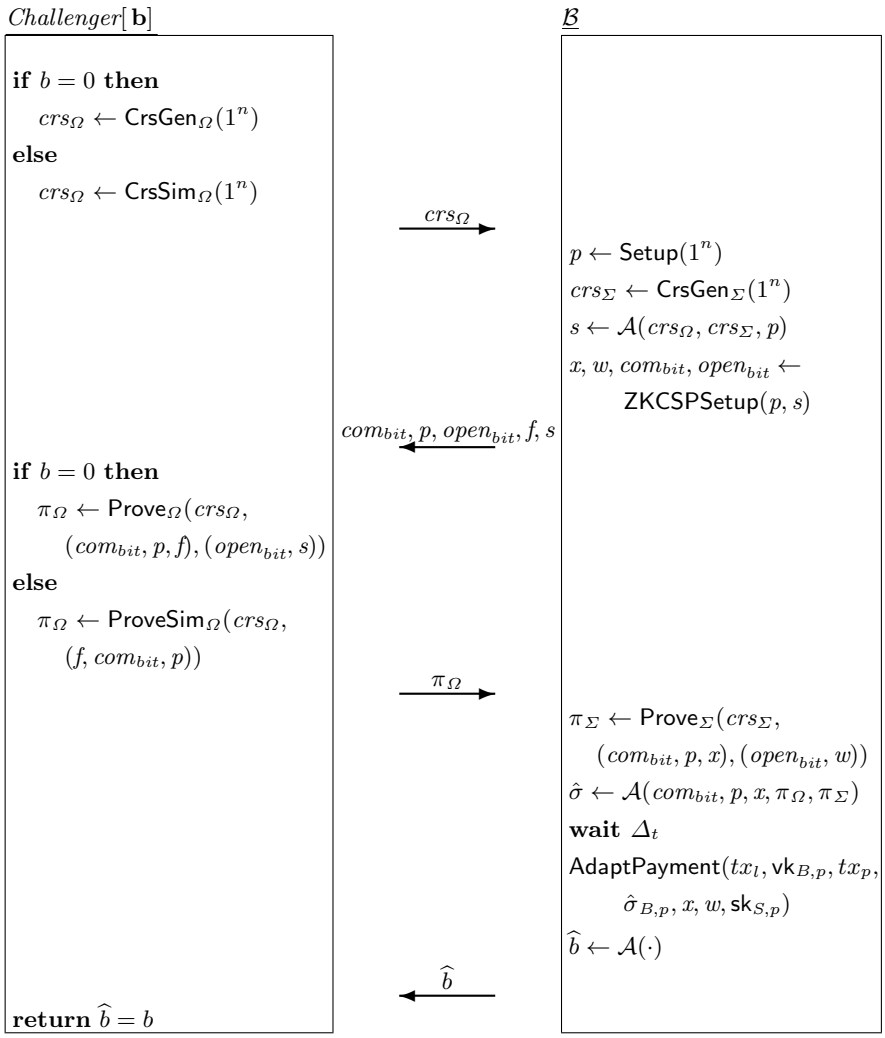

Fig. 41: Game showing how adversary $\mathcal{B}$ would use adversary $\mathcal{A}$ in order to break the zero-knowledge of the proof π_Ω.

$Sim_2(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetup}(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProveSim}_2(f, s, x, w, com_{bit}, p, open_{bit}, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_{bit}, p, x)$
4 : **wait** Δ_t
5 : $\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
6 :

$\mathsf{ZKCSPProveSim}_2(f, x, com_{bit}, p, crs_\Omega, crs_\Sigma)$

1 : $\pi_\Omega \leftarrow \mathsf{ProveSim}_\Omega(crs_\Omega, (f, com_{bit}, p))$
2 : $\pi_\Sigma \leftarrow \mathsf{ProveSim}_\Sigma(crs_\Sigma, (x, com_{bit}, p))$
3 : **return** π_Ω, π_Σ

Fig. 42: Pseudocode of the second simulator.

$Challenger[\mathbf{b}]$

$p \leftarrow \mathsf{Setup}(1^n)$
$crs_\Omega \leftarrow \mathsf{CrsSim}_\Omega(1^n)$
if $b = 0$ **then**
  $crs_\Sigma \leftarrow \mathsf{CrsGen}_\Sigma(1^n)$
else
  $crs_\Sigma \leftarrow \mathsf{CrsSim}_\Sigma(1^n)$
$s \leftarrow \mathcal{A}(crs_\Omega, crs_\Sigma, p)$
$x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetup}(p, s)$
if $b = 0$ **then**
  $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProveSim}_1(f, s, x, w, com_{bit}, p, open_{bit}, crs_\Omega, crs_\Sigma)$
else
  $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProveSim}_2(f, s, x, w, com_{bit}, p, open_{bit} crs_\Omega, crs_\Sigma)$
$\hat{\sigma} \leftarrow \mathcal{A}(com_{bit}, p, x, \pi_\Omega, \pi_\Sigma)$
wait Δ_t
$\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
$\widehat{b} \leftarrow \mathcal{A}(\cdot)$
return $\widehat{b} = b$

Fig. 43: Game where adversary $\mathcal{A}$ tries to distinguish between Sim_1 and Sim_2.

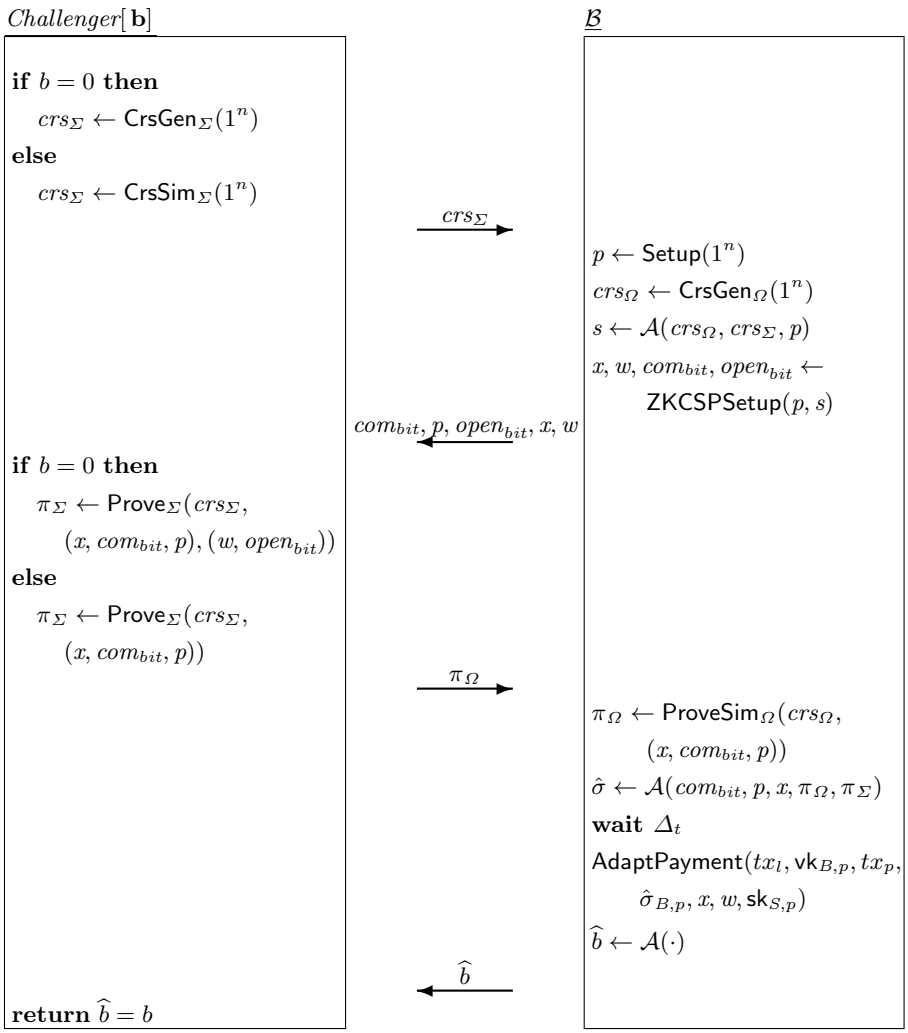

Fig. 44: Game showing how adversary $\mathcal{B}$ would use $\mathcal{A}$ in order to break the zero-knowledge of proof π_Σ.

$Sim_3(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetupSim}_3(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProveSim}_2(f, x, com_{bit}, p, open_{bit}, crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_{bit}, p, x)$
4 : **wait** Δ_t
5 : **abort**

Fig. 45: Pseudocode of the third simulator.

$Challenger[\mathbf{b}]$

$p \leftarrow \mathsf{Setup}(1^n)$
$crs_\Sigma \leftarrow \mathsf{CrsSim}_\Sigma(1^n)$
$crs_\Omega \leftarrow \mathsf{CrsSim}_\Omega(1^n)$
$s \leftarrow \mathcal{A}(crs_\Omega, crs_\Sigma, p)$
$x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetupSim}_3(p, s)$
$\hat{\sigma} \leftarrow \mathcal{A}(com_{bit}, p, x, \pi_\Omega, \pi_\Sigma)$
wait Δ_t
if $b = 0$ **then**
  $\mathsf{AdaptPayment}(tx_l, \mathsf{vk}_{B,p}, tx_p, \hat{\sigma}_{B,p}, x, w, \mathsf{sk}_{S,p})$
else
  abort
$\widehat{b} \leftarrow \mathcal{A}(\cdot)$
return $\widehat{b} = b$

Fig. 46: Game where adversary $\mathcal{A}$ tries to distinguish between Sim_3 and Sim_2.

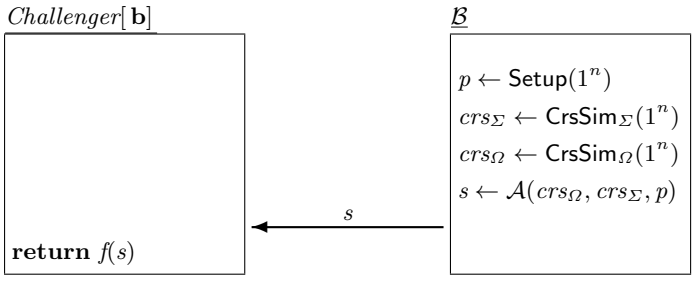

Fig. 47: Game showing how adversary $\mathcal{B}$ would use $\mathcal{A}$ in order to break the one-wayness of the predicate f.

$Sim_4(s, f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetupSim}_4(p, s)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProveSim}_2(f, x, com_{bit}, p, open_{bit} crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_{bit}, p, x)$
4 : **wait** Δ_t
5 : **abort**

$\mathsf{ZKCSPSetupSim}_4(p, s)$

1 : $(com_{bit}, open_{bit}) \leftarrow \mathsf{Commit}(p, f(s))$
2 : $(x, w) \leftarrow \widehat{\mathsf{GenR}}(1^n)$
3 : **return** x, w, com_{bit}

Fig. 48: Pseudocode of the fourth simulator.

$Challenger[\mathbf{b}]$

$p \leftarrow \mathsf{Setup}(1^n)$
$crs_\Sigma \leftarrow \mathsf{CrsSim}_\Sigma(1^n)$
$crs_\Omega \leftarrow \mathsf{CrsSim}_\Omega(1^n)$
$s \leftarrow \mathcal{A}(crs_\Omega, crs_\Sigma, p)$
if $b = 0$ **then**
  $x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetupSim}_3(p, s)$
else
  $x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetupSim}_4(p, s)$
$\hat{\sigma} \leftarrow \mathcal{A}(com_{bit}, p, x, \pi_\Omega, \pi_\Sigma)$
wait Δ_t
abort
$\widehat{b} \leftarrow \mathcal{A}(\cdot)$
return $\widehat{b} = b$

Fig. 49: Game where adversary $\mathcal{A}$ tries to distinguish between Sim_3 and Sim_4.

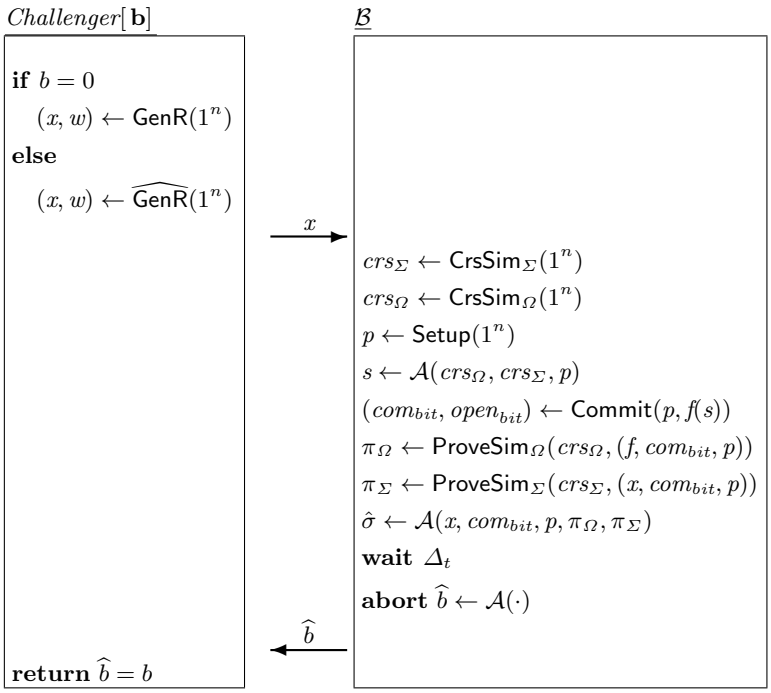

Fig. 50: Game showing how adversary $\mathcal{B}$ would use $\mathcal{A}$ in order to break the indistinguishability of relations R_{AS} and $\widehat{\mathsf{R}_{\mathsf{AS}}}$.

$Sim_5(f, crs_\Omega, crs_\Sigma, p)$

1 : $x, w, com_{bit}, open_{bit} \leftarrow \mathsf{ZKCSPSetupSim}_5(p,)$
2 : $\pi_\Omega, \pi_\Sigma \leftarrow \mathsf{ZKCSPProveSim}_2(f, x, com_{bit}, p, open_{bit} crs_\Omega, crs_\Sigma)$
3 : $\hat{\sigma}_{B,p} \leftarrow Buyer(\pi_\Omega, \pi_\Sigma, com_{bit}, p, x)$
4 : **wait** Δ_t
5 : **abort**

$\mathsf{ZKCSPSetupSim}_5(p)$

1 : $(com_{bit}, open_{bit}) \leftarrow \mathsf{Commit}(p, 0)$
2 : $(x, w) \leftarrow \widehat{\mathsf{GenR}}(1^n)$
3 : **return** x, w, com_{bit}

Fig. 51: Pseudocode of the fifth simulator.

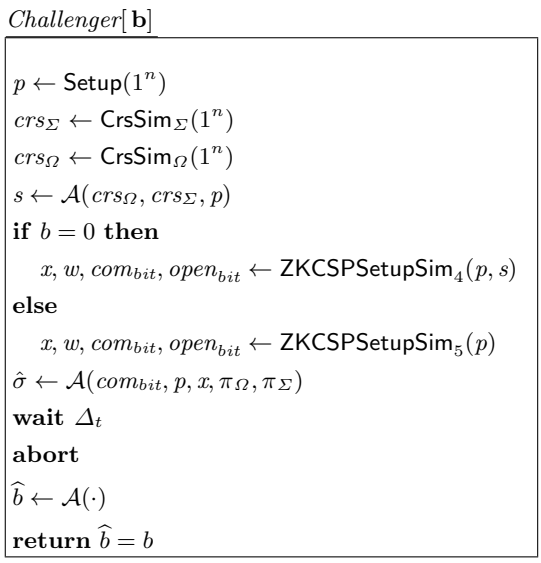

Fig. 52: Game where adversary $\mathcal{A}$ tries to distinguish between Sim_4 and Sim_5.

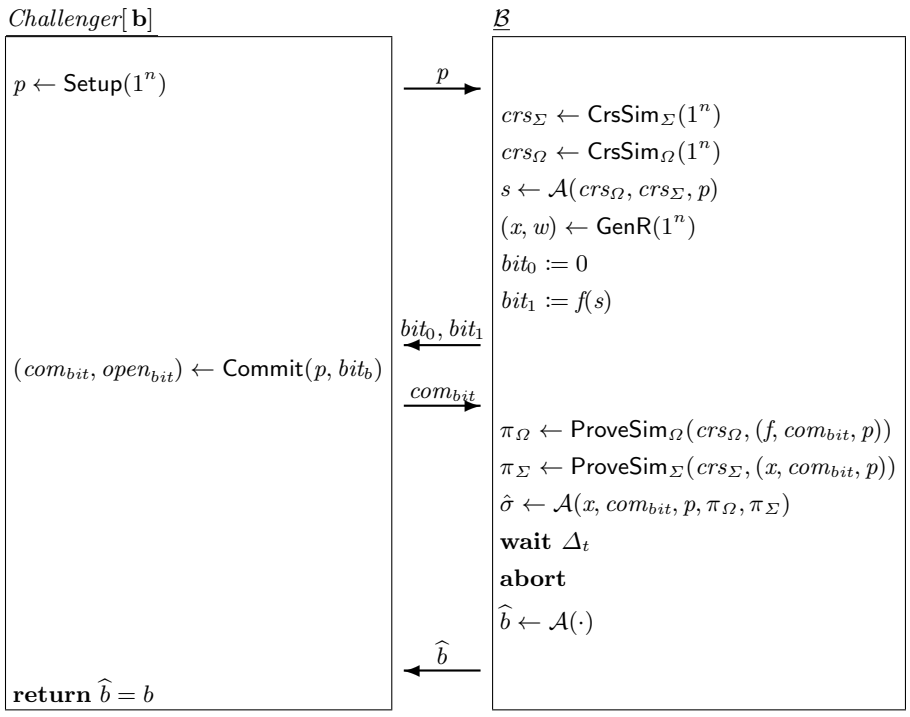

Fig. 53: Game showing how adversary $\mathcal{B}$ would use $\mathcal{A}$ in order to break the hiding property of the commitment scheme.