

Non-Conversation-Based Zero Knowledge*

JOËL ALWEN
Università di Salerno
84084 Fisciano (SA)
ITALY
jfa237@nyu.edu

GIUSEPPE PERSIANO
Università di Salerno
84084 Fisciano (SA)
ITALY
giuper@dia.unisa.it

IVAN VISCONTI
Università di Salerno
84084 Fisciano (SA)
ITALY
visconti@dia.unisa.it

Submission to TCC 2007

Abstract

In a zero-knowledge proof system Π a honest verifier V accepts or rejects a proof basing its decision on the common input, its randomness and the exchanged messages. Π is conversation based if an observer D guesses V ' decision only knowing the common input and the exchanged messages and without knowing V 's randomness.

In this paper we prove a separation between conversation-based and non-conversation based zero knowledge. In particular we show that a variant of 3-round conversation-based black-box concurrent zero knowledge with sequential soundness in the Bare Public-Key model already known to be impossible for non-trivial languages, is instead possible using a non-conversation-based protocol.

The result is obtained considering an impatient verifier, that is a natural extension of the non-aborting verifier previously used in [Rosen, CRYPTO 2000].

*The work presented in this paper has been supported in part by the European Commission through the IST Program under contract IST-2002-507932 ECRYPT and through the FP6 program under contract FP6-1596 AEOLUS.

1 Introduction

A zero-knowledge proof system [?] for a language L , is a game between two probabilistic polynomial-time algorithms, a prover P and a verifier V where P tries to convince V about the validity of a statement “ $x \in L$ ”. The final output of V is referred to as V ’s decision and it consists of a bit b . $b = 1$ specifies that V accepted the proof given by P , while $b = 0$ specifies that V rejected the proof given by P . V ’s decision is based on his randomness r_v , the common input x and the transcript τ_r of the exchanged messages. V ’ decision therefore can be denoted as the output of a predicate $\rho(x, r_v, \tau_r) = b$.

A natural question is whether a party D that simply observes the communication can successfully guess V ’s decision. Obviously when a honest prover P plays with a honest verifier V , V outputs 1 with the probability p (usually $p = 1$) given by the completeness of the proof system. However, in case P could be not honest, the completeness of the proof system does not help D for guessing V ’s decision.

Looking at ρ ’s input, r_v is the only element that is not known to D , thus D can not guess V ’s decision when this is influenced by r_v . Concretely, in almost all known zero-knowledge proof systems, r_v is not used by ρ , therefore, V ’s decision can be computed by any party that knows the common input x and reads the transcript of the exchanged messages. We say that these proof systems are conversation based. For instance, consider the constant-round zero-knowledge protocol for Hamiltonicity. V ’s decision is only based on the correct opening of the commitments sent by P according to the challenge of V and all this information is in the exchanged messages.

It is very easy to obtain a non-conversation-based zero-knowledge proof system from a conversation-based one. Simply let V send in the first round the public key of an encryption scheme, then all messages sent by P are encrypted with this public key and decrypted by V that knows the private key. Obviously the resulting protocol is still a zero-knowledge proof system but now D can not guess V ’s decision. In this case we say that the zero-knowledge proof system is non-conversation-based.

Non-conversation-based zero-knowledge proof systems exist (simply by using the previous transformation) and can be trivially (since it follows by definition) used to hide V ’s decision to an observer. An important question is therefore the following one.

Question 1: is there in cryptography a non-trivial problem that can be solved with a non-conversation-based protocols but that can not be solved by a conversation-based protocol ?

1.1 Related Work

Non-conversation-based protocols have been previously used in [?, ?] for achieving non-interactive zero-knowledge in the secret key model and in the registered public-key model [?], that is a relaxed assumption with respect to the secret key model. In the registered public-key model there exists a conversation-based non-interactive computational zero-knowledge argument system [?], but the non-conversation-based construction of [?] is more efficient and gives statistical zero knowledge. These results say that non-conversation-based protocols seem to be actually useful in cryptography and therefore deserve further investigations.

Concurrent zero knowledge in the BPK model. The setting in which we are going to discuss the difference between conversation-based and non-conversation-based zero knowledge is the Bare Public Key (BPK in short) model that has been proposed in [?] for achieving round-efficient concurrent [?] and resettable [?] zero knowledge where the traditional adversarial verifier is powered by interleaving and resetting capabilities. Since the notion of soundness is subtle in this model [?], different papers [?, ?, ?, ?] studied the round complexity for achieving a given notion of black-box zero knowledge and of soundness in the BPK model. This study is shown in Table 1 that specifies the exact round complexity for sequential, concurrent and resettable zero knowledge and for one-time, sequential and concurrent soundness. In particular in [?], the authors showed the impossibility of 3-round black-box concurrent (and thus resettable) zero knowledge with sequential soundness in the BPK model. The result of [?]

is first given for conversation-based protocols and then extended to non-conversation-based ones. Unfortunately, the authors found a weakness in this extension [?], thus while table 1 still holds for conversation-based protocols, the value in the entry for sequential soundness and black-box concurrent/resettable zero knowledge is still open in case non-conversation-based protocols are considered.

	3-Round OTS	3-Round SS	4-Round CS
sZK	[MR Crypto 01]	[DPV Crypto 04]	Folklore
cZK	[MR Crypto 01]	Impossible [APV Crypto 05] for conv-based	[DPV Crypto 04]
rZK	[MR Crypto 01]	Impossible [APV Crypto 05] for conv-based	[DPV Crypto 04]

Figure 1: The Round Complexity of Black-Box Zero Knowledge in the BPK model.

Concurrent zero knowledge with non-aborting verifiers. The impossibility of constant-round black-box concurrent zero knowledge in the plain model was proved in [?] extending the previous lower bound of [?] that showed the impossibility of 7-round black-box concurrent zero knowledge in the plain model. An interesting gap between these two impossibility results is discussed in [?]. Indeed, in [?] the impossibility is proved designing an adversarial verifier V^* that with a given probability p aborts its interactions with P , independently of the correctness of the proofs played so far. Since this is not required in the impossibility result of [?], it is still an interesting open problem the round complexity of black-box concurrent zero knowledge with non-aborting verifiers, i.e., with verifiers that do not abort as long as the played proofs are accepting. As discussed in [?], with this restriction on the maliciousness of V^* , constant-round black-box concurrent zero knowledge in the plain model for non-trivial languages is still possible.

1.2 Our Contribution

In this paper we answer affirmatively to Question 1. First of all, we extend the notion of non-aborting verifier to that of impatient verifier. This extension is pretty natural as while a non-aborting verifier does not send an abort as long as the received proofs are correct, we require that an impatient verifier is a non-aborting verifier that moreover aborts its execution with P as soon as an incorrect proof is received. We refer to as *impatient zero knowledge* the relaxed notion of concurrent zero-knowledge that only concerns impatient verifiers V^* .

We then show that in the BPK model, the impossibility result proved in [?] for 3-round conversation-based impatient¹ zero knowledge with sequential soundness, does not hold in the non-conversation-based case. Indeed, we show that in the BPK model there exists a 3-round non-conversation-based impatient zero-knowledge argument system with sequential soundness. These results shows a separation that explicitly says that non-conversation-based protocols can actually be more secure than conversation-based ones and can be used to solve non-trivial tasks where conversation-based protocols fail.

We achieve these results using a new technique that can be of independent interest. This technique allows the simulator to extract one bit of one of the secret keys of V^* depending on the fact that V^* accepted the proof or not. Obviously this extraction procedure still requires the rewind capabilities, otherwise P^* could do the same thus violating the soundness of the protocol.

2 Basic Tools

We will consider \mathcal{NP} -languages L and denote by R_L the corresponding polynomial-time relation such that $x \in L$ if and only if there exists w such that $R_L(x, w) = 1$. Also a *negligible* function $\nu(n)$ is a function such that for

¹The impossibility result proved in [?] holds for conversation-based protocols and any concurrent verifier, but actually does not use more than an impatient verifier.

any constant $c < 0$ and for all sufficiently large n , $\nu(n) \leq n^c$.

We assume that the reader is familiar with the notion of an *interactive Turing machine* (see [?]) and of *computationally indistinguishable* and *statistically close* ensemble of random variables.

Signature schemes. We assume the existence of a secure signature scheme $(\text{SGen}, \text{Sig}, \text{Ver})$. Here SGen denotes the key generator algorithm that receives the security parameter k (in unary) and returns a pair (pk, sk) of public and private keys; Sig is the signature algorithm that takes as input a message m and a private key sk and returns a signature s of m ; and Ver is the signature verification algorithm that takes as input a message m , a signature s and a public key pk and verifies that s is a valid signature. The scheme is secure in the sense that no probabilistic polynomial-time algorithm that has access to a signature oracle but not to the private key can forge the signature of a message m for which it has not queried the oracle.

In particular we will use signature schemes that are secure with respect to subexponential-time adversaries. For a formal definition, see Appendix A.

We assume that signatures of k -bit messages produced by using keys with security parameter k have length k . This is not generally true as for each signature scheme we have a constant a such that signatures of k -bit messages have length k^a but this has the advantage of not overburdening the notation. Our results still hold if this assumption is removed.

Standard secure signature schemes exist under the assumption of the existence of one-way functions [?]. In our case we need the existence of functions that are one-way with respect to subexponential-time adversaries.

Encryption schemes. An encryption scheme is a triple of efficient algorithms $\text{ES} = (\text{EGen}, \text{Enc}, \text{Decr})$. The key generator algorithm EGen on input a security parameter outputs a pair (pk, sk) of public and private keys. The public key pk is used to encrypt a string m by computing $\text{Enc}(\text{pk}, m)$.

Semantic security [?] is defined by considering the following experiment for encryption scheme $\text{ES} = (\text{EGen}, \text{Enc}, \text{Decr})$ involving a two-part adversary $A = (A_0, A_1)$. The key generator EGen is run on input 1^k and keys (pk, sk) are given in output. Two $\text{POLY}(k)$ -bit strings ω_0 and ω_1 and an auxiliary information aux are returned by A_0 on input pk . Then b is taken at random from $\{0, 1\}$ and an encryption ξ of ω_b is computed. We say that adversary A is *successful for ES* if the probability that A_1 outputs b on input $\text{pk}, \omega_0, \omega_1, \xi$ and aux is non-negligibly (in k) greater than $1/2$. We say that ES is η -secure if no adversary running in time $o(2^{k^\eta})$ is successful. The classical notion of semantic security is instead obtained by requiring that no polynomial-time adversary is successful. It is known how to construct semantic secure encryption schemes on top of enhanced trapdoor permutations, η -security can be obtained by strengthening the assumption of the enhanced trapdoor permutations that have to be secure against subexponential-time adversaries.

Commitment schemes. A commitment scheme can be seen as the digital equivalent of a sealed envelope.

If a party A , called the committer, wants to commit to some message m she just puts it into the sealed envelope, so that whenever A wants to reveal the message, she opens the envelope. This primitive must meet some basic requirements. First of all the digital envelope should *hide* the message: no party other than A should be able to learn m from the commitment (this is often referred in the literature as the hiding property). Second, the digital envelope should be *binding* in the sense that once the commitment has been produced A cannot change his mind about m .

Here we only focus on non-interactive commitment schemes where there is one message from the sender to the receiver to commit to a message and another message from the sender to the receiver to open the previous commitment. More specifically, a commitment scheme is a pair of probabilistic polynomial-time algorithms: the commitment algorithm Com and the verification algorithm Dec . The committer obtains a pair (c, d) of commitment and decommitment keys by running Com on input the message m . The commitment c is published by the committer. It is useful to think of c as the sealed envelop containing the message m . To reveal the commitment,

the committer publishes the triple (c, d, m) . The verification algorithm Dec then is run on input the triple to verify that the commitment c was properly opened as m .

The hiding property requires that the commitment c does not reveal any information on the committed message m to an adversary that has access to c .

The binding property requires that an adversary can not produce a commitment c for which there exists two messages m_0 and m_1 and two decommitment keys (d_0, d_1) such that c can be opened as m_0 using d_0 and as m_1 using d_1 (that is, $\text{Dec}(c, d_0, m_0) = 1$ and $\text{Dec}(c, d_1, m_1) = 1$).

In a *statistically/perfectly hiding commitment scheme* the hiding property holds regardless of the computational power of the adversary. In a *statistically/perfectly binding commitment scheme* the binding property holds regardless of the computational power of the adversary. For a formal definition see Appendix A.

Non-interactive perfectly binding commitment schemes exist under the assumption that one-way permutations exist [?]. When an initial message of the receiver is allowed, it is possible to construct non-interactive perfectly binding commitment schemes on top of any one-way function [?] and non-interactive perfectly hiding commitment schemes on top of collections of claw-free functions [?] or collision resistant hash functions [?].

We will also consider an extractable commitment scheme, i.e., a statistically/perfectly binding commitment scheme such that there exists an extractor algorithm that on input a commitment outputs the committed message with overwhelming probability in time $O(2^{k^\gamma})$ where k is the security parameter of the commitment scheme and γ is a positive constant. The hiding property of this commitment scheme is required to hold against polynomial-time adversaries. Such commitment scheme exists, for instance under the assumption that there exist one-way permutations that are one-way with respect to polynomial-time adversaries but such that they can be inverted in subexponential time [?]. We refer to these schemes as γ -extractable commitment schemes.

For the definitions of interactive argument/proof systems, witness indistinguishability and ZAPs please see Appendix A.

3 The BPK Model

Several round-efficient protocols have been shown to be secure in the Bare Public-Key model (the BPK model, for short) introduced by Canetti et al. [?]. This model makes very minimal set-up and network assumptions. More specifically, the BPK model requires that verifiers register their public keys in a public file during a set-up stage. This stage does not require any interaction among the players. Moreover, no trusted third party is assumed, nor physical assumptions are made, and the underlying communication network is assumed to be (adversarially) asynchronous. The BPK model only assumes that no prover-verifier interaction starts during the set-up stage. It is therefore a relaxed set-up assumption that is very attractive when impossibility results in the plain model hold.

For a formal definition of the BPK model and of concurrent zero-knowledge with sequential soundness in this model, please see Appendix B.

4 Subprotocols

In this section we present both known and new tools along with their implementations that we will first analyze individually and then later combine in the non-conversation-based protocol.

4.1 Coin Flipping

The first and simplest of the subprotocols we look at is a 3-round coin-flipping protocol based on a perfectly hiding commitment schemes (Com, Dec) . To simplify the notation we assume that the decommitment key also includes the committed message itself.

Subprotocols: commitment scheme (Com, Dec) .

Players: A and B .

Common Input: security parameter 1^k .

Preprocessing:

B0 B sends to A the initial receiver message for the perfectly hiding commitment scheme (Com, Dec) .

Protocol:

A1.1) A randomly selects a k -bit random string $m_1 \leftarrow \{0, 1\}^k$.

A1.2) A computes $(c, d) \leftarrow \text{Com}(m_1)$ and sends c to B .

B2) B selects a k -bit random string $m_2 \leftarrow \{0, 1\}^k$ and sends it to A .

A3) A sends d to B .

Result:

If B verifies that d is a valid decommitment of c to m_1 then both parties accept $m_3 \leftarrow m_1 \oplus m_2$ as the output of the protocol. Otherwise the protocol terminates with a failure.

Figure 2: Coin Flipping Protocol based on a Perfectly Hiding Commitment Scheme.

Lemma 4.1 *The output of the coin flipping protocol is unpredictable to A before round B2 is played and to B before round A3 is played.*

PROOF. Assume A predicts the output m_3 before round 2. Since A is a PPT algorithm, it can only select a polynomial number $l = \text{POLY}(k)$ of strings over the 2^k possible strings of $\{0, 1\}^k$ trying to predict m_3 . We show how to design an adversary B^* that plays honest B algorithm but rewinds A , and breaks the binding of the commitment scheme. The contradicting assumption of this proof implies that A after a commitment c succeeds with non-negligible probability p . Since B^* 's message is uniformly distributed, the expected number of times that the second round has to be sent again for obtaining other $l + 1$ successes of A on c is $(l + 1)/p$ which is polynomial in k . Since the number of successes is $l + 1$ but the number of selected strings is l , there must be two executions with the same c and same output m_3 . The probability that in these two executions B^* messages are the same is negligible since the messages of B^* are uniformly distributed over $\{0, 1\}^k$. Therefore, it holds that c has been opened in two different ways.

Let us now consider the other case. The unpredictability of m_3 for B follows from the fact the commitment scheme is perfectly hiding, therefore in case B selects $l = \text{POLY}(k)$ strings, it has only a negligible probability $l/2^k$ of success. \square

4.2 Interactive Argument System for \mathcal{NP} in the BPK Model

Here we show a simple interactive argument system between a prover P and a verifier V in which an \mathcal{NP} statement " $x \in L$ " is proved. This specific protocol will be later used as subprotocol of our main result. The interactive argument makes use of two cryptographic primitives. A secure signature scheme $(\text{SGen}, \text{Sig}, \text{Ver})$ with a security parameter of 1^k and an η -extractable statistically binding commitment scheme $(\text{Com}', \text{Dec}')$ where $\eta > 0$. (This will then be used in the proof of soundness by means of complexity leveraging, i.e., the telescopic use of cryptographic assumptions.) Moreover, the coin-flipping protocol of Fig. 2 and the ZAP of Fig. 6 are used as subprotocols.

Subprotocols: Coin-flipping protocol, η -extractable commitment scheme $(\text{Com}', \text{Dec}')$, γ -secure signature scheme $(\text{SGen}, \text{Sig}, \text{Ver})$, ZAP $(\text{ZG}, \text{ZP}, \text{ZV})$.

Players: Prover P , Verifier V .

Common Input: security parameter 1^k , an \mathcal{NP} statement $x \in L$.

Private Input P : a witness w to $x \in L$.

Preprocessing:

V0.1) V generates $(\text{spk}, \text{ssk}) \leftarrow \text{SGen}(1^k)$.

V0.2) V computes $z \leftarrow \text{ZG}(1^k)$.

V0.3) V plays round **B0** of the coin-flipping protocol computing message $B0$.

V0.4) V puts $(\text{spk}, z, B0)$ in its entry of the public file.

Protocol:

P1.1) P plays rounds **A1.1**, **A1.2** of the coin-flipping protocol.

V2.1) V plays round **B2** of the coin-flipping protocol.

repeat

- **Pi)** P randomly sets $m_i = 0^k$ and sends it to V .

- **Vi)** V computes $s = \text{Sig}(m_i, \text{ssk})$ and sends s to P .

until $m_i \neq 0^k$

P3.1) P plays round **A3** of the coin-flipping protocol, let m_3 be the output.

P3.2) P computes $(c, d) \leftarrow \text{Com}'(0^k)$ a commitment to 0^k .

P3.3) P computes $z_1 \leftarrow \text{ZP}(z, "x \in L \text{ OR } (c, \cdot) = \text{Com}'(\text{Sig}(m_3, \text{ssk}))", w)$.

P3.4) P sends c and z_1 to V .

Result: If V successfully verifies that $\text{ZV}(z, z_1, "x \in L \text{ OR } (c, \cdot) = \text{Com}'(\text{Sig}(m_3, \text{ssk}))") = 1$ then it accepts $x \in L$ to be true. Otherwise it rejects.

Figure 3: 3-Round Sequentially Sound Argument System for \mathcal{NP} in the BPK model.

Lemma 4.2 *The protocol depicted in Fig. 3 is a 3-round sequentially sound interactive argument system for \mathcal{NP} in the BPK model.*

PROOF.

Completeness. The completeness of this interactive argument system is clear because P is given a witness w to the theorem and receives z which it can then use in round **P3.3** to generate a valid ZAP.

Soundness. The proof of this property proceeds by contradiction. Specifically we begin by assuming that there exists a malicious prover P^* which can convince the honest verifier V of a false statement with a non-negligible probability. In particular this implies that V has accepted z_1 to be a valid ZAP of the required statement. This leaves two possibilities. Either P^* has managed to generate an accepting ZAP for a false statement or the statement is in fact true.

However the first possibility implies that P^* can be used to break the soundness property of the ZAP protocol depicted in Fig. 6. Indeed, this can be achieved by first forwarding to P^* the message z received from player A in the first round of Fig. 6 and finally forwarding to player A of Fig. 6 message z_1 computed by P^* . Each other message can be played with P^* by simply running the honest verifier algorithm.

Therefore we can assume the theorem “ $x \in L$ OR $(c, \cdot) = \text{Com}'(\text{Sig}(m_3, \text{ssk}))$ ” to be true. Since by the original assumption “ $x \in L$ ” is a false theorem it holds that “ $(c, \cdot) = \text{Com}'(\text{Sig}(m_3, \text{ssk}))$ ” is true. Once again this leads to a contradiction though, since now P^* can be used (as a black-box) to construct a machine B which breaks the security of the signature scheme. Such a machine could proceed as follows:

1. On input spk' it skips step **0.1** and performs step **0.2, 0.3, 0.4**, sending spk' instead of the value that honest V generates in step **0.1**.
2. It honestly plays round **2.1**.
3. It asks a signature of m_i to the oracle for the signature scheme and sends it back to P^* in round V_i .
4. Upon receiving z from P^* , running in time at most $O(2^{k^\eta})$, B extracts the message committed in c getting $\text{Sig}(m'_3, \text{ssk})$, i.e., a signature of the output of the coin flipping protocol.

By setting $\eta = \gamma/2$ we have that B computes a signature in time $O(2^{k^\eta}) = o(2^{k^\gamma})$ and thus breaks the security of the signature scheme. Notice that m'_3 is unpredictable to P^* when he sends the messages P_i , otherwise we can easily break the protocol of Fig. 2, which security we have already proved in Lemma 4.1. Since B has the same probability of producing $\text{Sig}(m'_3, \text{ssk})$ as P^* does of convincing V of a false statement, we can conclude that by security of the signature scheme the protocol depicted in Fig. 3 is one-time sound.

Sequential soundness follows by observing that a success of P^* in a given session over a sequential but still polynomial number of sessions, allows us to break the same cryptographic assumptions of the one-time case with non-negligible probability. Indeed, the only complication is the need of guessing the index of the session where P^* succeeds, this however can only decrease by a polynomial factor the probability of success of our reductions. \square

4.3 Encoding ssk in \mathcal{NP} statements

In Fig. 4 we present a method with which a player V can encode a secret information (in this case ssk) in the form of k pairs of proofs of \mathcal{NP} statements. This will be a crucial tool for the simulation of the non-conversation-based protocol as the simulator will use in a given session the pairs received in another session. This will allow it to extract (guessing sequentially all bits b_i) the secret key of the signature scheme. When the simulators will know such a key, it will be able to simulate all sessions corresponding to this a key in a straight-line fashion.

Since all pairs e_i^0, e_i^1 are encrypted with epk , by the semantic security of the encryption scheme, P^* is not able to distinguish between any two such values. Since in our final protocol P^* will be only a sequential adversary (in contrast with the simulator that instead has rewind capabilities that thus include concurrent sessions), it will not be able of using the encryptions received in a sessions for successfully completing other sessions.

We finally remark that V computes both the first and second round of the ZAP. In general, this means that the soundness of the ZAP computed by V can not be claimed and so the indistinguishability for V of the ZAPs that itself computes. This is precisely why for the second round V uses the randomness given by P and also uses the honest prover algorithm of the ZAP.

Obviously, a malicious V^* could deviate from this behavior, but we will prevent this in the larger protocol by requiring that the verifier of the larger protocol proves that all computations performed in steps **V2.1, V2.2, V2.3** have been done using the honest V algorithm of Fig. 4 (which includes the use of the honest prover algorithm of the ZAP). The resulting computed ZAPs will be therefore sound and indistinguishable to V^* itself with respect to

honestly computed ZAPs, provided that V^* forgets that actually it computed the ZAP (rewinds will remove this information from V^* 's view). The same trick is applied for the commitments and the encryptions computed by V .

Subprotocols: ZAP (ZG, ZP, ZV), γ -secure signature scheme ($S\text{Gen}, \text{Sig}, \text{Ver}$), semantically-secure encryption scheme ($E\text{Gen}, \text{Enc}, \text{Decr}$), η -extractable commitment scheme (Com', Dec').

Players: V, P .

Common Input: security parameter 1^k .

Note: we assume wlog that algorithms $ZP, \text{Enc}, \text{Com}$ needs k random bits each for their execution.

Preprocessing:

V0.1) V computes $(\text{epk}, \text{esk}) \leftarrow E\text{Gen}(1^k)$, and $(\text{spk}, \text{ssk}) \leftarrow S\text{Gen}(1^k)$, let wlog $\text{ssk} = b_1 b_2 \dots b'_k$ for $b_i \in \{0, 1\}$.

V0.3) V computes $z = ZG(1^k)$.

V0.4) V sends epk, spk and z to P .

Encoding spk in an \mathcal{NP} statement:

P1) P randomly selects $m_0, r_0, r_1, r_2 \leftarrow \{0, 1\}^k$ and sends all strings to V .

V2.1) V computes $s \leftarrow \text{Sig}(m_0, \text{ssk})$, the signature of the message m_0 .

V2.2) V computes $(c, d) \leftarrow \text{Com}'_{r_0}(s)$, a commitment to s using r_0 for the random coins.

V2.3) for $i \in \{1, \dots, k'\}$ V computes the pairs $(e_i^{b_i}, e_i^{1-b_i})$ where:

$$e_i^{b_i} = \text{Enc}^{r_2}(\text{epk}, (c, ZP^{r_1}(z, "x \in L \text{ OR } (c, \cdot) = \text{Com}'_{r_0}(\text{Sig}(m_0, \text{ssk}))", (s, d)))) \text{ and}$$

$$e_i^{1-b_i} = \text{Enc}^{r_2}(\text{epk}, 0^{l_i});$$

where ZP^{r_1} is the second round of a ZAP using r_1 as random coins, where l_i is the bit-length of the statement being encrypted in $e_i^{b_i}$ and Enc^{r_2} it the encryption algorithm that uses r_2 as random coins.

V2.4) V sends all pairs $(e_i^0, e_i^1)_{1 \leq i \leq k'}$ to P .

Figure 4: Method of Encoding ssk as an \mathcal{NP} Statement.

5 The Non-Conversation-Based Protocol

In this section we present the 3-round sequentially sound impatient zero-knowledge argument system in the BPK model. We stress that this result can not be obtained with a conversation-based protocol, thus this result shows a separation that proves that in some contexts, non-conversation-based protocols are more secure than conversation-based ones.

First we define two properties that a verifier V in a zero knowledge protocol can enjoy as well as the notion of a concurrent block of sessions.

Definition 5.1 *Let V be a PPT algorithm playing the verifier role in a zero-knowledge protocol. Then we call V a non-aborting verifier if V never aborts given an accepting transcript.*

Note that the above property is exactly that of the non-aborting verifiers as used in the impossibility result of [?] and explicitly mentioned in [?].

Definition 5.2 Let V be a \mathcal{PPT} Turing machine playing the verifier role in a zero-knowledge protocol. Then we call V an impatient verifier if V is a non-aborting verifier that immediately aborts given a rejecting transcript.

Hence forth we assume that all adversarial verifiers are impatient.

Definition 5.3 Let V^* be an impatient verifier playing against a prover P . We call the set of completed sessions BK a concurrent block of sessions if there exists no other incomplete session between the first round of the first initiated session in BK and last round of last completed session in BK .

From the above definition, we have that the interaction between P and V^* can be described as the sequential execution of concurrent blocks of sessions.

5.1 Description

In this section we will describe the 3-round protocol which is defined to run in the BPK model. Subsequently we will show both zero knowledge in the concurrent setting for impatient verifiers as well as sequential soundness.

We describe the main protocol by stating in which roles (and at which points) V and P play the previous subprotocols more or less in parallel.

- **Encoding s_{sk} as an \mathcal{NP} Statement:** Both players play the roles with their names but instead of sending the output of the preprocessing stage to P , V places it in F .
- **Interactive Argument System for \mathcal{NP} of Fig. 3.** Rather than sending its last message to V in step $P3.4$ of Fig. 3, P encrypts it with the public key e_{pk} that V adds to its entry of the public file (as we specify below).
- **ZAP:** Player P plays role A and V plays role B . The purpose of this subprotocol is to ensure that V has correctly encoded the secret key of the signature scheme in the encryptions given in the second round. Moreover, this has been done using the honest algorithms and the randomness provided by the prover. This means that there exists now a valid back door for the simulator. Notice that without the ZAP, P could not verify the correctness of the message played by the verifier since part of it is encrypted.

For a detailed description of the composed protocol see Fig. 5.

We now extend the proofs of completeness and soundness of the original interactive argument system for \mathcal{NP} showing that they also hold for the composed protocol. Finally in Section 5.1 we will analyze its zero knowledge property.

Completeness. As before, completeness, is trivially verifiable. Steps $P1.1 - P1.4$ can be performed by an efficient uniform Turing machine (i.e. it requires neither private input nor special computational capabilities). Further, given a witness w for “ $x \in L$ ” as well as the public file F containing s_{pk} , z and e_{pk} the prover P can perform all verifications as required in step $P3.1$. Using z the honest prover can compute the required second message for the ZAP. As a honest prover would have computed and sent a valid commitment to m_1 in step $P1.2$ it can also produce the valid decommitment d . The string m_3 and the commitment c_2 are trivial to generate as they require no extra input. Finally, by using e_{pk} P can correctly encrypt c_2 and the computed ZAP.

Since V knows the decryption key e_{sk} it can decrypt the ZAP and, using z , verify its correctness thus accepting the theorem $x \in L$.

Sequential soundness. The proof of sequential soundness relies heavily on the sequential soundness of the underlying interactive argument system. It begins (by contradiction) assuming the existence of a malicious prover P^* which can convince the honest verifier V of a false theorem. We can construct a malicious prover P^{**} for

Subprotocols: ZAP (ZG, ZP, ZV), γ -secure signature scheme ($S\text{Gen}, \text{Sig}, \text{Ver}$), semantically-secure encryption scheme ($E\text{Gen}, \text{Enc}, \text{Decr}$), η -extractable commitment scheme (Com', Dec'), perfectly hiding commitment scheme (Com, Dec).

Note: we assume wlog that algorithms $ZP, \text{Enc}, \text{Com}'$ needs k random bits each for their execution.

Common Input: security parameter 1^k , a statement “ $x \in L$ ” and a polynomial sized public file F pairing V ’s identity with a public key $\text{pk} = (\text{epk}, \text{spk}, z, B0)$ where $(\text{epk}, \text{esk}) \leftarrow E\text{Gen}(1^k)$, $(\text{spk}, \text{ssk}) \leftarrow S\text{Gen}(1^k)$, $z \leftarrow ZG(1^k)$, and $B0$ is the output of round $V0.4$ in Fig. 3 .

Prover’s Auxiliary Input: w , a witness to “ $x \in L$ ”.

Verifier’s Auxiliary Input: $\text{sk} = (\text{esk}, \text{ssk})$, the corresponding secret keys to pk , where $\text{ssk} = b_1 b_2 \dots b'_k$.

Protocol:

P1.1) P randomly selects $m_0, m_1, r_0, r_1 \leftarrow \{0, 1\}^k$.

P1.2) P computes $(c, d) \leftarrow \text{Com}(m_1)$.

P1.3) P computes $z' \leftarrow ZG(1^k)$.

P1.4) P sends $(m_0, c, r_0, r_1, r_2, z)$ to V .

V2.1) V randomly selects $m_2 \leftarrow \{0, 1\}^k$.

V2.2) V computes $s = \text{Sig}(m_0, \text{ssk})$.

V2.3) V computes k pairs of encryptions $\{e_i^0, e_i^1\}_{i \leq k'}$ where $\forall i \exists b_i \in \{0, 1\}$ such that $\text{ssk} = (b_1 \dots b'_k)$ and the following holds true:

- $(c_1, d_1) = \text{Com}'_{r_0}(s)$.
- $e_i^{b_i} = \text{Enc}^{r_2}(\text{epk}, (c_1, ZP^{r_1}(z, “x \in L \vee (c_1, \cdot) = \text{Com}'(\text{Sig}(m_0, \text{ssk}))”, (d_1, s))))$ where:
 - $\text{Com}'_{r_0}(s)$ is a commitment to s using randomness r_0 ;
 - ZP^{r_1} is ZP computed with randomness r_1 ;
- $e_i^{1-b_i} = \text{Enc}^{r_2}(\text{epk}, 0^l)$ where l is the length in bits of the message encrypted in $e_i^{b_i}$.

V2.4) V computes $z_1 = ZP(z', “\forall i \leq k' \exists b_i \in \{0, 1\}$ such that $e_i^{b_i}$ is “well formed” $\wedge e_i^{1-b_i} = \text{Enc}^{r_2}(\text{epk}, 0^l) \wedge \text{ssk} = b_1 b_2 \dots b'_k”, (s, d_1))$.

V2.5) V sends $(s, m_2, \{e_i^0, e_i^1\}_{i \leq k'}, z_1)$ to P .

P3.1) P verifies that s is a valid signature of m_0 corresponding to the public key spk and that z_1 is accepted by ZV with z as first round. If this is not the case the P aborts the protocol.

P3.2) P computes $m_3 = m_1 \oplus m_2$ and $(c_2, d_2) = \text{Com}'(0^k)$.

P3.3) P computes $e = \text{Enc}(\text{epk}, (c_2, ZP(z, “x \in L \vee (c_2, \cdot) = \text{Com}'(\text{Sig}(\text{ssk}, m_3))”, w)))$.

P3.4) P sends (d, e) to V .

V4) V verifies that d is a valid decommitment, computes m_3 , verifies that e is a valid encryption of a message with the correct form, verifies that the contained ZAP is correct and accepting, and if all verifications succeed accepts the proof for “ $x \in L$ ”.

Figure 5: 3-Round Black-Box Impatient Knowledge with Sequential Soundness for Impatient Verifiers in the BPK Model.

the argument system depicted in Fig. 3 which sequential soundness has been proved in Lemma 4.2. P^{**} plays as a man-in-the-middle, indeed it is a verifier of P^* for protocol 5 but it gives a proof to a honest verifier V of protocol 3. First of all, P^{**} put all preprocessing messages received from V in its entry of the public file. Moreover, it honestly generates the key pair for the encryption scheme and appends the public key to its entry of the public file. Then P^{**} receives the first round from P^* that contains the commitment of m_1 that is thus forwarded to V . Then P^{**} receives the answer from V (i.e., message $V2.1$ of Fig. 3), and uses it in round $V2.1$ of Fig. 5. P^{**} then sends m_0 (i.e., the message received from P^* in round $P1.4$ of Fig. 5) in round Pi of Fig. 3, obtains a signature of m_0 and can play the honest verifier algorithm for the remaining messages of Fig. 5. The only additional difference is that P^{**} decrypts the last message received from in Fig. 5 and sends the decrypted message to the verifier of Fig. 5.

From the above discussion we have that P^{**} has a non-negligible probability of convincing the verifier of Fig. 3 on the validity of the false statement which contradicts Lemma 4.2.

Impatient Zero Knowledge. We now give an extended informal description of the simulator algorithm together with a short look at its correctness, efficiency and indistinguishability. In Appendix C we have a detailed description of the simulators algorithm together with the data structures it uses and a formal analysis of the simulator’s properties.

The simulator S begins a session π by playing the honest provers algorithm until it knows the outcome of the coin-flipping subprotocol for π (which we will call m_3^π) i.e. the value m_3 which it computes after step $V2.5$ when it receives m_2 . Essentially S has three strategies for successfully completing π . The simplest one (we denote it as strategy 3) assumes that S has already extracted ssk and stored it in its local variable ssk_i for the identity i being used by V^* for π . In this case S can complete the simulation in a straight-line fashion by generating the value $(c_2, d_2) = \text{Com}'(\text{Sig}(m_3^\pi, \text{ssk}_i))$ which it uses as a witness for the final ZAP in round $P3.3$ of π instead of the witness to the theorem $x \in L$ as the honest prover would.

Of course, initially S does not know the secret signature key for any identity. So it must use a different strategy by means of a lookahead which starts with a rewind of V^* to the step ($P1.1$) at the beginning of π . Then S sets a new value for $m_0 = m_3^\pi$ while playing all other steps of this new session honestly up until step $V2.5$ at which point S gets a valid signature $s = \text{Sig}(m_3^\pi, \text{ssk})$ as well as the k' encryption pairs $\{e_j^0, e_j^1\}$. Now the lookahead is complete and S rewinds V^* back to step $V2.5$ in session π which it could previously not complete.

At this point one of two situations can occur. Either π is the last unfinished (open) session in the concurrent block of sessions between S and V^* or there is at least one other incomplete session.

1. In the first case (i.e. when S can not detect aborts) the simulator will use the following strategy 1. In step $P3.3$, S sets $(c_2, d_2) = \text{Com}'(s)$ which allows S to produce a valid ZAP of the required statement by using d_2 as a witness rather than some witness to $x \in L$. Thus π is guaranteed to have been complete with success.
2. In the second case S is able to detect an abort by V^* . This follows directly from the assumption that V^* is impatient. The non-aborting property implies that any session which V^* begins will also be completed (unless S causes V^* to have a rejecting view). Further the impatient property implies that since V^* just played round $V2.5$ for any other open session V^* still has an accepting view of that session. Therefore by sending message $P3.4$ for π the simulator can detect whether V^* accepts this message as valid by requesting a next message from V^* . Since V^* is impatient it will immediately abort if S sent a “bad” message. Therefore, as long as there is some other open session to which V^* must respond, a failure by V^* to deliver another message implies that it has aborted where as the successful delivery of the next message implies that π was completed with an accepting view.

At first glance one might ask why strategy 1 alone does not suffice. The problem (as is generally the case for concurrent zero knowledge protocols) is that rewinds can be very expensive (requiring exponentially many other sessions to be solved first) which was shown in [?]. Thus S requires some other mechanism in order to be able to put a bound on the number of rewinds it will need to complete the simulation. Actually there is a further reason

why strategy 1 alone can not work. If the simulator were limited to this method then it would not make use of the encryption pairs $\{e_j^0, e_j^1\}$ nor would there be any reason to encrypt the ZAP and the commitment in the final round and thus all these elements could be removed from the zero-knowledge protocol without altering its properties. We would therefore obtain the protocol depicted in Fig. 3 which is conversation based. However this contradicts the impossibility result for conversation-based zero knowledge in [?]. The protocol of Fig. 3 can be actually proved to be only sequential zero knowledge and this result was already obtained in [?].

Therefore S has a second strategy which will allow it to straight-line simulate any subsequent session for the identity i of V^* . However this strategy can only be used for sessions where an abort (on the part of V^*) can be detected by S . In such a case S uses a novel method (strategy 2) to interpret whether V^* aborted or not in such a way as to deduce information about the value of ssk one bit at a time. This will (after k' rewinds) allow S to deduce ssk for identity i , store it in ssk_i and straight-line simulate with strategy 3 any further session for that identity, thus eliminating the need for any future expensive rewinds.

A detailed description of strategy 2 follows. Recall that S has just received the round-2 message from V^* in session π and has just complete the lookahead. Now for every $j \in [1, k']$ S does the following: it selects a random bit b_j . Then it plays the honest provers algorithm for steps $P3.1$ until $P3.4$ and it sends $e_j^{b_j}$ (this is obtained by the lookahead) to V^* rather than e in step $P3.4$. If V^* aborts (i.e. fails to deliver and further messages) then S sets the j -th bit of ssk_i to $1 - b_j$ otherwise (if V^* does not abort) S sets the j -th bit to be b_j . Finally S rewinds V^* back to $V2.5$ and repeats for the process for the next value of j . Thus after k' such loops S will deduced the value of ssk and have stored it in ssk_i . (This is because for each pair of encryptions only one is of the correct form where as the other is simply an encryption of a 0 string. However, which is which for a pair j depends upon what the value of the j -th bit of ssk is.) Finally although S can not decrypt the pairs it can verify that they are of the correct form because of the ZAP delivered by V^* along with the pairs during the lookahead in step $V2.5$.) At the end of strategy 2, S completes π by using ssk_i (thus strategy 3) to compute $(c_2, d_2) = \text{Com}'(\text{Sig}(m_3^{\pi}, \text{ssk}_i))$ and d_2 will be its witness for the ZAP in round $P3.4$.

So far we have (informally) discussed the simulator's ability to complete any given session. The efficiency of S lies in the fact that strategy 1 is used only once per concurrent block, therefore it is not damaged by concurrency. Strategy 2 instead is used just once per entry of the public file which size is polynomially bounded. Strategy 3 does not require any rewind. What remains to be shown is that its output is truly indistinguishable from that of the honest prover. In all three cases (strategy 1, 2 and 3) the output of S for π is generated by following the honest provers algorithm up until the last step $P3.3$. In $P3.3$ the value of c_2 is different to that of P in that a different value is being committed to and a different witness is being used to prove the ZAP. Assuming a verifier can distinguish between playing against S and P via the committed value in round-3, such a verifier would break the hiding property of the commitment scheme Com' . If on the other hand there existed a verifier which could distinguish via the ZAP of step $P3.3$ then this verifier would break the witness indistinguishability property of the ZAP protocol. Thus it can be concluded that the output of S and P are indistinguishable when playing against any non-aborting and impatient V^* .

6 Conclusions

In this paper we have shown that non-conversation-based protocols can solve non-trivial cryptographic tasks that instead can not be solved by conversation-based protocols. We wonder whether there are other interesting cases in which such separations hold.

References

A Definitions

Interactive Argument/Proof Systems. An *interactive proof (resp., argument) system* [?] for a language L is a pair of interactive Turing machines $\langle P, V \rangle$, satisfying the requirements of *completeness* and *soundness*. Informally, completeness requires that for any $x \in L$, at the end of the interaction between P and V , V rejects with negligible probability. Soundness requires that for any $x \notin L$, for any computationally unbounded (resp., probabilistic polynomial-time) P^* , at the end of the interaction between P^* and V , V accepts with negligible probability. We denote by $\langle P, V \rangle(x)$ the output of the verifier V when interacting on common input x with prover P . Also, sometimes we will use the notation $\langle P(w), V \rangle(x)$ to stress that prover P receives as additional input witness w for $x \in L$.

Formally, we have the following definition.

Definition A.1 A pair of interactive Turing machines $\langle P, V \rangle$ is an interactive proof system for the language L , if V is probabilistic polynomial-time and there exists a negligible function $\nu(\cdot)$ such that

1. **COMPLETENESS:** For every $x \in L$ and valid witness w ,

$$\text{Prob}[\langle P(w), V \rangle(x) = 1] \geq 1 - \nu(|x|).$$

2. **SOUNDNESS:** For every $x \notin L$ and for every P^*

$$\text{Prob}[\langle P^*, V \rangle(x) = 1] \leq \nu(|x|).$$

If the soundness condition holds only with respect to probabilistic polynomial time P^* then $\langle P, V \rangle$ is called an argument.

Since all protocols we give are actually argument (rather than proof) systems, we will now focus on argument systems only. Also from now on we assume that all interactive Turing machines are probabilistic polynomial-time.

Witness Indistinguishability. The notion of witness-indistinguishability [?] applies to interactive proof/argument systems for \mathcal{NP} languages and requires that no information is revealed to malicious verifiers about which witness is being used during the execution of the argument.

Definition A.2 Let L be an \mathcal{NP} -language. An interactive argument $\langle P, V \rangle$ for L is witness indistinguishable if for every pair (w_1, w_2) such that $R_L(x, w_1) = R_L(x, w_2) = 1$, and all polynomial-time verifiers V^* running on input any auxiliary information z (that can include w_1 and w_2) the probability distributions $\text{view}_{V^*}^P(x, w_1, \cdot)$ and $\text{view}_{V^*}^P(x, w_2, \cdot)$ are indistinguishable.

Commitment schemes. Here we give the formal definition of non-interactive commitment scheme.

Definition A.3 (Com, Dec) is a non-interactive commitment scheme if:

- **efficiency:** Com and Ver are probabilistic polynomial-time algorithms;
- **completeness:** for all m it holds that

$$\text{Prob}((\text{com}, \text{dec}) \leftarrow \text{Com}(m) : \text{Dec}(\text{com}, \text{dec}, m) = 1) = 1;$$

- **binding:** for any \mathcal{PPT} algorithm sen^* there is a negligible function ν such that

$$\begin{aligned} & \text{Prob}((\text{com}, m_0, m_1, \text{dec}_0, \text{dec}_1) \leftarrow \text{sen}(1^n) \wedge m_0 \neq m_1 \wedge \\ & \text{Dec}(\text{com}, \text{dec}_0, m_0) = \text{Dec}(\text{com}, \text{dec}_1, m_1) = 1) \leq \nu(n); \end{aligned}$$

- **hiding:** for all m_0, m_1 where $|m_0| = |m_1|$ the probability distributions:

$$\{(\text{com}_0, \text{dec}_0) \leftarrow \text{Com}(m_0) : \text{com}_0\} \text{ and } \{(\text{com}_1, \text{dec}_1) \leftarrow \text{Com}(m_1) : \text{com}_1\}$$

are computationally indistinguishable.

In a statistically/perfectly hiding commitment scheme the hiding property holds regardless of the computational power of the adversary. In a statistically/perfectly binding commitment scheme the binding property holds regardless of the computational power of the adversary.

η -Secure signature schemes.

Definition A.4 An η -secure signature scheme is a triple of algorithms $(\text{SGen}, \text{Sig}, \text{Ver})$ such that

1. **correctness:** for all messages $m \in \{0, 1\}^k$,

$$\text{Pr}[(\text{pk}, \text{sk}) \leftarrow \text{SGen}(1^k); \hat{m} \leftarrow \text{Sig}(m, \text{sk}) : \text{Ver}(m, \hat{m}, \text{pk}) = 1] = 1.$$

2. **unforgeability:** for all algorithms A running in time $o(2^{k^\eta})$ it holds that

$$\text{Pr}[(\text{pk}, \text{sk}) \leftarrow \text{SGen}(1^k); (m, \hat{m}) \leftarrow A^{\mathcal{O}(\text{sk})}(\text{pk}) : m \notin \text{Query and } \text{Ver}(m, \hat{m}, \text{pk}) = 1]$$

is negligible in k where $\mathcal{O}(\text{sk})$ is a signature oracle that on input a message returns as output a signature of the message and Query is the set of signature requests submitted by A to \mathcal{O} .

ZAPs. Here we review the notion of ZAP [?] (see Fig. 6), i.e., a 2-round public coin witness indistinguishable proof system. For now the only assumption we will make concerning the theorems being proven in the ZAP is that they concern \mathcal{NP} statements. Their exact nature will be clarified (and our assumption justified) later on during the composition of the subprotocols.

Definition A.5 A triple of polynomial-time algorithms (ZG, ZP, ZV) is a ZAP for the NP-language L with polynomial-time relation R_L iff:

1. **Completeness:** given a witness w for “ $x \in L$ ” and $z \leftarrow ZG(1^k)$ then $ZV(z, ZP(z, x, w), x) = 1$ with probability 1.
2. **Soundness:** for all $x \notin L$, with overwhelming probability over $z \leftarrow ZG(1^k)$, there exists no z' such that $ZV(z, z', x) = 1$.
3. **Witness-Indistinguishability:** let w_1, w_2 such that $(x, w_1) \in R_L$ and $(x, w_2) \in R_L$. Then $\forall z$, the distributions on $ZP(z, x, w_1)$ and on $ZP(z, x, w_2)$ are computationally indistinguishable.

In [?] a ZAP is presented under the assumption that non-interactive zero-knowledge proof systems exist, thus the existence of ZAPs is implied by the existence of enhanced trapdoor permutations.

Players: A and B .

Common Input: security parameter 1^k .

Protocol:

A1) A computes $z \leftarrow ZG(1^k)$ and sends it to B .

B2.1) B selects the \mathcal{NP} statement T along with the witness w .

B2.2) B computes $z_1 \leftarrow ZP(z, T, w)$ and sends (z_1, T) to A .

Result ZAP : A outputs $ZV(z, z_1, T)$.

Figure 6: ZAP Protocol.

B The BPK Model

Here we describe the BPK model, proposed in [?] and that we consider for our results. We give the definitions for concurrent zero-knowledge argument systems with sequential soundness since these are the arguments that we will later use. For further details, see [?, ?, ?].

The BPK model assumes that: 1) there exists a polynomial-size collection of records associating identities with public keys in the public file F ; 2) an (honest) prover is an interactive deterministic polynomial-time algorithm that takes as input a security parameter 1^k , F , an n -bit string x , such that $x \in L$ where L is an \mathcal{NP} -language, an auxiliary input w , a reference to an entry of F and a random tape; 3) an (honest) verifier V is an interactive deterministic polynomial-time algorithm that works in the following two stages: 1) in a first stage on input a security parameter 1^k and a random tape, V generates a key pair (pk, sk) and stores its identity associated with pk in an entry of the file F ; 2) in the second stage, V takes as input sk , a statement $x \in L$ and a random string, V performs an interactive protocol with a prover, and outputs “accept” or “reject”; 4) the first interaction of a prover and a verifier starts after all verifiers have completed their first stage.

Definition B.1 Given an \mathcal{NP} -language L and its corresponding relation R_L , we say that a pair of probabilistic polynomial-time algorithms $\langle P, V \rangle$ is **complete** for L , if for all n -bit strings $x \in L$ and any witness w such that $(x, w) \in R_L$, the probability that V on input x when interacting with P on input x and w , outputs “reject” is negligible in n .

Malicious provers and attacks in the BPK model. Let s be a positive polynomial and P^* be a probabilistic polynomial-time algorithm that takes as first input 1^n .

P^* is an s -sequential malicious prover if it runs in at most $s(n)$ stages in the following way: in stage 1, P^* receives a public key pk and outputs an n -bit string x_1 . In every even stage, P^* starts from the final configuration of the previous stage, sends and receives messages of a single interactive protocol on input pk and can decide to abort the stage in any moment and to start the next one. In every odd stage $i > 1$, P^* starts from the final configuration of the previous stage and outputs an n -bit string x_i .

Given an s -sequential malicious prover P^* and a honest verifier V , a *sequential attack* is performed in the following way: 1) the first stage of V is run on input 1^n and a random string so that a pair (pk, sk) is obtained; 2) the first stage of P^* is run on input 1^n and pk and x_1 is obtained; 3) for $1 \leq i \leq s(n)/2$ the $2i$ -th stage of P^* is run letting it interact with V which receives as input sk, x_i and a random string r_i , while the $(2i + 1)$ -th stage of P^* is run to obtain x_i .

Definition B.2 Given a complete pair $\langle P, V \rangle$ for an \mathcal{NP} -language L in the BPK model, then $\langle P, V \rangle$ is a **sequentially sound interactive argument system for L** if for all positive polynomials s , for all s -sequential malicious provers P^* and for any false statement “ $x \in L$ ” the probability that in an execution of a sequential attack, V outputs “accept” for such a statement is negligible in n .

Definition B.3 Let $\langle P, V \rangle$ be an interactive argument system for a language L . We say that a probabilistic polynomial-time adversarial verifier V^* is a **concurrent adversary in the BPK model** if on input polynomially many values $\bar{x} = x_1, \dots, x_{\text{POLY}(n)}$, it first generates the public file F with $\text{POLY}(n)$ public keys and then concurrently interacts with $\text{POLY}(n)$ number of independent copies of P (each with a valid witness for the statement), with common input \bar{x} and without any restrictions over the scheduling of the messages in the different interactions with P . Moreover we say that the transcript of such a concurrent interaction consists of \bar{x} and the sequence of prover and verifier messages exchanged during the interaction. We refer to $\text{view}_{V^*}^P(\bar{x})$ as the random variable describing the content of the random tape of V^* and the transcript of the concurrent interactions between P and V^* .

Definition B.4 Let $\langle P, V \rangle$ be an interactive argument or proof system for a language L in the BPK model. We say that $\langle P, V \rangle$ is black-box **concurrent zero knowledge** if there exists a probabilistic polynomial-time algorithm S such that for each polynomial-time concurrent adversary V^* , let $S_{V^*}(\bar{x})$ be the output of S on input \bar{x} and black-box access to V^* , then if $x_1, \dots, x_{\text{POLY}(n)} \in L$, the ensembles $\{\text{view}_{V^*}^P(\bar{x})\}$ and $\{S_{V^*}(\bar{x})\}$ are computationally indistinguishable.

C The Simulator

We now continue with the detailed description and analysis of the simulator’s algorithm.

Data types and functions. The simulator’s algorithm employs several constants, variables, functions and complex data types which will be elaborated upon in this section along with their roles in the algorithm.

Constants. Two constants are used in the description of the simulator S . D which is the maximum number of concurrent open sessions V^* will open (i.e. the maximum depth V^* will reach) and I which is the number of different identities in the public file F . Note that although technically speaking D is not a fixed value (since it is only polynomially bounded in k but not actually predefined) the simulator can be implemented to make a reasonable guess at D ’s value at the beginning of the interaction. If this guess turns out to be too low S can easily adjust its internal value of D which will involve no more than resizing the dimensions of a matrix. D and I are mainly important for the final analysis of S .

Simple variables. Two types of simple variables are used by S . The first, T , contains a string representing V^* ’s current view of the transcript thus far. At the end of the execution T will contain a complete transcript of the interaction between the two parties indistinguishable from the transcript of an interaction between the honest prover and V^* . T is initialized as an empty transcript and its value is updated explicitly through out the algorithm. The second type of simple variables are the set $\{ssk_i\}_{i \in I}$ which are used to store the value of each identities secret signature key once they have been discovered via strategy 2. Each of these variables are initialized to $0^{k'}$ and there values are updated explicitly one bit at the end of strategy 2.

Functions. The algorithm has 4 functions, 3 of which are explicitly implemented below while the 4-th is only described as its implementation is clear but long-winded. For each concurrent block of sessions a call is made to the simulator's `solve()` functions. This method consists of a do-until loop which reads one message at a time from V^* calling one of two handler functions to process each new message as it arrives and updating T with all new changes in V^* 's view. A new message from V^* can be of two types. Either the message initiates a new (concurrent) session in which case the handler `init.handler()` is called, or the message is a round-2 response by V^* to some already existing (open) session in which case `rnd2.handler()` is called. The last function explicitly called in the code of the simulator is `NumOpenSessions()` which takes a transcript T as an argument and returns the number of open sessions T . Both handlers as well as `solve()` are implemented below.

Structures and complex data types. S makes use of 1 data structure called `msg` which is used to store a message from V^* in `solve()`. A `msg` contains a record `sid` which holds the unique ID of the session to which the message belongs. Further a `msg` can be of either type `init` (when V^* is initiating a new concurrent session) in which case `msg` contains the record `ident` which holds the identity from the public file F for that session or a `msg` can be of type `round` (when V^* is sending its round-2 message to an existing protocol) in which case `msg` contains the record `rnd2` which holds the message sent by V^* at step $V2.5$ for some open session.

The simplest of the complex data types used by the simulator is hash table ID which maps unique session IDs to a pair (i, d) where i is the identity in F used by V^* for that session and d is the sessions depth. (The depth can be understood to be the number of open sessions plus 1 at the point where S receives the `init` messages. Thus the first session has depth 1, the second depth 2 and so on until a maximum depth of D .)

The core data type of the algorithm is a 2 dimensional matrix M of size I by D . Each entry is a 3-tuple which stores current state (and, when relevant, auxiliary information) for the session with identity i and depth d . A session can be in one of 4 states symbolized by the first value of the tuple:

- **n:** “not seen” which is the initial state of any session and implies that no session has been requested by V^* with this identity and at this depth. In this case the second and third elements of the 3-tuple are never used.
- **b:** “beginning” which implies that a session with this identity and depth has been begun but no round-2 has been received yet and no lookahead process has been started for it yet. In this case the second element of the tuple is not used but the third element T contains the transcript (from V^* 's point of view) up to and including the `init` message by V^* for this session. (Note it does not include the round-1 message by S for this session.) Once a round-2 message has been received a lookahead process is initiated which begins by rewinding V^* to T and playing a new round-1 message.
- **l:** “lookahead” which implies that a lookahead for the session with this identity at this depth is currently running. In this case the second element of the 3-tuple contains the round message of S (where $m_0 = m_3^\pi$) which is stored so that it can be replayed whenever a session with this identity at this session is initiated until the lookahead is completed. This way it can be guaranteed that only two rewinds per lookahead are required. Once to begin and once to end. Any session for which a lookahead has been initiated will only be rewound again once a round-2 message has been received by S . (Later on during the analysis of the runtime of S this fact will be vital to establish that at most a polynomial number of rewinds are required before terminating or discovering at least one new secret signature key.) The third element of the tuple T contains the transcript which lead up to the point where the lookahead was initiated. (I.e. up to the reception of the round-2 message of session π by S .) Once the lookahead has been completed V^* is rewound to T and either strategy 1 or 2 is used to play a round-3 message.
- **d:** “done” which implies that the S has already used strategy 2 to extracted the secret signature key for this identity and stored it in `sski`. In this case S can straight-line any session at with this identity (in particular

at this depth). Once an entry of the matrix has reached this state it will no longer change for the rest of the execution.

Often the elements of the 3-tuples in M will be referenced individually. In this case we use the notation $M[i, d]_{state}$, $M[i, d]_{rnd1}$, and $M[i, d]_{transcript}$ to refer to the first, second and third elements respectively of entry for the session with identity i and depth d .

The last data type used is a list C . It is a list of identity-depth pairs (i, d) which will be used to keep track of the order in which lookaheads are initiated for different sessions. Whenever a new lookahead is started the corresponding values (i, d) are appended to C . A search of C begins at the oldest pair and progress till the most recently added pair is reached. We say that a comes after b in C if the a was appended to C before b .

As a quick reference we give a table with all constants, variables unimplemented functions, records and complex data types in Fig. 7.

D	Maximum number of sessions in concurrent block.
I	Number of identities in the public file F .
T	Stores the transcript of the current view of V^* .
$\{ssk_i\}_{i \leq I}$	Stores the secret signature keys extracted by the simulator for each identity in F .
$\text{NumOpenSessions}(T)$	Input: a transcript T , Output: number of unfinished sessions in T .
msg	The record of containing a message from V^* with an sid entry. msg can be of type $init$ in which case it contains $ident$ or of type $round$ in which case it contains $rnd2$.
ID	A hash table with unique session IDs as keys and the corresponding identity-depth pair (i, d) as values.
M	A 2D matrix of size I by D with entries indexed by an identity and depth pair where each storing a 3-tuple.
$M[i, d]$	A 3-tuple $(state, rnd1, transcript)$ where $state$ can take on the values: n for “Not Seen”, b for “Beginning”, l for “Lookahead”, and d for “Done”. When $state = l$ then $rnd1$ contains a round-1 message and when $state = l$ or b then $transcript$ contains a transcript.
C	Is a list of identity-depth pairs where “ a comes before b ” implies that element a was added before b .

Figure 7: Constants, Variables, Functions, Structures and Data Types.

C.1 The Simulator Algorithm

Now we present the detailed implementation of the three functions `solve()`, `init.handler()` and `rnd2.handler()`.

Comments to `solve()`.

- (1) Store the identity and depth of the new session being initiated.
- (2) Call the handler routine to generate a round-1 message.
- (3) I.e. when m is of type “rnd2”.
- (4) Call the handler routine to generate a round-3 message or to start a lookahead.

Algorithm solve():

```

solve ()
  T ← empty transcript;
  do
    get m = next message from V*;
    T ← T + m;
    if m is of type “init”
(1)   ID[m.sid] = (m.ident, NumOpenSessions(T), T);
(2)   T ← init.handler(ID[m.sid]depth, ID[m.sid]ident, T);
(3)   else
(4)   T ← rnd2.handler(m.rnd2, ID[m.sid]depth, ID[m.sid]ident, T);
  until NumOpenSessions(T) = 0;
  output (T);

```

Figure 8: Simulator solve () Function.

Comments to init.handler ().

- (1) If this identity-depth pair is already in a *lookahead* state then play the round-1 message already stored in the matrix and add it to the transcript.
- (2) If this identity-depth is in the *done* state (i.e. V^* 's secret key has already been discovered) then don't set the state to *beginning* and store the current transcript in M because the simulator can already straight-line simulate for this session and therefore will never need to rewind back the beginning of this session after a lookahead.

Comments to rnd2.handler ().

- (1) If the current session *state* (in M) is *done* then the simulator works out the result of the coin flipping (m_3) and plays the honest provers algorithm except that it uses the secret key ssk_i (which it has already extracted) in order to set c_2 such that the simulator now has a new witness to the ZAP rather than the witness to the theorem $x \in L$.
- (2) If the current session *state* is *beginning* then the simulator needs to start a lookahead in order to solve it. This is done by first rewinding V^* to the beginning of the session and playing a new first round where m_0 has been set to equal the result of the coin-flipping (m_3) of the current session. Then the simulator updates the variable T to V^* 's current view and finally it stores the transcript which lead to this lookahead, in M so that it can later return to this point when the lookahead is complete.
- (3) If the current session *state* is *lookahead* then the simulator can now complete the lookahead process. This is done by first rewinding to the point where the lookahead was initiated. And then deciding upon which of the two strategies it will use depending on whether it can detect aborts.
- (4) If there is at least one other un-finished session then aborts can be detected since in such a case V^* is expected to continue with the other session(s). If this is not the case then the simulator uses strategy 1 (i.e. it uses s for its round-3 message to set c_2 in such a way that it now has an alternative witness to the ZAP.

Algorithm `init.handler()`:

```

init.handler(ident : i, depth : d, transcript : T)
(1) if  $M[i, d]_{state} = l$ 
    send round-1 stored in  $M[i, d]_{rnd1}$ ;
     $T \leftarrow T + M[i, d]_{rnd1}$ ;
    else
    play prover alg. P1.1 - P1.4 resulting in  $rnd1 \leftarrow \text{round-1}$ ;
(2) if  $M[i, d]_{state} \neq d$ 
     $M[i, d] = (b, rnd1, T)$ ;
     $T \leftarrow T + rnd1$ ;
return (T);

```

Figure 9: Simulator `init.handler()` Function.

- (5) Since V^* 's aborts can be detected the simulator uses strategy 2.
- (6) For strategy 2 the simulator first extracts the encryption pairs $\{e_i^0, e_i^1\} \forall i \leq k'$ and plays them back to V^* .
- (7) Next the simulator detects an abort (or not) and extracts V^* 's secret key which it stores in `sski`.
- (8) The simulator updates the state of all depths for the identity i to *done* in M since at any depth, if identity i is used the session can now be straight-line simulated.
- (9) If any previous lookaheads were initiated for the identity i then V^* is rewound the point where the earliest such lookahead was initiated and any subsequent lookaheads in the stack are erased from memory.
- (10) V^* has been rewound to a session which required a lookahead for completion. This session is now completed by the simulator in playing the honest prover algorithm except for using the secret key `sski` in order to set c_2 such that the simulator now has a new witness to the ZAP rather than the witness to the theorem $x \in L$.

C.2 Analysis

Now we give a rigorous analysis of the simulator showing three properties: *completeness* (i.e. that all sessions are completed by some method), *indistinguishability* (to playing against the honest prover) and *efficiency* (i.e. that the runtime of the simulator is polynomially related to that of the verifier). If S satisfies all three properties we can conclude that it is a simulator for the composed protocol in Fig. 5.

Completeness. Of the 3 properties of S this is the most straightforward to show. It can be verified by carefully looking at the algorithm and noting the following: any session V^* may initiate will be either in a “beginning” state or in a “done” state. In the later case S will use the straight-line implantation method (i.e. it will use the already extracted secret signature key for this identity to create a witness for the final ZAP). Otherwise, for a session in the state “beginning”, S will use a look-ahead to gain some relevant information (s and $\{e_j^0, e_j^1\}_{j \leq k'}$). At this point the session is either the last unsolved session or not, i.e. either S generates a round-3 message using strategy 1 or strategy 2 both of which result in S having a witness to the statement for the final ZAP. Thus for any possible type of new session S has a method for completing the it such that the round-3 message of the protocol would be accepted by the honest verifier in step $V4$.

Indistinguishability. In order to show this property we will need two tools in the form of lemmas. The first states that a session solved with strategy 1 is indistinguishable to the session being solved by P and the second states that strategy 2 correctly extracts the secret signature key for the identity used by V^* . Once these have been proven the main proof can begin. It assumes by contradiction that there exists some algorithms V^* which can distinguish between S and P . Then using hybrid arguments and a careful analysis of all cases produce contradictions to lemmas 4.1 and C.1 and the proof is concluded. We begin with the lemma for strategy 1.

Lemma C.1 *For all PPT algorithms the transcript of a session between S , using strategy 1, and any V^* is indistinguishable from the transcript of the same session being played between P and V^* .*

PROOF. Assume by contradiction, that there exists some pair of algorithms D and V^* which can tell the difference between S playing strategy 1 and P playing honestly with a non-negligible probability in the following experiment: V^* plays against each of the parties producing two transcripts which D can then tell apart with non-negligible probability. Since both S and P play the exact same (honest provers) algorithm until calculating:

$$e = \text{Enc}(\text{epk}, (c_2, ZP(z, "x \in L \vee (c_2, \cdot) = \text{Com}'(\text{Sig}(m_3, \text{ssk})), \cdot)))$$

for the last part of their round-3 messages D clearly can only use the value of e to tell the difference between S and P .

We now look at a hybrid algorithm called H which does the following: given the same input as P (i.e. also a witness to $x \in L$) H runs S 's algorithm until the end of the session when it must decide upon how to set e for its round-3 message. It does this by using S 's strategy 1 algorithm for the value of c_2 and P 's algorithm for the ZAP. In other words $(c_2, d_2) = \text{Com}'(\text{Sig}(m_3, \text{ssk}))$ but the witness to $x \in L$ is used for the ZAP when calculating e . Since D and V^* could distinguish between S and P they must, by a hybrid argument, at least either be able to distinguish between S and H or H and P .

If D and V^* can distinguish between H and P they can be used to break the hiding property of the commitment scheme Com' as follows: first a transcript is produced using V^* by playing P 's algorithm against V^* up until the beginning of round-3. Next c is given where c is the commitment of a random message in the set $\{\text{Sig}(m_3, \text{ssk}), 0^l\}$ where l is the number of bits of $\text{Sig}(m_3, \text{ssk})$. Finally the session between V^* and P is completed as P would normally play except that c_2 is replaced by c when calculating e . Now D can be used to distinguish between commitments in the message set. If, on input the transcript, D decides that it was generated by H playing against V^* then with probability $(1/2)$ plus a non-negligible advantage $c = \text{Sig}(m_3, \text{ssk})$ while if D decides that the transcript was generated by P playing against V^* then with probability $(1/2)$ plus a non-negligible advantage $c = 0^l$. Therefore the existence of D and V^* contradict the the hiding property of the commitment scheme.

If, on the other hand, D and V^* can distinguish between S and H then they can be used to break the witness indistinguishability property of the ZAP protocol as follows: first a transcript is produced by running H 's protocol against V^* up until the beginning of round-3. Next z is given where z is a round-2 message of a ZAP (with the first round ZV) for the same statement as the ZAP in step P3.3 of the composed protocol. (Note that z has two possible (types of) witnesses: a witness to $x \in L$ or a witness to " c_2 is a commitment of a signature of m_3 with ssk ".) Finally the session between V^* and H is completed just as H would normally play except that in step P3.3 the ZAP is replaced by z when calculating the value of e . Now D can be used to distinguish between the witnesses of z by running it on input the transcript. If it decides that the transcript was generated by H then with probability $(1/2)$ plus a non-negligible advantage the witness used for z is a witness to $x \in L$, otherwise with probability $(1/2)$ plus a non-negligible advantage the witness used for z is d_2 for $(c_2, \cdot) = \text{Com}'(\text{Sig}(m_3, \text{ssk}))$. Therefore the existence of D and V^* contradict the witness indistinguishability property of the ZAP.

Since no algorithms exist which can distinguish between S , using strategy 1 for a session, and H nor between H and P , there is also no algorithm which can distinguish between S 's strategy 1 and P with more than a negligible probability of success. \square

Intuitively lemma C.1 will be used in the proof of indistinguishability by pointing out that a distinguishing V^* which uses a round-3 message generated by strategy 1 to tell the difference between S and P is contradicting the statement of the lemma.

Now we give a second lemma concerning the correctness of strategy 2.

Lemma C.2 *A simulator S using strategy 2 playing a session π against V^* will terminate the session such that $\text{ssk}_i = \text{ssk}$.*

Recall that S obtains the value of ssk_i by running a lookahead to obtain the encryption pairs $\{e_j^0, e_j^1\}$ along with the second round of a ZAP z stating that “the j -th encryption pair is of the correct form corresponding to the value of the j -th bit ssk for all $j \leq k'$ ”. Then for each pair S selects a random bit b and substitutes e_j^b for e in S 's round-3 message of π . If V^* aborts then S sets ssk_i 's j -th bit to $1 - b$. Otherwise it sets the bit to b .

PROOF. The proof of lemma C.2 goes by contradiction. Assume that S , using strategy 2, plays a session π against V^* such that S terminates with $\text{ssk}_i \neq \text{ssk}$. Let j be the first bit position where the two strings differ and let d be the value of the j -th bit of ssk_i . We show that the existence of j leads to contradictions.

By the correctness of the ZAP z sent by V^* during the lookahead in step V2.5 the value e_j^d , obtained by S during the same step of the lookahead, is either of the form:

$$\text{Enc}^{r_2}(\text{epk}, (c_2, ZP^{r_1}(z, "x \in L \vee (c_2, \cdot) = \text{Com}'(\text{Sig}(m_0, \text{ssk}))", (d_2, s))))$$

or of the form:

$$\text{Enc}^{r_2}(\text{epk}, 0^l)$$

where r_0, r_1 and r_2 are random strings chosen by S and l is the length of the previously encrypted message. Notice that, by substituting e_j^d of the first form for e in the round-3 message of π , the transcript is identically distributed to that of π being played with strategy 1 by S . (In both cases all steps are played honestly except that c_2 , a commitment of s , is used as the witness for the ZAP.) Therefore, by lemma C.1, if S plays round-3 with the above substitution, V^* accepts whenever e_j^d has the first form as long as it would accept when playing against P . Since V^* is non-aborting it would always accept when playing with P (as P knows a witness to $x \in L$ and can thus always provide an accepting ZAP in step P3.3) and so V^* would always accept when playing against an S using the above substitution in the last round.

Now we distinguish between the two forms e_j^d can have and show that both lead to contradictions. The first part is quite straight forward while if e_j^d takes on the second form we need to differentiate between whether S set $b = d$ or $b = 1 - d$ when running the strategy 2 algorithm for the j -th bit of ssk_i . In both cases the contradiction reached is that the j -th bit of ssk_i turns out to be $1 - d$ although we defined it to be d . Now for the details.

We begin by assuming that e_j^d has the first form. By the correctness of z the first form is taken by e_j^d if and only if d is the j -th bit of ssk . However this contradicts ssk and ssk_i differing in their j -th bits since we just showed they both equal d .

So we assume that e_j^d has the second form. Now there are two cases. When S ran strategy 2 to work out the j -th bit of ssk_i it either set $b = d$ or $b = 1 - d$. In the first case V^* would reject a round-3 message with e_j^b substituted for e since the second form for e_j^d is an encryption of only 0s. Thus S would set the j -th bit of ssk_i equal to $1 - d$. However this contradicts the above definition of d as being the j -th bit of ssk_i . In the second case (if $b = 1 - d$) S send e_j^{1-d} substituted for e in its round-3 message. Now by the correctness of z , e_j^{1-d} has the opposite form of e_j^d i.e. the first form. So, by the same reasoning as before when e_j^d had the first form, by lemma C.1, V^* would accept this round-3 message with e_j^{1-d} . However this would result in S setting the j -th bit of ssk_i to $1 - d$ which is again a contradiction of the definition of d .

Thus it has been shown that however j is chosen further reasoning leads to a contradiction. Therefore it can be concluded that no such j exist and in particular $\text{ssk}_i = \text{ssk}$. \square

Now we have all the tools we need to complete the general proof of the indistinguishability of S and P .

Theorem C.3 *There exists no pair of PPT algorithms D and V^* with V^* non-aborting and impatient which have a non-negligible probability in k to distinguish between an interaction with S and an interaction with P .*

PROOF. Assume by contradiction that there exists some PPT D and V^* , a non-aborting and impatient verifier, which can tell with probability $(1/2)$ plus a non-negligible advantage whether V^* interacted with S or with P via the following experiment: V^* runs the protocol depicted in Fig. 5 against an either S or P producing a transcript. On input this transcript D decides whether V^* was playing against S or P . Using hybrid arguments, there must exist some round sent by S in contrast to P in a given session that makes the transcripts distinguishable to D .

Now there are two cases. Case 1 is that m is a round-1 message and case 2 is that m is a round-3 message.

1. $m =$ **Round-1 Message:** By looking at the code of S it can be determined that there are only three places where a round-1 message is sent. So we have the following subcases:
 - (a) If the current session state is either “done” or “not seen” then the honest prover’s algorithm is used to generate and send m and so it can clearly not be used to distinguish between S and P .
 - (b) If the current session state is “beginning” then m is sent after a rewind at the beginning of a lookahead process. In this case the only difference between the honest prover’s algorithm and S ’s algorithm is that instead of selecting m_0 at random as P does S uses m_3 for m_0 , i.e the outcome of the coin flipping of the session. Note that since V^* is first rewound by S to a point before the coin-flipping started, it has no extra information about the value m_3 . Thus D can not detect any difference in the distributions of the m_0 s sent by P and S as the output of the coin-flipping protocol has the uniform distribution since the commitment scheme used by the prover/simulator is perfectly hiding.
 - (c) The last time a round-1 message is sent by S is when a newly initiated session is already in the state “lookahead”. In this case the round-1 message has been stored in the matrix M and is sent from memory. But since M has exactly one entry per identity-depth pair no stored round-1 message will be played more than once in any view of V^* . Therefore, at any given moment during the execution, all m_0 s in round-1 messages in V^* ’s view are independent of each other. So, from V^* ’s point of view, playing a round-1 message from memory is exactly the same as S starting the lookahead at that moment. Therefore the same argument leading to a contradiction as in the previous subcase holds here too.
2. $m =$ **Round-3 Message:** By looking at the code of S it can be determined that there are only three places where a round-3 message is sent. So we have the following subcases:
 - (a) First we consider a round-3 message which is sent when a session is solved by S with strategy 1. A D and V^* which can use such an m to distinguish between an interaction with S and with P with non-negligible probability contradicts the result of lemma C.1.
 - (b) Next we consider a round-3 message which is sent when a session is solved by S with strategy 2. The only difference between this round-3 message and one sent when S uses strategy 1 is that c_2 , a commitment of a signature of m_3 uses ssk_i rather than ssk for the signature. However, by lemma C.2 $\text{ssk}_i = \text{ssk}$ and so, as in the previous subcase, the same contradiction of lemma C.1 is reached.
 - (c) Finally we consider a round-3 message which is sent when a session is solved by S straight-lining. This time m is generated with the exact same algorithm as in the subcase where strategy 2 is used and so, once again, the same contradiction of lemma C.1 is reached.

In other words for any m which D and V^* use to distinguish between S and P all possible value which S gives m uses either the exact same algorithm as P or leads to a contradiction. Therefore no such m exists and so no pair D and V^* exists which has a non-negligible probability of distinguishing between S and P . \square

Note that as we have just shown no \mathcal{PPT} algorithm can tell whether a transcript came from playing against P or S and since the final output of S is the last view of the verifier, this output too is indistinguishable from the transcript the verifier interacting with P . In other words as an application of theorem C.3 we have that S is a simulator for P .

Efficiency. Between any two messages of V^* S will perform a constant number of instructions. Thus, to show that S has a polynomially related runtime to that of V^* , it suffices to compute the worst case number of rewinds S will perform. This is done listing up all the different situations in which S will rewind V^* and calculating how often each one can occur within a concurrent block of sessions. Thus we have the following cases:

1. **Start Lookahead:** This is the case where a rewind is done to start a lookahead. In this case the state of the session is changed from “beginning” to “lookahead” in M . Therefore this type of rewind can occur a maximum of $D * I$ times (which is the number of entries of M) before at least one lookahead is completed. (Recall that D is the maximum depth V^* will reach and I is the number of identities in F , thus $D * I$ is polynomial in k .)
2. **Finish a Lookahead with Strategy 1:** Once a round-2 message for a lookahead is received by S it rewinds back to the session it was trying to solve when it started the lookahead. If this is the last open session then it is solved with out another rewind using strategy 1. In such a case the simulator has completed all open session in the concurrent block and thus terminates outputting T . Therefore this type of rewind can occur at most one time during the S execution solving the concurrent block and requires 1 rewind.
3. **Finish a Lookahead with Strategy 2:** If a lookahead is completed by S using strategy 2 then a new secret signature key is discovered. This required 1 rewind to return to the session being solved, k rewinds for the k bits of the key, at most 1 rewind to the earliest lookahead for identity i listed in C , and finally 1 last rewind to the point just before round-3 is played for the session in order to finish it. Therefore at most $k + 3$ rewinds are required to finish a session with strategy 2. Since there are I identities, S will finish at most I sessions with this method since then all secret signature keys have been discovered and everything can now be straight-lined. In other words at most $I * (k + 3)$ rewinds can be incurred with this method.

Now we are ready to compute the worst case possible number of rewinds for a concurrent block of sessions. At most $D * I$ rewinds are required before at least one lookahead is completed. At most I lookaheads can be completed with strategy 2 and at most 1 with strategy 1 requiring $k' + 3$ rewinds and one rewind respectively. Thus S will perform at most $D * I(I(k' + 3) + 1)$ rewinds before it terminates. Since both D , I and k' are polynomial in k so is the total number of rewinds of S within a concurrent block of sessions. Therefore it can be concluded that S has a polynomially related runtime to V^* .

Algorithm rnd2.handler():

```
rnd2.handler(message: rnd2, ident : i, depth : d, transcript : T)
  select case  $M[i, d]_{state}$ 
(1)  d: compute  $m_3$  from  $T$  and  $rnd2$ ;
      play prover alg. P3.1 - P3.4 except use  $ssk_i$  to generate
       $(c_2, d_2) = Com'(Sig(m_3, ssk))$  resulting in  $rnd3 \leftarrow$  the round-3
      message sent to  $V^*$ ;
       $T \leftarrow T + rnd3$ ;
(2)  b: compute  $m_3$  from  $T$  and  $rnd2$ ;
      rewind  $V^*$  to  $M[i, d]_{transcript}$ ;
      play prover alg. P1.1 - P1.4 except set  $m_0 \leftarrow m_3$  in  $rnd1$ ,
      the round-1 message sent to  $V^*$ ;
       $temp \leftarrow T$ ;
       $T \leftarrow M[i, d]_{transcript} + rnd1$ ;
       $M[i, d] = (l, rnd1, temp)$ ;
       $C \leftarrow C + M[i, d]$ ;
(3)  l: rewind  $V^*$  to  $M[i, d]_{transcript}$ ;
      if NumOpenSessions( $T$ ) = 1;
(4)  read out signature  $s$  from  $rnd2$ ;
      play prover alg. P3.1 - P3.4 except use  $s$  to generate
       $(c_2, d_2) = Com'(Sig(m_3, ssk))$  resulting in  $rnd3 \leftarrow$  the round-3
      message sent to  $V^*$ ;
(5)  else for all  $j \leq k'$ 
(6)  read out  $\{e_j^0, e_j^1\}$  from  $rnd2$ ;
      select  $b \leftarrow \{0, 1\}$  at random;
      play prover alg. P3.1 - P3.4 except replace  $e$  with  $e_j^b$ ;
(7)  if  $V^*$  plays a new message
      set bit  $j$  of  $ssk_i$  to  $b$ ;
      else
      set bit  $j$  of  $ssk_i$  to  $1 - b$ ;
      rewind  $V^*$  to  $M[i, d]_{transcript}$ ;
(8)  for all  $t \leq D$ 
      set  $M[i, j] = (d, 0, 0)$ ;
(9)  if  $C$  contains any other lookaheads for identity  $i$ ;
      let  $l$  be the first such lookahead in  $C$ ;
      for all lookaheads in  $C$  which come after  $l$  in  $C$  and
      whose state is not done set their entry in  $M$  to  $(n, 0, 0)$ 
      and remove them from  $C$ ;
       $d \leftarrow$  depth of  $l$ ;
      rewind  $V^*$  to  $M[i, d]_{transcript}$ ;
(10) play prover alg. P3.1 - P3.4 except use  $ssk_i$  to generate
       $(c_2, d_2) = Com'(Sig(m_3, ssk))$  resulting in  $rnd3 \leftarrow$  the round-3
      message sent to  $V^*$ ;
       $T \leftarrow M[i, d]_{transcript} + rnd3$ ;
```

Figure 10: Simulator rnd2.handler() Function.