Lillian Kråkmo

Privacy Preserving Protocols and Security Proof Techniques

Thesis for the degree of Philosophiae Doctor

Trondheim, August 2009

Norwegian University of Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Mathematical Sciences
PREFACE

This thesis is submitted in partial fulfillment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The work has been carried out at the Department of Mathematical Sciences as part of the project “ICT and Mathematics”, which aims to strengthen the theoretical research in information and communication technologies at NTNU. The project is funded by NTNU, the Faculty of Computer Science, Mathematics and Electronics and the involved departments. My supervisors have been Dr. Aslak Bakke Buan at the Department of Mathematical Sciences and Prof. Stig Frode Mjølsnes at the Department of Telematics.

My topic of research has been privacy preserving cryptographic protocols and mathematical techniques for proving their security. As a particularly powerful tool in this context, I have studied the framework of universally composable security. The work has resulted in five papers, comprising the main part of this thesis. The first four papers are concerned with how security of different tasks can be expressed and achieved within this model, starting with basic cryptographic building blocks, and moving on to more complex protocols. The fifth paper proposes an application of formal methods to proving universally composable security, as a potential first step towards automatic verification of such proofs.

Looking back at my years as a PhD student, I am truly grateful for this opportunity to learn so much about cryptography, research and life in general. I sincerely thank my supervisors Dr. Aslak Bakke Buan and Prof. Stig Frode Mjølsnes for their valuable guidance and encouragement. A special thanks goes to my co-author Dr. Kristian Gjøsteen, for willingly sharing his expertise in cryptography, for always providing me with interesting research problems, and for pointing me in the right direction whenever I got stuck.

My colleagues at the department all deserve a great thanks, for creating such a nice work environment. I thank the technical/administrative staff for always being helpful, and for providing me with excellent computer and coffee facilities. In particular, I thank my fellow PhD students, for keeping me company and making these years memorable.

I am also grateful to my family and friends, for trying to understand my PhD problems and occasionally helping me forget them. My parents deserve a special thanks, for their love and support, and for countless hours of babysitting.

Finally, I thank Frantz and Sverre, for standing by me, and for constantly reminding me of the truly important things in life.

Lillian Kråkmo
Trondheim, June 2009
CONTENTS

Introduction

Paper I:
Universally Composable Signcryption
Kristian Gjøsteen and Lillian Kråkmo
Published in Proceedings of EuroPKI 2007, volume 4582 of LNCS

Paper II:
Universally Composable Blind Signatures in the Plain Model
Aslak Bakke Buan, Kristian Gjøsteen and Lillian Kråkmo
Preprint

Paper III:
Round-Optimal Blind Signatures from Waters Signatures
Kristian Gjøsteen and Lillian Kråkmo
Published in Proceedings of ProvSec 2008, volume 5324 of LNCS

Paper IV:
A Universally Composable Anonymous Online Service
Lillian Kråkmo
Preprint

Paper V:
Secure Messaging from Signcryption:
An Application of Formal Methods to Universal Composability
Lillian Kråkmo
Preprint
INTRODUCTION

As more and more people have come to rely on the internet for conducting their everyday activities, an increasing degree of trust is placed with the communication systems providing the numerous services, and in particular with the underlying cryptographic protocols. According to [21], a cryptographic protocol is a ‘distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve some specific security objective’. Typically, such algorithms involve data transfer across insecure networks, and should maintain some level of security even if some entities deviate from their specified behavior. Examples of cryptographic protocols include protocols for entity authentication, email, electronic voting and online banking. As demonstrated by these examples, security objectives of such algorithms are diverse in content and complexity. An entity authentication protocol should merely assure one entity of another’s identity. On the other hand, a protocol for online banking may use an entity authentication protocol as a subroutine for providing just one out of a wide range of required security properties.

SECURITY OF CRYPTOGRAPHIC PROTOCOLS

It is an empirical fact that designing and analyzing cryptographic protocols is a difficult and error-prone task. While a detailed survey is out of scope, we will highlight some trends within this area of research. A defining example in this context is the Needham-Schroeder public key protocol for mutual entity authentication [24], which was broken by Lowe [20] almost 20 years after its publication, despite the fact that Abadi, Burrows and Needham were able to formally prove its correctness [6]. Interestingly, rather than benefiting from stronger tools for protocol analysis, Lowe’s attack resulted from a modified view regarding the power of an adversary. While the authors of [6] worked under the assumption that users of a network are honest, and only considered attacks by outsiders, Lowe also considered attacks by insiders, i.e. by regular users of the network. Undoubtedly, Lowe’s approach better reflects modern networks like the internet, where users are numerous and cannot necessarily trust one another [12].

The above example illustrates some of the subtleties involved in designing cryptographic protocols, and pinpoints the importance of a rigorous security model. Traditionally, we have seen two main approaches for security analysis of protocols. In computational models, cryptographic primitives are treated as algorithms, and adversaries are computationally bounded entities with access to the inputs and outputs of these algorithms. Security is typically defined in a probabilistic sense, and relies on computational intractability assumptions. A variety of such models have been proposed [16, 17, 5]. A common feature of these models is that security
notions tend to get relatively complex, even for simple protocols. Furthermore, the security proofs require some level of human creativity, since breaking the security of the protocol in question must be reduced to breaking some underlying hard problem. Symbolic models, on the other hand, treat cryptographic primitives as symbolic operations, which immediately guarantee a set of idealized security properties. As a result, protocol analysis becomes considerably simpler. Examples of such models are the Dolev-Yao model [13], the BAN logic [6] and the Spi-calculus [1]. However, when it comes to guaranteeing security for protocols running in realistic settings, the computational approach is the only one that is obviously sound.

In most of the above works, security definitions aim to capture one particular task, like entity authentication or public key encryption. It may be argued that, rather than a problem-specific approach, we need a general analytical framework, allowing different tasks to be handled within the same model. Such a framework facilitates a methodological specification of security objectives, and may improve our understanding of the required properties and and their formalizations. Consequently, a general framework may lead to simpler and less error-prone formulation of security definitions. Moreover, while protocols were formerly analyzed in a stand-alone setting, modern communication systems are more likely characterized by multiple protocols running in parallel, and protocols using other protocols as subroutines. Indeed, proving security in such a scenario is feasible only in the context of a unified framework [8].

The need for such a framework was first pointed out by Yao in [2]. A few years later, Goldreich, Micali and Wigderson [15] suggested to define security for the general task of secure function evaluation, by comparing the protocol in question to an ideal process, involving a trusted party. This idea was further developed by the works of Goldwasser and Levin [18], Micali and Rogaway [22], Beaver [4] and numerous others, and resulted in the trusted-party paradigm, which is now considered to be fundamental in the field of protocol analysis. Within this paradigm, a protocol is considered secure if it emulates the ideal process for the task at hand, which may be considered as a formal specification of the required security properties. The high level idea is that, if a protocol emulates the ideal process, then any effect caused by an adversary interacting with the protocol can also be caused by an adversary interacting with the ideal process. We can then conclude that, since the ideal process is designed to withstand any attack, this also holds for the protocol.

While the initial works following this paradigm treat protocols as stand-alone entities, we have lately seen a development of models that guarantee security-preserving composition of protocols [25, 3, 7]. Notably, these models combine the computational and the symbolic approach to protocol analysis, in order to obtain both soundness and simplification of analysis. In such models, primitives are represented by idealized abstractions. These abstractions are realizable by actual protocols, and may also be deployed as subroutines by higher-level protocols, thus allowing for a more mechanical security analysis. At the same time, soundness
is guaranteed by a strong composition theorem, which ensures that a protocol using an abstraction retains its security when the abstraction is replaced by a realizing protocol. The work presented in this thesis is mainly concerned with one specific such model, namely the framework of universally composable (UC) security, developed by Canetti [7].

**The UC Framework**

The UC framework provides a general method for formulating security definitions for cryptographic tasks, and for determining whether a given protocol is secure. As opposed to conventional models, this framework guarantees that a protocol maintains its security within any context, i.e. even if run concurrently with arbitrary other protocols in an adversarially controlled manner. Moreover, the involved composition theorem allows for a modular design and analysis of protocols. While a brief review of the framework is given herein, we refer to [7] for a full overview.

**The Computational Model.** We start by a high-level description of the underlying computational model. In the UC framework, a network of communicating computer programs is represented by a system of interactive Turing machines (ITMs). Each ITM has certain tapes that are writable by other ITMs: an incoming communication tape, which models communication over an insecure network, in addition to an input tape and a subroutine output tape, which model local subroutine calls. We note that, while an ITM corresponds to a computer program, an ITM instance (ITI) corresponds to an instance of a program running on some specific data.

A system of ITMs \((I, C)\) consists of an initial ITM \(I\) and a control function \(C\). An execution of such a system starts when an instance of \(I\), called the initial ITI, is invoked on some external input, and proceeds by a series of activations of ITIs. In each activation, only one ITI is active, and this ITI may write to the tapes of at most one other ITI, in addition to doing local computations. Once the active ITI enters a special wait state, the ITI whose tapes were written to becomes active. If no tapes of other ITIs were written to, the initial ITI is activated. The control function determines whether an ITI can write to a certain tape of another ITI. An execution ends when the initial ITI halts. The output of an execution is the output of the initial ITI.

Some comments are in order concerning the above modeling. First, in order to let an ITI specify which ITI it wishes to address, each ITI is equipped with a permanent identity. If there is no ITI with the specified identity in the system, then an ITI with this identity is created. Second, each identity consists of a session id (SID) and a party id (PID). A set of ITIs in an execution constitute a protocol instance if they are instances of the same ITM and have the same SID. The PIDs are used to identify different parties within a protocol instance.

**The Model of Protocol Execution.** We proceed by describing the model of protocol execution. This model is parametrized by three ITMs: \(\pi\) represents the
protocol to be executed, $A$ is the adversary, and $Z$ is the environment. While $A$ represents attacks directly aimed at the instance of $\pi$ in question, $Z$ represents the other protocol instances running in the system, including those using the instance in question as a subroutine. Given $\pi, A$ and $Z$, the model is defined as a system of ITMs, where $Z$ is the initial ITM, and the control function is defined as follows:

- $Z$ first invokes the adversary $A$. In following activations, $Z$ provides inputs either to $A$ or some instance of $\pi$. These instances are required to have the same SID, i.e. they are all parties of the same protocol instance.
- The adversary may either write some message to some party’s incoming communication tape, corrupt some party, or write some message to $Z$’s subroutine output tape. We note that no restrictions are made on the written messages. $A$ corrupts a party if and only if instructed to do so by $Z$, and does so by writing some specific message on that party’s incoming communication tape. The party’s response to this message is determined by the corruption model, which is specified in the party’s program.
- The parties of $\pi$ may write a message to $A$’s incoming communication tape or $Z$’s subroutine output tape. Moreover, they may invoke new ITIs as subroutines, provide these with inputs, and read their outputs. The subroutine ITIs may invoke new ITIs as subroutines of their own, and so on.

An execution of the system ends when $Z$ halts. Without loss of generality, we assume that $Z$’s output consists of a single bit. The model of protocol execution is illustrated in Figure 1.

![Figure 1](image_url)

**Figure 1.** The model of protocol execution. For each ITI, a line coming from above represents the input tape, a line coming from below represents the subroutine output tape, and a line coming from either side represents the incoming communication tape.

Regarding the model of protocol execution, note that $Z$ and $A$ may interact freely after each activation of some party. This feature reflects the continual flow
of information between protocol executions that run concurrently, and turns out
to be essential for the composition theorem to hold. We also emphasize that parties can only send messages to $A$, and not directly to each other. This means that $A$ is in total control of the network. In particular, the messages delivered by $A$ need not be related to the messages it receives from the parties.

It is stressed that the behavior of parties upon corruption is not defined by the general model, but specified as part of the protocol description. The most common corruption model is that of Byzantine corruption, where a corrupted party sends its entire state to $A$, and follows $A$’s instructions in all remaining activations. Note that, while in general, $Z$ may instruct $A$ to corrupt parties adaptively throughout the protocol execution, it is sometimes useful to consider a static corruption model, where the identities of the corrupt parties are fixed by $Z$ in advance.

**Ideal Protocols.** Security of a protocol is defined by comparison with an ideal protocol for the task at hand. This protocol is formulated within the above model, but is different in the sense that it involves an ideal functionality. The ideal functionality, denoted by $\mathcal{F}$, is a special ITM behaving as a joint subroutine of several ITIs. $\mathcal{F}$ serves as a trusted party and incorporates the required properties of a protocol for the task. Given $\mathcal{F}$, the ideal protocol, denoted by IDEAL$_{\mathcal{F}}$, is defined as follows: When a party receives an input, it simply forwards this input to the instance of $\mathcal{F}$ carrying the local SID. Oppositely, when a party receives an output from $\mathcal{F}$, this is copied to the local output. The parties in the ideal protocol are thus often referred to as dummy parties for $\mathcal{F}$. $\mathcal{F}$ follows its instructions for generating outputs to parties from the given inputs. Moreover, $\mathcal{F}$ may be instructed to send messages to the adversary, and may also receive messages from the adversary. In the ideal protocol, corruption is modeled by specific messages sent between the adversary and $\mathcal{F}$.

To reflect the special role played by the adversary in this setting, we will refer to it as the ideal adversary and denote it by $S$. The communication between $\mathcal{F}$ and $S$ represents the influence/information that an adversary is allowed to obtain. For instance, $S$ may delay certain outputs to parties, or learn the length of a message being encrypted. The model of execution for an ideal protocol is depicted in Figure 2.

**Realizing Functionalities.** In the UC framework, protocol security is expressed by the notions of UC-emulation of protocols and UC-realization of ideal functionalities. In order to define these concepts, we need a probability ensemble describing the output of the environment $Z$ when interacting with a protocol $\pi$ and an adversary $A$. We denote this ensemble by

$$\text{EXEC}_{Z,A,\pi} = \{\text{EXEC}_{Z,A,\pi}(n, z)\}_{n \in \mathbb{N}, z \in \{0,1\}^n},$$

where $n$ is the security parameter and $z$ is the external input of $Z$.

**Definition 1.** A protocol $\pi$ UC-emulates a protocol $\phi$ if for any probabilistic polynomial time (PPT) adversary $A$ there exists a PPT adversary $S$ such that
Figure 2. The model of execution for an ideal protocol $\phi$.

for any PPT environment $Z$ we have

$$|\text{Prob}[\text{EXEC}_{Z,A,\pi} = 1] - \text{Prob}[\text{EXEC}_{Z,S,\phi} = 1]| = \nu(n),$$

for some negligible function $\nu$.

$\pi$ UC-realizes an ideal functionality $F$ if it UC-emulates the ideal protocol $\text{IDEAL}_{F}$.

Informally, we say that $\pi$ UC-realizes an ideal functionality $F$ if, for any adversary $A$ there exists an ideal adversary $S$ such that no environment $Z$ can tell whether it is interacting with $\pi$ and $A$ or with $\text{IDEAL}_{F}$ and $S$. In particular, this means that the input/output behavior of good parties is the same in both scenarios, and that any information learnt from $A$ in interactions with $\pi$ can be imitated by $S$ in interactions with $\text{IDEAL}_{F}$. Intuitively, if any influence/information gained by $A$ can be imitated by $S$, who only obtains what is explicitly permitted by $F$, then we can conclude that $A$ is harmless.

$S$ is often referred to as a simulator, since in typical proofs of UC-realization, $S$ runs a simulated copy of $A$.

The Composition Theorem. Let $\pi$ be a protocol in which parties make subroutine calls to some protocol $\phi$, and assume that $\rho$ is a protocol that UC-emulates $\phi$. The composed protocol $\pi^{\rho/\phi}$ is then defined as the protocol $\pi$ where calls to $\phi$ are replaced by calls to $\rho$, i.e. inputs to some instance of $\phi$ are treated as inputs to an instance of $\rho$ with the same identity, and outputs from some instance of $\rho$ are treated as outputs from the corresponding instance of $\phi$.

Theorem 2. Let $\pi$, $\phi$ and $\rho$ be protocols such that $\rho$ UC-emulates $\phi$. Then $\pi^{\rho/\phi}$ UC-emulates $\pi$.

The above result is the composition theorem in its most general form. A corollary of particular interest concerns the notion of hybrid protocols. An $F$-hybrid protocol is a protocol where, in addition to communicating via the adversary, parties can make subroutine calls to instances of $\text{IDEAL}_{F}$. For the special case
of protocol composition where the replaced protocol $\phi$ is an $F$-hybrid protocol, we denote the composed protocol by $\pi^{\rho/F}$.

**Corollary 3.** Let $\pi$ be an $F$-hybrid protocol, and let $\rho$ be a protocol that UC-realizes $F$. Then $\pi^{\rho/F}$ UC-emulates $\pi$. Moreover, if $\pi$ UC-realizes an ideal functionality $G$, then so does $\pi^{\rho/F}$.

In order to illustrate the above corollary, the $F$-hybrid protocol $\pi$ and the composed protocol $\pi^{\rho/F}$ are depicted in Figure 3. For graphical clarity, the dummy parties of $F$ are omitted.

![Figure 3. The $F$-hybrid protocol $\pi$ and the composed protocol $\pi^{\rho/F}$.](image)

The composition theorem facilitates a modular approach to design and analysis of protocols. When designing protocols for complex tasks, one may assume that parties have secure access to ideal functionalities for different subtasks, which greatly simplifies protocol analysis. Then, when implementing the protocol, each functionality can be replaced by a realizing protocol. Security of the obtained protocol is then guaranteed by the above corollary.
Our Work

A main focus of our work is how the security of various cryptographic tasks may be captured within the UC framework, starting with cryptographic primitives, and moving on to more complex protocols. Seeing that UC security is a powerful notion, an interesting question is how it relates to conventional security definitions. An implication from UC security of a given task to a widely accepted notion would provide some evidence that the proposed functionality incorporates a sufficient level of security. Oppositely, an implication in the other direction would indicate that the UC notion is not overly restrictive. Moreover, since such a result would reduce proving UC security to proving security in the standard model, it allows for simpler proofs, as the former task tends to be more complex.

In Paper I, we define a functionality for signcryption, which is a primitive proposed by Zheng [28]. The purpose of signcryption is to obtain both confidentiality and authenticity of message delivery/storage in a logically single step, thus potentially reducing the cost compared to the standard “sign-then-encrypt” method. We show that a signcryption protocol realizes the functionality if and only if the corresponding signcryption scheme is secure with respect to indistinguishability and unforgeability, adapted from the analogous notions for public key encryption and digital signatures. Furthermore, with applications such as email and instant messaging in mind, we propose a functionality for secure messaging, and prove that this functionality can be realized using functionalities for signcryption and a public key infrastructure.

The idea of blind signatures was proposed by Chaum [11] as a key ingredient for anonymous electronic cash applications. Blind signatures allow a bank to issue signatures without seeing the content of the signed documents, and at the same time prevent users from forging signatures. Paper II revisits a functionality for blind signatures proposed by Fischlin [14], and presents an alternative formulation, which may allow for a larger class of realizing protocols. Also, under the assumption that honest key generation may be enforced, we show that our functionality is realized by a blind signature protocol if and only if the corresponding blind signature scheme is secure with respect to blindness and non-forgeability, as defined by Juels, Luby and Ostrovsky [19].

Along with the problem of defining functionalities for cryptographic tasks comes the question of how to design efficient protocols realizing them. This question is addressed in Paper III, where we present a round-optimal blind signature scheme. Prior to our work, Fischlin proposed a round-optimal blind signature scheme in the common reference string model [14]. While Fischlin’s scheme relies on generic non-interactive zero-knowledge (NIZK) proofs, making it quite impractical, our scheme is a concrete construction based on Waters signatures [27]. In order to obtain provable security, the user is required to compute a moderate number of NIZK proofs as part of the signature generation protocol. These are obtained by compiling a dedicated Σ-protocol, using a technique developed by Damgård et al. [26]. Consequently, in addition to a common reference string, our scheme requires a registered public key for the signer.
Having considered UC security in the context of some basic cryptographic building blocks, a natural next step is to investigate the behavior of more complex protocols within this framework. This is the main purpose of **Paper IV**. As an example of a protocol of realistic complexity, consider an online newspaper offering subscriptions on a pay-per-read basis. Such a service raises a number of security concerns. As a matter of privacy protection, the subscribers may not wish to reveal their identities to the newspaper. On the other hand, the newspaper should be ensured that a user pays for the provided service. This may be obtained by involving a bank, which guarantees the validity of a payment issued by a user. The bank may also allow for deposits of these payments by the newspaper. To prevent the bank from learning where a user has spent his money, it should be infeasible to link specific payments to specific users. From the bank’s point of view, the involved payments should be unforgeable.

In **Paper IV**, we propose a functionality for an **anonymous online service**, incorporating the above properties. We also construct a hybrid protocol realizing the functionality, which uses functionalities for several cryptographic tasks. The security properties concerning payments and deposits are provided by using a dedicated blind signature functionality. Also, the pairwise communication between parties in the protocol requires various levels of privacy. In particular, the communication between the server (i.e. the newspaper) and a user ought to be anonymous at the user’s end. One might consider using the secure messaging functionality from **Paper I** for this purpose, along with a functionality for an anonymous network. However, according to our functionality, which aims to capture a general anonymous online service, a typical session between the server and a user involves sending several messages back and forth. It thus seems more convenient to use an anonymous secure channel, where asymmetric encryption is used for key distribution, while symmetric encryption is used for data exchange. Accordingly, **Paper IV** also defines a functionality for an anonymous secure channel. Moreover, by using an additional functionality for an anonymous network, we adapt a secure channel proposed by Nagao et al. [23], and obtain a protocol realizing the functionality.

As can be seen from the security arguments in **Paper IV**, proving security within the UC framework is a tedious task. Although protocol analysis is simplified by assuming secure access to ideal functionalities, it still requires that security proofs be obtained within a full-fledged cryptographic model. This problem is addressed by Canetti and Herzog in [10, 9], where they demonstrate how symbolic analysis within a simple model can be used to argue about the UC security of a concrete protocol. **Paper V** presents an alternative approach to solve this problem, by using state machine theory. As an example, we revisit the secure messaging protocol from **Paper I**, and use our technique to prove that the protocol realizes the secure messaging functionality. Most of the steps involved in the proof boil down to mechanical manipulations of state machines, which should be easy to implement on a computer. Our approach can thus be considered as a first step towards automatic verification of such proofs. We emphasize that this paper...
presents ongoing work. For instance, our approach currently only applies to protocols where the Turing machines representing the parties are all deterministic. Eliminating this restriction is an object of further study.

REFERENCES


Paper I

Universally Composable Signcryption
Kristian Gjøsteen and Lillian Kråkmo
Published in Proceedings of EuroPKI 2007, volume 4582 of LNCS
UNIVERSALLY COMPOSABLE SIGNCRYPTION

KRISTIAN GJØSTEEN AND LILLIAN KRÅKMO

Abstract. One of the challenges within public-key based cryptosystems is providing the user with a convenient interface, while retaining security. In the framework of universally composable security, we propose an ideal functionality for secure messaging, with a user-friendly interface. We also show how to realize it using a cryptographic primitive and a public key infrastructure.

1. Introduction

Signcryption was first proposed by Zheng [7] as a primitive for achieving both confidentiality and authenticity of message delivery/storage in a logically single step, with the aim of reducing the cost compared to the standard “sign-then-encrypt” method. Regarding security definitions for signcryption schemes, several approaches have been taken. An overview of the different models is provided in [5].

In general, composing several (possibly identical) protocols into a larger protocol may not preserve security. Universally composable security is a framework proposed by Canetti [3] as a way to define security for protocols such that security-preserving composition is possible. This allows for a modular design and analysis of protocols.

For each cryptographic task, an ideal functionality can be defined, which incorporates the required properties of a protocol for the task and the allowed actions of an adversary. A protocol is said to securely realize the ideal functionality if, loosely speaking, any effect caused by an adversary attacking the protocol can be obtained by an adversary attacking the ideal functionality. When designing complex protocols, one can allow the involved parties to have secure access to ideal functionalities. Then, when implementing the protocol, each ideal functionality is replaced by a protocol securely realizing the functionality. The composition theorem then guarantees security. We refer to [3] for a complete overview of this framework.

In Section 2 of this paper, we review the properties of a signcryption scheme and define what it means for a signcryption scheme to be secure. Based on these security requirements, we construct an ideal functionality for signcryption, which is defined in Section 3. This section also presents a natural ideal functionality for secure messaging, which is a suitable model for applications such as secure email.
and secure instant messaging. Given functionalities for a public key infrastructure and signcryption, we construct a protocol that integrates these services and securely realizes the secure messaging functionality.

Finally, in Section 4 we show that a signcryption scheme satisfies our security definitions if and only if the corresponding protocol securely realizes the signcryption functionality. We note that our results are only valid in the static corruption case. This is discussed further in Section 5.

2. Signcryption

In this section, we define what a signcryption scheme is, and describe our security model for such schemes (adapted from the corresponding notions for public-key encryption and digital signatures).

The following definition of a signcryption scheme is similar to the one given in [5].

Definition 1. A signcryption scheme $SC$ is a 4-tuple of algorithms $(K_s, K_r, S, U)$ with the following properties:

- $K_s$ is a probabilistic algorithm, taking as input a security parameter $\tau$ (encoded as $1^\tau$) and returning a key pair $(sk^s, pk^s)$ for the sender. $sk^s$ is the secret key, while $pk^s$ is the public key.
- $K_r$ is a probabilistic algorithm, taking as input a security parameter $\tau$ (encoded as $1^\tau$) and returning a key pair $(sk^r, pk^r)$ for the receiver. $sk^r$ is the secret key, while $pk^r$ is the public key.
- $S$, the signcryption algorithm, is probabilistic. Its inputs are a sender’s private key $sk^s$, a receiver’s public key $pk^r$ and a plaintext $m$, and its output is a ciphertext $c$.
- $U$, the unsigncryption algorithm, is deterministic. Its inputs are a sender’s public key $pk^s$, a receiver’s secret key $sk^r$ and a ciphertext $c$. Its output is a plaintext $m$ or the symbol $\bot$, indicating that the signcryption is invalid.

It is required that $U(pk^s, sk^r, S(sk^s, pk^r, m)) = m$ for all plaintexts $m$ and all key pairs $(sk^s, pk^s)$ and $(sk^r, pk^r)$ output by $K_s$ and $K_r$.

Our security model for signcryption schemes corresponds to the ADR model presented in [5]. We note that we do not consider non-repudiation. Therefore we need only consider unforgeability between honest users. We need two experiments described in Figure 1.

The first experiment $Exp_{SC,A}^{\text{ind-cca2}}$ concerns privacy of messages, and adapts the notion IND-CCA2 from public-key encryption. In the beginning of the experiment, the adversary $A$ is given two public keys $pk^s$ and $pk^r$ belonging to the target sender and the target receiver, respectively. $A$ is composed of a find-stage algorithm $A_1$ and a guess-stage algorithm $A_2$. $A_1$ finds two messages $m_0$ and $m_1$ of the same length, while $A_2$ is given a challenge ciphertext $c$ and guesses whether $c$ is a signcryption of $m_0$ or $m_1$.

Both $A_1$ and $A_2$ have access to a flexible signcryption oracle $O_S$, which performs signcryption under the fixed key $sk^s$ and an arbitrary key $pk^r$, and a
The scheme $\mathcal{SC}$ is said to be secure with respect to IND-CCA2 if the advantage $\text{Adv}_{\mathcal{SC},A}^{\text{ind-cca2}}(\tau)$ is negligible in $\tau$, whenever $A$’s runtime and number of oracle queries are polynomially bounded in $\tau$.

The second experiment $\text{Exp}_{\mathcal{SC},A}^{\text{ext-cma}}(\tau)$ concerns unforgeability of messages, and adapts the notion EXT-CMA from digital signatures. This experiment starts with the adversary $A$ being given the target sender’s public key $pk^s$ and the target receiver’s public key $pk^r$. $A$’s job is to produce a ciphertext $c$ such that $c$ is a valid ciphertext with respect to the target sender and the target receiver. $A$ has access to the oracles $\mathcal{O}_S$ and $\mathcal{O}_U$ described above. It is required that $c$ was not output by $\mathcal{O}_S$ on input of $pk^r$.

A is said to win if the experiment returns 1. We define the success rate of $A$ in breaking $\mathcal{SC}$ with respect to EXT-CMA as

$$\text{Succ}_{\mathcal{SC},A}^{\text{ext-cma}}(\tau) = \Pr[\text{Exp}_{\mathcal{SC},A}^{\text{ext-cma}}(\tau) = 1].$$

The scheme $\mathcal{SC}$ is said to be secure with respect to EXT-CMA if the advantage $\text{Succ}_{\mathcal{SC},A}^{\text{ext-cma}}(\tau)$ is negligible in $\tau$, whenever $A$’s runtime and number of oracle queries are polynomially bounded in $\tau$. 

We now present a second notion for privacy of messages, adapting the notion "real-or-random" for symmetric encryption given in [1]. The idea is that no adversary should be able to distinguish a signcryption of a known message from a signcryption of a hidden random string. In the experiment $\text{Exp}_\text{ror-cca2}(\tau)$, the adversary $A$ has access to the oracle $O^b_{\text{ror}}$ (initialized with a hidden bit $b$) which takes as input a message $m$. If $b = 0$, it outputs a signcryption of a randomly chosen string of length $|m|$ under $sk^s$ and $pk^r$. A new random string is chosen for each query. If $b = 1$, it outputs a signcryption of $m$ under $sk^s$ and $pk^r$. $A$'s challenge is to guess the hidden bit $b$. As before, the adversary has access to $O_S$ and $O_U$.

In this experiment we require that $A$ does not query $O_U$ with any of the ciphertexts output by $O^b_{\text{ror}}$ together with $pk^s$ and $pk^r$.

We define the advantage of $A$ in breaking $SC$ with respect to ROR-CCA2 as

$$\text{Adv}_{SC,A}^{\text{ror-cca2}}(\tau) = \left| \Pr[\text{Exp}_{SC,A}^{\text{ror-cca2}}(\tau) = 1 | b = 1] - \Pr[\text{Exp}_{SC,A}^{\text{ror-cca2}}(\tau) = 1 | b = 0] \right|.$$ 

The scheme $SC$ is said to be secure with respect to ROR-CCA2 if the advantage $\text{Adv}_{SC,A}^{\text{ror-cca2}}(\tau)$ is negligible in $\tau$, whenever $A$’s runtime and number of oracle queries are polynomially bounded in $\tau$.

The following theorem is a straight-forward adaptation of a theorem in [6].

**Theorem 2.** A signcryption scheme $SC$ is secure with respect to IND-CCA2 if and only if it is secure with respect to ROR-CCA2.

### 3. Secure Messaging in the UC Framework

In this section, we define an ideal functionality $F_{SC}$ for signcryption, based on the security definitions given in the previous section. With applications such as email and instant messaging in mind, we also define an ideal functionality $F_{SM}$ for secure messaging, which arises naturally from the required properties of such applications. We also show that, given a public key infrastructure and a secure signcryption protocol, we can construct a protocol $\pi_{SM}$ that securely realizes $F_{SM}$ in the $(F_{CA}, F_{SC})$-hybrid model, where $F_{CA}$ is an ideal functionality providing a public key infrastructure.

We point out that in the definitions of the ideal functionalities, we only consider the case of static corruption. In other words, when executing a protocol, it is known from the start which parties are corrupt and which are honest.

The ideal certification authority functionality $F_{CA}$ is defined in Figure 2, and is similar to the one given by Canetti in [2]. Next, in Figure 3, the ideal signcryption functionality $F_{SC}$ is defined, and then, in Figure 4, the ideal secure messaging functionality $F_{SM}$ is defined. The protocol $\pi_{SM}$ is described in Figure 5.

The proof of the following result is given in Appendix A.

**Theorem 3.** The protocol $\pi_{SM}$ securely realizes $F_{SM}$ in the $(F_{CA}, F_{SC})$-hybrid model.
Functionality $F_{CA}$

$F_{CA}$ proceeds as follows, with parties $P_1, \ldots, P_n$ and an ideal adversary $S$.

**CA.Register**

Upon receiving the first message ($CA.\text{Register}, sid, v$) from some party $P_i$, send ($CA.\text{Register}, sid, P_i, v$) to $S$. Upon receiving ($Ok, sid, P_i$) from $S$, record the pair $(P_i, v)$, and output ($CA.\text{Registered}, sid, v$) to $P_i$.

**CA.Retrieve**

Upon receiving a message ($CA.\text{Retrieve}, sid, P_i$) from party $P_j$, send ($CA.\text{Retrieve}, sid, P_i, P_j$) to $S$, and wait for an ($Ok, sid, P_i, P_j$) from $S$. Then, if there is a recorded pair $(P_i, v)$ output ($CA.\text{Retrieved}, sid, P_i, v$) to $P_j$. Otherwise output ($CA.\text{Retrieved}, sid, P_i, \bot$) to $P_j$.

**Figure 2.** The ideal certification authority functionality $F_{CA}$.

4. **Securely Realizing $F_{SC}$**

This section contains our main result. The protocol $\pi_{SC}$ given in Figure 6 is constructed in a natural way from the signcryption scheme $SC$. We show that $SC$ is secure if and only if $\pi_{SC}$ securely realizes $F_{SC}$ under static corruption.

The proof of the following theorem is given in Appendix B.

**Theorem 4.** Let $SC$ be a signcryption scheme. If $\pi_{SC}$ securely realizes $F_{SC}$, then $SC$ is secure with respect to both IND-CCA2 and EXT-CMA.

Finally, we prove the following result:

**Theorem 5.** Let $SC$ be a signcryption scheme. If $SC$ is secure with respect to both IND-CCA2 and EXT-CMA, then $\pi_{SC}$ securely realizes $F_{SC}$.

**Proof.** Assume that $SC$ is secure with respect to both IND-CCA2 and EXT-CMA. By Theorem 2, the former is equivalent to being secure with respect to ROR-CCA2. We will show that for every adversary $A$ interacting with parties running $\pi_{SC}$, there is an ideal adversary $S$ such that no environment $Z$ can tell whether it is interacting with $A$ and $\pi_{SC}$ or with $S$ and the ideal protocol for $F_{SC}$, $\text{IDEAL}_{F_{SC}}$.

As usual, $S$ runs a simulated copy of $A$, and forwards all messages from $Z$ to $A$ and back. Since there is no communication between parties in the protocol $\pi_{SC}$, the only way that $S$ affects the output of $Z$ is by communication with $F_{SC}$.

Upon receiving a message from a corrupt party, $F_{SC}$ forwards the message to $S$. $S$ sends this message to $A$ on the correct input tape, and when $A$ replies to this message, $S$ forwards the reply to $F_{SC}$.

**Simulating $SC.\text{KeyGen}$:** Upon receiving the first message ($SC.\text{KeyGen}, sid$) from some party $P_i$, $F_{SC}$ sends ($SC.\text{KeyGen}, sid, P_i$) to $S$. $S$ runs the algorithms $K_s$ and $K_r$. He obtains a sender’s key pair $(sk_{i}^{s}, pk_{i}^{s})$ and a receiver’s key pair $(sk_{i}^{r}, pk_{i}^{r})$, records $(P_i, sk_{i}^{s}, pk_{i}^{s}, sk_{i}^{r}, pk_{i}^{r})$, and sends ($SC.\text{Key}, sid, P_i, (pk_{i}^{s}, pk_{i}^{r})$) to $F_{SC}$. 
Functionality $F_{SC}$

$F_{SC}$ proceeds as follows, with parties $P_1, \ldots, P_n$ and an ideal adversary $S$.

Upon receiving a message from a corrupt party, $F_{SC}$ forwards the message to $S$, and when $S$ replies to this message, $F_{SC}$ forwards the reply to the corrupt party.

**SC.KeyGen**

Upon receiving the first message $(SC.KeyGen, sid)$ from some party $P_i$, send $(SC.KeyGen, sid, P_i)$ to $S$. Upon receiving $(SC.Key, sid, P_i, (pk_i^s, pk_i^r))$ from $S$, output $(SC.Key, sid, (pk_i^s, pk_i^r))$ to $P_i$ and record $(P_i, (pk_i^s, pk_i^r))$.

**SC.Encrypt**

Upon receiving $(SC.Encrypt, sid, pk^r, m)$ from $P_i$, do:

1. If $pk^r = pk_j^r$ for some $j$ and $P_j$ is honest, then send $(SC.Encrypt, sid, pk_i^s, pk_j^r, |m|)$ to $S$. Otherwise send $(SC.Encrypt, sid, pk_i^s, pk^r, m)$ to $S$.

2. Upon receiving $(SC.Ciphertext, sid, pk_i^s, pk^r, c)$ from $S$ such that there is no recorded entry $(pk_i^s, pk^r, m', c)$ for any $m'$, output $(SC.Ciphertext, sid, pk^r, m)$ to $P_i$. If $pk^r = pk_j^r$ for some $j$ and $P_j$ is honest, then record the entry $(pk_i^s, pk_j^r, m, c)$.

**SC.Decrypt**

Upon receiving $(SC.Decrypt, sid, pk^s, c)$ from $P_j$, do:

1. Send $(SC.Decrypt, sid, pk^s, pk_j^r, c)$ to $S$. Upon receiving $(SC.Plaintext, sid, pk^s, pk_j^r, m'/ot, c)$ from $S$, continue.

2. If an entry $(pk^s, pk_j^r, m, c)$ is recorded, then output $(SC.Plaintext, sid, pk^s, m, c)$ to $P_j$.

3. Otherwise, if $pk^s = pk_i^s$ for some $i$ and $P_i$ is honest, then output $(SC.Plaintext, sid, pk^s, \bot, c)$ to $P_j$.

4. Otherwise, output $(SC.Plaintext, sid, pk^s, m'/\bot, c)$ to $P_j$.

**Figur 3.** The ideal signcryption functionality $F_{SC}$.

**Simulating SC.Encrypt:** Assume that $F_{SC}$ receives $(SC.Encrypt, sid, pk^r, m)$ from some party $P_i$. If $pk^r = pk_j^r$ for some $j$, and $P_j$ is honest, then the procedure given in 1 takes place, otherwise the procedure given in 2 takes place.

1. $F_{SC}$ sends $(SC.Encrypt, sid, pk_i^s, pk^r, |m|)$ to $S$. $S$ chooses a random message $m'$ of length $|m|$, and encrypts $m'$ using the signcryption function $S$ with $sk_i^s$ and $pk^r$. He obtains $c$, and checks that there is no recorded entry $(pk_i^s, pk^r, c)$. If there is such an entry, he encrypts $m'$ again, until the above condition holds. He then sends $(SC.Ciphertext, sid, pk_i^s, pk^r, c)$ to $F_{SC}$, and records $(pk_i^s, pk^r, c)$.

2. $F_{SC}$ sends $(SC.Encrypt, sid, pk_i^s, pk^r, m)$ to $S$. $S$ encrypts $m$ using the signcryption function $S$ with $sk_i^s$ and $pk^r$. He obtains $c$, and checks
Functionality $F_{SM}$

$F_{SM}$ proceeds as follows, with parties $P_1, \ldots, P_n$ and an ideal adversary $S$.

Upon receiving a message from a corrupt party, $F_{SM}$ forwards the message to $S$, and when $S$ replies to this message, $F_{SM}$ forwards the reply to the corrupt party.

**SM.Register**

Upon receiving the first message ($SM\.Register, sid$) from some party $P_i$, send ($SM\.Register, sid, P_i$) to $S$. Upon receiving ($Ok, sid, P_i$) from $S$, output ($SM\.Registered, sid$) to $P_i$ and record $P_i$.

**SM.Encrypt**

Upon receiving ($SM\.Encrypt, sid, P_j, m$) from some party $P_i$, do:

1. If there is a recorded entry $P_j$, then continue. Otherwise, output ($SM\.Encrypt\.Error, sid, P_i$ not registered) to $P_i$.
2. If $P_j$ is honest, then send ($SM\.Encrypt, sid, P_i, P_j, |m|$) to $S$. Otherwise, send ($SM\.Encrypt, sid, P_i, P_j, m$) to $S$.
3. Upon receiving ($SM\.Ciphertext, sid, P_i, P_j, c$) from $S$ such that there is no recorded entry ($P_i, P_j, m', c$) for any $m'$, check if there is a recorded entry $P_j$. If there is, then continue. Otherwise, output ($SM\.Encrypt\.Error, sid, P_j$ not registered) to $P_i$.
4. Output ($SM\.Ciphertext, sid, P_j, m, c$) to $P_i$. If $P_j$ is honest, then record the entry ($P_i, P_j, m, c$).

**SM.Decrypt**

Upon receiving ($SM\.Decrypt, sid, P_i, c$) from some party $P_j$, do:

1. If there is a recorded entry $P_j$, then continue. Otherwise, output ($SM\.Decrypt\.Error, sid, P_j$ not registered) to $P_j$.
2. Send ($SM\.Decrypt, sid, P_i, P_j, c$) to $S$.
3. Upon receiving ($SM\.Plaintext, sid, P_i, P_j, m'/\bot, c$) from $S$, check if there is a recorded entry $P_j$. If there is, then continue. Otherwise, output ($SM\.Decrypt\.Error, sid, P_i$ not registered) to $P_j$.
4. If an entry ($P_i, P_j, m, c$) is recorded, then output ($SM\.Plaintext, sid, P_i, m, c$) to $P_j$.
5. Otherwise, if $P_i$ is honest, then output ($SM\.Plaintext, sid, P_i, \bot, c$) to $P_j$.
6. Otherwise, output ($SM\.Plaintext, sid, P_i, m'/\bot, c$) to $P_j$.

**Figure 4.** The ideal secure messaging functionality $F_{SM}$. 

...that there is no recorded entry ($pk_i^s, pk^r, c$). If there is such an entry, he encrypts $m$ again, until the above condition holds. He then sends ($SC\.Ciphertext, sid, pk_i^s, pk^r, c$) to $F_{SC}$. 

Protocol $\pi_{SM}$

$\pi_{SM}$ proceeds as follows, with parties $P_1, \ldots, P_n$ and an adversary $A$.

SM.Register

1. Upon the first input $(\text{SM.Register, } sid)$, $P_i$ sends $(\text{SC.KeyGen, } sid)$ to $\mathcal{F}_{SC}$.
2. Upon receiving $(\text{SC.Key, } sid, (pk_i^s, pk_i^r))$ from $\mathcal{F}_{SC}$, $P_i$ sends $(\text{CA.Register, } sid, (pk_i^s, pk_i^r))$ to $\mathcal{F}_{CA}$.
3. Upon receiving $(\text{CA.Registered, } sid, (pk_i^s, pk_i^r))$ from $\mathcal{F}_{CA}$, $P_i$ outputs $(\text{SM.Registered, } sid)$.

SM.Encrypt

1. Upon input $(\text{SM.Encrypt, } sid, P_j, m)$, $P_i$ checks if he has received $(\text{CA.Registered, } sid, (pk_i^s, pk_i^r))$. If he has, then he sends $(\text{CA.Retrieve, } sid, P_j)$ to $\mathcal{F}_{CA}$. Otherwise, he outputs $(\text{SM.Encrypt.Error, } sid, P_i \text{ not registered})$.
2. If $P_i$ receives $(\text{CA.Retrieved, } sid, P_j, (pk_j^s, pk_j^r))$ from $\mathcal{F}_{CA}$, then $P_i$ sends $(\text{SC.Encrypt, } sid, pk_j^s, m)$ to $\mathcal{F}_{SC}$. Otherwise, if $P_i$ receives $(\text{CA.Retrieved, } sid, P_j, \bot)$ from $\mathcal{F}_{CA}$, then $P_i$ outputs $(\text{SM.Encrypt.Error, } sid, P_j \text{ not registered})$.
3. Upon receiving $(\text{SC.Ciphertext, } sid, pk_j^r, m, c)$ from $\mathcal{F}_{SC}$, $P_i$ outputs $(\text{SM.Ciphertext, } sid, P_j, m, c)$.

SM.Decrypt

1. Upon input $(\text{SM.Decrypt, } sid, P_i, c)$, $P_j$ checks if he has received $(\text{CA.Registered, } sid, (pk_j^s, pk_j^r))$. If he has, then he sends $(\text{CA.Retrieve, } sid, P_i)$ to $\mathcal{F}_{CA}$. Otherwise, he outputs $(\text{SM.Decrypt.Error, } sid, P_j \text{ not registered})$.
2. If $P_j$ receives $(\text{CA.Retrieved, } sid, P_i, (pk_j^s, pk_j^r))$ from $\mathcal{F}_{CA}$, then $P_j$ sends $(\text{SC.Decrypt, } sid, pk_j^r, c)$ to $\mathcal{F}_{SC}$. Otherwise, if $P_j$ receives $(\text{CA.Retrieved, } sid, P_i, \bot)$ from $\mathcal{F}_{CA}$, then $P_j$ outputs $(\text{SM.Decrypt.Error, } sid, P_i \text{ not registered})$.
3. Upon receiving $(\text{SC.Plaintext, } sid, pk_j^s, pk_j^r, m', c)$ from $\mathcal{F}_{SC}$, $P_j$ outputs $(\text{SM.Plaintext, } sid, P_i, m'/\bot, c)$.

Simulating SC.Decrypt: Upon receiving $(\text{SC.Decrypt, } sid, pk^s, c)$ from some party $P_j$, $\mathcal{F}_{SC}$ sends $(\text{SC.Decrypt, } sid, pk^s, pk_j^r, c)$ to $\mathcal{S}$. If $pk^s = pk_i^s$ for some $i$, and $P_i$ is honest, then the procedure given in 1 takes place, otherwise the procedure given in 2 takes place.

1. $\mathcal{S}$ chooses an arbitrary message $m'$ and sends $(\text{SC.Plaintext, } sid, pk^s, pk_j^r, m', c)$ to $\mathcal{F}_{SC}$.
2. $\mathcal{S}$ decrypts $c$ using the unsigncryption function $U$ with $pk^s$ and $sk_j^r$, obtains $m'/\bot$ and sends $(\text{SC.Plaintext, } sid, pk^s, pk_j^r, m'/\bot, c)$ to $\mathcal{F}_{SC}$.
Protocol $\pi_{SC}$

$\pi_{SC}$ proceeds as follows, with parties $P_1, \ldots, P_n$ and an adversary $A$.

**SC.KeyGen**

Upon the first input ($SC.KeyGen, \text{sid}$), $P_i$ runs the algorithms $K_s$ and $K_r$. He obtains a sender’s key pair $(sk_i^s, pk_i^s)$ and a receiver’s key pair $(sk_i^r, pk_i^r)$, and outputs ($SC.Key, \text{sid}, (pk_i^s, pk_i^r)$).

**SC.Encrypt**

Upon input ($SC.Encrypt, \text{sid}, pk_r^r, m$), $P_i$ obtains $c = S(sk_i^s, pk_r^r, m)$ and outputs ($SC.Ciphertext, \text{sid}, pk_r^r, m, c$).

**SC.Decrypt**

Upon input ($SC.Decrypt, \text{sid}, pk_s^s, c$), $P_i$ obtains $m/\bot = U(pk_s^s, sk_r^r, c)$ and outputs ($SC.Plaintext, \text{sid}, pk_s^s, m/\bot, c$).

---

**Figure 6.** The signcryption protocol $\pi_{SC}$.

It is clear that $S$ simulates $SC.KeyGen$ perfectly.

As for $SC.Encrypt$, if $P_i$ and $P_j$ are honest, and $pk_r^r = pk_j^r$ for some $j$, we observe that in $\pi_{SC}$, the real message $m$ is encrypted, while in $\text{IDEAL}_{F_{SC}}$, a random message $m'$ is encrypted. Moreover, in $\text{IDEAL}_{F_{SC}}$, for a fixed key pair $(pk_s^s, pk_r^r)$, a message will never be encrypted into the same ciphertext twice. In $\pi_{SC}$ this can be shown to happen with negligible probability.

As for $SC.Decrypt$, we observe that the simulation is perfect if $P_i$ or $P_j$ is corrupt, or $pk_s^s \neq pk_i^s$ for all $i$. Otherwise, the following separates $\pi_{SC}$ from $\text{IDEAL}_{F_{SC}}$: In $\pi_{SC}$, the decryption of a ciphertext $c$ is determined by the unsigncryption function $U$, while in $\text{IDEAL}_{F_{SC}}$, the decryption of $c$ is determined according to whether there is a recorded tuple $(pk_s^s, pk_r^r, m, c)$. If there is, then $m$ is returned, otherwise the symbol $\bot$ is returned. This implies that if $P_i$ has encrypted a message to $P_j$ and obtained $c$ at an earlier stage, then the corresponding messages will be output both in $\text{IDEAL}_{F_{SC}}$ and in $\pi_{SC}$. Otherwise, there are two scenarios to consider: If $c$ is an invalid encryption according to the unsigncryption function $U$, then the output will be $\bot$ both in $\text{IDEAL}_{F_{SC}}$ and $\pi_{SC}$. However, if $c$ is valid, the decryption of $c$ according to $U$ will be output in $\pi_{SC}$, while $\bot$ will be output in $\text{IDEAL}_{F_{SC}}$.

Throughout the proof, we assume without loss of generality that the parties $P_1, \ldots, P_t$ are honest, while the remaining parties $P_{t+1}, \ldots, P_n$ are corrupt.

We want to show that if there exists an environment $Z$ able to distinguish $\text{IDEAL}_{F_{SC}}$ and $\pi_{SC}$ with non-negligible probability, then we can construct an adversary $A^{\text{tor}}$ breaking the ROR-CCA2 security or an adversary $A^{\text{ext}}$ breaking the EXT-CMA security of our signcryption scheme. To this end, we will consider a series of games, where we gradually modify the behavior of the honest parties.

Game 0: In this game, $Z$ interacts with the protocol $\pi_{SC}$ running with parties $P_1, \ldots, P_n$. 
Game 1: This game is the same as Game 0, with the following modifications: The honest parties \( P_1, \ldots, P_t \) are now simulated by \( P \). When receiving messages on the input tape corresponding to an honest party \( P_i \), \( P \) runs the algorithms \( K_s, K_r, S \) and \( U \), and outputs messages to \( Z \), exactly as \( P_i \) would do in the protocol \( \pi_{SC} \). \( P \) records \( (P_i, sk_i^s, pk_i^r, sk_i^r, pk_i^t) \) to keep track of which keys belong to which party. Moreover, every time a message \( m \) from some honest party \( P_i \) to another honest party \( P_j \) is encrypted into some ciphertext \( c \), \( P \) records \( (pk_i^s, pk_j^r, m, c) \).

From \( Z \)'s point of view, there is no difference between Game 0 and Game 1. Therefore, if we let \( G_i \) denote the output of \( Z \) when taking part in Game \( i \), we have

\[
\left| \Pr[G_0 = 1] - \Pr[G_1 = 1] \right| = 0.
\]

Game 2: This game is the same as Game 1, with the following modification: When receiving messages \( (SC.\text{Decrypt}, sid, pk_i^s, c) \) on the input tape corresponding to \( P_j \), where both \( P_i \) and \( P_j \) are honest parties, \( P \) responds as follows: If an entry \( (pk_i^s, pk_j^r, m, c) \) is recorded, \( P \) outputs \( (SC.\text{Plaintext}, sid, pk_i^s, m, c) \). Otherwise, \( P \) outputs \( (SC.\text{Plaintext}, sid, pk_i^s, \perp, c) \).

The proof of the following lemma is given in Appendix C.

**Lemma 6.** There is an adversary \( A^{ext} \) such that

\[
\left| \Pr[G_1 = 1] - \Pr[G_2 = 1] \right| \leq t(t-1) \text{Succ}_{SC,A^{ext}}^{ext-cma}(\tau).
\]

Game 3: This game is the same as Game 2, with the following modification: When \( P \) receives a message \( (SC.\text{Encrypt}, sid, pk_i^r, m) \) on the input tape corresponding to \( P_1 \), \( P \) chooses a random message \( m' \) of length \( |m| \), obtains \( c = S(sk_1^s, pk_2^r, m') \), outputs \( (SC.\text{Ciphertext}, sid, pk_i^r, m, c) \) on the output tape corresponding to \( P_1 \), and records \( (pk_i^s, pk_j^r, m, c) \).

For each of the games Game 4, . . . , Game \( t(t-1) + 2 \), \( P \) encrypts random messages for an additional pair of senders and receivers. Consequently, in Game \( t(t-1) + 2 \), \( P \) always encrypts random messages for honest parties.

The proof of the following lemma is given in Appendix D.

**Lemma 7.** For all \( i \) such that \( 2 \leq i \leq t(t-1) + 1 \), there is an adversary \( A^{ror} \) such that

\[
\left| \Pr[G_i = 1] - \Pr[G_{i+1} = 1] \right| = \text{Adv}_{SC,A^{ror}}^{ror-cca2}(\tau).
\]

Game \( t(t-1) + 3 \): In this game, \( Z \) interacts with the ideal protocol IDEAL_{SC} running with parties \( P_1, \ldots, P_n \).

From \( Z \)'s point of view, there is only one difference between Game \( t(t-1) + 2 \) and Game \( t(t-1) + 3 \): In the latter, for a fixed key pair \( (pk^s, pk^r) \), the same ciphertext will never be output twice. In the former, this can be shown to happen with negligible probability. This follows from the next lemma, whose proof is given in Appendix E.
Lemma 8. Let \((sk^s, pk^s) \leftarrow K_s, (sk^r, pk^r) \leftarrow K_r\), and let \(m\) be a randomly chosen message. Then there is an adversary \(A_{ror}\) such that
\[
Pr[c = c'|c, c' \leftarrow S(sk^s, pk^r, m)] \leq 2 \text{Adv}^{ror-cca2}_{SC,A_{ror}}(\tau).
\]

Hence we have
\[
\left|Pr[G_{t(t-1)+2} = 1] - Pr[G_{t(t-1)+3} = 1]\right| = \delta(\tau),
\]
for some negligible function \(\delta(\tau)\). Finally, the triangle inequality gives
\[
\left|Pr[G_0 = 1] - Pr[G_{t(t-1)+3} = 1]\right| \leq \left|Pr[G_0 = 1] - Pr[G_1 = 1]\right| + \left|Pr[G_1 = 1] - Pr[G_2 = 1]\right| + \cdots + \left|Pr[G_{t(t-1)+2} = 1] - Pr[G_{t(t-1)+3} = 1]\right|
\]
\[
= t(t-1)\text{Succ}_{SC,A_{ext}}^{ext-cma}(\tau) + t(t-1)\text{Adv}^{ror-cca2}_{SC,A_{ror}}(\tau) + \delta(\tau).
\]

Since \(t \leq n\) is polynomially bounded in \(\tau\), the expression on the right-hand side is negligible in \(\tau\), and we conclude that no environment \(Z\) can distinguish interaction with \(\text{IDEAL}_{F_{SC}}\) and \(S\) from interaction with \(\pi_{SC}\) and \(A\). This completes the proof.

5. Concluding Remarks

We have proposed ideal functionalities for signcryption and secure messaging, and described a protocol \(\pi_{SM}\) that securely realizes \(F_{SM}\) in the \((F_{CA}, F_{SC})\)-hybrid model. In addition, we have proved that for signcryption schemes, security with respect to IND-CCA2 and EXT-CMA is necessary and sufficient for securely realizing \(F_{SC}\). This provides some evidence that IND-CCA2 and EXT-CMA are the correct security notions for signcryption.

Since non-repudiation is not always required, we have not incorporated it in \(F_{SC}\) due to space limitations.

We have proved our results only with static corruption, since it seems impossible to do better. However, it is conceivable that some kind of non-committing encryption can be used to get security against adaptive adversaries. Since \(F_{SC}\) can be securely realized using ideal functionalities for public-key encryption and digital signatures, it may be possible to replace the encryption functionality (providing security against static corruption) with an ideal functionality for non-committing encryption (providing security against adaptive corruption) [4], and realize a functionality \(F'_{SC}\) with adaptive corruption.

References


We show that for every adversary $A$ interacting with parties running $\pi_{SM}$ in the $(\mathcal{F}_{CA}, \mathcal{F}_{SC})$-hybrid model, there is an ideal adversary $S$ such that no environment $Z$ can tell whether it is interacting with $A$ and $\pi_{SM}$ or with $S$ and IDEAL$_{FSM}$.

$S$ runs a simulated copy of $A$, and forwards all messages from $Z$ to $A$ and back. Due to space limitations, we do not describe the behavior of $S$ upon corruption of parties.

**Simulating SM.Register:** Upon the first input $(\text{SM.Register}, sid)$ from some party $P_i$, $F_{SM}$ sends $(\text{SM.Register}, sid, P_i)$ to $S$. $S$ sends $(\text{SC.KeyGen}, sid, P_i)$ to $A$ in the name of $F_{SC}$. Upon receiving $(\text{SC.Key}, sid, P_i, (pk_i^s, pk_i^r))$ from $A$, $S$ sends $(\text{CA.Register}, sid, P_i, (pk_i^s, pk_i^r))$ to $A$ in the name of $F_{CA}$, and waits for an $(\text{Ok}, sid, P_i)$ from $A$. $S$ records the entry $(P_i, (pk_i^s, pk_i^r))$, and sends $(\text{Ok}, sid, P_i)$ to $F_{SM}$.

**Simulating SM.Encrypt:** Upon input $(\text{SM.Encrypt}, sid, P_j, m)$ from some party $P_i$, $F_{SM}$ checks if there is a recorded entry $P_i$. If there is no such entry, $F_{SM}$ outputs $(\text{SM.Encrypt.Error}, sid, P_i$ not registered) to $P_i$. Otherwise, the following procedure takes place:

$F_{SM}$ sends $(\text{SM.Encrypt}, sid, P_i, P_j, |m|)$ to $S$. $S$ sends $(\text{CA.Retrieve}, sid, P_j, P_i)$ to $A$ in the name of $F_{CA}$, and waits for an $(\text{Ok}, sid, P_j, P_i)$ from $A$. $S$ checks if there is a recorded entry $(P_j, (pk_j^s, pk_j^r))$. If there is no such entry, he randomly chooses a key $pk_j^r$ and a message $m'$ of length $|m|$, obtains $c = S(sk_i^s, pk_j^r, m')$, and sends $(\text{SM.Ciphertext}, sid, P_i, P_j, c)$ to $F_{SM}$. Otherwise, the following procedure takes place:

$S$ sends $(\text{SC.Ciphertext}, sid, P_i, P_j, c)$ to $A$ in the name of $F_{SC}$. Upon receiving $(\text{SC.Ciphertext}, sid, P_i, P_j, c)$ from $A$, $S$ sends $(\text{SM.Ciphertext}, sid, P_i, P_j, c)$ to $F_{SM}$.

**Simulating SM.Decrypt:** Upon input $(\text{SM.Decrypt}, sid, P_i, c)$ from some party $P_j$, $F_{SM}$ checks if there is a recorded entry $P_j$. If there is no such entry, $F_{SM}$ outputs $(\text{SM.Decrypt.Error}, sid, P_j$ not registered) to $P_j$. Otherwise, the following procedure takes place:

$F_{SM}$ sends $(\text{SM.Decrypt}, sid, P_i, P_j, c)$ to $S$. $S$ sends $(\text{CA.Retrieve}, sid, P_i, P_j)$ to $A$ in the name of $F_{CA}$, and waits for an $(\text{Ok}, sid, P_i, P_j)$ from $A$. $S$ checks
UNIVERSALLY COMPOSABLE SIGNCRYPTION 13

if there is a recorded entry \((P_i, (pk_i^s, pk_j^r))\). If there is no such entry, he sends \((\text{SM}.\text{Plaintext}, sid, P_i, P_j, \bot)\) to \(\mathcal{F}_{\text{SM}}\). Otherwise, the following procedure takes place:

\(S\) sends \((\text{SC}.\text{Decrypt}, sid, pk_i^s, pk_j^r, c)\) to \(A\) in the name of \(\mathcal{F}_{\text{SC}}\). Upon receiving \((\text{SC}.\text{Plaintext}, sid, pk_i^s, pk_j^r, m', \bot, c)\) from \(A\), \(S\) sends \((\text{SM}.\text{Plaintext}, sid, P_i, P_j, m'/ \bot, c)\) to \(\mathcal{F}_{\text{SM}}\).

Now assume that IDEAL\(_{\mathcal{F}_{\text{SM}}}\) runs with the same instance of \(A\) as \(\pi_{\text{SM}}\). It is clear from the simulation of SM.Encrypt that an entry \((P_i, P_j, m, c)\) is recorded by \(\mathcal{F}_{\text{SM}}\) if and only if an entry \((pk_i^s, pk_j^r, m, c)\) is recorded by \(\mathcal{F}_{\text{SC}}\). Moreover, the condition "\(pk^s_i = pk^r_j\) for some \(i\)" in SC.Decrypt, is automatically satisfied by \(\pi_{\text{SM}}\). So SM.Decrypt behaves exactly as SC.Decrypt in the protocol, and consequently the output when running with IDEAL\(_{\mathcal{F}_{\text{SM}}}\) will be the same as when running with \(\pi_{\text{SM}}\). This completes the proof.

APPENDIX B. PROOF OF THEOREM 4

We first prove that if \(\pi_{\text{SC}}\) securely realizes \(\mathcal{F}_{\text{SC}}\), then \(\text{SC}\) is secure with respect to ROR-CCA2. We assume that there exists an adversary \(A^{\text{ror}}\) breaking the ROR-CCA2 security of \(\text{SC}\), i.e.

\[
\text{Adv}_{\text{SC}, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = \Pr[\text{Exp}_{\text{SC}, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = 1|b = 1] - \Pr[\text{Exp}_{\text{SC}, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = 1|b = 0] = \delta(\tau),
\]

for some non-negligible function \(\delta(\tau)\). We construct an environment \(Z\) and an adversary \(A\) such that \(Z\) can distinguish interaction with \(\pi_{\text{SC}}\) from interaction with IDEAL\(_{\mathcal{F}_{\text{SC}}}\).

No parties are corrupted, and \(A\) is never activated by \(Z\). Since there is no communication between parties, \(A\) never gets any input.

\(Z\) proceeds as follows:

1. \(Z\) chooses two parties \(P_i\) and \(P_j\), sends the message \((\text{SC}.\text{KeyGen}, sid)\) to each of them, and obtains key pairs \((pk_i^s, pk_i^r)\) and \((pk_j^s, pk_j^r)\).

2. \(Z\) chooses \(b \in \{0, 1\}\) at random and runs \(A^{\text{ror}}\) with input \((pk_i^s, pk_j^r)\). He answers \(A^{\text{ror}}\)'s queries to \(O_S/O_U/O_b\) by sending appropriate messages to \(P_i/P_j/P_i\). In particular, if \(b = 0\), \(Z\) chooses a random message of appropriate length and asks \(P_i\) to encrypt this message. If \(b = 1\), \(Z\) asks \(P_i\) to encrypt the message in \(A^{\text{ror}}\)'s query.

3. Finally, when \(A^{\text{ror}}\) outputs \(b'\), \(Z\) outputs \(1\) if \(b' = b\) and \(0\) otherwise.

We observe that when \(Z\) interacts with \(\pi_{\text{SC}}, A^{\text{ror}}\)'s environment in \(\text{Exp}_{\text{SC}, A^{\text{ror}}}^{\text{ror-cca2}}(\tau)\) is perfectly simulated.

When \(Z\) interacts with IDEAL\(_{\mathcal{F}_{\text{SC}}}\), \(A^{\text{ror}}\) does not run in its expected environment, but whatever happens, \(A^{\text{ror}}\) gets no information about \(b\), and \(Z\) outputs \(1\) with probability \(\frac{1}{2}\).
Let \( Z_{\pi_{\mathsf{SC}}} \) and \( Z_{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SC}}}} \) denote \( Z \)'s output when interacting with \( \pi_{\mathsf{SC}} \) and \( \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SC}}} \), respectively. The above argument gives

\[
\left| \Pr[Z_{\pi_{\mathsf{SC}}} = 1] - \Pr[Z_{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SC}}}} = 1] \right| = \left| \Pr[Z_{\pi_{\mathsf{SC}}} = 1 | b = 1] \cdot \Pr[b = 1] + \Pr[Z_{\pi_{\mathsf{SC}}} = 1 | b = 0] \cdot \Pr[b = 0] - \frac{1}{2} \right|
\]

\[
= \left| \frac{1}{2} (\Pr[Z_{\pi_{\mathsf{SC}}} = 1 | b = 1] + (1 - \Pr[Z_{\pi_{\mathsf{SC}}} = 0 | b = 0]) - 1) \right|
\]

\[
= \frac{1}{2} \left| (\Pr[Z_{\pi_{\mathsf{SC}}} = 1 | b = 1] - \Pr[Z_{\pi_{\mathsf{SC}}} = 0 | b = 0]) \right|
\]

\[
= \frac{1}{2} \delta(\tau).
\]

Hence we have constructed an environment with non-negligible probability of distinguishing interaction with \( \pi_{\mathsf{SC}} \) from interaction with \( \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SC}}} \).

It remains to show that if \( \pi_{\mathsf{SC}} \) securely realizes \( \mathcal{F}_{\mathsf{SC}} \), then \( \mathsf{SC} \) is secure with respect to \( \mathsf{EXT-CMA} \). We assume that there exists an adversary \( A^{\text{ext}} \) breaking the \( \mathsf{EXT-CMA} \) security of \( \mathsf{SC} \), i.e.

\[
\text{Succ}_{\mathsf{SC}, A}^{\text{ext-cma}}(\tau) = \Pr[\text{Exp}_{\mathsf{SC}, A}^{\text{ext-cma}}(\tau) = 1] = \delta(\tau),
\]

for some non-negligible function \( \delta(\tau) \). We construct an environment \( Z \) and an adversary \( A \) such that \( Z \) can distinguish interaction with \( \pi_{\mathsf{SC}} \) from interaction with \( \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SC}}} \).

No parties are corrupted, and \( A \) is never activated by \( Z \). Since there is no communication between parties, this means that \( A \) never gets any input.

\( Z \) proceeds as follows:

1. \( Z \) chooses two parties \( P_i \) and \( P_j \), sends the message \((\mathsf{SC}.\text{KeyGen}, \text{sid})\) to each of them, and obtains key pairs \((pk_s^i, pk_r^i)\) and \((pk_s^j, pk_r^j)\).
2. \( Z \) runs \( A^{\text{ext}} \) with input \((pk_s^i, pk_r^j)\). He answers \( A^{\text{ext}} \)'s queries to \( \mathcal{O}_S \) and \( \mathcal{O}_U \) by sending appropriate messages to \( P_i \) and \( P_j \), respectively.
3. Finally, when \( A^{\text{ext}} \) outputs \( c \), \( Z \) sends \((\mathsf{SC}.\text{Decrypt}, \text{sid}, pk_s^i, c)\) to \( P_j \).
   
   If \( P_j \) outputs a message \( m \), \( Z \) outputs 1. Otherwise, if \( P_j \) outputs the symbol \( \bot \), \( Z \) outputs 0.

We observe that when \( Z \) interacts with \( \pi_{\mathsf{SC}} \), \( A^{\text{ext}} \)'s environment in \( \text{Exp}_{\mathsf{SC}, A}^{\text{ext-cma}}(\tau) \) is perfectly simulated. When \( Z \) interacts with \( \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SC}}} \), then \( P_j \), when asked to decrypt \( c \) in the last step, will output \( \bot \) even if \( c \) is a valid ciphertext. Hence \( Z \) always outputs 0 in this case. So we get

\[
\left| \Pr[Z_{\pi_{\mathsf{SC}}} = 1] - \Pr[Z_{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SC}}}} = 1] \right| = \left| \Pr[\text{Exp}_{\mathsf{SC}, A}^{\text{ext-cma}}(\tau) = 1] - 0 \right|
\]

\[
= \delta(\tau).
\]

This completes the proof.
APPENDIX C. PROOF OF LEMMA 6

We define the event $F$ in a game as follows:

$F$: At some point during the game, some honest party $P_j$ is asked to decrypt a valid ciphertext $c$ from some honest party $P_i$, but $c$ was not a result of some query $(\text{SC.Encrypt}, \text{sid}, pk^j_r, m)$ sent to $P_i$.

We observe that unless $F$ occurs, Game 1 and Game 2 proceed identically, hence we have

$$\left| \Pr[G_1 = 1] - \Pr[G_2 = 1] \right| \leq \Pr[F].$$

We now construct an adversary $A^{\text{ext}}$ trying to break the EXT-CMA security of our signcryption scheme. $A^{\text{ext}}$ has been given a target sender’s public key $pk^s$ and a target receiver’s public key $pk^r$, and attempts to produce a ciphertext $c$ such that $U(pk^s, sk^r, c) \neq \perp$. $A^{\text{ext}}$ runs simulated copies of $Z$, $A$ and the corrupt parties. $A^{\text{ext}}$ also simulates $P$, and behaves exactly as $P$, with the following exceptions: $A^{\text{ext}}$ chooses random numbers $k, l \in \{1, \ldots, t\}$ such that $k \neq l$, and lets $P_k$ and $P_l$ be its target sender and target receiver, respectively. Thus, when $P_k$ is asked to generate keys, $A^{\text{ext}}$ lets $pk^*_k = pk^s$, runs $K_r$ to obtain $(sk^r_k, pk^r_k)$, outputs $(pk^*_k, pk^r_k)$ and records $(P_k, \ldots, pk^*_k, sk^r_k, pk^r_k)$. When $P_l$ is asked to generate keys, a corresponding procedure takes place, and $A^{\text{ext}}$ records $(P_l, sk^r_l, pk^r_l, \ldots, pk^r_l)$. When receiving a message $(\text{SC.Encrypt}, \text{sid}, pk^j_r, m)$ on the input tape corresponding to $P_k$, $A^{\text{ext}}$ answers using its flexible signcryption oracle. Similarly, when receiving a message $(\text{SC.Decrypt}, \text{sid}, pk^j_r, c)$ on the input tape corresponding to $P_l$, $A^{\text{ext}}$ answers using its flexible unsigncryption oracle.

If $F$ occurs, then $Z$ has produced a forgery, which can be detected by $A^{\text{ext}}$. In particular, if $F$ involves the sender $P_k$ and the receiver $P_l$, then $A^{\text{ext}}$ has obtained a forgery against its target sender and target receiver. If this happens, $A^{\text{ext}}$ outputs the forged ciphertext and halts. Since the numbers $k, l$ are chosen at random, we get

$$\text{Succ}_{\text{SC}, A^{\text{ext}}}^{\text{ext-cma}}(\tau) = \frac{\Pr[F]}{t(t-1)}.$$

Hence we have

$$\left| \Pr[G_1 = 1] - \Pr[G_2 = 1] \right| \leq t(t-1)\text{Succ}_{\text{SC}, A^{\text{ext}}}^{\text{ext-cma}}(\tau),$$

and the proof is complete.

APPENDIX D. PROOF OF LEMMA 7

Using Game 2 and Game 3, we will construct an adversary $A^{\text{ror}}$ trying to break the ROR-CCA2 security of our signcryption scheme. $A^{\text{ror}}$ has been given a target sender’s public key $pk^s$ and a target receiver’s public key $pk^r$. $A^{\text{ror}}$ runs simulated copies of $Z$, $A$ and the corrupt parties. $A^{\text{ror}}$ also simulates $P$, and behaves exactly as $P$, with the following exceptions: $A^{\text{ror}}$ lets $P_1$ be the target sender and $P_2$ the target receiver. Thus, when $P_1$ is asked to generate keys, $A^{\text{ror}}$ lets $pk^*_1 = pk^s$, ...
runs \( K_r \) to obtain \((sk^r_1, pk^r_1)\), outputs \((pk^s_1, pk^r_1)\) and records \((P_1, \cdot, pk^s_1, \cdots)\). Correspondingly, when \( P_2 \) is asked to generate keys, \( A^{\text{ror}} \) lets \( pk^s_2 = pk^r \), runs \( K_s \) to obtain \((sk^s_2, pk^s_2)\), outputs \((pk^s_2, pk^r_2)\) and records \((P_2, sk^s_2, pk^s_2, \cdots)\). When receiving a message \((\text{SC.Encrypt}, sid, pk^s_2, m)\) on the input tape corresponding to \( P_1 \), \( A^{\text{ror}} \) answers using its real-or-random oracle. Moreover, when receiving a message \((\text{SC.Encrypt}, sid, pk^s_2, m)\) on the same input tape, and \( j \neq 2 \), \( A^{\text{ror}} \) answers using its flexible signcryption oracle. In addition, when receiving a message \((\text{SC.Decrypt}, sid, pk^s_2, c)\) on the input tape corresponding to \( P_2 \), and \( P_2 \) is a corrupt party, \( A^{\text{ror}} \) answers using its flexible unsigncryption oracle. Finally, \( A^{\text{ror}} \) copies \( Z \)'s output to its own output tape.

We observe that when the real-or-random oracle encrypts the real message, \( A^{\text{ror}} \) simulates \( P \)'s behavior in Game 2 perfectly. On the other hand, when the oracle encrypts a random message, \( P \)'s behavior in Game 3 is perfectly simulated.

We compute the advantage of \( A^{\text{ror}} \) as follows:

\[
\text{Adv}_{SC, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = \left| \Pr[\text{Exp}_{SC, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = 1|b = 1] - \Pr[\text{Exp}_{SC, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = 1|b = 0] \right|
\]

\[
= \left| \Pr[G_2 = 1] - \Pr[G_3 = 1] \right|.
\]

A similar argument can be applied to any pair of games Game \( i \) and Game \( i+1 \), where \( 2 \leq i \leq t(t-1) + 1 \). Consequently, for all such \( i \) we can construct an adversary \( A^{\text{ror}} \) such that

\[
\left| \Pr[G_i = 1] - \Pr[G_{i+1} = 1] \right| = \text{Adv}_{SC, A^{\text{ror}}}^{\text{ror-cca2}}(\tau),
\]

and the proof is complete.

**Appendix E. Proof of Lemma 8**

Assume that, for \((sk^s, pk^s) \leftarrow K_s\), \((sk^r, pk^r) \leftarrow K_r\), and a randomly chosen message \( m \),

\[
\Pr[c = c'|c, c' \leftarrow S(sk^s, pk^r, m)] = \delta(\tau).
\]

We will construct an adversary \( A^{\text{ror}} \) trying to break the ROR-CCA2 security of our signcryption scheme. Upon input \((pk^s, pk^r)\), \( A^{\text{ror}} \) chooses a random message \( m \), queries its real-or-random oracle \( \mathcal{O}^b_{\text{ror}} \) with this message twice, and receives two ciphertexts \( c \) and \( c' \). If \( c = c' \), \( A^{\text{ror}} \) outputs 1, otherwise \( A^{\text{ror}} \) outputs 0. Assume that \( m \) has length \( l \). The advantage of \( A^{\text{ror}} \) is then given by

\[
\text{Adv}_{SC, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = \left| \Pr[\text{Exp}_{SC, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = 1|b = 1] - \Pr[\text{Exp}_{SC, A^{\text{ror}}}^{\text{ror-cca2}}(\tau) = 1|b = 0] \right|
\]

\[
= \left| \delta(\tau) - \frac{1}{2^l} \delta(\tau) \right|
\]

\[
= \frac{2^l - 1}{2^l} \delta(\tau)
\]

\[
\geq \frac{1}{2} \delta(\tau).
\]
So we have
\[ \Pr[c = c'|c, c' \leftarrow S(sk^s, pk^r, m)] \leq 2\text{Adv}_S^{\text{top-cca2}}(\tau), \]
which completes the proof.
Paper II

Universally Composable Blind Signatures in the Plain Model
Aslak Bakke Buan, Kristian Gjøsteen and Lillian Kråkmo
Preprint
UNIVERSALLY COMPOSABLE BLIND SIGNATURES IN THE
PLAIN MODEL

ASLAK BAKKE BUAN, KRISTIAN GJÖSTEEN, LILLIAN KRÅKMO

ABSTRACT. In the framework of universally composable security, we define an ideal functionality for blind signatures, as an alternative to a functionality recently proposed by Fischlin. Fischlin proves that his functionality cannot be realized in the plain model, but this result does not apply to our functionality. We show that our functionality is realized in the plain model by a blind signature protocol if and only if the corresponding blind signature scheme is secure with respect to blindness and non-forgeability, as defined by Juels, Luby and Ostrovsky.

1. INTRODUCTION

The idea of blind signatures was proposed by Chaum [4] as a key ingredient for anonymous electronic cash applications. Blind signatures allow a bank to issue signatures without seeing the content of the signed documents, and at the same time prevent users from forging signatures. Pointcheval and Stern [7] first defined non-forgeability for blind signature schemes. Juels, Luby and Ostrovsky (JLO) [6] further formalized the concept by giving a general definition of a blind signature scheme and formulating the required security properties: blindness and non-forgeability. Informally, a scheme has blindness if it is infeasible for a malicious signer to determine the order of which two messages are signed by interaction with an honest user. A scheme has non-forgeability if, given \( l \) interactions with an honest signer, it is infeasible for a malicious user to produce more than \( l \) valid signatures.

Universally composable (UC) security is a framework proposed by Canetti [2] as a way to define security for protocols such that security-preserving composition is possible. This allows for a modular design and analysis of protocols. For each cryptographic task, an ideal functionality can be defined, which incorporates the required properties of a protocol for the task and the allowed actions of an adversary. A protocol is said to securely realize the ideal functionality if, loosely speaking, any effect caused by an adversary attacking the protocol can be obtained by an adversary attacking the ideal functionality. When designing complex protocols, one can allow the involved parties to have secure access to ideal functionalities. Then, when implementing the protocol, each ideal functionality is replaced by a protocol securely realizing the functionality. The composition
We refer to \cite{ref1} for a complete overview of this framework.

Since UC security is a powerful and useful notion, an interesting question is how it relates to conventional security notions. Fischlin \cite{ref2} addresses this question in the context of blind signatures. The author defines an ideal functionality for blind signatures, $F_{BlSig}$, and shows that blind signature schemes realizing $F_{BlSig}$ in the plain model do not exist. He does this by showing that $F_{BlSig}$ can be used to realize the functionality $F_{com}$ for commitment schemes, and then applying a well-known impossibility result \cite{ref3}. To realize $F_{BlSig}$, Fischlin has to work in the common reference string model.

One somewhat awkward artefact of Fischlin’s functionality is that any realizing protocol must encode the entire message to be signed into the first protocol message. This restricts any realizing protocol to a maximal message length, otherwise blindness would be violated. We could extend this by signing a hash of the message instead of the message itself. If we use a collision resistant hash function, this clearly does not degrade the real security of any such scheme. Unfortunately, the modified protocol no longer realizes the functionality. This can be considered an artefact of the UC framework, not of the specific functionality. Nonetheless, it is undesirable and we would like to allow blind signature protocols that do not encode the entire message to be signed into the first protocol message.

To achieve this, we propose a new blind signature functionality. The main change is that while Fischlin’s functionality requires, even for corrupt users, that the message to be signed be included in the signing command given to the ideal functionality, our functionality does not look at the message specified by a corrupt user, but instead gives him a “free signature”. Only when message/signature pairs are verified, the functionality can learn what message was signed. As a consequence, it can not be used to realize $F_{com}$, meaning that Fischlin’s impossibility result does not apply in our case. Even so, our functionality still captures the essential features of blind signature schemes.

We can prove that a blind signature protocol realizes our functionality in the plain model if and only if the corresponding blind signature scheme satisfies weak blindness and non-forgeability, as originally defined by JLO. We note that, in this paper, we refer to JLO’s version of blindness as weak blindness, reflecting the fact that the adversary is not allowed to choose his target keys.

On the negative side, our functionality requires the signer to be honest during key generation. In the UC framework, this corresponds to the property of weak blindness, since the adversary does not choose his own keys. We believe that in some cases this correctly models the real world, for instance in a scenario where the key generation for a bank is performed by a financial supervisory authority.

As an alternative to this corruption model, we may allow corruption of the signer before the key generation takes place, but demand that the signer reveal his public and secret keys to a trusted third party for verification. A possible scenario is one where the bank generates its own keys, but then shows them to a financial supervisory authority. We express this in the UC framework by adding
an incorruptible trusted third party to the protocol, to whom the signer sends his public and secret keys (over a confidential channel) before any user issues signing requests. With this requirement, it is clear that our result also holds for a slightly stronger version of blindness, where the adversary chooses his target keys, but then reveals them to the simulator.

If we are willing to return to the common reference string model, it should be possible to relax this requirement even further, by including in the public key a commitment to the secret key along with a proof that this commitment is correct. (In the CRS model with a carefully chosen commitment scheme, the simulator can extract the secret key, essentially reducing everything to the case of blindness.) In this case, the main advantage of our functionality compared to Fischlin’s functionality is that we allow protocols where the user’s first message does not contain an extractable copy of the message to be signed. This means that the functionality could be realized by blind signature protocols accepting messages of arbitrary length.

Another, minor difference between the functionalities is that our functionality lets the environment decide whether or not the signer should grant a signature to a user. This is a vital property if the functionality is to be used in a bigger protocol, but it could of course be added to Fischlin’s functionality.

Our main contribution in this paper is a more flexible blind signature functionality that allows a larger class of realizing protocols, while still capturing the essence of blind signatures.

In Section 2 of this paper, we review the properties of a blind signature scheme and give formal definitions of blindness and non-forgeability. In Section 3 we present our ideal functionality for blind signatures and prove our main result.

2. Blind Signatures

Our definition of a blind signature scheme corresponds to the one given by Juels, Luby and Ostrovsky in [6].

Definition 1 (Blind Signature Scheme). A blind signature scheme BS is a four-tuple $(\text{Signer}, \text{User}, \text{Gen}, \text{Verify})$ with the following properties:

- $\text{Gen}$ is a probabilistic polynomial time algorithm, which takes as input a security parameter $\tau$ (encoded as $1^{\tau}$) and outputs a pair $(sk, pk)$ of secret and public keys.
- $\text{Signer}$ and $\text{User}$ are a pair of probabilistic polynomial time interactive Turing machines, given as common input a public key $pk$. In addition, $\text{Signer}$ is given a corresponding secret key $sk$, and $\text{User}$ is given a message $m$. The length of all inputs must be polynomial in the security parameter $\tau$. $\text{Signer}$ and $\text{User}$ interact according to the protocol. At the end of the interaction, $\text{Signer}$ outputs either completed or not completed and $\text{User}$ outputs either fail or $\sigma(m)$.
- $\text{Verify}$ is a deterministic polynomial time algorithm, which takes as input a public key $pk$, a message $m$ and a signature $\sigma(m)$, and outputs
either accept or reject, indicating whether \( \sigma(m) \) is a valid signature on the message \( m \).

It is required that for any message \( m \), and for all key pairs \((sk, pk)\) output by \( \text{Gen} \), if both \( \text{Signer} \) and \( \text{User} \) follow the protocol, then \( \text{Signer}(sk, pk) \) outputs completed, \( \text{User}(pk, m) \) outputs \( \sigma(m) \), and \( \text{Verify}(pk, m, \sigma(m)) \) outputs accept.

The security of blind signature schemes is formally defined below. We note that, throughout this paper, weak blindness corresponds to the original definition of blindness given by JLO.

**Definition 2** (Weak Blindness). Consider the experiment \( \text{Exp}_{BS,A}^{wb}(\tau) \) (steps 1, 2, \ldots, 6) in Figure 1, where \( A \) is an algorithm which controls \( \text{Signer} \) but not \( \text{User} \). We define the advantage of \( A \) in breaking \( BS \) with respect to weak blindness as

\[
\text{Adv}_{BS,A}^{wb}(\tau) = \left| \Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0] \right|.
\]

The scheme \( BS \) is said to be secure with respect to weak blindness if, for all probabilistic polynomial time \( A \), \( \text{Adv}_{BS,A}^{wb}(\tau) \) is negligible in \( \tau \).

We now introduce a slightly stronger definition, blindness, in which the adversary determines the key pair \((sk, pk)\), and hands it over to us.

**Definition 3** (Blindness). Consider the experiment \( \text{Exp}_{BS,A}^{b}(\tau) \) (steps 1', 2, \ldots, 6) in Figure 1, where \( A \) is an algorithm which controls \( \text{Signer} \) but not \( \text{User} \). We define the advantage of \( A \) in breaking \( BS \) with respect to blindness as

\[
\text{Adv}_{BS,A}^{b}(\tau) = \left| \Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0] \right|.
\]

The scheme \( BS \) is said to be secure with respect to blindness if, for all probabilistic polynomial time \( A \), \( \text{Adv}_{BS,A}^{b}(\tau) \) is negligible in \( \tau \).

An even stronger notion, strong blindness, is defined in the same manner, except that the adversary is only required to output a public key \( pk \) in the first step of the experiment.

**Definition 4** (Non-Forgeability). Consider the experiment \( \text{Exp}_{BS,A}^{nf}(\tau) \) in Figure 1, where \( A \) is an algorithm which controls \( \text{User} \) but not \( \text{Signer} \). We define the success rate of \( A \) in breaking \( BS \) with respect to non-forgery as

\[
\text{Succ}_{BS,A}^{nf}(\tau) = \Pr[k > l].
\]

The scheme \( BS \) is said to be secure with respect to non-forgery if, for all probabilistic polynomial time \( A \), \( \text{Succ}_{BS,A}^{nf}(\tau) \) is negligible in \( \tau \).

We now present another notion for blindness, adapting the notion ”real-or-random” for symmetric encryption given in [1]. The idea is that, when interacting with an honest user, it should be infeasible for a malicious signer to tell whether a known message or a hidden random string is being signed. (Note that there does not seem to be a natural “strong” version of this notion.)
**Universally Composable Blind Signatures in the Plain Model**

Let $A$ engage in two parallel interactive protocols, the first with $User(pk_{i}, m_{0})$ and the second with $User(pk_{i}, m_{1})$. If the first $User$ outputs $(m_{0}, m_{1})$ and the second $User$ outputs $(m_{1}, m_{b})$, then give $(m_{0}, m_{1})$ to $A$ as additional input.

A outputs a bit $b'$.

**Definition 5** ((Weak) Real-or-Random Blindness). Consider the experiment $Exp_{SS, A}^{wror/por} (\tau)$ in Figure 1, where $A$ is an algorithm which controls Signer but not User. We define the advantage of $A$ in breaking $SS$ with respect to (weak) real-or-random blindness as

$$Adv_{SS, A}^{wror/por} (\tau) = |Pr[b' = 1|b = 1] - Pr[b' = 1|b = 0]|.$$
The scheme BS is said to be secure with respect to real-or-random blindness if, for all probabilistic polynomial time $A$, $\text{Adv}_w^{\text{wror/ror}}(\tau)$ is negligible in $\tau$.

Adapting of a result from [1], we obtain the following theorem, the proof of which is given in Appendix A:

**Theorem 1.** A blind signature scheme BS is secure with respect to (weak) blindness if and only if it is secure with respect to (weak) real-or-random blindness.

3. **Universally Composable Blind Signatures**

Our ideal functionality for blind signatures, $F_{BS}$, is defined in Figure 2. The protocol $\pi_{BS}$ is defined in Figure 3.

For ease of presentation, the session id (SID), which should be present in all messages, is not included. The first message sent to the functionality must be (KeyGen). We consider a static corruption model, but require that corruption take place immediately after the (KeyGen) message has been processed, which amounts to the key generation being honest. In addition to generating keys, the ideal adversary $S$ produces signature generation and verification facilities for the functionality: $\Pi(m)$ simulates a conversation between an honest signer and an honest user, producing a signature $\sigma$ on $m$. $\pi(m, \sigma)$ outputs 1 if $\sigma$ is a valid signature on $m$, and 0 otherwise.

When the signer receives a signature request from a user, the environment determines whether or not the user is entitled to a signature. If the functionality is to be used in a bigger protocol, allowing this decision to depend on outer circumstances may be useful. For instance, it may depend on the balance of the user’s bank account.

In the plain model, parties running a protocol have authenticated communication channels, but messages are potentially delayed by the adversary. To handle this, we let $S$ delay messages between parties. $S$ also decides when to inform the respective parties about the outcome of a signature request. This allows $S$ to get the order of these messages right, that is, according to the real protocol.

Our functionality keeps track of signatures generated by corrupt users by means of a *free signature count*. In Fischlin’s functionality, when a corrupt user produces a signature by interaction with an honest signer, this signature is stored together with the message $m$ input by the user. However, since the user is corrupt, we do not know if $m$ really is the message being signed. Assume that the user instead signs $m'$, a message never seen by $F_{BSig}$, and obtains a valid pair $(m', \sigma)$. Upon verification, $(m', \sigma)$ will be rejected by $F_{BSig}$, while accepted by the real protocol. Our functionality overcomes this problem, by increasing the free signature count every time an honest signer completes a protocol with a corrupt user. If the free signature count is more than zero, a pair $(m, \sigma)$ may be accepted upon verification, even if $F_{BS}$ has never seen $m$ before. We argue that $F_{BS}$ still incorporates the required properties of a blind signature protocol, as it still prevents a user from obtaining more valid signatures than generated by interaction with an honest signer.
\(\mathcal{F}_{BS}\) proceeds as follows, with signer ˜Q, users ˜P₁, . . . , ˜Pₙ and an ideal adversary S.

On message (KeyGen) from the signer ˜Q:
1. Send (KeyGen) to S and wait.
2. Upon receipt of (Key, pk, \(\Pi, \pi\)) from S, store (pk, \(\Pi, \pi\)) and send (Key, pk) to ˜Q and stop.

On message (Sign, pk, \(m, \sigma\)) from a user ˜Pᵢ:
1. Send (Sign, Pᵢ) to S.
2. Upon receipt of (Sign, Pᵢ, ack) from S, send (Sign, Pᵢ) to ˜Pᵢ.
3. Upon receipt of (Sign, Pᵢ, denied) from S, send (Sign, Pᵢ, denied) to S.
   Wait for (Sign, Pᵢ, denied, ack) from S, and output (Sign, denied) to ˜Pᵢ.
4. Upon receipt of (Sign, Pᵢ) from S, send (Sign, Pᵢ) to S.
   1. Upon receipt of (Signature, Pᵢ, Q completed) from S, send (Signature, Pᵢ) to ˜Q. If ˜Q is honest and ˜Pᵢ is corrupt, increase the free signature count.
   2. Upon receipt of (Signature, Pᵢ, Q not completed) from S, send (Signature, Pᵢ, not completed) to ˜Q.
   3. Upon receipt of (Signature, Pᵢ, Pᵢ completed) from S, if Pᵢ is honest, \(\sigma \leftarrow \Pi(m)\). If \((m, \sigma, 0)\) is stored, then stop. Otherwise, store \((m, \sigma, 1)\) and send (Signature, \(\sigma\)) to ˜Pᵢ.
   4. Upon receipt of (Signature, Pᵢ, fail) from S, send (Signature, fail) to ˜Pᵢ.

On message (Verify, pk, \(m, \sigma\)) from an honest user ˜Pᵢ:
1. If \((m, \sigma, 1)\) is stored, send (Verify) to ˜Pᵢ and stop.
2. If \((m, \sigma, 0)\) is stored, send (Verify, reject) to ˜Pᵢ and stop.
3. If \(\pi(m, \sigma) = 1\) and Q is corrupt, store \((m, \sigma, 1)\), send (Verify) to ˜Pᵢ and stop.
4. If \(\pi(m, \sigma) = 1\) and the free signature count is larger than zero, decrease the signature count, store \((m, \sigma, 1)\), send (Verify) to ˜Pᵢ and stop.
5. Store \((m, \sigma, 0)\) and send (Verify, reject) to ˜Pᵢ and stop.

**Figure 2.** Blind signature functionality \(\mathcal{F}_{BS}\)

The protocol \(\pi_{BS}\) is described in Figure 3. The user initiates a protocol, and upon receiving the first message from a user, the signer lets the environment decide whether he should engage in the protocol, analogously to the formulation of \(\mathcal{F}_{BS}\). We now prove our main result:

**Theorem 2.** The blind signature scheme BS is secure with respect to weak blindness and non-forgeability if and only if the protocol \(\pi_{BS}\) securely realizes \(\mathcal{F}_{BS}\) in the plain model.
\[ \pi_{BS} \] is defined as follows, with signer \( Q(\tau) \) and users \( P_1, \ldots, P_n \).

**Q(\tau):**
1. Upon the first input (\texttt{KeyGen}), run the algorithm \( \texttt{Gen}(\tau) \), obtain a key pair \((sk, pk)\) and output (\texttt{Key}, \( pk \)).
2. Upon receiving \((x)\) from \( P_i \), send \((\texttt{Sign}, P_i?)\) to \( Z \) and wait.
3. Upon input \((\texttt{Sign}, P_i, \text{denied})\) from \( Z \), send \((\texttt{Sign}, \text{denied})\) to \( P_i \) and stop.
4. Upon input \((\texttt{Sign}, P_i)\) from \( Z \), run \texttt{Signer} and give it \( x \) as input on its communication tape. Forward any output on \texttt{Signer}'s communication tape to \( P_i \), and vice versa. If \texttt{Signer}(\( sk, pk \)) outputs \textit{completed}, then output \((\texttt{Signature}, P_i)\). Otherwise, if \texttt{Signer} outputs \textit{not completed}, output \((\texttt{Signature}, P_i, \textit{not completed})\).

**P_i:**
1. Upon input \((\texttt{Sign}, pk, m)\), run \texttt{User}(\( pk, m \)). When \texttt{User}(\( pk, m \)) outputs \( x \) on its communication tape, send \((x)\) to \( Q \) and wait.
2. Upon receiving \((\texttt{Sign}, \text{denied})\) from \( Q \), output \((\texttt{Sign}, \text{denied})\).
3. Upon receiving \((y)\) from \( Q \), forward \( y \) to \texttt{User}'s communication tape. Forward any output on \texttt{User}'s communication tape to \( Q \), and vice versa. If \texttt{User} outputs \( \sigma \), then output \((\texttt{Signature}, \sigma)\). Otherwise, if \texttt{User} outputs \textit{fail}, output \((\texttt{Signature}, \text{fail})\).
4. Upon input \((\texttt{Verify}, pk, m, \sigma)\), run the algorithm \texttt{Verify}, obtain accept/reject and output \((\texttt{Verify})/(\texttt{Verify}, \text{reject})\).

**Figure 3.** Blind signature protocol \( \pi_{BS} \)

**Proof. Only if:** Assuming that the blind signature scheme \( BS \) is secure with respect to weak blindness and non-forgeryability, we show that for every adversary \( A \) interacting with parties running \( \pi_{BS} \) in the plain model, there is an ideal adversary \( S \) such that no environment \( Z \) can tell whether it is interacting with \( A \) and \( \pi_{BS} \) or with \( S \) and the ideal protocol \( \text{IDEAL}_{F_{BS}} \).

As usual, \( S \) runs a simulated copy of \( A \), and forwards all messages from \( Z \) to \( A \) and back. When \( A \) corrupts a party \( P \), \( S \) corrupts \( \tilde{P} \). When \( \tilde{P} \) is corrupt, any input from \( Z \) meant for \( \tilde{P} \) goes directly to \( S \), who forwards it to \( A \) on the input tape corresponding to \( P \), and the other way around. Moreover, \( S \) can send messages to \( F_{BS} \) in the name of \( P \), and messages from \( F_{BS} \) meant for \( \tilde{P} \) go to \( S \). \( S \) runs as described below.

In the following, when a party \( P \) controls another party \( P' \), the notation "\( P|P' \)" should be read as "\( P \), in the name of \( P' \)."
Algorithm $S(\tau)$:

- Upon receiving $(\text{KeyGen}, Q)$ from $\mathcal{F}_{BS}$, $S$ runs $Gen(\tau)$, obtains a key pair $(sk, pk)$ and stores $(sk, pk)$. $S$ then produces signature generation and verification facilities $\Pi$ and $\pi$, and sends $(\text{Key}, Q, pk, \Pi, \pi)$ to $\mathcal{F}_{BS}$.
- Upon receiving $(\text{Sign}, P_i)$ from $\mathcal{F}_{BS}$, when both $Q$ and $P_i$ are honest: $S$ simulates honest $Q$ and honest $P_i$, and allows $A$ to delay any communication between $S|Q$ and $S|P_i$.

1. $S$ chooses a random message $\tilde{m}$, and runs $User(pk, \tilde{m})$. When $User(pk, \tilde{m})$ outputs $x$ on its communication tape, $S|P_i$ sends $(x)$ to $S|Q$. When $S|Q$, receives $(x)$, $S$ sends $(\text{Sign}, P_i, \text{ack})$ to $\mathcal{F}_{BS}$.
2. Upon receipt of $(\text{Sign}, P_i, \text{denied})$ from $\mathcal{F}_{BS}$, $S|Q$ sends $(\text{Sign}, \text{denied})$ to $S|P_i$. When $S|P_i$ receives $(\text{Sign}, \text{denied})$, $S$ sends $(\text{Sign}, P_i, \text{denied}, \text{ack})$ to $\mathcal{F}_{BS}$.
3. Upon receipt of $(\text{Sign}, P_i)$ from $\mathcal{F}_{BS}$, $S$ runs $Signer(sk, pk)$ and gives it $x$ as input on its communication tape. $S|Q$ forwards any output on $Signer$’s communication tape to $S|P_i$, and vice versa. Similarly, $S|P_i$ forwards any output on $User$’s communication tape to $S|Q$, and vice versa.
4. If $Signer$ outputs completed, then $S$ sends $(\text{Signature}, P_i, Q \text{ completed})$ to $\mathcal{F}_{BS}$. Otherwise, if $Signer$ outputs not completed, then $S$ sends $(\text{Signature}, P_i, Q \text{ not completed})$ to $\mathcal{F}_{BS}$.
5. If $User$ outputs $\sigma$, then $S$ sends $(\text{Signature}, P_i, P_i \text{ completed})$ to $\mathcal{F}_{BS}$. Otherwise, if $User$ outputs fail, then $S$ sends $(\text{Signature}, P_i, P_i \text{ fail})$ to $\mathcal{F}_{BS}$.

- Upon receiving $(\text{Sign}, P_i)$ from $\mathcal{F}_{BS}$, when $Q$ is corrupt and $P_i$ is honest: $S$ simulates honest $P_i$.

1. $S$ chooses a random message $\tilde{m}$, and runs $User(pk, \tilde{m})$. When $User(pk, \tilde{m})$ outputs $x$ on its communication tape, $S|P_i$ sends $(x)$ to $A$.
2. $S$ sends $(\text{Sign}, P_i, \text{ack})$ to $\mathcal{F}_{BS}$, and $S|Q$ receives $(\text{Sign}, P_i ?)$ from $\mathcal{F}_{BS}$.
3. If $S|P_i$ receives $(\text{Sign}, \text{denied})$ from $A$, then $S|Q$ sends $(\text{Sign}, P_i, \text{denied})$ to $\mathcal{F}_{BS}$. Upon receiving $(\text{Sign}, P_i, \text{denied})$ from $\mathcal{F}_{BS}$, $S$ sends $(\text{Sign}, P_i, \text{denied}, \text{ack})$ to $\mathcal{F}_{BS}$.
4. If $S|P_i$ receives $(y)$ from $A$, then $S|Q$ sends $(\text{Sign}, P_i)$ to $\mathcal{F}_{BS}$, and $S$ receives $(\text{Sign}, P_i)$ from $\mathcal{F}_{BS}$. $S|P_i$ forwards $y$ to $User$’s communication tape. $S|P_i$ forwards any output on $User$’s communication tape to $A$, and vice versa.
5. If $User$ outputs $\sigma$, then $S$ sends $(\text{Signature}, P_i, P_i \text{ completed})$ to $\mathcal{F}_{BS}$. Otherwise, if $User$ outputs fail, then $S$ sends $(\text{Signature}, P_i, P_i \text{ fail})$ to $\mathcal{F}_{BS}$.
6. $S$ sends $(\text{Signature}, P_i, Q \text{ not completed})$ to $\mathcal{F}_{BS}$, and $S|Q$ receives $(\text{Signature}, P_i, Q \text{ not completed})$ from $\mathcal{F}_{BS}$.

- When $Q$ is honest and $P_i$ is corrupt: $S$ simulates honest $Q$. 


(1) When $S|Q$ receives $(x)$ from $A$, $S$ chooses a random message $\tilde{m}$ and $S|\tilde{P}_i$ sends $(\text{Sign}, pk, \tilde{m})$ to $F_{BS}$. Upon receipt of $(\text{Sign}, P_i)$ from $F_{BS}$, $S$ sends $(\text{Sign}, P_i, \text{ack})$ to $F_{BS}$.

(2) Upon receipt of $(\text{Sign}, P_i, \text{denied})$ from $F_{BS}$, $S|Q$ sends $(\text{Sign}, \text{denied})$ to $A$. $S$ sends $(\text{Sign}, P_i, \text{denied, ack})$ to $F_{BS}$, and $S|\tilde{P}_i$ receives $(\text{Sign}, \text{denied})$ from $F_{BS}$.

(3) Upon receipt of $(\text{Sign}, P_i)$ from $F_{BS}$, $S$ runs $\text{Signer}(sk, pk)$ and gives it $x$ as input on its communication tape. $S|Q$ forwards any output on $\text{Signer}$’s communication tape to $A$, and vice versa.

(4) If $\text{Signer}$ outputs completed, then $S$ sends $(\text{Signature}, P_i, Q \text{ completed})$ to $F_{BS}$. Otherwise, if $\text{Signer}$ outputs not completed, then $S$ sends $(\text{Signature}, P_i, Q \text{ not completed})$ to $F_{BS}$.

(5) $S$ sends $(\text{Signature}, P_i, P_i \text{ fail})$ to $F_{BS}$, and $S|\tilde{P}$ receives $(\text{Signature}, \text{fail})$ from $F_{BS}$.

We want to show that if there exists an environment $Z$ able to distinguish $\text{IDEAL}_{F_{BS}}$ and $\pi_{BS}$ with non-negligible probability, then we can construct an adversary $A^{\text{ror}}$ breaking the real-or-random blindness or an adversary $A^{\text{af}}$ breaking the non-forgery of our scheme. To this end, we will consider a series of games, where we gradually modify the behavior of the involved parties.

Game 0: In this game, $Z$ interacts with the protocol $\pi_{BS}$ running with parties $Q, P_1, \ldots, P_n$.

Game 1: This game is the same as Game 0, with the following modifications:

The honest parties are now simulated by $P$. In addition, $P$ keeps track of the produced signatures: If an honest $P_i$ generates a signature $\sigma$ on a message $m$, $P$ stores $(m, \sigma, 1)$. However, if $P_i$ is corrupt and $Q$ is honest, and $Q$ outputs $(\text{Signature}, P_i)$, $P$ increases the free signature count.

From $Z$’s point of view, there is no difference between Game 0 and Game 1. Therefore, if we let $G_i$ denote the output of $Z$ when taking part in Game $i$, we have

$$\left| \Pr[G_0 = 1] - \Pr[G_1 = 1] \right| = 0.$$ 

Game 2: This game is the same as Game 1, with the following modifications: If an honest $P_i$ generates a signature $\sigma$ on a message $m$, and $(m, \sigma, 0)$ is stored, then $P$ stops. Upon input $(\text{Verify}, pk, m, \sigma)$ to an honest $P_i$, $P$ responds as follows:

1. If $(m, \sigma, 1)$ is stored, $P|P_i$ outputs (Verify) and stops.
2. If $(m, \sigma, 0)$ is stored, $P|P_i$ outputs (Verify, reject) and stops.
3. If $\pi(m, \sigma) = 1$ and $Q$ is corrupt, $P$ stores $(m, \sigma, 1)$, then $P|P_i$ outputs (Verify) and stops.
4. If $\pi(m, \sigma) = 1$ and the free signature count is larger than zero, $P$ decreases the signature count and stores $(m, \sigma, 1)$, then $P|P_i$ outputs (Verify) and stops.
5. $P$ stores $(m, \sigma, 0)$, then $P|P_i$ outputs (Verify, reject) and stops.

We now prove the following lemma:
Lemma 1. There is an adversary $A_{nf}$ such that
\[ \left| \Pr[G_1 = 1] - \Pr[G_2 = 1] \right| \leq \text{Succ}_{BS,A_{nf}}^nf(\tau). \]

Proof. We simplify the proof by defining the event $F$ in a game as follows:

$F$: At some point during the game, while $Q$ is honest, some honest party $P_i$ is asked to verify a valid pair $(m, \sigma)$, but there is no recorded entry $(m, \sigma, 1)$, and the free signature count is zero.

We note that, unless $F$ occurs, Game 1 and Game 2 proceed identically. In particular, if it ever happens in Game 2 that an honest $P_i$ generates a signature $\sigma$ on a message $m$, and $(m, \sigma, 0)$ is stored, then $(m, \sigma, 0)$ must have been stored while $Q$ was honest, which implies that $F$ occurred. Hence we have
\[ \left| \Pr[G_1 = 1] - \Pr[G_2 = 1] \right| \leq \Pr[F]. \]

We now construct an adversary $A_{nf}$ trying to break the non-forgeability of our scheme. $A_{nf}(1^{\tau}, pk)$ runs simulated copies of $Z$, $A$ and the corrupt parties. $A_{nf}$ also simulates $P$, and behaves exactly as $P$, with the following exceptions: When $Q$ is asked to generate keys, instead of running Gen, $Q$ simply outputs $(\text{Key}, pk)$. When $Q$ engages in a protocol with some $P_i$, $A_{nf}$ engages in a protocol with $\text{Signer}(sk, pk)$, forwarding inputs from $P_i$ to $\text{Signer}(sk, pk)$ and vice versa. If $F$ occurs, we deduce that for some numbers $k, l$ such that $k > l$, $Z$ has produced $k$ valid pairs $(m, \sigma)$, while $Q$ has output completed $l$ times. Since $A_{nf}$ stores the valid signatures and controls the free signature count, $A_{nf}$ can detect $F$ and output the $k$ valid pairs $(m, \sigma)$. Hence we have
\[ \text{Succ}_{BS,A_{nf}}^nf(\tau) \geq \Pr[F], \]
which leads to
\[ \left| \Pr[G_1 = 1] - \Pr[G_2 = 1] \right| \leq \text{Succ}_{BS,A_{nf}}^nf(\tau), \]
by which the proof is complete. \qed

Game 3: In this game, $Z$ interacts with the ideal protocol $\text{IDEAL}_{F_{BS}}$ running with parties $\tilde{Q}, \tilde{P}_1, \ldots, \tilde{P}_n$.

We observe that Game 2 and Game 3 differ only in the case where $P_i$ is honest: When signing a message $m$, in Game 2, $Q$ runs a protocol with $\text{User}(pk, m)$, while in Game 3, $Q$ runs a protocol with $\text{User}(pk, \tilde{m})$, where $\tilde{m}$ is a randomly chosen message.

We prove the following lemma:

Lemma 2. There is an adversary $A_{ror}$ such that
\[ \left| \Pr[G_2 = 1] - \Pr[G_3 = 1] \right| = \text{Adv}_{BS,A_{ror}}(\tau). \]

Proof. We construct an adversary $A_{ror}$ trying to break the real-or-random blindness of our scheme: $A_{ror}(1^{\tau}, sk, pk)$ runs simulated copies of $Z$, $A$ and the corrupt parties. $A_{ror}$ also simulates $P$. We note that, since we require that $Q$ be honest during key generation, $A_{ror}$ controls $Q$ at this stage. When $Q$ is asked to
generate keys, $\texttt{A}^{\text{ror}}$ simply outputs $(\text{Key}, pk)$ in the name of $Q$. $\texttt{A}^{\text{ror}}$ behaves exactly as $P$, with the following exception: When $Q$ engages in a protocol to sign some message $m_1$ input by some honest user $P_1$, $\texttt{A}^{\text{tor}}$ outputs $m_1$ and engages in a protocol with User$(pk, m_b)$, where $b$ is $\texttt{A}^{\text{tor}}$‘s challenge bit and $m_0$ is a randomly chosen message. $\texttt{A}^{\text{tor}}$ forwards messages from $Q$ to User$(pk, m_b)$ and the other way around. If $\texttt{A}^{\text{tor}}$ obtains $\sigma$ as additional input, then $\texttt{A}^{\text{tor}}$ outputs $(\text{Signature}, \sigma)$ in the name of $P$. Finally, when $\mathcal{Z}$ outputs a bit $b'$, $\texttt{A}^{\text{tor}}$ outputs $b'$.

We observe that, if $\texttt{A}^{\text{tor}}$’s challenge bit $b = 0$, then $\mathcal{Z}$’s environment in Game 3 is perfectly simulated, while if $b = 1$, $\mathcal{Z}$’s environment in Game 2 is perfectly simulated. Hence, we compute $\texttt{A}^{\text{tor}}$’s advantage as follows:

$$\text{Adv}_{\text{BS}, \texttt{A}^{\text{tor}}}^{\text{ror}}(\tau) = \left| \Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0] \right|$$

$$= \left| \Pr[G_2 = 1] - \Pr[G_3 = 1] \right|,$$

and the proof is complete.

By a standard hybrid argument, if there exists an effective distinguisher $\mathcal{Z}$ between Game 0 and Game 3, then, for some $i$, $0 \leq i \leq 2$, there exists an effective distinguisher $\mathcal{Z}_i$ between Game $i$ and Game $i + 1$. As shown above, such a $\mathcal{Z}_i$ would imply the existence of an effective $\texttt{A}^{\text{nf}}$ or an effective $\texttt{A}^{\text{tor}}$. Hence this part of the proof is complete, and we proceed with the opposite direction.

If: We start by assuming that there exists an effective adversary $\texttt{A}^{\text{tor}}$, and construct an environment $\mathcal{Z}$ trying to distinguish interaction with $\pi_{\text{BS}}$ from interaction with IDEAL$_{\text{FBS}}$. $\mathcal{Z}$ runs as follows:

1. $\mathcal{Z}$ sends $(\text{KeyGen})$ to $Q$, and obtains $pk$.
2. $\mathcal{Z}$ instructs $A$ to corrupt $Q$, so that $sk$ may be obtained from $A$.
3. $\mathcal{Z}$ runs $\texttt{A}^{\text{tor}}(\tau, sk, pk)$.
4. $\mathcal{Z}$ chooses $b \in \{0, 1\}$ at random. Each time $\texttt{A}^{\text{tor}}$ outputs a message $m_1$, if $b = 0$, $\mathcal{Z}$ chooses a random message $m_0$ of appropriate length and sends $(\text{Sign}, pk, m_0)$ to $P_1$. Otherwise, if $b = 1$, $\mathcal{Z}$ sends $(\text{Sign}, pk, m_1)$ to $P_1$. $\mathcal{Z}$ now instructs $A$ to let $\texttt{A}^{\text{tor}}$ run a protocol with $P_1$ on $Q$’s behalf. If $P_1$ outputs $(\text{Signature}, \sigma)$, $\mathcal{Z}$ gives $\sigma$ to $\texttt{A}^{\text{tor}}$ as additional input.
5. When $\texttt{A}^{\text{tor}}$ outputs a bit $b'$, $\mathcal{Z}$ outputs 1 if $b' = b$ and 0 otherwise.

We note that, if $\mathcal{Z}$ interacts with $\pi_{\text{BS}}$, $\texttt{A}^{\text{tor}}$’s environment in $\text{Exp}_{\text{BS}, \texttt{A}^{\text{tor}}}(\tau)$ is perfectly simulated. If $\mathcal{Z}$ interacts with IDEAL$_{\text{FBS}}$, then $\texttt{A}^{\text{tor}}$ does not run in its expected environment, but whatever happens, $\texttt{A}^{\text{tor}}$ gets no information about
the bit $b$, so $b' = b$ with probability $\frac{1}{2}$. Hence we get

$$\left| \Pr[Z_{\pi_{BS}} = 1] - \Pr[Z_{\text{IDEAL}_{F_{BS}}} = 1] \right|$$

$$= \left| \Pr[Z_{\pi_{BS}} = 1|b = 1] \cdot \Pr[b = 1] + \Pr[Z_{\pi_{BS}} = 1|b = 0] \cdot \Pr[b = 0] - \frac{1}{2} \right|$$

$$= \frac{1}{2} \left( \Pr[Z_{\pi_{BS}} = 1|b = 1] + (1 - \Pr[Z_{\pi_{BS}} = 0|b = 0]) - 1 \right)$$

$$= \frac{1}{2} \left( \Pr[Z_{\pi_{BS}} = 1|b = 1] - \Pr[Z_{\pi_{BS}} = 0|b = 0] \right)$$

$$= \frac{1}{2} \text{Adv}_{BS,A_{\text{xor}}}(\tau),$$

which completes this part of the proof.

Finally, assuming that there exists an effective adversary $A_{\text{nf}}$, we construct an environment $Z$ trying to distinguish interaction with $\pi_{BS}$ from interaction with $\text{IDEAL}_{F_{BS}}$. $Z$ runs as follows:

1. $Z$ sends $(\text{KeyGen})$ to $Q$, and obtains $pk$.
2. $Z$ runs $A_{\text{nf}}(\tau, pk)$.
3. Each time $A_{\text{nf}}$ engages in a protocol, $Z$ instructs $A$ to corrupt a party $P_i$ and let $A_{\text{nf}}$ run a protocol with $Q$ on $P_i$’s behalf. (When $Q$ outputs $(\text{Sign}, P_i)$, $Z$ replies with $(\text{Sign}, P_i)$).
4. Assume that $Q$ outputs $l$ messages on the form $(\text{Signature}, P_i)$ for some $i$, and that $A_{\text{nf}}$ outputs $k$ valid pairs $(m_i, \sigma_i)$. If $k > l$, $Z$ sends $(\text{Verify}, pk, m_i, \sigma_i)$ to an honest party $P_j$ for each $i = 1, \ldots, k$. For $i = k$, if $P_j$ replies with $(\text{Verify}, \text{reject})$, then $Z$ outputs 0. Otherwise, if $P_2$ replies with $(\text{Verify})$, then $Z$ outputs 1.

We note that, if $Z$ interacts with $\pi_{BS}$, then $A_{\text{nf}}$’s environment in $\text{Exp}_{BS,A_{\text{nf}}}(\tau)$ is perfectly simulated, and if $A_{\text{nf}}$ outputs $k > l$ valid pairs then $Z$’s output will be 1. If $Z$ interacts with $\text{IDEAL}_{F_{BS}}$, then $A_{\text{nf}}$ does not run in its expected environment, but whatever happens, we know that $Z$’s output will be 0 in this case.

If $A_{\text{nf}}$ outputs $k > l$ valid pairs $(m_i, \sigma_i)$, then if $Z$ interacts with $\pi_{BS}$, $Z$’s output will be 1, while if $Z$ interacts with $\text{IDEAL}_{F_{BS}}$, $Z$’s output will be 0. So we get

$$\left| \Pr[Z_{\pi_{BS}} = 1] - \Pr[Z_{\text{IDEAL}_{F_{BS}}} = 1] \right| = \left| \Pr[k > l] - 0 \right|$$

$$= \text{Succ}_{BS,A_{\text{nf}}}(\tau),$$

by which the proof is complete.

\[\square\]

References

In order to simplify the proof, we introduce the notion left-or-right blindness, which is similar to weak blindness, except that the step where $A$ outputs two messages $(m_0, m_1)$ and interacts with User$(pk, m_b)$ and User$(pk, m_{1-b})$ can be repeated polynomially many times.

Definition 6 (Left-or-Right Blindness). Consider the following experiment, where $A$ is an algorithm which controls Signer but not User:

\[ \text{Exp}_{BS,A}^{lor}(\tau): \]

1. $(sk, pk) \leftarrow \text{Gen}(\tau)$.
2. $b \leftarrow \{0, 1\}$.
3. Run $A(1^n, sk, pk)$.
4. When $A$ outputs two messages $(m_0, m_1)$: Let $A$ engage in two parallel interactive protocols, the first with User$(pk, m_b)$ and the second with User$(pk, m_{1-b})$. If the first User outputs $\sigma(m_b)$ and the second User outputs $\sigma(m_{1-b})$, then give $(\sigma(m_0), \sigma(m_1))$ to $A$ as additional input. This step can be repeated polynomially many (in $\tau$) times.
5. $A$ outputs a bit $b'$.

We define the advantage of $A$ in breaking $BS$ with respect to left-or-right blindness as

\[ \text{Adv}_{BS,A}^{lor}(\tau) = \left| Pr[b' = 1|b = 1] - Pr[b' = 1|b = 0] \right|. \]

The scheme $BS$ is said to be secure with respect to left-or-right blindness if, for all probabilistic polynomial time $A$, $\text{Adv}_{BS,A}^{lor}(\tau)$ is negligible in $\tau$.

The proof consists of showing the following relations among the security notions for blind signature schemes:

1. Weak blindness $\Rightarrow$ left-or-right blindness
2. Left-or-right blindness $\Rightarrow$ real-or-random blindness
(3) Real-or-random blindness ⇒ left-or-right-blindness
(4) Left-or-right blindness ⇒ weak blindness

**Weak Blindness ⇒ Left-or-Right Blindness:** Assume that there exists an effective adversary $A^{\text{lor}}$ with respect to left-or-right blindness. We will construct an effective adversary $A^{\text{wb}}$ with respect to weak blindness, using $A^{\text{lor}}$ as a subroutine. In more detail, we consider the experiment $\text{Exp}_{\text{SS}, A^{\text{wb}}}^{\text{wb}}(\tau)$, where $A^{\text{wb}}$ proceeds as given below.

Algorithm $A^{\text{wb}}(1^\tau, sk, pk)$:

(1) $j \leftarrow \{0, \ldots, n\}$, where $n$ is the number of times that $A^{\text{lor}}$ repeats step 4 in $\text{Exp}_{\text{SS}, A^{\text{lor}}}^{\text{lor}}(\tau)$.

(2) Run $A^{\text{lor}}$ on input $(1^\tau, sk, pk)$.

(3) Denote by $(m_{00}, m_{10}), \ldots, (m_{0(n-1)}, m_{1(n-1)})$ the message pairs output by $A^{\text{lor}}$ during the run. When $A^{\text{lor}}$ outputs $(m_{0i}, m_{1i})$:

- If $0 \leq i < j$, let $A^{\text{lor}}$ engage in two parallel interactive protocols, the first with $\text{User}(pk, m_{0i})$ and the second with $\text{User}(pk, m_{1i})$. If $\text{User}(pk, m_{0i})$ outputs $\sigma(m_{0i})$ and $\text{User}(pk, m_{1i})$ outputs $\sigma(m_{1i})$, then $A^{\text{lor}}$ is given $(\sigma(m_{0i}), \sigma(m_{1i}))$ as additional input.

- If $i = j$, then output $(m_{0i}, m_{1i})$. Forward inputs from $\text{User}(pk, m_{0i})$ and $\text{User}(pk, m_{1(i-b)i})$ to the respective communication tapes of $A^{\text{lor}}$, and the other way around. If $(\sigma(m_{0i}), \sigma(m_{1i}))$ is obtained as additional input, forward this to $A^{\text{lor}}$.

- If $j < i < n$, let $A^{\text{lor}}$ engage in two parallel interactive protocols, the first with $\text{User}(pk, m_{1i})$ and the second with $\text{User}(pk, m_{0i})$. If $\text{User}(pk, m_{0i})$ outputs $\sigma(m_{0i})$ and $\text{User}(pk, m_{1i})$ outputs $\sigma(m_{1i})$, then $A^{\text{lor}}$ is given $(\sigma(m_{0i}), \sigma(m_{1i}))$ as additional input.

(4) When $A^{\text{lor}}$ outputs a bit $b'$, output $b'$.

For $0 \leq i \leq n$, define $P_i$ as the probability that $A^{\text{lor}}$ outputs 1 given that, for the first $i$ message pairs $(m_0, m_i)$ output during a run, $A^{\text{wb}}$ lets $A^{\text{lor}}$ interact with $\text{User}(pk, m_0)$ and $\text{User}(pk, m_1)$, respectively. Then, for the last $n - i$ message pairs, $A^{\text{wb}}$ lets $A^{\text{lor}}$ interact with $\text{User}(pk, m_1)$ and $\text{User}(pk, m_0)$, respectively.

We note that if $i = 0$, $A^{\text{wb}}$ simulates $A^{\text{lor}}$’s environment in $\text{Exp}_{\text{SS}, A^{\text{lor}}}^{\text{lor}}(\tau)$ in the case where $b = 1$, and if $i = n$, $A^{\text{wb}}$ simulates $A^{\text{lor}}$’s environment in the case where $b = 0$. So by assumption we have

$$|P_0 - P_n| = \delta(\tau)$$

for some $\delta(\tau)$ non-negligible in $\tau$. We can write

$$\left| (P_0 - P_1) + (P_1 - P_2) + \cdots + (P_{n-1} - P_n) \right| = \delta(\tau).$$
We note that if $A^{wb}$’s challenge bit $b = 0$, the probability that $A^{lor}$ outputs 1 is $P_{j+1}$, and if $b = 1$, the probability that $A^{lor}$ outputs 1 is $P_j$. Hence we get
\[
\text{Adv}^{wb}_{BS,A^{wb}}(\tau) = \left| \Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0] \right|
\]
\[
= \left| \Pr[b' = 1 | b = 1 \land j = 0] \cdot \Pr[j = 0] + \cdots + \Pr[b' = 1 | b = 1 \land j = n - 1] \cdot \Pr[j = n - 1] - \Pr[b' = 1 | b = 0 \land j = 0] \cdot \Pr[j = 0] - \cdots - \Pr[b' = 1 | b = 0 \land j = n - 1] \cdot \Pr[j = n - 1] \right|
\]
\[
= \frac{1}{n} (P_0 + \cdots + P_{n-1} - (P_1 + \cdots + P_n))
\]
\[
= \frac{1}{n} ((P_0 - P_1) + \cdots + (P_{n-1} - P_n))
\]
\[
= \frac{\delta(\tau)}{n}.
\]
Since $\delta(\tau)$ is non-negligible in $\tau$, and $n$ is polynomial in $\tau$, $\frac{1}{n}\delta(\tau)$ is also non-negligible in $\tau$.

\textbf{Left-or-Right Blindness} $\Rightarrow$ \textbf{Real-or-Random Blindness}: In this part, assuming that there exists an effective adversary $A^{lor}$ with respect to real-or-random blindness, we consider $\text{Exp}_{BS,A^{lor}}(\tau)$, where $A^{lor}$ runs the following procedure.

Algorithm $A^{lor}(1^\tau, sk, pk)$:
1. Run $A^{lor}$ on input $(1^\tau, sk, pk)$.
2. Each time $A^{lor}$ outputs a message $m_1$:
   - Choose a random message $m_0$ and output $(m_0, m_1)$.
   - Forward inputs from $\text{User}(pk, m_0)$ to $A^{lor}$’s communication tape, and the other way around. Run the protocol with $\text{User}(pk, m_{1-b})$ honestly (to ensure that a signature is produced). If $(\sigma(m_0), \sigma(m_1))$ is obtained as additional input, forward $\sigma(m_1)$ to $A^{lor}$.
3. When $A^{lor}$ outputs a bit $b'$, output $b'$.

We observe that when $A^{lor}$’s challenge bit $b = 0$, $A^{lor}$ simulates $A^{lor}$’s environment in $\text{Exp}_{BS,A^{lor}}(\tau)$ in the case where $A^{lor}$’s challenge bit $b = 0$, and correspondingly for the case $b = 1$. Hence we get
\[
\text{Adv}^{lor}_{BS,A^{lor}}(\tau) = \text{Adv}^{lor}_{BS,A^{lor}}(\tau),
\]
which, by assumption, is non-negligible in $\tau$.

\textbf{Real-or-Random Blindness} $\Rightarrow$ \textbf{Left-or-Right Blindness}: Assuming that there exists an effective adversary $A^{lor}$ with respect to left-or-right blindness, our goal is to construct an adversary $A^{lor}$ breaking the real-or-random blindness of our scheme. To simplify this part of the proof, we define the following games:
Game 1: This game is the same as $\text{Exp}^{\text{lor}}_{\text{BS}, A}(\tau)$, except that we always have $b = 0$. Game 2: This game is the same as Game 1, with the following modifications: Each time $A$ outputs a message pair, say $(m_0, m'_1)$, we choose a random message $m_1$, and let $A$ engage in protocols with $\text{User}(pk, m_0)$ and $\text{User}(pk, m_1)$, respectively. Moreover, we run internally a protocol between honest $\text{Signer}(sk, pk)$ and honest $\text{User}(pk, m'_1)$, and obtain $\sigma(m'_1)$. If $\text{User}(pk, m_0)$ outputs $\sigma(m_0)$ and $\text{User}(pk, m_1)$ outputs $\sigma(m_1)$, then we give $(\sigma(m_0), \sigma(m'_1))$ to $A$ as additional input.

Game 3: This game is the same as Game 1, with the following modifications: Each time $A$ outputs a message pair, say $(m'_0, m'_1)$, we choose two random messages $m_0$ and $m_1$, and let $A$ engage in protocols with $\text{User}(pk, m_0)$ and $\text{User}(pk, m_1)$, respectively. Moreover, we run two protocols internally, one between honest $\text{Signer}(sk, pk)$ and honest $\text{User}(pk, m'_0)$, the other between honest $\text{Signer}(sk, pk)$ and honest $\text{User}(pk, m'_1)$. We thus obtain $\sigma(m'_0)$ and $\sigma(m'_1)$. If $\text{User}(pk, m_0)$ outputs $\sigma(m_0)$ and $\text{User}(pk, m_1)$ outputs $\sigma(m_1)$, then we give $(\sigma(m'_0), \sigma(m'_1))$ to $A$ as additional input.

We start by constructing a distinguisher between Game 1 and Game 3, $A^{13}$, using $A^{\text{lor}}$ as a subroutine. That is, $A^{13}$ participates in either Game 1 or Game 3 and outputs a bit $b$.

Algorithm $A^{13}(1^\tau, sk, pk)$:

1. Run $A^{\text{lor}}$ on input $(1^\tau, sk, pk)$.
2. $\tilde{b} \leftarrow \{0, 1\}$
3. Each time $A^{\text{lor}}$ outputs a message pair, say $(\tilde{m}_0, \tilde{m}_1)$:
   - Output $(\tilde{m}_b, \tilde{m}_{1-b})$.
   - Engage in two protocols, say with $\text{User}_0$ and $\text{User}_1$, respectively. Forward outputs from $\text{User}_0$ and $\text{User}_1$ to the respective communication tapes of $A^{\text{lor}}$, and the other way around. If some pair $(\sigma(\tilde{m}_0), \sigma(\tilde{m}_1))$ is obtained as additional input, forward this to $A^{\text{lor}}$.
4. When $A^{\text{lor}}$ outputs a bit $b'$, if $b' = \tilde{b}$, then output 1, otherwise output 0.

We observe that, if $A^{13}$ participates in Game 1, $A^{13}$ simulates $A^{\text{lor}}$’s environment in $\text{Exp}^{\text{lor}}_{\text{BS}, A^{\text{lor}}}(\tau)$, with $\tilde{b}$ acting as the challenge bit. On the other hand, if $A^{13}$ participates in Game 3, $A^{\text{lor}}$ gets no information about $\tilde{b}$, so in this case $A^{13}$ outputs 1 with probability $\frac{1}{2}$. In line with the former definitions, we compute the advantage of $A^{13}$ as follows, where $b''$ denotes the output of $A^{13}$:

$$\text{Adv}_{\text{BS}, A}^{13}(\tau) = \left| \Pr[b'' = 1|\text{Game 1}] - \Pr[b'' = 1|\text{Game 3}] \right|,$$
where
\[
\Pr[b'' = 1 | \text{Game 1}] = \Pr[b' = 1 | \text{Game 1} \land \tilde{b} = 1] \cdot \Pr[\tilde{b} = 1] \\
+ \Pr[b' = 0 | \text{Game 1} \land \tilde{b} = 0] \cdot \Pr[\tilde{b} = 0] \\
= \Pr[b' = 1 | \text{Game 1} \land \tilde{b} = 1] \cdot \frac{1}{2} \\
+ \left(1 - \Pr[b' = 1 | \text{Game 1} \land \tilde{b} = 0]\right) \cdot \frac{1}{2}.
\]

So we get
\[
\text{Adv}_{13}^{\text{BS, } A} (\tau) = \left| \frac{1}{2} \left( \Pr[b' = 1 | \text{Game 1} \land \tilde{b} = 1] - \Pr[b' = 1 | \text{Game 1} \land \tilde{b} = 0]\right) + \frac{1}{2} - \frac{1}{2} \right| \\
= \frac{1}{2} \cdot \text{Adv}_{12}^{\text{BS, } A} (\tau),
\]

which, by assumption, is non-negligible in \( \tau \). This means that \( A^{13} \) is an effective distinguisher between Game 1 and Game 3.

By a standard hybrid argument, if there exists an effective \( A^{13} \), then there exists an effective distinguisher \( A^{12} \) between Game 1 and Game 2 or an effective distinguisher \( A^{23} \) between Game 2 and Game 3. We complete this part of the proof by showing that, using either \( A^{12} \) or \( A^{23} \), we can construct an effective adversary \( A^{\text{ror}} \) with respect to real-or-random blindness.

First, we assume that there exists an effective \( A^{12} \) and consider the experiment \( \text{Exp}_{\text{BS, } A^{\text{ror}}}^{\text{ror}} (\tau) \), where \( A^{\text{ror}} \) proceeds as described below.

Algorithm \( A^{\text{ror}}(1^\tau, sk, pk) \):
1. Run \( A^{12} \) on input \( (1^\tau, sk, pk) \).
2. When \( A^{12} \) outputs a message pair, say \((\tilde{m}_0, \tilde{m}_1)\), output \( \tilde{m}_1 \).
3. Simulate honest \( \text{User} (pk, \tilde{m}_0) \) internally, and engage in a protocol, say with \( \text{User}_1 \). Forward inputs from \( \text{User}(pk, \tilde{m}_0) \) and \( \text{User}_1 \) to the respective communication tapes of \( A^{12} \), and the other way around. If \( \text{User}(pk, \tilde{m}_0) \) outputs \( \sigma(\tilde{m}_0) \) and \( \sigma(\tilde{m}_1) \) is obtained as additional input, forward \( (\sigma(\tilde{m}_0), \sigma(\tilde{m}_1)) \) to \( A^{12} \).
4. When \( A^{12} \) outputs a bit \( b' \), output \( b' \).

We note that if \( A^{\text{ror}} \)'s challenge bit \( b = 0 \), then \( A^{\text{ror}} \) simulates \( A^{12} \)'s environment in Game 2, while if \( b = 1 \), \( A^{\text{ror}} \) simulates \( A^{12} \)'s environment in Game 1. We compute the advantage of \( A^{\text{ror}} \) as follows:
\[
\text{Adv}_{\text{BS, } A}^{\text{ror}} (\tau) = \left| \Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]\right| \\
= \left| \Pr[b' = 1 | \text{Game 1}] - \Pr[b' = 1 | \text{Game 2}]\right| \\
= \text{Adv}_{12}^{\text{BS, } A} (\tau),
\]

which, by assumption, is non-negligible in \( \tau \).

Now, in a similar manner, we assume that there exists an effective \( A^{23} \) and consider the experiment \( \text{Exp}_{\text{BS, } A^{\text{ror}}}^{\text{ror}} (\tau) \), where \( A^{\text{ror}} \) proceeds as follows:
Algorithm $A^{\lor}(1^\tau, sk, pk)$:

1. Run $A^{23}$ on input $(1^\tau, sk, pk)$.
2. When $A^{23}$ outputs a message pair, say $(\tilde{m}_0, \tilde{m}_1)$, output $\tilde{m}_0$.
3. Randomly choose a message $\tilde{m}$. Simulate internally $User(pk, \tilde{m})$ in an honest manner, and engage in a protocol, say with $User_0$. Forward inputs from $User_0$ and $User(pk, \tilde{m})$ to the respective communication tapes of $A^{23}$, and the other way around. Moreover, simulate internally a protocol between honest $Signer(sk, pk)$ and honest $User(pk, \tilde{m}_0)$, so that a signature $\sigma(\tilde{m}_0)$ is obtained. If $User(pk, \tilde{m})$ outputs $\sigma(\tilde{m})$ and $\sigma(\tilde{m}_0)$ is obtained as additional input, forward $(\sigma(\tilde{m}_0), \sigma(\tilde{m}_1))$ to $A^{23}$.
4. When $A^{23}$ outputs a bit $b'$, output $b'$.

We note that if $A^{\lor}$'s challenge bit $b = 0$, then $A^{\lor}$ simulates $A^{23}$'s environment in Game 3, while if $b = 1$, $A^{\lor}$ simulates $A^{23}$'s environment in Game 2. We compute the advantage of $A^{\lor}$ as follows:

$$Adv^{\lor}_{BS,A}(\tau) = \left| Pr[b' = 1|b = 1] - Pr[b' = 1|b = 0]\right|$$

$$= Pr[b' = 1|\text{Game 2}] - Pr[b' = 1|\text{Game 3}]$$

$$= Adv^{23}_{BS,A}(\tau),$$

which, by assumption, is non-negligible in $\tau$.

**Left-or-Right Blindness $\Rightarrow$ Weak Blindness:** Assume that there exists an effective adversary $A^{wb}$ with respect to weak blindness, and consider the experiment $Exp^{l_{\lor}}_{BS,A_{lor}}(\tau)$, where $A^{lor}$ proceeds as described below.

Algorithm $A^{lor}(1^\tau, sk, pk)$:

1. Run $A^{wb}$ on input $(1^\tau, sk, pk)$.
2. When $A^{wb}$ outputs a message pair $(m_0, m_1)$, output $(m_0, m_1)$.
3. Forward inputs from $User(pk, m_b)$ and $User(pk, m_{1-b})$ to the respective communication tapes of $A^{wb}$, and the other way around. If $(\sigma(m_0), \sigma(m_1))$ is obtained as additional input, forward this to $A^{wb}$.
4. When $A^{wb}$ outputs a bit $b'$, output $b'$.

It is clear that when $A^{lor}$’s challenge bit $b = 0$, $A^{lor}$ simulates $A^{wb}$’s environment in $Exp^{wb}_{BS,A_{wb}}(\tau)$ in the case where $A^{wb}$’s challenge bit $b = 0$, and correspondingly for the case where $b = 1$. Hence we get

$$Adv^{lor}_{BS,A_{lor}}(\tau) = Adv^{wb}_{BS,A_{wb}}(\tau),$$

which, by assumption, is non-negligible in $\tau$. 
Paper III

Round-Optimal Blind Signatures from Waters Signatures
Kristian Gjøsteen and Lillian Kråkmo
Published in Proceedings of ProvSec 2008, volume 5324 of LNCS
ROUND-OPTIMAL BLIND SIGNATURES FROM WATERS SIGNATURES

KRISTIAN GJØSTEEN AND LILLIAN KRÅKMO

Abstract. We present a round-optimal blind signature scheme based on Waters' signature scheme. Our construction resembles that of Fischlin [9], but does not rely on generic non-interactive zero-knowledge proofs. In addition to a common reference string, our scheme requires a registered public key for the signer.

1. Introduction

The idea of blind signatures was proposed by Chaum [8] as a key ingredient for anonymous electronic cash applications. Blind signatures allow a bank to issue signatures without seeing the content of the signed documents, and at the same time prevent users from forging signatures. The security of blind signatures was first formalized by Pointcheval and Stern [16] and later by Juels, Luby and Ostrovsky [12], resulting in the notions blindness and non-forgery. Since then, a number of blind signature schemes have been proposed, some in the random oracle model [16, 1, 4, 5], and some without random oracles [7, 13, 14, 9, 11]. Most of the above mentioned schemes use three or more moves, and proving security under concurrent executions of the signature generation protocol has often been difficult. Notably, this problem is avoided in schemes requiring only two moves, i.e. round-optimal schemes.

Recently, Fischlin [9] proposed a round-optimal blind signature scheme in the common reference string model. This scheme uses generic non-interactive zero-knowledge (NIZK) proofs, which makes it quite impractical. Our contribution is a concrete round-optimal scheme based on Waters’ signature scheme [18]. Waters’ scheme is weakly unforgeable, in the sense that signatures may easily be randomized. This property makes Waters’ scheme a natural starting point for constructing a blind signature scheme. In our scheme, to obtain a blind signature on a message, the user computes a commitment to the message, based on Waters’ hash function. The signer’s response is essentially a signature on the commitment. Finally, the user obtains a blind signature from the signer’s response, by simultaneously unblinding the commitment and randomizing the resulting Waters signature.

In order to obtain provable security, the user is also required to compute a NIZK proof that the commitment was honestly generated. The proof is obtained by applying linear encryption [6] as an extractable commitment scheme, and by
compiling a suitable Σ-protocol using a technique developed by Damgård et al. [17]. Consequently, we need a common reference string and a registered public key for the signer.

The main drawback with our scheme is that a moderate number of NIZK proofs must be generated and verified as part of the signature generation protocol. A major advantage, however, is that verifying signatures is no more expensive than for Waters signatures.

2. Preliminaries

2.1. Bilinear Groups. Let $G$ be a multiplicative cyclic group of prime order $p$, and let $g$ be a generator of $G$. We say that $G$ is a bilinear group if the group operation in $G$ is efficiently computable, and if there exists a multiplicative cyclic group $G_1$ of order $p$ and an efficiently computable non-degenerate bilinear map $e : G \times G \to G_1$, i.e. for all $u, v \in G$ and $a, b \in \{0, \ldots, p - 1\}$, we have $e(u^a, v^b) = e(u, v)^{ab}$, and $e(g, g) \neq 1$.

2.2. Signature Schemes and Their Security. We refer to [12] for a formal definition of a signature scheme $S$, and note that we use the following notation: $S = (Gen, Sign, Verify)$, where $Gen(1^\tau)$ outputs $(sk, pk)$, $Sign(sk, m)$ outputs $\sigma$, and $Verify(pk, m, \sigma)$ outputs accept/reject.

We need the notion existential unforgeability under an adaptive chosen message attack (UF-CMA) as defined by Goldwasser, Micali and Rivest in [10], which is defined by the experiment $\text{Exp}_{S,A}^{uf-cma}(\tau)$, given in Figure 1. The experiment $\text{Exp}_{S,A}^{uf-cma}(\tau)$ starts with the adversary $A$ being given the target sender’s public key $pk$. $A$’s job is to produce a message/signature pair $(m, \sigma)$ such that $\sigma$ is a valid signature on $m$ with respect to $pk$. $A$ has access to the signing oracle $O_S$, which takes a message as input and outputs a signature on the message under $sk$. It is required that $O_S$ was never queried with the message $m$.

$A$ is said to win if the experiment returns 1. We define the success rate of $A$ in breaking $S$ with respect to UF-CMA as:

$$\text{Succ}_{S,A}^{uf-cma} = \Pr \left[ \text{Exp}_{S,A}^{uf-cma} = 1 \right].$$

Definition 1. The scheme $S$ is said to be $(t, q, \epsilon)$-secure with respect to UF-CMA if no $A$ running in time $t$ and making at most $q$ oracle queries has success rate at least $\epsilon$.

Waters’ Signature Scheme: The security of Waters’ signature scheme is based on the Computational Diffie Hellmann assumption and does not rely on random oracles [18]. We review the scheme below.

Key Generation: Let $G$ be a bilinear group of order $p$, where $p$ has length $\tau$, and let $e : G \times G \to G_1$ be the corresponding efficiently computable bilinear map. To generate the public key, the algorithm $Gen_W$ chooses a random generator $g \in G$ and a random $\alpha \in \{0, \ldots, p - 1\}$ and lets $g_1 = g^\alpha$. Additionally, for some appropriate $n$, it chooses random $g_2, u', u_1, \ldots, u_n \in G$ and lets $U = (u_1, \ldots, u_n)$. The public key is $(g, g_1, g_2, u', U)$ and the secret key is $g_2^\beta$. 

**Signature Generation:** Upon input of a message \( m \) of length \( n \), the algorithm \( \text{Sign}_W \) chooses a random \( r \in \{0, \ldots, p-1 \} \) and computes the signature \( \sigma = (\sigma_1, \sigma_2) = (g_2^r (u' \prod_{i=1}^{n} u_i^{m_i})^r, g^r) \), where \( m_i \) denotes the \( i \)’th bit of \( m \).

**Signature Verification:** To verify a signature \( \sigma = (\sigma_1, \sigma_2) \) on a message \( m \), the algorithm \( \text{Verify}_W \) checks that \( e(\sigma_1, g) = e(\sigma_2, u' \prod_{i=1}^{n} u_i^{m_i}) \cdot e(g_1, g_2) \). If this holds, it outputs accept, otherwise it outputs reject.

Assume that a signature \( (\sigma_1, \sigma_2) \) on a message \( m \) is generated according to the above scheme. Note that, if we randomly choose \( r^* \in \{0, \ldots, p-1 \} \) and let \( (\sigma_1^*, \sigma_2^*) = (\sigma_1 (u' \prod_{i=1}^{n} u_i^{m_i})^{r^*}, \sigma_2 g^{r^*}) \), \( (\sigma_1^*, \sigma_2^*) \) is a new, uniformly distributed signature on \( m \). This property is exploited in our blind signature scheme.

The computational Diffie-Hellman (CDH) problem is reviewed in Appendix A. Waters’ scheme is known to be \((t, q, \epsilon)\)-secure with respect to UF-CMA if the \( (t, \frac{e}{10(n+1)q}) \)-CDH assumption holds in \( G \).

### 2.3. Public Key Encryption Schemes and Their Security.

We refer to [3] for a formal definition of a public key encryption scheme \( \mathcal{PKE} \), and note that we use the following notation: \( \mathcal{PKE} = (\text{Gen}, \text{Enc}, \text{Dec}) \), where \( \text{Gen}(1^\tau) \) outputs \((\text{sk}, \text{pk}) \), \( \text{Enc}(\text{pk}, m) \) outputs \( c \), and \( \text{Dec}(\text{sk}, c) \) outputs \( m/\bot \).

In our work we need the security notion real-or-random indistinguishability under a chosen plaintext attack (ROR-CPA), which is defined by the experiment \( \text{Exp}_{\mathcal{PKE}, A}(\tau) \) given in Figure 1. In the experiment \( \text{Exp}_{\mathcal{PKE}, A}(\tau) \), the adversary \( A \) has access to the oracle \( \mathcal{O}_{\text{ror}}^b \) (initialized with a hidden bit \( b \)) which takes as input a message \( m \). If \( b = 0 \), it outputs an encryption of a randomly chosen string of length \( |m| \) under \( \text{pk} \). A new random string is chosen for each query. If \( b = 1 \), it outputs an encryption of \( m \) under \( \text{pk} \). A’s challenge is to guess the hidden bit \( b \).

We define the advantage of \( A \) in breaking \( \mathcal{PKE} \) with respect to ROR-CPA as

\[
\text{Adv}_{\mathcal{PKE}, A}^{\text{ror-cpa}}(\tau) = \Pr [\text{Exp}_{\mathcal{PKE}, A}(\tau) = 1 | b = 1] - \Pr [\text{Exp}_{\mathcal{PKE}, A}(\tau) = 1 | b = 0].
\]

**Definition 2.** The scheme \( \mathcal{PKE} \) is said to be \((t, q, \epsilon)\)-secure with respect to ROR-CPA if no \( A \) running in time \( t \) and making at most \( q \) oracle queries has advantage at least \( \epsilon \).

**Linear Encryption:** Linear encryption was proposed by Boneh, Boyen and Shacham in [6] as a natural extension of ElGamal encryption. While ElGamal encryption relies on the Decision Diffie Hellman (DDH) problem, Linear encryption relies on the Decision Linear Diffie Hellman (DLDH) problem, which is believed to be hard even in bilinear groups where the DDH problem is easy. The DLDH problem is reviewed in Appendix A.

In the Linear encryption (LE) scheme, \( \text{Gen}_L \) outputs \((\text{sk}_L, \text{pk}_L) \), where \( \text{pk}_L \) is a triple of randomly chosen generators \( \alpha_1, \alpha_2, \beta \in G \), where \( G \) is a group of prime order \( p \), and \( p \) has length \( \tau \). \( \text{sk}_L \) is the exponents \( a_1, a_2 \in \{0, \ldots, p-1 \} \) such that \( \alpha_1^{a_1} = \alpha_2^{a_2} = \beta \). \( \text{Enc}_L \) takes as input a message \( m \in G \), chooses random values \( r, s \in \{0, \ldots, p-1 \} \), and outputs the triple \((y_1, y_2, y_3) = (\alpha_1^r, \alpha_2^s, \beta^{r+s}m) \). \( \text{Dec}_L \) takes a ciphertext \((y_1, y_2, y_3) \) as input and outputs \( y_3 y_1^{-a_1} y_2^{-a_2} \).
It can be shown that, for all $t,q$ polynomial in $\tau$, the LE scheme is $(t,q,\epsilon)$-secure with respect to ROR-CPA, for some $\epsilon$ negligible in $\tau$, if the corresponding holds for the DLDH problem in $G$.

2.4. Setup Assumptions.

The Common Reference String Model: Let $D$ be a probabilistic polynomial-time algorithm which takes a security parameter $1^\tau$ as input and outputs a value $crs$, chosen according to some publicly known distribution. In the common reference string (CRS) model, we may assume that all parties have access to a trusted functionality $F_D^{crs}$, which initially runs $D(1^\tau)$ and obtains $crs$, and later gives $crs$ to any party asking for it.

The Registered Public Key Model: We also review the registered public-key model, according to [17], and note that its relation to the common reference string model is discussed in [2].
Let $KS$ be a probabilistic polynomial-time algorithm which takes a security parameter $1^\tau$ as input and outputs a pair $(sk, pk)$ of private and public keys.

In the registered public-key model, we may assume that all parties have access to a trusted functionality $F_{reg}^{KS}$, which can be invoked to register own key pairs and to retrieve the public keys of others. In order to register a key pair, the registrant privately sends $F_{reg}^{KS}$ the random coins $r$ used to create his key pair. $F_{reg}^{KS}$ then runs $KS(1^\tau, r)$, stores the resulting public key together with the identity of the registrant, and later gives the public key to any party asking for it.

2.5. Compilation of $\Sigma$-Protocols in the Registered Public Key Model. According to [17], a $\Sigma$-protocol for a relation $R$ is an interactive proof-system for the language $L_R = \{x | \exists w : (x, w) \in R\}$. The conversations are on the form $(a, e, z)$, where $a$ and $z$ are messages sent by the prover $P$, while $e$ is a random challenge sent by the verifier $V$. Additionally, a $\Sigma$-protocol has the properties relaxed special soundness and special honest-verifier zero-knowledge. We refer to [17] for a formal definition of these properties, and note that perfect honest-verifier zero-knowledge is a stronger variant of honest-verifier zero-knowledge, where the conversations output by the simulator are identically distributed as conversations between $P$ and $V$.

We now briefly review the technique developed by Damgård et al. [17] for compiling $\Sigma$-protocols into non-interactive zero-knowledge arguments. Their technique works in the registered public-key model, and we refer to [17] for a formal definition of a non-interactive system for the relation $R$ with key setup $KS$, along with the desired properties of such a system: correctness, zero-knowledge and soundness.

We note that the compilation technique only applies to $\Sigma$-protocols with the additional property of linear answer, i.e. it requires that $P$'s final message $z$ be a sequence of integers which are linearly obtained from the challenge $e$.

The high-level idea of the technique is the following: It is assumed that $V$ has initially registered a public key $pk_{KS}$ with the functionality $F_{reg}^{KS}$ described above. $pk_{KS}$ is on the form $(pk, c)$, where $pk$ is a public key of a homomorphic encryption scheme, and $c$ is an encryption under $pk$ of a randomly chosen challenge $e$. The corresponding private key is $(sk, e)$, where $sk$ is the private key corresponding to $pk$. To compute a proof, $P$ first obtains $pk_{KS}$ from $F_{reg}^{KS}$, and computes the first message $a$ according to the $\Sigma$-protocol. Then, $P$ exploits the homomorphic property of the encryption scheme, and the fact that the $\Sigma$-protocol has linear answer, to obtain an encrypted response to the challenge $e$ encrypted in $c$. The encrypted response may in turn be decrypted and checked as usual by $V$. This technique is illustrated in Chapter 4.2, where we give a detailed description of the compiled $\Sigma$-protocol used in our blind signature scheme.

As for showing that the compiled protocol has the desired properties, we note that correctness of the above system follows directly from completeness of the involved $\Sigma$-protocol. Furthermore, since a simulator running $V$ obtains the random coins intended for $F_{reg}^{KS}$, he obtains $V$'s private key, and in particular the
challenge \(e\). Hence, to simulate a proof for a statement \(x\), he may run an honest-verifier simulator for the \(\Sigma\)-protocol on input \((x, e)\) to obtain a conversation \((a, e, z)\) from which a correctly distributed proof is directly obtained. This means that zero-knowledge (for arbitrary verifiers) of the compiled protocol follows from honest-verifier zero-knowledge of the original \(\Sigma\)-protocol. Proving soundness is more involved, but it basically boils down to the assumed security of the involved encryption scheme. We refer to [17] for detailed proofs of the above properties for the general construction. As for the particular construction used in our blind signature scheme, proofs are given in the appendices.

3. Blind Signature Schemes and Their Security

Our definition of a blind signature scheme corresponds to the one given by JLO in [12], modified to fit our model, where all parties are assumed to have access to the trusted functionalities \(\mathcal{F}_{D_{\text{crs}}}^D\) and \(\mathcal{F}_{\text{reg}}^{KS}\) defined earlier, and where the signer is initially required to register a public key \(pk_{KS}\) with \(\mathcal{F}_{\text{reg}}^{KS}\).

**Definition 3** (Blind Signature Scheme). A blind signature scheme \(\text{BS}\) is a tuple \((\text{Gen}, \text{Signer}, \text{User}, \text{Verify}, KS, D)\) with the following properties:

- \(\text{Gen}\) is a probabilistic polynomial time algorithm, which takes as input a security parameter \(\tau\) (encoded as \(1^\tau\)), and outputs a pair \((sk, pk)\) of secret and public keys.
- \(\text{Signer}\) and \(\text{User}\) are a pair of polynomially-bounded probabilistic interactive Turing machines, given as common input a public key \(pk\). In addition, \(\text{Signer}\) is given a corresponding secret key \(sk\), and \(\text{User}\) is given a message \(m\). The length of all inputs must be polynomial in the security parameter \(\tau\). \(\text{Signer}\) and \(\text{User}\) interact according to the protocol. At the end of the interaction, \(\text{Signer}\) outputs either completed or not completed and \(\text{User}\) outputs either fail or \(\sigma(m)\).
- \(\text{Verify}\) is a deterministic polynomial time algorithm, which takes as input a public key \(pk\), a message \(m\) and a signature \(\sigma(m)\), and outputs either accept or reject, indicating whether \(\sigma(m)\) is a valid signature on the message \(m\).
- \(D\) and \(KS\) are the algorithms parameterizing \(\mathcal{F}_{D_{\text{crs}}}^D\) and \(\mathcal{F}_{\text{reg}}^{KS}\).

It is required that for any message \(m\), and for all key pairs \((sk, pk)\) output by \(\text{Gen}\), if both \(\text{Signer}\) and \(\text{User}\) follow the protocol, then \(\text{Signer}(sk, pk)\) outputs completed, \(\text{User}(pk, m)\) outputs \(\sigma(m)\), and \(\text{Verify}(pk, m, \sigma(m))\) outputs accept.

JLO defined security of a blind signature scheme using the notions *blindness* and *non-forgeability*. Informally, a scheme has *blindness* if it is infeasible for a malicious signer to determine the order of which two messages are signed by interaction with an honest user. A scheme has *non-forgeability* if, given \(l\) interactions with an honest signer, it is infeasible for a malicious user to produce more than \(l\) valid signatures.
Non-forgery for blind signature schemes in our model is formally defined using the experiment $\text{Exp}_{BS,A}^{\text{nf}}(\tau)$ given in Figure 1. $A$ is said to win the experiment if $k > l$. We define the success rate of the adversary $A$ in breaking $BS$ with respect to non-forgery as

$$\text{Succ}_{BS,A}^{\text{nf}} = \Pr[k > l].$$

**Definition 4** (Non-Forgeability). The scheme $BS$ is said to be $(t, q, \epsilon)$-secure with respect to non-forgery if no $A$ running in time $t$, engaging in at most $q$ protocols where Signer outputs completed, has success rate at least $\epsilon$.

As for blindness in our model, we need the experiment $\text{Exp}_{BS,A}^{\text{b}}(\tau)$ given in Figure 1. We define the advantage of $A$ in breaking $BS$ with respect to blindness as

$$\text{Adv}_{BS,A}^{\text{b}}(\tau) = \left| \Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0] \right|.$$ 

**Definition 5** (Blindness). The scheme $BS$ is said to be $(t, \epsilon)$-secure with respect to blindness if no $A$ running in time $t$ has advantage at least $\epsilon$.

4. **OUR BLIND SIGNATURE SCHEME**

In this section we present our blind signature scheme, which is based on Waters’ signature scheme. We note that a similar scheme is proposed by Okamoto [14].

4.1. **A Sketch of Our Scheme.** We start by briefly outlining the idea of our scheme. To obtain a blind signature on a message $m$, the user commits to $m$ and sends the resulting commitment $c$ to the user, along with a proof $\pi$ that $c$ was honestly generated. The signer responds with $\sigma'$, which is essentially a signature on the commitment. In the final step, the user obtains a blind signature $\sigma$ on $m$, by simultaneously unblinding the commitment and randomizing the resulting signature. The signature generation protocol is sketched in Figure 2, where we note that the key pair $(sk_{BS}, pk_{BS})$ is generated exactly as $(sk_W, pk_W)$ in Waters’ signature scheme.

We proceed by explaining the high-level idea of the proof $\pi$. Let $A^{\text{nf}}$ be an adversary trying to break the non-forgable security of our scheme. In order to achieve provable security, $\pi$ is constructed such that a simulator running a copy of $A^{\text{nf}}$ can extract the message $m$ and the exponent $t$. This extractability is obtained by having the user commit to $m$ and $t$ by encrypting them, using a public key obtained from the common reference string. Given $m$ and $t$, the simulator can use a signing oracle for Waters’ signature scheme to obtain a correctly distributed response $(\sigma'_1, \sigma'_2)$, hence the non-forgable security of our scheme reduces to the UF-CMA security of Waters’ scheme.

The proof $\pi$ should convince the verifier (in this case the signer) that $c$ was honestly generated. If we let $t_{\tau-1} \ldots t_0$ be the bit representation of $t$, that is, $t = \sum_{i=0}^{\tau-1} t_i 2^i$, this amounts to proving that $c = g^{\sum_{i=0}^{\tau-1} t_i 2^i} \prod_{i=1}^{n} u_i^{m_i}$ for known bits $t_i$, $0 \leq i \leq \tau - 1$, and $m_i$, $1 \leq i \leq n$. This is obtained by having the prover (in this case User) commit to each of the values $g^{t_i}$, $0 \leq i \leq \tau - 1$, and $u_i^{m_i}$,
Signer

\[ \text{pk}_{BS} = (g, g_1, g_2, u', U) \]
\[ \text{sk}_{BS} = g_2 \]

User

\[ \text{pk}_{BS} = (g, g_1, g_2, u', U) \]
\[ m = m_1m_2 \ldots m_n \in \{0, 1\}^n \]

Choose \( t \leftarrow \{0, \ldots, p - 1\} \)

Let \( c = g^t \prod_{i=1}^{n} u_i^{m_i} \)

Check that \( u'c \neq 1 \)

Compute a proof \( \pi \) of correctness of \( c \)

\[ \leftarrow (c, \pi) \]

Verify \( \pi \)

Choose \( r \leftarrow \{0, \ldots, p - 1\} \)

Let \( \sigma_1' = g_2^r (u'c)^r \)

Let \( \sigma_2' = g^r \)

Output \( (\sigma_1', \sigma_2') \)

\[ \rightarrow (\sigma_1, \sigma_2) \]

Choose \( r' \leftarrow \{0, \ldots, p - 1\} \)

Let \( \sigma_1 = \sigma_1' \sigma_2'^{-t} (u' \prod_{i=1}^{n} u_i^{m_i})^{-r'} \)

Let \( \sigma_2 = \sigma_2' g^{r'} \)

Check that

\[ e(\sigma_1, g) = e(\sigma_2, u' \prod_{i=1}^{n} u_i^{m_i}) \cdot e(g_1, g_2) \]

Output \( (\sigma_1, \sigma_2) / \text{fail} \)

---

**Figure 2.** The signature generation protocol.

1 \( \leq i \leq n \), prove correctness of each of the involved commitments, and then prove that \( c \) in fact contains the committed values.

Let \( \text{Enc}_L(m, r, s) \) denote the Linear encryption of the message \( m \in G \) with randomness \((r, s)\) under the public key \((\alpha_1, \alpha_2, \beta)\). Recall that, with this notation, \( \text{Enc}_L(m, r, s) = (\alpha_1^r, \alpha_2^s, \beta^{r+s} m) \), and note that this encryption scheme is homomorphic, i.e.

\[ \text{Enc}_L(m, r, s) \cdot \text{Enc}_L(m', r', s') = \text{Enc}_L(mm', r + r', s + s). \]

Let \( t_{l-1} \ldots t_0 \) be the bit representation of \( t \), that is, \( t = \sum_{i=0}^{l-1} t_i 2^i \). The prover commits to \( g^{t_i}, 0 \leq i \leq \tau - 1 \), by choosing random values \( r_i, s_i \in \{0, \ldots, p - 1\} \)
and computing
\[ T_i = \text{Enc}_L(g^{t_i}, r_i, s_i) = (\alpha_{r_i}^{r_i}, \alpha_{s_i}^{s_i}, \beta_{r_i+s_i}^{r_i}g^{t_i}). \]
Similarly, the prover commits to \( u_i^{m_i} \), \( 1 \leq i \leq n \), by choosing random values \( r_i', s_i' \in \{0, \ldots, p - 1\} \) and computing
\[ M_i = \text{Enc}_L(u_i^{m_i}, r_i', s_i') = (\alpha_{r_i'}^{r_i'}, \alpha_{s_i'}^{s_i'}, \beta_{r_i'+s_i'}^{r_i'+s_i'}u_i^{m_i}). \]
Correctness of each of the above commitments can be proved using a suitable \( \Sigma \)-protocol. Moreover, since we want our proof \( \pi \) to be non-interactive, we apply the technique developed by Damgård et al. for compiling \( \Sigma \)-protocols into NIZK proofs.

As for proving that \( c \) in fact contains the committed values, note that, by letting \( r^* = \sum_{i=0}^{l-1} r_i2^i + \sum_{i=1}^{n} r'_i \) and \( s^* = \sum_{i=0}^{l-1} s_i2^i + \sum_{i=1}^{n} s'_i \), we have
\[ \prod_{i=0}^{l-1} T_i^{2^i} \prod_{i=1}^{n} M_i = (\alpha_{r^*}^{r^*}, \alpha_{s^*}^{s^*}, \beta_{r^*+s^*}^{r^*+s^*}g^{\prod_{i=1}^{n} u_i^{m_i}}), \]
i.e. \( \prod_{i=0}^{l-1} T_i^{2^i} \prod_{i=1}^{n} M_i \) is a commitment to \( g^{\prod_{i=1}^{n} u_i^{m_i}} \). This means that the prover can prove the correctness of \( c \) by opening this commitment, that is, by including \( r^* \) and \( s^* \) in \( \pi \). In this way, assuming that the verifier has accepted all of the above NIZK proofs, he can conclude that \( c \) was honestly generated if and only if
\[ \text{Enc}_L(c, r^*, s^*) = \prod_{i=0}^{l-1} T_i^{2^i} \prod_{i=1}^{n} M_i. \]

4.2. The Protocol \( \text{compile}(\Sigma_{OR}) \). The proof \( \pi \) in our blind signature scheme includes proofs of correctness of several commitments, all on the form \( (y_1, y_2, y_3) = (\alpha_1^{r}, \alpha_2^{s}, \beta^{r+s}u^b) \), where \( \alpha_1, \alpha_2, \beta \) and \( u \) are publicly known elements of a bilinear group \( G \) of known prime order \( p \), \( r, s \in \{0, \ldots, p - 1\} \) are secret exponents, and \( b \) is a secret bit. We proceed by constructing a \( \Sigma \)-protocol for proving correctness of a commitment on the above form. To this end, we apply the so-called OR-construction, briefly reviewed here according to [17]. Given \( \Sigma \)-protocols \( \Sigma_I \) and \( \Sigma_r \) for relations \( R_I \) and \( R_r \), the OR-construction yields a \( \Sigma \)-protocol \( \Sigma_{OR} \) for the relation \( R_{OR} \) defined by
\[ ((x_1, x_r), (w_1, w_r)) \in R_{OR} \iff (x_1, w_1) \in R_I \lor (x_r, w_r) \in R_r. \]
Let \( R_L \) be the relation defined by
\[ (x, w) = ((x_1, x_2, x_3), (r, s)) \in R_L \iff x_1 = \alpha_1^{r}, x_2 = \alpha_2^{s}, x_3 = \beta^{r+s}. \]
We note that, for a commitment \( (y_1, y_2, y_3) \), we have
\[ (y_1, y_2, y_3) = (\alpha_1^r, \alpha_2^s, \beta^{r+s}) \iff ((y_1, y_2, y_3), (r, s)) \in R_L \]
and
\[ (y_1, y_2, y_3) = (\alpha_1^r, \alpha_2^s, \beta^{r+s}u) \iff ((y_1, y_2, \frac{y_3}{u}), (r, s)) \in R_L. \]
This means that a suitable protocol $\Sigma_{OR}$ for our purpose is obtained by composing $\Sigma$-protocols for $R_L$, where we let $x_3 = y_3$ in one protocol and $x_3 = \frac{y_3}{u}$ in the other.

A $\Sigma$-protocol $\Sigma_L$ for $R_L$ is presented in Appendix B, where we also show that $\Sigma_L$ has completeness, relaxed special soundness and perfect honest-verifier zero-knowledge.

The protocol $\Sigma_{OR}$ is constructed in a standard way from the two suitable instances of $\Sigma_L$, and it is not hard to show that completeness, relaxed special soundness and perfect honest-verifier zero-knowledge of $\Sigma_{OR}$ follow from the corresponding properties of $\Sigma_L$. For completeness, a description of $\Sigma_{OR}$ and proofs of the respective properties are given in Appendix C. We simply note that $\Sigma_{OR}$ has conversations on the form $(a,e,(z_1,z_2,\ldots,z_6))$, where $z_i$ is linearly obtained from $e$ for all $i, 1 \leq i \leq 6$. This means that $\Sigma_{OR}$ has linear answer, hence we may apply the compilation technique of Damgård et al.

We now describe the compilation of the protocol $\Sigma_{OR}$. The homomorphic encryption scheme used in the compilation is Paillier’s encryption scheme [15]. We refer to [15] for a description of this scheme. It is assumed that the verifier has initially registered a public key with $\mathcal{F}_{KS}^{KS}$, where the key setup algorithm $KS$ is defined as follows:

$KS(1^\tau)$: Generate a keypair $(sk_P,pk_P)$ for Paillier encryption, by running the key generation algorithm with $2\tau$ as input. Choose a random challenge $e \in \{0,\ldots,p-1\}$. Also, let $e$ be a Paillier encryption of $e$ under $pk_P$. The public key is $(pk_P, e)$ and the private key is $(sk_P, e)$.

Due to the homomorphic property of Paillier encryption, and the fact that $\Sigma_{OR}$ has linear answer, it is possible to execute the prover’s side of the protocol given only the encryption $e$ of the challenge $e$. First, the prover computes his first message $a$. Then, denoting by $Enc_P(pk_P,m)$ a random Paillier encryption of $m$ under $pk_P$, and assuming that the component $z_i$ of his response $z$ is given by $z_i = u_i + v_i e$, he can compute an encryption of $z_i$ as $Enc_P(pk_P,u_i \cdot e^{v_i})$.

The compiled protocol is defined below. We remind the reader that $\Sigma_{OR}$ has conversations on the form $(a,e,(z_1,z_2,\ldots,z_6))$, where, for each $i, 1 \leq i \leq 6$, $z_i = u_i + v_i e$ for some $u_i,v_i \in \{0,\ldots,p-1\}$.

Protocol $\text{Compile}(\Sigma_{OR})$:

(1) Given $(x,w)$, $P$ gets $V$’s public key $(pk_P, e)$ from $\mathcal{F}_{KS}^{KS}$ and computes the first message $a$ according to $\Sigma_{OR}$. Then, for $i$ such that $1 \leq i \leq 6$, $P$ computes $Enc_P(pk_P,u_i \cdot e^{v_i})$, and lets $c_i$ be a randomization of the resulting encryption. $P$ sends $(x,\pi)$ to $V$, where $\pi = (a,(c_1,\ldots,c_6))$.

(2) On input $x$ and a proof $\pi = (a,(c_1,\ldots,c_6))$, for $i$ such that $1 \leq i \leq 6$, $V$ lets $z_i^\prime$ be the Paillier decryption of $c_i$ under $sk_P$. Then $V$ verifies that the conversation $(a,e,(z_1^\prime,\ldots,z_6^\prime))$ would be accepted by the verifier of $\Sigma_{OR}$ upon input $x$, and accepts or rejects accordingly.

We need a result from [17] based on the following assumption: ‘$\mathcal{H}_{\text{Paillier}}$ is 2-harder than $G_{\text{dlog}}$’. Due to space limitations, we refer to [17] for formal definitions of the involved terms. Loosely speaking, it is assumed that, given an algorithm.
A that solves the discrete logarithm (DLOG) problem for moduli of length \( \tau \), there is no algorithm that breaks Paillier encryption for moduli of length \( 2\tau \), with runtime comparable to that of \( A \). We note that, as defined in [17], \( G_{\text{dlog}} \) addresses the DLOG problem in the subgroup of \( \mathbb{Z}_p \) of order \( p' \), where \( p \) and \( p' \) are primes, and \( p = 2p' + 1 \). Since our \( \Sigma \)-protocol involves a bilinear group, we need a modified version of \( G_{\text{dlog}} \), addressing the DLOG problem in a general bilinear group of prime order. We call this modified version \( G_{\text{dlog}}^* \). By arguing as in [17], the following assumption seems reasonable.

**Assumption 6.** \( \mathcal{H}_{\text{Paillier}} \) is 2-harder than \( G_{\text{dlog}}^* \).

We obtain the following result, analogous to Theorem 1, Theorem 2 and Corollary 1 in [17]. We note that, when it comes to proving soundness, we slightly modify the definition used on [17] so it better suits our application. This modified definition, along with a proof of Theorem 7 is given in Appendix D.

**Theorem 7.** compile\((\Sigma_{\text{OR}})\) has correctness and perfect zero-knowledge (in the registered public-key model). Furthermore, under Assumption 6, compile\((\Sigma_{\text{OR}})\) is sound for \( \mathcal{O}(\log \tau) \) executions.

4.3. Our Scheme. A detailed description of our blind signature scheme is given below. Recall that, in our model, it is assumed that Signer and User have access to the trusted functionalities \( F_{\text{crs}}^D \) and \( F_{\text{reg}}^D \) defined earlier. Furthermore, Signer is initially required to register a public key \( pk_{KS} \) with the functionality \( F_{\text{reg}}^D \).

**Common Reference String:** The algorithm \( D \) takes \( 1^r \) as input, randomly chooses \( \alpha_1, \alpha_2, \beta \in G \) and lets \( crs = \alpha_1 || \alpha_2 || \beta \).

**Key Setup:** The algorithm \( KS \) works exactly as in compile\((\Sigma_{\text{OR}})\), i.e. the public and secret keys are \( pk_{KS} = (pk, c) \) and \( sk_{KS} = (sk, e) \).

**Key Generation:** The algorithm \( Gen \) works exactly as in Waters’ signature scheme, i.e. the public and secret keys are \( pk_{BS} = (g, g_1, g_2, u', U) \) and \( sk_{BS} = g_2^\beta \).

**Signature Generation:** User takes \( (pk_{BS}, m) \) as input and lets \( m_1 m_2 \ldots m_n \) be the bit representation of \( m \). He randomly chooses \( t \in \{0, \ldots, p - 1\} \), and lets \( c = g^t \prod_{i=1}^{n} u_i^{m_i} \). He checks that \( u'c \neq 1 \). If this holds, he continues. Otherwise, he starts over, choosing a new \( t \). Then he generates a proof \( \pi \) of correctness of \( c \) as follows: Let \( t_{r-1} \ldots t_0 \) be the bit representation of \( t \), that is, \( t = \sum_{i=0}^{r-1} t_i 2^i \).

User gets \( crs = \alpha_1 || \alpha_2 || \beta \) from \( F_{\text{crs}}^D \) and lets \( pk_L = (\alpha_1, \alpha_2, \beta) \). Then, for \( i, 0 \leq i \leq r - 1 \), he chooses random values \( r_i, s_i \in \{0, \ldots, p - 1\} \) and computes

\[
T_i = Enc_L(pk_L, g^t, r_i, s_i) = (\alpha_1^{r_i}, \alpha_2^{s_i}, \beta^{r_i+s_i} g^{t_i}).
\]

Similarly, for \( i, 1 \leq i \leq n \), he chooses random values \( r_i', s_i' \in \{0, \ldots, p - 1\} \) and computes

\[
M_i = Enc_L(pk_L, u_i^{m_i}, r_i', s_i') = (\alpha_1^{r_i'}, \alpha_2^{s_i'}, \beta^{r_i'+s_i'} u_i^{m_i}).
\]

He also computes

\[
r^* = \sum_{i=0}^{r-1} r_i 2^i + \sum_{i=1}^{n} r_i', s^* = \sum_{i=0}^{r-1} s_i 2^i + \sum_{i=1}^{n} s_i'.
\]
User then computes, for all $i$, $0 \leq i \leq \tau - 1$, a proof $\pi_T$ according to $\text{Compile}(\Sigma_{\text{OR}})$ on input $(T_i, (r_i, s_i))$. Moreover, for all $i$, $1 \leq i \leq n$, he computes a proof $\pi_{M_i}$ according to $\text{Compile}(\Sigma_{\text{OR}})$ on input $(M_i, (r'_i, s'_i))$. Finally, he lets $\pi = ((T_0, \pi_{T_0}), \ldots, (T_{i-1}, \pi_{T_{i-1}}), (M_1, \pi_{M_1}), \ldots, (M_n, \pi_{M_n}), r^*, s^*)$, sends $(c, \pi)$ to Signer and waits.

Signer gets $(pk_{\text{BS}}, sk_{\text{BS}})$ as input. Upon receiving $(c, \pi)$ from User, he verifies $\pi$ by the following procedure: First, he verifies each of the pairs $(T_i, \pi_T)$, $0 \leq i \leq \tau - 1$, and $(M_i, \pi_{M_i})$, $1 \leq i \leq n$, according to $\text{Compile}(\Sigma_{\text{OR}})$. If all pairs are accepted, he continues. Otherwise, he outputs not completed and stops. He gets $\text{crs} = \alpha_1 || \alpha_2 || \beta$ from $\mathcal{F}^D_{\text{crs}}$ and lets $pk_L = (\alpha_1, \alpha_2, \beta)$. He then checks if

$$\text{Enc}_L(pk_L, c, r^*, s^*) = \prod_{i=0}^{l-i} T^{2^i}_{i} \prod_{i=1}^{n} M_i.$$  

If this holds, he concludes that $c$ was honestly generated and continues. Otherwise, he outputs not completed and stops. He randomly chooses $r \in \{0, \ldots, p-1\}$. He then lets $\sigma'_1 = g^{\sigma_2^2} (u^c)^r$ and $\sigma'_2 = g^r$, outputs completed, sends $(\sigma'_1, \sigma'_2)$ to User and stops.

Upon receiving $(\sigma'_1, \sigma'_2)$ from Signer, User randomly chooses $r' \in \{0, \ldots, p-1\}$. He lets $\sigma_1 = \sigma'_1 \sigma'_2^{-t} (u' \prod_{i=1}^{n} u_i^{m_i})^{r'}$ and $\sigma_2 = \sigma'_2 g^{r'}$. He then checks if

$$e(\sigma_1, g) = e(\sigma_2, u' \prod_{i=1}^{n} u_i^{m_i}) \cdot e(g_1, g_2).$$  

If this holds, he outputs $(\sigma_1, \sigma_2)$ and stops. Otherwise, he outputs fail and stops.

**Signature Verification:** The algorithm Verify works exactly as in Waters’ scheme, i.e. to verify a signature $\sigma = (\sigma_1, \sigma_2)$ on a message $m$, it checks that

$$e(\sigma_1, g) = e(\sigma_2, u' \prod_{i=1}^{n} u_i^{m_i}) \cdot e(g_1, g_2).$$  

If this holds, it outputs accept, otherwise it outputs reject.

We note that the restriction on $l$ in the below theorem is due to the restriction to $O(\log \tau)$ executions in Theorem 7. However, the authors of [17] show that, under a stronger non-standard assumption, their compiled protocol for proving equality of discrete logarithms is sound for an arbitrary polynomial number of executions. Intuitively, a similar strategy should apply to the protocol compile$(\Sigma_{\text{OR}})$. Hence, under this assumption, we may achieve non-forgiveness for an arbitrary polynomial $l$. We refer to [17] for more details.

The proofs of the following results are given in Appendix E and Appendix F, respectively.

**Theorem 8.** Under Assumption 6, and if Waters’ scheme is $(t', q', \epsilon')$-secure with respect to UF-CMA, then our blind signature scheme is $(t, l, \epsilon)$-secure with respect to non-forgiveness, where $q' = l = O(\log \tau)$, $t' = t + \text{poly}(\tau)$ and $\epsilon' = (1 - \rho)\epsilon$, where $\rho$ is a negligible function in $\tau$. 
Theorem 9. If the LE scheme is \((t', 1, \epsilon')\)-secure with respect to ROR-CPA, then our blind signature scheme is \((t, \epsilon)\)-secure with respect to blindness, where \(t' = t + \text{poly}(\tau)\), and \(\epsilon' = \frac{\epsilon}{2n+2\tau}\).

REFERENCES


Appendix A. Complexity Assumptions

A.1. The Computational Diffie-Hellman Assumption. The Computational Diffie-Hellman (CDH) problem is defined as follows: Given a cyclic group $G$ of order $p$ and a generator $g \in G$, the challenger chooses random $a, b \in \{0, \ldots, p-1\}$ and outputs $(g, g^a, g^b)$. The adversary’s job is to find $g^{ab} \in G$.

We define the success rate of an adversary $A$ in solving the CDH problem in $G$ as

$$\text{Succ}_{\text{CDH}}^A = \Pr[A(g, g^a, g^b) = g^{ab}] .$$

Definition 10. The $(t, \epsilon)$-CDH assumption holds in $G$ if no adversary running in time $t$ has success rate at least $\epsilon$ in solving the CDH problem in $G$.

A.2. The Decision Linear Diffie-Hellman Assumption. The Decision Linear Diffie-Hellman (DLDH) problem is defined as follows: Given a cyclic group $G$ of order $p$ and generators $g_1, g_2, g_3 \in G$, the challenger chooses random $a, b, c \in \{0, \ldots, p-1\}$ and a random bit $\beta$. If $\beta = 0$, the challenger outputs $(g_1, g_2, g_3, g_1^a, g_2^b, g_3^c)$. Otherwise, if $\beta = 1$, he outputs $(g_1, g_2, g_3, g_1^a, g_2^b, g_3^{a+b})$. The adversary outputs a guess $\beta'$ of $\beta$.

We say that an adversary $A$ has an advantage at least $\epsilon$ in solving the DLDH problem in $G$ if

$$\text{Adv}_{\text{DLDH}}^A = \left| \Pr[A(g_1, g_2, g_3, g_1^a, g_2^b, g_3^c) = 1] - \Pr[A(g_1, g_2, g_3, g_1^a, g_2^b, g_3^{a+b}) = 1] \right| \geq 2\epsilon .$$

Definition 11. The $(t, \epsilon)$-DLDH assumption holds in $G$ if no adversary running in time $t$ has advantage at least $\epsilon$ in solving the DLDH problem in $G$.

Appendix B. The Protocol $\Sigma_L$

The $\Sigma$-protocol $\Sigma_L$ for the relation $R_L$ is presented in Figure 3.

$\Sigma_L$ obviously has completeness, since an honest $P$ is guaranteed to convince $V$. We prove that it also has special soundness. Assume that, for some $x = (x_1, x_2, x_3)$, there are accepting conversations $((a_1, a_2, a_3), e, (z_1, z_2)), ((a_1, a_2, a_3), e', (z_1', z_2'))$, where $e \neq e'$. We show that we can efficiently compute $w = (r, s)$ such that $(x, w) \in R$. From the above assumption we have

$$\alpha z_1 = x_1 a_1, \alpha z'_1 = x_1' a_1,$$

which implies that

$$\alpha^{z_1 - z_1'} = x_1^{e - e'} .$$

Assuming that

$$x_1 = \alpha_1^r ,$$
Choose \( r', s' \leftarrow \{0, \ldots, p-1\} \)

Let

\[
\begin{align*}
a_1 &= \alpha_1^{r'} \\
a_2 &= \alpha_2^{s'} \\
a_3 &= \beta^{r'} + s'
\end{align*}
\]

Choose \( e \leftarrow \{0, \ldots, p-1\} \)

Let

\[
\begin{align*}
z_1 &= r' + rc \\
z_2 &= s' + sc
\end{align*}
\]

Check if

\[
\begin{align*}
\alpha_{z_1}^{z_1} &= x_1^e a_1 \\
\alpha_{z_2}^{z_2} &= x_2^e a_2 \\
\beta^{z_1+z_2} &= x_3^e a_3
\end{align*}
\]

and accept or reject accordingly

---

**Figure 3.** The protocol \( \Sigma_L \).

\( r \) can be computed as

\[
r = \frac{z_1 - z_1'}{e - e'} \mod p.
\]

We also have

\[
\alpha_2^{z_2} = x_2^e a_2, \quad \alpha_2^{z_2'} = x_2^{e'} a_2,
\]

which implies that

\[
\alpha_2^{z_2-z_2'} = x_2^{e-e'}.
\]
Assuming that
\[ x_2 = \alpha_2^s, \]
s can be computed as
\[ s = \frac{z_2 - z_2'}{e - e'} \mod p. \]

Finally we have
\[ \beta^{z_1+z_2} = x_3^s a_3, \quad \beta^{z_1'+z_2'} = x_3'^s a_3, \]
which implies that
\[ \beta^{(z_1-z_1')+(z_2-z_2')} = x_3'^{e'-e}. \]

Raising both sides to the power \( \frac{1}{e-e'} \mod p \) we obtain
\[ x_3 = \beta^{r+s}, \]
so \( w = (r, s) \) can be computed such that \((x, w) \in R_L\).

Next we prove that \( \Sigma_L \) has perfect honest-verifier zero-knowledge, by constructing a simulator which on input \((x, e)\), where \( x = (x_1, x_2, x_3) \in L_{R_L}, \) outputs a conversation \( ((a_1^*, a_2^*, a_3^*), e, (z_1^*, z_2^*))\), which is identically distributed as conversations between honest \( P \) and \( V \) on input \( x \) and with challenge \( e \). The simulator \( S_L \) runs as follows on input \((x, e)\):

1. Choose \( z_1^*, z_2^* \leftarrow \{0, \ldots, p-1\} \).
2. Let \( a_1^* = \alpha_1^{z_1^*} x_1^{-e}, \quad a_2^* = \alpha_2^{z_2^*} x_2^{-e}, \quad a_3^* = \beta^{z_1^*+z_2^*} x_3^{-e} \).
3. Output \((a_1^*, a_2^*, a_3^*), e, (z_1^*, z_2^*)\).

Assuming that \((x_1, x_2, x_3) = (a_1^*, a_2^*, \beta^{r+s})\), we have
\[ a_1^* = \alpha_1^{z_1^*-re}, \quad a_2^* = \alpha_2^{z_2^*-se}, \quad a_3^* = \beta^{(z_1^*-re)+(z_2^*-se)} \]
Letting \( r' = z_1^*-re \) and \( s' = z_2^*-se \) we observe that the only difference between a conversation between \( P \) and \( V \) and a conversation produced by the simulator is the following: In the real conversation, the exponents \( r' \) and \( s' \) are randomly chosen, and determine \( z_1 \) and \( z_2 \) by \( z_1 = r'+re \) and \( z_2 = s'+se \), while in the fake conversation, \( z_1^* \) and \( z_2^* \) are randomly chosen, and determine \( r' \) by \( r' = z_1^*-re \) and \( s' \) by \( s' = z_2^*-se \). This means that the two conversations are identically distributed, and we conclude that \( \Sigma_L \) has perfect honest-verifier zero-knowledge.

**Appendix C. The Protocol \( \Sigma_{OR} \)**

In this section we describe the protocol \( \Sigma_{OR} \) constructed from two instances of the protocol \( \Sigma_L \). For a commitment \((y_1, y_2, y_3)\), the goal is to prove knowledge of \((r, s)\) such that either \(((y_1, y_2, y_3), (r, s)) \in R_L \) or \(((y_1, y_2, \frac{y_3}{y}), (r, s)) \in R_L \), without revealing which is the case. That is, a suitable relation \( R_{OR} \) can be defined by

\[ (x, w) = ((y_1, y_2, y_3), (r, s)) \in R_{OR} \Leftrightarrow ((y_1, y_2, y_3), (r, s)) \in R_L \lor ((y_1, y_2, \frac{y_3}{y}), (r, s)) \in R_L. \]
We note that, if \((y_1, y_2, y_3) \in L_R\), then we have \((y_1, y_2, \frac{u_2}{u}) \notin L_R\) with overwhelming probability, and vice versa. In any case, the simulator \(S_L\) defined above is applicable to both \((y_1, y_2, y_3)\) and \((y_1, y_2, \frac{u_2}{u})\) and produces accepting conversations for both statements.

The protocol \(\Sigma_{OR}\) is defined as follows for the case where \(((y_1, y_2, y_3), (r, s)) \in R_L\). In this case, the prover \(P\) and the verifier \(V\) get \(x = (y_1, y_2, y_3)\) as common input, and \(P\) gets \(w = (r, s)\) as private input.

1. \(P\) computes the first message \((a_1, a_2, a_3)\) in \(\Sigma_L\), using \((y_1, y_2, y_3)\) as input. He also chooses a random challenge \(e_1 \in \{0, \ldots, p - 1\}\) and runs the simulator \(S_L\) on input \(((y_1, y_2, \frac{u_2}{u}), e_1)\) to obtain a conversation \(((a_1^*, a_2^*, a_3^*), e_1, (z_1^*, z_2^*))\). He sends \(a = ((a_1, a_2, a_3), (a_1^*, a_2^*, a_3^*))\) to \(V\).

2. \(V\) chooses a random challenge \(e \in \{0, \ldots, p - 1\}\) and sends \(e\) to \(P\).

3. \(P\) lets \(e_0 = (e - e_1) \mod p\) and computes the response \((z_1, z_2)\) to the challenge \(e_0\) according to \(\Sigma_L\), using \((r, s)\) and the pair of random elements \((r', s')\) used in the construction of \((a_1, a_2, a_3)\). He sends \(z = (e_0, z_1, z_2, e_1, z_1^*, z_2^*)\) to \(V\).

4. \(V\) checks that \(e_0 + e_1 \equiv e \mod p\) and that \(((a_1, a_2, a_3), e_0, (z_1, z_2))\) and \(((a_1^*, a_2^*, a_3^*), e_1, (z_1^*, z_2^*))\) are accepting conversations in \(\Sigma_L\) on inputs \((y_1, y_2, y_3)\) and \((y_1, y_2, \frac{u_2}{u})\), respectively, and accepts or rejects accordingly.

For the case where \(((y_1, y_2, \frac{u_2}{u}), (r, s)) \in R_L\), \(P\) computes \((a_1, a_2, a_3)\) according to \(\Sigma_L\), using \(x = (y_1, y_2, \frac{u_2}{u})\) as input, picks \(e_0\) at random and runs \(S_L\) on input \(((y_1, y_2, y_3), e_0)\) to obtain a conversation \(((a_1^*, a_2^*, a_3^*), e_0, (z_1^*, z_2^*))\). He sends \(a = ((a_1, a_2, a_3), (a_1^*, a_2^*, a_3^*))\) to \(V\). Correspondingly, upon receiving \(e\) from \(V\), he lets \(e_1 = (e - e_0) \mod p\), computes the response \((z_1, z_2)\) to the challenge \(e_1\) according to \(\Sigma_L\), using \((r, s)\) and the pair of random elements \((r', s')\) used in the construction of \((a_1, a_2, a_3)\), and sends \(z = (e_0, z_1, z_2, e_1, z_1^*, z_2^*)\) to \(V\).

It is straightforward to verify that \(\Sigma_{OR}\) is complete. As for special soundness, assume that, for some \((y_1, y_2, y_3)\), there are accepting conversations \((a, e, z)\) and \((a, e', z')\) for challenges \(e\) and \(e'\) such that \(e \neq e'\). Assume that \(z\) contains challenges \(e_0\) and \(e_1\) such that \(e_0 + e_1 = e\) and that \(z'\) contains challenges \(e'_0\) and \(e'_1\) such that \(e'_0 + e'_1 = e'\). Since \(e \neq e'\), we must have \(e_0 \neq e'_0\) or \(e_1 \neq e'_1\).

Assume that \(e_0 \neq e'_0\). Then, for \((y_1, y_2, y_3)\), there are two accepting conversations in \(\Sigma_L\) with the same first message but with distinct challenges. Since \(\Sigma_L\) has special soundness, we can then compute \((r, s)\) such that \(((y_1, y_2, y_3), (r, s)) \in R_L\), which means that \(((y_1, y_2, y_3), (r, s)) \in R_{OR}\). A similar argument holds if \(e_1 \neq e'_1\).

Next we show that \(\Sigma_{OR}\) has special honest-verifier zero-knowledge. To this end, we consider the simulator \(S_{OR}\), which on input \((x, e)\) such that \(x = (y_1, y_2, y_3)\) runs as follows:

1. Choose a random challenge \(e_0 \in \{0, \ldots, p - 1\}\), and let \(e_1 = (e - e_0) \mod p\).
(2) Run the simulator $S_L$ on input $((y_1, y_2, y_3), e_0)$ and obtain $((a_1, a_2, a_3), e_0, (z_1, z_2))$.
(3) Run the simulator $S_L$ on input $((y_1, y_2, \frac{w_1}{u}), e_1)$ and obtain $((a'_1, a'_2, a'_3), e_1, (z'_1, z'_2))$.
(4) Output $((a_1, a_2, a_3), (a'_1, a'_2, a'_3), e, (e_0, (z_1, z_2), e_1, (z'_1, z'_2))$.

Assume that $(y_1, y_2, y_3) \in L_{OR}$, and in particular that $(y_1, y_2, y_3) \in L_{RL}$. In this case, it follows from perfect honest-verifier zero-knowledge of $\Sigma_L$ that the conversation produced in step 2 is identically distributed as the corresponding conversation produced by $P$ and $V$ in $\Sigma_{OR}$. As for the conversation in step 3, the corresponding conversation in $\Sigma_{OR}$ is also produced by $S_L$ on input $((y_1, y_2, \frac{w_1}{u}), e_1)$, so it is clear that these conversations are identically distributed. A similar argument holds if $(y_1, y_2, \frac{w_1}{u}) \in L_{RL}$. It follows that conversations output by $S_{OR}$ are identically distributed as conversations between honest $P$ and $V$ in $\Sigma_{OR}$, and we conclude that $\Sigma_{OR}$ has perfect honest-verifier zero-knowledge.

**APPENDIX D. PROOF OF THEOREM 7**

Correctness of $\text{compile}(\Sigma_{OR})$ is straightforward to verify given that $\Sigma_{OR}$ is complete. As for zero-knowledge, consider the algorithm $M$ running as follows: First, recall that $M$ receives $V$’s secret key, and in particular the challenge $e$, as input. Accordingly, to simulate a proof for $x = (y_1, y_2, y_3)$, $M$ runs the special honest-verifier simulator for $\Sigma_{OR}$ on input $(x, e)$ to obtain $(a, e, z) = (a, e, (z_1, \ldots, z_6))$. Then, for $i$ such that $1 \leq i \leq 6$, $M$ chooses a random $r_i \in \mathbb{Z}_n^*$, computes $c_i \leftarrow \text{Enc}_P(z_i, r_i)$ and outputs $\pi = (a, (c_1, \ldots, c_6))$. It now follows from perfect honest-verifier zero-knowledge of $\Sigma_{OR}$ that $\text{compile}(\Sigma_{OR})$ has perfect zero-knowledge.

As for soundness, we slightly modify the definition in [17] so it better suits our application. We consider the following experiment, where $\tilde{P}$ is a probabilistic, polynomial-time adversary:

1. $(sk_{KS}, pk_{KS}) \leftarrow KS(1^\tau)$
2. Run $\tilde{P}$ on input $(1^\tau, pk_{KS})$.
3. Repeat until $\tilde{P}$ stops: $\tilde{P}$ outputs $(x_i, \pi_i), 1 \leq i \leq m(k)$, for some polynomial $m$. Run $V(1^\tau, x_i, \pi_i, pk_{KS}), 1 \leq i \leq m(k)$. If $V(1^\tau, x_i, \pi_i, sk_{KS}) = 1$ for all $i$, then give 1 to $\tilde{P}$, otherwise give 0 to $\tilde{P}$.

$\tilde{P}$ is said to win if he produces at least one pair $(x_j, \pi_j)$, where $x_j \notin L_R$, but the corresponding collection of statement/proof pairs is still accepted in the above experiment. The system is sound if no $\tilde{P}$ wins with probability non-negligible in $\tau$. We say that the system is sound for $n(\tau)$ executions if $\tilde{P}$ always stops after at most $n(\tau)$ repetitions in step 3 of the above experiment.

In the original definition, $\tilde{P}$ is only allowed to output one statement/proof pair at a time. The reason why we allow for a polynomial number of pairs is that the prover in our blind signature scheme does not learn whether each pair is accepted or not.
It remains to prove that, under Assumption 1, $\text{compile}(\Sigma_{\text{OR}})$ is sound for $O(\log \tau)$ executions, according to the above definition. We refer to [17] for definitions of the involved terms. First we note that, under Assumption 1, $\mathcal{H}_{\text{Paillier}}$ is 2-harder than any fake-proof generator for $\Sigma_{\text{OR}}$. Assume that there is a dishonest prover $\tilde{P}$ breaking the soundness of $\text{compile}(\Sigma_{\text{OR}})$. We start by outlining the strategy of the proof: We will use $\tilde{P}$ to obtain a fake-proof generator $\tilde{G}_{\Sigma_{\text{OR}}} = \langle \tilde{G}_{\Sigma_{\text{OR}}}, \hat{g}_{\Sigma_{\text{OR}}} \rangle$ for $\Sigma_{\text{OR}}$. Then, using $\tilde{P}$ and an algorithm $A$ that completely breaks $\tilde{G}$ on instances of size $\tau$, we will construct an algorithm $A'$ that breaks $\mathcal{H}_{\text{Paillier}}$ on instances of size $2\tau$, with runtime comparable to that of $A$. This contradicts the assumption that $\mathcal{H}_{\text{Paillier}}$ is 2-harder than any fake-proof generator for $\Sigma_{\text{OR}}$.

The algorithm $\tilde{G}_{\Sigma_{\text{OR}}}$ takes $1^\tau$ as input and generates a public key $(pk, c)$ according to the protocol. $\tilde{G}_{\Sigma_{\text{OR}}}$ then runs $\tilde{P}$ on input $(1^\tau, (pk, c))$. Whenever $\tilde{P}$ outputs a collection of statement/proof pairs $(x_i, \pi_i)$, where $1 \leq i \leq m(\tau)$, for some polynomial $m$, $\tilde{G}_{\Sigma_{\text{OR}}}$ replies with a random bit. When $\tilde{P}$ stops, $\tilde{G}_{\Sigma_{\text{OR}}}$ uniformly chooses one of the statement/proof pairs, say $(x, \pi)$, which will be on the form $(x, (a, (c_1, \ldots, c_b)))$, and outputs $(x, a)$.

Note that, if $\tilde{P}$ outputs $O(\log \tau)$ collections of statement/proof pairs, there is a non-negligible probability that all of the bits sent to $\tilde{P}$ by $\tilde{G}_{\Sigma_{\text{OR}}}$ are identical to the bits sent to $\tilde{P}$ by the verifier in the real protocol. Moreover, since $\tilde{P}$ breaks soundness, there is a non-negligible probability that $\tilde{P}$ produces at least one pair $(x_i, \pi_i)$ such that $V(1^\tau, x, \pi_i, sk_{KS}) = 1$, where $x_i \notin L_R$. If there is such a pair, there is a non-negligible probability that $\tilde{G}_{\Sigma_{\text{OR}}}$ chooses this particular pair to produce his output. Hence, with overall non-negligible probability, $\tilde{G}_{\Sigma_{\text{OR}}}$ outputs $(x, a)$, such that $x \notin L_R$, and for which there exists exactly one challenge $e$ such that, for some $z$, the conversation $(a, e, z)$ would be an accepting conversation for $\Sigma_{\text{OR}}$. This $e$ must be identical to the plaintext encrypted inside $c$, since the corresponding proof would be accepted by the verifier in the compiled protocol.

Now, letting $\hat{g}_{\Sigma_{\text{OR}}}(x, a)$ be this unique $e$ if such an $e$ exists and $x \notin L_R$, and $0^\tau$ otherwise, we see that $\tilde{G}_{\Sigma_{\text{OR}}} = \langle \tilde{G}_{\Sigma_{\text{OR}}}, \hat{g}_{\Sigma_{\text{OR}}}, \hat{g}_{\Sigma_{\text{OR}}} \rangle$ defines a fake-proof generator for $\Sigma_{\text{OR}}$. Accordingly, Assumption 6 implies in particular that $\mathcal{H}_{\text{Paillier}}$ is 2-harder than $\tilde{G}_{\Sigma_{\text{OR}}}$.

Finally, given $A$ that completely breaks $\tilde{G}_{\Sigma_{\text{OR}}}$ on instances of size $\tau$ in time $T(\tau)$, we construct $A'$ breaking $\mathcal{H}_{\text{Paillier}}$ on instances of size $2\tau$ as follows: On input a $2\tau$-instance $(pk, c)$ for $\mathcal{H}_{\text{Paillier}}$, $A'$ runs $\tilde{P}$ on input $(pk, c)$, and obtains a pair $(x, a)$ exactly as described above for $\tilde{G}_{\Sigma_{\text{OR}}}$. $A'$ then runs $A$ on input $(x, a)$, and outputs the value $e$ returned by $A$.

Since $A$ is assumed to break $\mathcal{H}_{\text{Paillier}}$ completely, there is a non-negligible probability that $A'$ returns the plaintext encrypted inside $c$, and since $A'$ runs in time $T(\tau) + poly(\tau)$, we have a contradiction to the assumption that $\mathcal{H}_{\text{Paillier}}$ is 2-harder than any fake-proof generator for $\Sigma_{\text{OR}}$. 


Appendix E. Proof of Theorem 8

Proof. We assume that there is an adversary $A_{BS}^{nf}$ against the nonforgeability of our blind signature scheme, that engages in $l$ protocols where Signer outputs completed, runs in time $t$ and has advantage $\epsilon$, and construct an adversary $A_{W}^{nf}$ against the UF-CMA security of Waters’ scheme, that queries its real-or-random oracle $l$ times, runs in time $t + poly(\tau)$, and has advantage at least $(1 - \rho)\epsilon$, where $q' = l = O(\log \tau)$ and $\rho$ is a negligible function in $\tau$.

$A_{W}^{nf}$ receives $(1^\tau, pk_W)$ as input. Instead of honestly choosing $crs$ by running $D(1^\tau)$, he generates a keypair for the LE scheme, and uses the obtained public key to construct $crs$, i.e. if $pk_L = (\alpha_1, \alpha_2, \beta)$, he lets $crs = \alpha_1||\alpha_2||\beta$. He runs $KS(1^\tau)$ and obtains $(pk_{KS}, sk_{KS})$, and then he runs $A_{BS}^{nf}$ on input $(1^\tau, crs, pk_{KS}, pk_W)$.

When $A_{BS}^{nf}$ outputs a pair $(c, \pi)$, $A_{W}^{nf}$ verifies $\pi$ exactly as Signer would do in the protocol. In addition, using his secret key for the LE scheme, he decrypts each of the commitments $T_i$ and $M_i$, and obtains $t$ and $m$. He sends $m$ to the signing oracle $O^S$, and receives $(\sigma_1^*, \sigma_2^*)$. He lets $\sigma_1' = \sigma_1^*\sigma_2^t$, $\sigma_2' = \sigma_2^*$, and sends $(\sigma_1', \sigma_2')$ to $A_{BS}^{nf}$. Note that, since $(\sigma_1', \sigma_2') = (g_2^r (u' \prod_{i=1}^n u_i^{m_i})^{\tau}, g^r)$, for some randomly chosen $r \in \{0, \ldots, p - 1\}$, we have $(\sigma_1', \sigma_2') = (g_2^r (u' g_i^{\prod_{i=1}^n u_i^{m_i}})^r, g^r)$. Moreover, when the verification procedure involves checking that $Enc_L(c, r^*, s^*) = \prod_{i=0}^{l-1} T_i^2 \prod_{i=1}^n M_i$, it is guaranteed that $c = g^r \prod_{i=1}^n u_i^{m_i}$, so $(\sigma_1', \sigma_2')$ has the right form.

In order to simplify the proof, we define the event $E$ as follows: Some pair $(c^*, \pi^*)$ output by $A_{BS}^{nf}$ is accepted according to the protocol, but some commitment included in $\pi^*$ does not decrypt to a bit.

We observe that $A_{BS}^{nf}$’s environment in $Exp_{BS,A}^{nf}(\tau)$ is perfectly simulated unless $E$ occurs. Assuming that $E$ occurs, we show how to construct a prover $\tilde{P}$ breaking the soundness of $Compile(\Sigma_{OR})$. $\tilde{P}$ receives $(1^\tau, pk_{KS})$ as input. He runs $D(1^\tau)$ and obtains $crs$, generates a keypair $(pk_{BS}, sk_{BS})$ for the blind signature scheme according to the protocol, and runs $A_{BS}^{nf}$ on input $(1^\tau, crs, pk_{BS}, pk_{BS})$.

When $A_{BS}^{nf}$ outputs a pair $(c, \pi)$, $\tilde{P}$ checks that $Enc_L(c, r^*, s^*) = \prod_{i=0}^{l-1} T_i^2 \prod_{i=1}^n M_i$ exactly as Signer would do in the protocol, and outputs the statement/proof pairs $(T_i, \pi_T)$, $0 \leq i \leq \tau - 1$, and $(M_i, \pi_M), 1 \leq i \leq n$. He accepts the proof only if he receives 1. He constructs $(\sigma_1', \sigma_2')$ exactly as Signer would do in the protocol. We observe that, until the point where $E$ occurs, $A_{BS}^{nf}$’s environment in $Exp_{BS,A}^{nf}(\tau)$ is perfectly simulated. Moreover, when $E$ occurs, $\tilde{P}$ outputs some statement/proof pair, say $(x^*, \pi^*)$, such that $x^* \notin L_{ROr}$, but still the corresponding collection of statement/proof pairs is accepted by the verifier. This means that $\tilde{P}$ wins, which according to Theorem 7 happens with negligible probability. Hence, denoting by $Pr[E]$ the probability that $E$ occurs, we get

$$Pr[E] \leq \rho,$$

where $\rho$ is a negligible function in $\tau$.

Assume that $A_{BS}^{nf}$ outputs message/signature pairs $(m_i, \sigma(m_i))$, $1 \leq i \leq k$, after $A_{W}^{nf}$, acting as Signer, has output completed $l$ times, and assume that $A_{BS}^{nf}$
wins, that is, all the signatures are valid, no two messages are equal, and \(k > l\). This means that \(A_{WS}^\text{nf}\) has sent \(l\) queries to its signing oracle. Consequently, there is a pair, say \((m^*, \sigma^*)\), where \(m^*\) has not been queried to the signing oracle. \(A_{WS}^\text{nf}\) outputs \((m^*, \sigma^*)\), and we conclude that, unless \(E\) occurs, and if \(A_{BS}^\text{nf}\) wins, then \(A_{WS}^\text{nf}\) wins. Hence we get

\[
\text{Succ}_{BS,A}^\text{nf} \geq (1 - \rho)\epsilon,
\]

and the proof is complete.

\[\square\]

\section*{Appendix F. Proof of Theorem 9}

\textit{Proof.} We assume that there is an adversary \(A_{BS}^b\) against the blindness of our blind signature scheme, that runs in time \(t\) and has advantage \(\epsilon\), and construct an adversary \(A_{L}\) against the ROR-CPA security of the LE scheme, that queries its real-or-random oracle once, runs in time \(t + \text{poly}(\tau)\), and has advantage at least \(\frac{\epsilon}{2nT + 2\tau}\). To this end, we define a series of games as follows:

\begin{enumerate}
\item \(A_{BS}^b\) runs in the experiment \(\text{Exp}_{BS,A}^b(\tau)\), where \(b = 0\).
\item Game 0: \(A_{BS}^b\) runs in the experiment \(\text{Exp}_{BS,A}^b(\tau)\), where \(b = 0\).
\item Game 1: This game is the same as Game 0, but \(A_{BS}^b\)’s environment is now simulated by \(P\). That is, \(P\) runs \(D(1^\tau)\) and obtains \(\text{crs}\), and runs \(A_{BS}^b\) on input \((1^\tau, \text{crs})\). When \(A_{BS}^b\) outputs \((pk, r, (m_0, m_1))\), \(P\) runs \(KS(1^\tau, r)\) and obtains \((sk_{KS}, pk_{KS})\). He simulates \(\text{User}(pk, m_0)\) and \(\text{User}(pk, m_1)\), and if \(\text{User}(pk, m_0)\) outputs \(\sigma(m_0)\) and \(\text{User}(pk, m_1)\) outputs \(\sigma(m_1)\), then \(P\) gives \((\sigma(m_0), \sigma(m_1))\) to \(A_{BS}^b\).

From \(A_{BS}^b\) ’s point of view, there is no difference between Game 0 and Game 1. Therefore, if we let \(G_i\) denote the output of \(A_{BS}^b\) when taking part in Game \(i\), we have

\[
\left| \Pr[G_1 = 1] - \Pr[G_0 = 1] \right| = 0.
\]

\item Game 2: This game is the same as Game 1, with the following modification: When acting as \(\text{User}(pk, m_0)\) and \(\text{User}(pk, m_1)\), instead of honestly generating the proofs \(\pi_{T_i}\) and \(\pi_{M_i}\), \(P\) uses the simulator \(M\) described in the proof of Theorem 7. Note that the input \(sk_{KS}\) of \(M\) is obtained by \(P\), since \(P\) receives the random coins \(r\) used in the key generation.

Since \textit{compile}(\(\Sigma_{OR}\)) has perfect zero-knowledge, proofs generated by \(M\) has exactly the same distribution as honestly generated proofs, so we have

\[
\left| \Pr[G_2 = 1] - \Pr[G_1 = 1] \right| = 0.
\]

\item Game 3: This game is the same as Game 2, with the following modification: When acting as \(\text{User}(pk, m_0)\) and \(\text{User}(pk, m_1)\), instead of honestly generating all the commitments \(T_i\) and \(M_i\) and the opening \((r^*, s^*)\), \(P\) randomly selects \(r^*, s^* \in \{0, \ldots, p-1\}\), and lets \(C = \text{Enc}_k(g^t \prod_{i=1}^n w_{i}^{m_i}, r^*, s^*)\). The commitments \(T_0, \ldots, T_{\tau-1}\) and \(M_1, \ldots, M_{n-1}\) are generated exactly as before, while

\[
M_n = \frac{C}{\prod_{i=0}^{\tau-1} T_i^{2i} \prod_{i=1}^{n-1} M_i}.
\]


Observe that in Game 3 we have
\[
M_n = (\alpha_1 r^* - \sum_{i=0}^{r-1} r_i, \alpha_2 s^* - \sum_{i=0}^{s-1} s_i, \\
\beta(r^* - \sum_{i=0}^{r-1} r_i + \sum_{i=1}^{n} r'_i) + (s^* - \sum_{i=0}^{s-1} s_i - \sum_{i=1}^{n} s'_i) \tau_{u_n}).
\]

The only difference between Game 2 and Game 3 is the following: In Game 2, \(r_0, \ldots, r_{\tau-1}\) and \(r'_1, \ldots, r'_n\) are randomly chosen, and \(r^* = \sum_{i=0}^{\tau-1} r_i 2^i + \sum_{i=1}^{n} r_i\), while in Game 3, \(r^*\), \(r_0, \ldots, r_{\tau-1}\) and \(r'_1, \ldots, r'_{n-1}\) are randomly chosen, and \(r'_n = r^* - \sum_{i=0}^{\tau-1} r_i 2^i - \sum_{i=1}^{n} r'_i\). The corresponding holds for \(s^*, s_0, \ldots, s_{\tau-1}\) and \(s'_1, \ldots, s_n\). This means that, even if they are generated in a different manner, the distribution of these values is exactly the same in both games, so we have
\[
\left| \Pr[G_3 = 1] - \Pr[G_2 = 1] \right| = 0.
\]

Game 4: This game is the same as Game 3, with the following modification: When acting as \(User(pk, m_0)\) and \(User(pk, m_1)\), instead of letting \(T_0 = Enc_L(g^{t^0}, r_0, s_0)\), \(P\) chooses a random element \(\rho_0 \in G\) and lets \(T_0 = Enc_L(\rho_0, r_0, s_0)\).

In each of the games Game 5, ..., Game \(n + \tau + 3\), one more of the \(g^t\) is replaced by a random element in \(G\). Correspondingly, in each of the games Game \(\tau + 4, \ldots, n + \tau + 2\), one more of the \(u_i^{m_i}\) is replaced by a random element. Thus, in Game \(n + \tau + 2\), \(g^{t^0}, \ldots, g^{t^{\tau-1}}\) and \(u_1^{m_1}, \ldots, u_{n-1}^{m_{n-1}}\) are all replaced by random elements.

Game \(n + \tau + 3\): This game is the same as Game \(n + \tau + 2\), with the following modification: When acting as \(User\), \(P\) swaps \(m_0\) and \(m_1\).

Let \((c_1, \pi_1)\) and \((c_2, \pi_2)\) denote the messages sent by \(User\) in the first and the second protocol, respectively. The view of \(A_{BS}^t\) in a game is then described by \(((c_1, c_2), (\pi_1, \pi_2), (\sigma(m_0), \sigma(m_1)))\) if \(User\) outputs completed in both protocols, and \(((c_1, c_2), (\pi_1, \pi_2))\) otherwise. We argue that the distribution of \(A_{BS}^t\)'s view in Game \(n + l + 3\) is negligibly close to the distribution of \(A_{BS}^t\)'s view in Game \(n + l + 2\). First, we note that the proofs \(\pi_1\) and \(\pi_2\) are now independent of the message input to \(User\), i.e. \(\pi_1, \pi_2\) has exactly the same distribution in both games. Also, in both games, \(c_1\) and \(c_2\) are both on the form \(g^t \prod_{i=1}^{n} u_i^{m_i} - b\) for some \(b\) and some randomly chosen \(t \in \{0, \ldots, p-1\}\). Since the blinding factor \(g^t\) perfectly hides \(b\), \((c_1, c_2)\) has exactly the same distribution in Game \(n + \tau + 3\) as in Game \(n + \tau + 2\). \((c_1, c_2)\) is also independent of \((\pi_1, \pi_2)\), so the distribution of \(((c_1, c_2), (\pi_1, \pi_2))\) is exactly the same in both games.

Assume that \(A_{BS}^t\)'s response \((\sigma_1', \sigma_2')\) has the correct form in both protocols, i.e. in the first protocol \(\sigma_1' = g_2^{\sigma_2}(u'c_1)^r\) and \(\sigma_2' = g^r\), for some \(r \in \{0, \ldots, p-1\}\), and correspondingly for the second protocol. This means that, in both games, the signatures \(\sigma(m_0)\) and \(\sigma(m_1)\) have the respective forms \((g_2^{\sigma_2}(u' \prod_{i=1}^{n} u_i^{m_i}) u_{r_1 + r'_1}, g^{r_1 + r'_1})\) and \((g_2^{\sigma_2}(u' \prod_{i=1}^{n} u_i^{m_i}) u_{r_2 + r'_2}, g^{r_2 + r'_2})\), for some \(r_1, r_2 \in \{0, \ldots, p-1\}\) randomly chosen by \(User\) in the respective protocols. Since \(r_1\) and \(r_2\) perfectly hides \(r'_1\) and \(r'_2\), \((\sigma(m_0), \sigma(m_1))\) has exactly the same distribution in both games. Also, the distribution of \((\sigma(m_0), \sigma(m_1))\) is independent of both \((\pi_1, \pi_2)\) and \((c_1, c_2)\).
Consequently, in this case, \(((c_1, c_2), (\pi_1, \pi_2), (\sigma(m_0), \sigma(m_1)))\) has exactly the same distribution in both games.

Now assume that, in at least one of the protocols, \(A_{BS}^b\)'s response \((\sigma'_1, \sigma'_2)\) does not have the correct form. For instance, assume that in the first protocol \(\sigma'_1 = g_2^a (u'c_1)'\) and \(\sigma'_2 = g^r\), such that \(r \neq r^* \mod p\). It is easy to show that, for the resulting signature to be accepted, we must have \(u'c_1 = 1\), which is impossible, since \(User\) is required to choose \(t\) such that \(u'c_1 \neq 1\). Thus, \(A_{BS}^b\)'s view is described by \(((c_1, c_2), (\pi_1, \pi_2))\), which, by the above argument, has exactly the same distribution in both games.

Hence we have

\[
\left| \Pr[G_{n+\tau+4} = 1] - \Pr[G_{n+\tau+3} = 1] \right| = 0.
\]

Game \(n + \tau + 4\): This game is the same as Game \(n + \tau + 3\), with the following modification: When acting as \(User(pk, m_0)\) and \(User(pk, m_1)\), instead of choosing a random element \(\rho_0 \in G\) and letting \(T_0 = Enc_L(\rho_0, r_0, s_0)\), \(P\) lets \(T_0 = Enc_L(g^{b_0}, r_0, s_0)\).

In each of the games Game \(n + \tau + 5\), \ldots, Game \(n + 2\tau + 3\), one more of the \(M_i\) is generated by encrypting \(b_i^{\pi_i}\) instead of a randomly chosen element. Correspondingly, in each of the games Game \(n + 2\tau + 4\), \ldots, Game \(2n + 2\tau + 2\), one more of the \(M_i\) is generated by encrypting \(u_i^{b_1}\) instead of a randomly chosen element. Thus, in Game \(2n + 2\tau + 2\), \(T_0, \ldots, T_{l-1}\) and \(M_1, \ldots, M_{l-1}\) are all generated according to the protocol.

Game \(2n + 2\tau + 3\): This game is the same as Game \(2n + 2\tau + 2\), with the following modification: When acting as \(User(pk, m_0)\) and \(User(pk, m_1)\), instead of generating the opening \((r^*, s^*)\) and the commitment \(M_n\) as described in Game 3, \(P\) generates \((r^*, s^*)\) and \(M_n\) according to the protocol.

By arguing as for Game 2 and Game 3, we have

\[
\left| \Pr[G_{2n+2\tau+3} = 1] - \Pr[G_{2n+2\tau+2} = 1] \right| = 0.
\]

Game \(2n + 2\tau + 4\): This game is the same as Game \(2n + 2\tau + 3\), with the following modification: When acting as \(User(pk, m_0)\) and \(User(pk, m_1)\), instead of using the simulator \(M\) when generating the proofs \(\pi_{T_i}\) and \(\pi_{M_i}\), as described in Game 2, \(P\) generates these proofs according to the protocol.

Again, since \text{compile}(\Sigma_{OR})\) has perfect zero-knowledge, we have

\[
\left| \Pr[G_{2n+2\tau+4} = 1] - \Pr[G_{2n+2\tau+3} = 1] \right| = 0.
\]

Game \(2n + 2\tau + 5\): In this game \(A_{BS}^b\) runs in the experiment \(\text{Exp}_{BS,A}^b(\tau)\), where \(b = 1\).

In Game \(2n + 2\tau + 4\), \(P\) perfectly simulates \(A_{BS}^b\)'s environment in \(\text{Exp}_{BS,A}^b(\tau)\), where \(b = 1\), so we have

\[
\left| \Pr[G_{2n+2\tau+5} = 1] - \Pr[G_{2n+2\tau+4} = 1] \right| = 0.
\]

We now construct an adversary \(A_{L}^{\text{ror}}\) trying to break the real-or-random blindness of linear encryption. \(A_{L}^{\text{ror}}\) receives \(pk_L = (\alpha_1, \alpha_2, \beta)\) as input, and randomly
chooses an index $i \in \{3, \ldots, n + \tau + 1, n + \tau + 3, \ldots, 2n + 2\tau + 1\}$. First, assume that $i = 3$. $A_{L}^{\text{ofr}}$ then behaves exactly as $P$ in Game 3, with the following exceptions: $A_{L}^{\text{ofr}}$ lets $crs = \alpha_1 || \alpha_2 || \beta$, and instead of generating $T_0$ as in Game 3, $A_{L}^{\text{ofr}}$ sends $g_0^0$ to its real-or-random oracle, receives $T_0'$, and lets $T_0 = T_0'$. Finally, when $A_{BS}^b$ outputs a bit $b'$, $A_{L}^{\text{ofr}}$ outputs $b'$.

We observe that, if $A_{L}^{\text{ofr}}$’s challenge bit $b = 0$, then $A_{BS}^b$’s environment in Game 4 is perfectly simulated, while if $b = 1$, $A_{BS}^b$’s environment in Game 3 is perfectly simulated. Hence we get

$$\left| \Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0] \right| = \left| \Pr[G_3 = 1] - \Pr[G_4 = 1] \right|.$$

Defining the behavior of $A_{L}^{\text{ofr}}$ for each index $i$ in an analogous manner, and taking into account that $i$ is randomly chosen among less than $2n + 2\tau$ possible values, we get

$$\text{Adv}_{LE,A_{L}^{\text{ofr}}}^{\text{ofr}}(\tau) \geq \frac{1}{2n + 2\tau} \left( \sum_{i=3}^{n + \tau + 1} \left| \Pr[G_i = 1] - \Pr[G_{i+1} = 1] \right| + \sum_{i=n + \tau + 3}^{2n + 2\tau + 1} \left| \Pr[G_{i+1} = 1] - \Pr[G_i = 1] \right| \right).$$

We have shown that, for $i \in \{0, 1, 2, n + \tau + 2, 2n + 2\tau + 2, 2n + 2\tau + 3, 2n + 2\tau + 4\}$, we have

$$\left| \Pr[G_{i+1} = 1] - \Pr[G_i = 1] \right| = 0.$$

Hence we may write

$$\text{Adv}_{LE,A_{L}^{\text{ofr}}}^{\text{ofr}}(\tau) \geq \frac{1}{2n + 2\tau} \sum_{i=0}^{2n + 2\tau + 5} \left| \Pr[G_{i+1} = 1] - \Pr[G_i = 1] \right|$$

$$\geq \frac{1}{2n + 2\tau} \sum_{i=0}^{2n + 2\tau + 5} \Pr[G_{i+1} = 1] - \Pr[G_i = 1]$$

$$= \frac{1}{2n + 2\tau} \left| \Pr[G_{2n + 2\tau + 5} = 1] - \Pr[G_0 = 1] \right|$$

$$= \frac{1}{2n + 2\tau} \text{Adv}_{BS,A_{L}^b}^b(\tau),$$

which completes the proof. □
Paper IV

A Universally Composable Anonymous Online Service

Lillian Kråkmo

Preprint
A UNIVERSALLY COMPOSABLE ANONYMOUS ONLINE SERVICE

LILLIAN KRÅKMO

Abstract. In the framework of universally composable security, we define an ideal functionality for an anonymous online service. Moreover, we propose a dedicated hybrid protocol, and prove that it realizes the functionality. The protocol uses functionalities for several cryptographic primitives, including a functionality for an anonymous secure channel. We show how such a functionality can be realized, by adapting a secure channel based on the KEM-DEM framework. Our constructions may be viewed as case study protocols, exemplifying protocols of realistic complexity within this security model.

1. Introduction

Universally composable (UC) security is a framework proposed by Canetti [2] as a way to define security for protocols such that security-preserving composition is possible. This allows for a modular design and analysis of protocols. For each cryptographic task, security is defined in terms of an ideal functionality, which incorporates the required properties of a protocol for the task and the allowed actions of an adversary. A protocol is said to securely realize the ideal functionality if, loosely speaking, any effect caused by an adversary attacking the protocol can be obtained by an adversary attacking the ideal functionality. When designing complex protocols, one can allow the involved parties to have secure access to ideal functionalities. Then, when implementing the protocol, each ideal functionality can be replaced by a protocol securely realizing the functionality. The composition theorem then guarantees security. We refer to [2] for a complete overview of this framework.

In Section 2 of this paper, we define a functionality for an anonymous online service. In order to realize this functionality, we need a functionality for a secure channel, which is anonymous in one end. Such a functionality is presented in Section 3. In this section, we also propose a protocol realizing an anonymous secure channel, based on the KEM-DEM framework. Our protocol adapts the construction of Nagao et al. [6] to a setting where one of the involved parties wishes to remain anonymous, by using an additional functionality for an anonymous network. Finally, in Section 4, we present our protocol for an anonymous online service, and prove that it realizes our functionality.
2. The Anonymous Online Service Functionality

The functionality for an anonymous online service, FAOS, is presented in Figure 1. It involves a bank $B$, a server $Q$ and users $P_1, \ldots, P_n$. The purpose of the functionality is to allow for anonymous purchase of services, by means of establishing a secure channel between the server and a user, which is anonymous at the user’s end. Payments are handled as follows: Each time the user initiates a session with the server, a payment request is issued to the bank, which answers by either yes or no. If the answer is yes, then a session is established, and the server’s balance is increased. Furthermore, the server may make deposits, whereupon his balance is decreased, and the bank is duly notified.

3. A Universally Composable Anonymous Secure Channel

In order to realize our functionality for an anonymous online service, we need a functionality for a secure channel. This functionality, which is denoted by $F_{ASC}$, is defined in Figure 2. It resembles the functionality for a secure channel suggested by Canetti and Krawczyk [3], except that the session initiator is anonymized. Also, while the functionality in [3] is designed to handle a single communication session, and the security in the case of multiple sessions is argued by the UC with joint state theorem [4], our functionality immediately handles multiple sessions for a single pair of users. Although the former approach allows for a more modular analysis and may be preferable in the general case, the latter turns out to be more convenient for our application. In particular, the anonymous online service functionality is designed to handle multiple sessions between the server and a user, while requiring only one registration by the server.

Another modification from the formulation in [3], is that data exchange in the case where either the sender or the receiver is corrupt, is explicitly handled by the functionality. In detail, if one of the involved parties is corrupt, the message to be sent is revealed to the ideal adversary. Keeping in mind that the ideal adversary receives all inputs meant for corrupt parties anyway, this property is not needed for the simulation of the protocol to work. However, it may be argued that this formulation more intuitively represents the security requirements for an anonymous secure channel protocol. Furthermore, every time a session is established, our functionality notifies the involved parties. This feature allows for a clearer presentation of protocols using $F_{ASC}$ as a subroutine, as parties may be instructed to wait until a session is established before they start sending data across the channel.

Note that, in $F_{ASC}$, it may happen that the ideal adversary receives two messages on the form $(ASC.\text{Send}, sid, ssid, P_i, \text{partner}, |m|)$ before a message $(ASC.\text{Send}, sid, ssid, P_i, \text{partner}, \text{ack})$ is sent in the opposite direction. In this case, upon the first $(ASC.\text{Send}, sid, ssid, P_i, \text{partner}, \text{ack})$ from the ideal adversary, the functionality delivers the message $m$ that was first received from $P_i$. That is, the ideal adversary is not allowed to change the order of messages sent across the channel.
In [6], Nagao et al. propose a universally composable secure channel based on the KEM-DEM framework. In this section, we show how their hybrid protocol can be adapted to fit our purpose, by using an additional functionality for an anonymous network. This functionality is given in Figure 3. It essentially re-names the registrating party, by choosing a random identity $ID$. In subsequent activations, the functionality redirects messages meant for $ID$ to the registrating party, and messages sent from this party appear to come from $ID$. Furthermore, the functionality $F_{CA}$ used by Nagao et al. is replaced by an anonymized variant, $F_{ACA}$, which is defined in Figure 4. For completeness, the KEM-DEM functionality proposed by Nagao et al. is presented in Figure 5. Finally, our hybrid protocol $\pi_{ASC}$ for an anonymous secure channel is described in Figure 6.

We highlight the fact that, since the anonymous network functionality is designed to handle only one registration, a new copy of the functionality is invoked for each session. When implementing the protocol, these copies can be replaced by a single protocol handling multiple registrations. Security is then guaranteed by the UC with joint state theorem [4].

Since the functionality $F_{ASC}$ ensures that messages sent across the channel arrive at the recipient in the correct order, the protocol $\pi_{ASC}$ must also provide this guarantee. This is obtained by having each party maintain two variables, ssid $- sn$ and ssid $- psn$, to be interpreted as “sequence number” and “partner’s sequence number”, respectively, for the session with identifier ssid. By having the sender include the current value of ssid $- sn$ in the encrypted message, the receiver can check if the decrypted message includes the current value of ssid $- psn$, and accept the message only if this is the case.

The remaining part of this section is dedicated to proving the following result.

**Theorem 1.** $\pi_{ASC}$ securely realizes $F_{ASC}$ in the $(F_{ACA}, F_{KEM-DEM}, F_{AN})$-hybrid model under static corruption.

**Proof.** We show that for every adversary $A$ interacting with parties running $\pi_{ASC}$ in the $(F_{ACA}, F_{KEM-DEM}, F_{AN})$-hybrid model, there is an ideal adversary $S$ such that no environment $Z$ can tell whether it is interacting with $A$ and $\pi_{ASC}$ or with $S$ and IDEAL$F_{ASC}$.

As usual, $S$ runs a simulated copy of $A$ and forwards all messages from $Z$ to $A$ and back. When $A$ corrupts a party $P$, $S$ corrupts $\tilde{P}$. When $\tilde{P}$ is corrupt, any input from $Z$ meant for $\tilde{P}$ goes directly to $S$, who forwards it to $A$ on the input tape corresponding to $P$, and the other way around. Moreover, $S$ can send messages to $F$ in the name of $\tilde{P}$, and messages from $F$ meant for $\tilde{P}$ go to $S$. In the following, when an entity $E$ controls another entity $E'$, the notation “$E|E'$” should be read as “$E$, in the name of $E'$”.

**When both $P_i$ and $P_j$ are honest:** $S$ simulates honest $P_i$, honest $P_j$ and functionalities $F_{ACA}, F_{KEM-DEM}$ and $F_{AN}$. 

**When one is corrupt:** $S$ simulates honest $P_i$, honest $P_j$ and functionalities $F_{ACA}, F_{KEM-DEM}$ and $F_{AN}$.
Simulating the set-up: Upon receiving \((\text{ASC}.\text{Register}, \text{id}_1, P_i)\) from \(F_{\text{ASC}}\), \(S|F_{\text{KEM-DEM}}\) sends \((\text{KEM}.\text{KeyGen}, \text{id}_2)\) to \(A\), and waits to receive \((\text{KEM}.\text{Key}, \text{id}_2, \text{pk}_i)\) from \(A\). Then \(S|F_{\text{ACA}}\) sends \((\text{ACA}.\text{Register}, \text{id}_3, P_i, \text{pk}_i)\) to \(A\). Upon receiving \((\text{ACA}.\text{Register}, \text{id}_3, \text{ack})\) from \(A\), \(S|F_{\text{ACA}}\) stores \((P_i, \text{pk}_i)\) and sends \((\text{ASC}.\text{Registered}, \text{id}_1, P_i, \text{ack})\) to \(F_{\text{ASC}}\).

Simulating the session set-up: Upon receiving \((\text{ASC}.\text{EstablishSession}, \text{id}_1, \text{ssid}, P_i)\) from \(F_{\text{ASC}}\), \(S\) chooses a random identity \(ID\) and stores \((ID, ssid)\). Then \(S|F_{\text{ACA}}\) sends \((\text{ACA}.\text{Retrieve}, \text{id}_3, P_i)\) to \(A\). Upon receiving \((\text{ACA}.\text{Retrieve}, \text{id}_3, \text{ack})\) from \(A\), \(S|F_{\text{ACA}}\) checks that there is a stored entry \((P_i, \text{pk}_i)\). \(S|F_{\text{KEM-DEM}}\) then sends \((\text{KEM}.\text{Encrypt}, \text{id}_2, \text{ssid}, \text{pk}_i, C_0)\) from \(A\), and stores \((ID, ssid, C_0, \text{active})\). \(S|F_{\text{AN}}\) then sends \((\text{AN}.\text{Send}, \text{id}_4, \text{ssid}), (\text{ASC}.\text{EstablishSession}, \text{id}_1, \text{ssid}, C_0, ID, P_i)\) to \(A\), and \(S\) sends \((\text{ASC}.\text{EstablishSession}, \text{id}_1, \text{ssid}, \text{partner}, \text{ack})\) to \(F_{\text{ASC}}\).

If \(S|F_{\text{AN}}\) receives \((\text{AN}.\text{Send}, (\text{id}_4, \text{ssid}), m', ID')\) from \(A\), where \(ID'\) represents a corrupt party, then \(S|F_{\text{AN}}\) sends \((\text{AN}.\text{Receive}, (\text{id}_4, \text{ssid}), m')\) to \(A\) on the input tape corresponding to \(ID'\). Otherwise, if \(ID' = P_i\), and \(m'\) is on the form \((\text{ASC}.\text{EstablishSession}, \text{id}_1, \text{ssid}, C_0^*, ID^*)\), then \(S|F_{\text{KEM-DEM}}\) sends \((\text{KEM}.\text{Decrypt}, \text{id}_2, \text{ssid}, C_0^*)\) to \(A\). Upon receiving \((\text{KEM}.\text{Decrypt}, \text{id}_2, \text{ssid}, \text{ack})\) from \(A\), \(S|F_{\text{KEM-DEM}}\) stores \((P_i, \text{ssid}, C_0^*, \text{active})\), and \(S\) sends \((\text{ASC}.\text{EstablishSession}, \text{id}_1, \text{ssid}, P_i, \text{ack})\) to \(F_{\text{ASC}}\).

Simulating the data exchange: Upon receiving \((\text{ASC}.\text{Send}, \text{id}_1, \text{ssid}, P_i, \text{partner}, |M|)\) from \(F_{\text{ASC}}\), \(S\) checks that there is a stored entry \((P_i, \text{ssid}, C_0^*, \text{active})\) for some \(C_0^*\), and sets \(|m| = |M| + l\), where \(l\) is the number of bits used to represent a party identity and a sequence number. \(S|F_{\text{KEM-DEM}}\) then sends \((\text{DEM}.\text{Encrypt}, \text{id}_2, \text{ssid}, |m|)\) to \(A\), and waits to receive \((\text{DEM}.\text{Ciphertext}, \text{id}_2, \text{ssid}, c)\) from \(A\). \(S|F_{\text{AN}}\) then sends \((\text{AN}.\text{Send}, (\text{id}_4, \text{ssid}), c, P_i, ID)\) to \(F_{\text{AN}}\). The next time \(S|F_{\text{AN}}\) receives a message \((\text{AN}.\text{Send}, (\text{id}_4, \text{ssid}), c', ID')\) from \(A\), if \(ID'\) represents a corrupt party, then \(S|F_{\text{AN}}\) sends \((\text{AN}.\text{Receive}, (\text{id}_4, \text{ssid}), c')\) to \(A\) on the input tape corresponding to \(ID'\). Otherwise, if \(ID' = ID\), and there is a stored entry \((ID, ssid, C_0, \text{active})\) for some \(C_0\), then \(S|F_{\text{KEM-DEM}}\) sends \((\text{DEM}.\text{Decrypt}, \text{id}_2, \text{ssid}, c')\) to \(A\), and waits to receive \((\text{DEM}.\text{Plaintext}, \text{id}_2, \text{ssid}, m')\) from \(A\). If \(C_0 = C_0\) and \(c' = c\), then \(S\) sends \((\text{ASC}.\text{Send}, \text{id}_1, \text{ssid}, P_i, \text{partner}, \text{ack})\) to \(F_{\text{ASC}}\).
Upon receiving (ASC.Send, sid_1, ssid, partner, P_i, |M|) from FASC, S checks that there is a stored entry (ID, ssid, C_0, active) for some C_0, and sets |m| = |M| + l, where l is the number of bits used to represent an identity and a sequence number. S[F_KEM-DEM] then sends (DEM.Encrypt, sid_2, ssid, |m|) to A, and waits to receive (DEM.Ciphertext, sid_2, ssid, c) from A. S[F_AN] then sends (AN.Send, (sid_4, ssid), c, ID, P_i) to F_AN. The next time S[F_AN] receives a message (AN.Send, (sid_4, ssid), c', ID') from A, if ID' represents a corrupt party, then S[F_AN] sends (AN.Receive, (sid_4, ssid), c') to A on the input tape corresponding to ID'. Otherwise, if ID' = P_i, and there is a stored entry (P_i, ssid, C_0, active) for some C_0, then S[F_KEM-DEM] sends (DEM.Decrypt, sid_2, ssid, c') to A, and waits to receive (DEM.Plaintext, sid_2, ssid, m') from A. If C_0 = C_0' and c' = c, then S sends (ASC.Send, sid_1, ssid, partner, P_i, ack) to F_ASC.

Simulating the session ending: Upon receiving (ASC.ExpireSession, sid_1, ssid, P_i/ partner) from FASC, S removes (P_i, ssid, C_0, active)/(ID, ssid, C_0, active).

When P_i is corrupt, while P_j is honest: As explained earlier, inputs to S[P_i] are forwarded to A[P_i], and outputs from A[P_i] are output by S[P_i]. S simulates honest P_j and functionalities F_ACA, F_KEM-DEM and F_AN. S can perfectly simulate F_ACA, since all the information obtained by F_ACA in the real protocol is obtained by S in the simulated protocol. Regarding F_AN, S does not obtain P_j’s identity, but this is also perfectly hidden from A, so S can still perfectly simulate A’s view in the real protocol. As for F_KEM-DEM, when P_i is corrupt, S obtains every message m being sent between P_i and P_j. By storing sequence numbers and increasing them as P_i and P_j do in the real protocol, S can perfectly simulate the messages expected by A from F_KEM-DEM. In the following, when we say that A[P_i] or S[ID^*] sends a message to S[F], where F is one of the above functionalities, this implies that S proceeds according to the description of this functionality, and allows A to delay messages, determine keys, redirect messages etc. whenever appropriate.

Simulating the set-up: The first time S[P_i] receives some input, S[P_i] sends (ASC.Register, sid_1) to F_ASC, and upon receiving (ASC.Register, sid_1, P_i) from F_ASC, S sends (ASC.Register, sid_1, ack) to F_ASC.

Simulating the session set-up: Upon receiving (ASC.EstablishSession, sid_1, ssid, P_i) from F_ASC, S sets a variable P_i – ssid – sn equal to 1, and sends (ASC.EstablishSession, sid_1, ssid, P_i, ack) to F_ASC. S chooses a random identity ID^*, stores (ID^*, ssid), and S[ID^*] sends (AN.Register, (sid_4, ssid)) to S[F_AN]. Upon receiving (AN.Registered, (sid_4, ssid), ID) from S[F_AN], S[ID^*] sends (ACA.Retrieve, sid_3, P_i) to S[F_ACA].
Upon receiving \((\text{ACA.Retrieved}, sid_3, P_i, pk_i)\) from \(S|\mathcal{F}_{\text{ACA}}\), \(S|\mathcal{F}_{\text{KEM-DEM}}\) sends \((\text{KEM.Encrypt}, sid_2, ssid, pk_i)\) to \(S|\mathcal{F}_{\text{KEM-DEM}}\). Upon receiving \((\text{EncryptedSharedKey}, sid_2, ssid, pk_i, C_0)\) from \(S|\mathcal{F}_{\text{KEM-DEM}}\), \(S|ID^*\) stores \((ID^*, ssid, \text{active})\), and sets a variable \(ID^* - ssid - sn\) equal to 1. \(S|ID^*\) then sends \((\text{AN.Send}, (sid_4, ssid), (\text{ASC.EstablishSession}, sid_1, C_0, ID), P_i)\) to \(S|\mathcal{F}_{\text{AN}}\), and sends \((\text{ASC.EstablishSession}, sid_1, ssid, partner, ack)\) to \(\mathcal{F}_{\text{ASC}}\).

Simulating the data exchange: When \(A|P_i\) successfully sends a message to \(S|ID^*\) (i.e. when \(S|ID^*\) receives \((\text{AN.Receive}, (sid_4, ssid), c)\) from \(S|\mathcal{F}_{\text{AN}}\), \((ID^*, ssid, \text{active})\) is stored, \(S|ID^*\) sends \((\text{DEM.Decrypt}, sid_2, ssid, c)\) to \(S|\mathcal{F}_{\text{KEM-DEM}}\), receives \((\text{DEM.Plaintext}, sid_2, ssid, m)\) from \(S|\mathcal{F}_{\text{KEM-DEM}}\), and \(m = M||P_i||P_i - ssid - sn\), then \(S\) increases \(P_i - ssid - sn\), \(S|P_i\) sends \((\text{ASC.Send}, sid_1, ssid, M)\) to \(\mathcal{F}_{\text{ASC}}\), and upon receiving \((\text{ASC.Send}, sid_1, ssid, P_i, partner, M)\) from \(\mathcal{F}_{\text{ASC}}\), \(S\) sends \((\text{ASC.Send}, sid_1, ssid, P_i, partner, ack)\) to \(\mathcal{F}_{\text{ASC}}\). Upon receiving \((\text{ASC.Send}, sid_1, ssid, partner, P_i, M)\) from \(\mathcal{F}_{\text{ASC}}\), \(S\) sends \((\text{ASC.Send}, sid_1, ssid, partner, P_i, c, ack)\) to \(\mathcal{F}_{\text{ASC}}\). \(S|ID^*\) sets \(m = M||ID||ID^* - ssid - sn\), increases \(ID^* - ssid - sn\), and sends \((\text{DEM.Encrypt}, sid_2, ssid, m)\) to \(S|\mathcal{F}_{\text{KEM-DEM}}\). Upon receiving \((\text{DEM.Ciphertext}, sid_2, ssid, c)\) from \(S|\mathcal{F}_{\text{KEM-DEM}}\), \(S|ID^*\) sends \((\text{AN.Send}, (sid_4, ssid), c, P_i)\) to \(S|\mathcal{F}_{\text{AN}}\).

Simulating the session ending: Upon receiving \((\text{ASC.ExpireSession}, sid, ssid, partner)\) from \(\mathcal{F}_{\text{ASC}}\), \(S\) removes \((ID^*, ssid, \text{active})\), and \(S|ID^*\) sends \((\text{KEM.ExpireSession}, sid_2, ssid)\) to \(S|\mathcal{F}_{\text{KEM-DEM}}\).

When \(P_i\) is honest, while \(P_j\) is corrupt: As explained earlier, inputs to \(S|\tilde{P}_j\) are forwarded to \(A|P_j\), and outputs from \(A|P_j\) are output by \(S|\tilde{P}_j\). \(S\) simulates honest \(P_i\) and functionalities \(\mathcal{F}_{\text{ACA}}, \mathcal{F}_{\text{KEM-DEM}}\) and \(\mathcal{F}_{\text{AN}}\). \(S\) obtains \(P_j\)'s identity and all the messages sent between \(P_i\) and \(P_j\), so all functionalities can be perfectly simulated. In the following, when we say that \(A|P_j\) or \(S|P_i\) sends a message to \(S|\mathcal{F}\), where \(\mathcal{F}\) is one of the above functionalities, this implies that \(S\) proceeds according to the description of this functionality.

Simulating the set-up: Upon receiving \((\text{ASC.Register}, sid_1)\) from \(\mathcal{F}_{\text{ASC}}\), \(S|P_i\) sends \((\text{KEM.KeyGen}, sid_2)\) to \(S|\mathcal{F}_{\text{KEM-DEM}}\). Upon receiving \((\text{KEM.Key}, sid_2, pk_i)\) from \(S|\mathcal{F}_{\text{KEM-DEM}}\), \(S|P_i\) sends \((\text{ACA.Register}, sid_3, pk_i)\) to \(S|\mathcal{F}_{\text{ACA}}\). Upon receiving \((\text{ACA.Registered}, sid_3)\) from \(S|\mathcal{F}_{\text{ACA}}\), \(S\) sends \((\text{ASC.Register}, sid_1, ack)\) to \(\mathcal{F}_{\text{ASC}}\).
Simulating the session set-up: When $A|P_j$ successfully establishes a session with $P_i$ (i.e. when $S|P_i$ receives $(\text{AN.Receive}, (\text{sid}_4, \text{ssid}))$, $(\text{ASC.EstablishSession}, \text{sid}_1, \text{ssid}, C'_0, \text{ID'})$) from $\mathcal{F}_{\text{AN}}$, $S|P_i$ sends $(\text{KEM.Decrypt}, \text{sid}_2, \text{ssid}, C'_0)$ to $S|\mathcal{F}_{\text{KEM-DEM}}$, and $(\text{KEM.Decrypt}, \text{sid}_2, \text{ssid}, \text{ok})$ is returned from $S|\mathcal{F}_{\text{KEM-DEM}}$, $S$ stores ($P_i, \text{ssid}, \text{active}$) and sets variables $P_i - \text{ssid} - sn$ and $\text{ID'} - \text{ssid} - sn$ equal to 1. $S|P_j$ then sends $(\text{ASC.EstablishSession}, \text{sid}_1, \text{ssid}, P_i)$ to $\mathcal{F}_{\text{ASC}}$. Upon receiving $(\text{ASC.EstablishSession}, \text{sid}_1, \text{ssid}, P_i)$ from $\mathcal{F}_{\text{ASC}}$, $S$ sends $(\text{ASC.EstablishSession}, \text{sid}_1, \text{ssid}, \text{partner}, \text{ack})$ to $\mathcal{F}_{\text{ASC}}$, and then $S$ sends $(\text{ASC.EstablishSession}, \text{sid}_1, \text{ssid}, P_i, \text{ack})$ to $\mathcal{F}_{\text{ASC}}$.

Simulating the data exchange: Upon receiving $(\text{ASC.Send}, \text{sid}_1, \text{ssid}, P_i, \text{partner}, M)$ from $\mathcal{F}_{\text{ASC}}$, $S$ sends $(\text{ASC.Send}, \text{sid}_1, \text{ssid}, P_i, \text{partner}, \text{ack})$ to $\mathcal{F}_{\text{ASC}}$. $S|P_i$ sets $m = M||P_i||P_i - \text{ssid} - sn$, increases $P_i - \text{ssid} - sn$, and sends $(\text{DEM.Encrypt}, \text{sid}_2, \text{ssid}, m)$ to $S|\mathcal{F}_{\text{KEM-DEM}}$. Upon receiving $(\text{DEM.Ciphertext}, \text{sid}_2, \text{ssid}, c)$ from $S|\mathcal{F}_{\text{KEM-DEM}}$, $S|P_i$ sends $(\text{AN.Send}, (\text{sid}_4, \text{ssid}), c, \text{ID'})$ to $S|\mathcal{F}_{\text{AN}}$. When $A|P_j$ successfully sends a message to $S|P_i$ (i.e. when $S|P_i$ receives $(\text{AN.Receive}, (\text{sid}_4, \text{ssid}), c)$ from $S|\mathcal{F}_{\text{AN}}$, $(P_i, \text{ssid}, \text{active})$ is stored), $S|P_i$ sends $(\text{DEM.Decrypt}, \text{sid}_2, \text{ssid}, c)$ to $S|\mathcal{F}_{\text{KEM-DEM}}$, receives $(\text{DEM.Plaintext}, \text{sid}_2, \text{ssid}, m)$ from $S|\mathcal{F}_{\text{KEM-DEM}}$, and $m = M||\text{ID'}||\text{ID'} - \text{ssid} - sn$, $S$ increases $\text{ID'} - \text{ssid} - sn$, $S|P_j$ sends $(\text{ASC.Send}, \text{sid}_1, \text{ssid}, m)$ to $\mathcal{F}_{\text{ASC}}$, and upon receiving $(\text{ASC.Send}, \text{sid}_1, \text{ssid}, \text{partner}, P_i, m)$ from $\mathcal{F}_{\text{ASC}}$, $S$ sends $(\text{ASC.Send}, \text{sid}_1, \text{ssid}, \text{partner}, P_i, \text{ack})$ to $\mathcal{F}_{\text{ASC}}$.

Simulating the session ending: Upon receiving $(\text{ASC.ExpireSession}, \text{sid}, \text{ssid}, P_i)$ from $\mathcal{F}_{\text{ASC}}$, $S$ removes $(P_i, \text{ssid}, \text{active})$, and $S|P_i$ sends $(\text{KEM.ExpireSession}, \text{sid}_2, \text{ssid})$ to $S|\mathcal{F}_{\text{KEM-DEM}}$.

When both $P_i$ and $P_j$ are corrupt: In this case, $S$ obtains the inputs to both $P_i$ and $P_j$, and also controls their outputs. $S$ simulates functionalities $\mathcal{F}_{\text{ACA}}$, $\mathcal{F}_{\text{KEM-DEM}}$ and $\mathcal{F}_{\text{AN}}$. The functionality $\mathcal{F}_{\text{ASC}}$ is never activated.

It can be verified that the simulation done by $S$ is perfect, i.e. no environment $\mathcal{Z}$ can tell whether it is interacting with $A$ and $\mathcal{F}_{\text{ASC}}$ or with $S$ and IDEAL$\mathcal{F}_{\text{ASC}}$.

We conclude that $\pi_{\text{ASC}}$ securely realizes $\mathcal{F}_{\text{ASC}}$ in the $(\mathcal{F}_{\text{KEM-DEM}}, \mathcal{F}_{\text{ACA}}, \mathcal{F}_{\text{AN}})$-hybrid model under static corruption. □

4. A Universally Composable Anonymous Online Service

In this section, we present a protocol for an anonymous online service, and prove that it realizes the functionality $\mathcal{F}_{\text{AOS}}$. In addition to the functionalities $\mathcal{F}_{\text{ASC}}$ and $\mathcal{F}_{\text{ACA}}$ given in Section 3, our protocol makes use of a functionality for anonymous blind signatures and a functionality for a standard (non-anonymous) secure channel.

The functionality for anonymous blind signatures, $\mathcal{F}_{\text{ABS}}$, is defined in Figure 7. It is based on the functionality proposed by Fischlin [5], but has been subject
to some modifications in order to suit our application. For instance, our functionality lets the environment decide whether a signature should be granted to a user, upon learning the user’s identity. This property is essential for our purpose, where the functionality is used by a protocol realizing the anonymous online service functionality. Recall that, in the latter, the environment determines whether the bank should establish a session with a specified user.

Furthermore, we highlight the fact that the user is anonymized, in the sense that his identity is not revealed to the ideal adversary. As a consequence, realizing this functionality requires a network where the user is identified to the communicating party, i.e. the bank, but anonymized with respect to outsiders. Let $F_{ASC}$ denote a functionality for a secure channel incorporating these properties. Intuitively, $F_{ASC}^*$ can be realized using $F_{ASC}$ in combination with $F_{ACA}$ and a dedicated functionality for digital signatures, which does not reveal the identity of an honest signer. In such a setting, the bank may establish a session with the user using $F_{ASC}$, and then have the user identify himself by signing a randomly chosen challenge.

Obviously, in order to realize $F_{ABS}$, we also need a blind signature scheme. This scheme should be secure with respect to blindness and non-forgeability, as defined by Buan et al. in [1]. Note that, in the definition of blindness, the adversary is required to output the secret key. Accordingly, we need some set-up assumption allowing the secret key to be extracted by a simulator. One option is to apply the scheme proposed by Gjøsteen and Kråkmo in [7], along with an additional key registration functionality, to which the signer sends the random coins used for generating his key pair. Alternatively, if this scheme is used in combination with an extra common reference string functionality, one may have the signer commit to the secret key, and include the commitment in the public key, together with a proof of correctness. With a carefully chosen commitment scheme, a simulator can construct the common reference string in such way that the secret key may be extracted. We note that, since the scheme in [7] is only weakly non-forgeable, the non-forgeability condition in $F_{ABS}$ must be modified in order to obtain a realizing protocol. Anyway, the resulting functionality can still be used to realize $F_{AOS}$.

The anonymous online service functionality also incorporates a communication session between the bank $B$ and the server $Q$, allowing the server to make deposits. For this purpose, we equip $B$ and $Q$ with a functionality for a secure channel, which is presented in Figure 8. This functionality corresponds to the one proposed by Canetti and Krawczyk in [3], except that it handles multiple sessions. As demonstrated by Nagao et al. in [6], it can be realized by using the KEM-DEM framework.

The functionalities $F_{ABS}$, $F_{ACA}$, $F_{SC}$ and $F_{ASC}$ constitute the building blocks necessary to construct our hybrid protocol $\pi_{AOS}$ for an anonymous online service. A detailed description of the protocol is given in Figures 9 and 10.
Concerning the description of π\textsubscript{AOS}, we point out that the challenge \( c \) picked by the server \( Q \) is chosen from a sufficiently large set. That is, we assume throughout the paper that an honest server never chooses the same challenge twice.

We note that, instead of modifying Fischlin’s functionality to fit our purpose, we may have considered using the functionality for blind signatures proposed in [1] as a starting point. As explained in [1], the main difference between these functionalities concerns the signature generation for corrupt users. While Fischlin’s functionality requires that even corrupt users specify the message to be signed, the latter disregards this message, and instead gives the user a “free signature”. This feature makes the functionality unsuitable as a building block for an anonymous online service. To illustrate our point, assume throughout this paragraph that \( \pi^*\text{AOS} \) is the protocol \( \pi\text{AOS} \) where \( F_{\text{ABS}} \) has been replaced by a functionality similar to the one in [1], granting free signatures to corrupt users. Consider the scenario where the user \( P_i \) is corrupt, while the bank \( B \) and the server \( Q \) are honest. In the ideal protocol, if \( B \) responds negatively to \( P_i \)’s payment request, no session will be established between \( P_i \) and \( Q \). In the hybrid protocol, the blind signature functionality should ensure that this cannot happen, i.e. if \( B \) refused to sign some challenge \( c \), it should be impossible for \( P_i \) to come up with a signature \( \sigma \) such that \((c, \sigma)\) is accepted upon verification.

While the non-forgeability condition in Fischlin’s functionality implicitly prevents such an event (under the assumption that the functionality has never seen \( c \) before), the functionality in [1] fails to do so. First, note that the ideal adversary determines the verification function, so in principle, this function may accept all message/signature pairs. Now assume that, during some session, \( P_i \) obtains a signature \( \sigma \) on some challenge \( c' \), but does not send \( \sigma \) to \( Q \). Instead, he starts a new session, and receives a new challenge \( c \) from \( Q \), such that \((c, \sigma)\) is accepted by the verification algorithm. Assume that \( B \) refuses to sign this time, and that \( P_i \) sends \( \sigma \) to \( Q \). Since \( P_i \)’s free signature count is positive at this point, the pair \((c, \sigma)\) will be accepted by the functionality. This means that \( P_i \) may establish a session with \( Q \), even if \( B \) refused to sign the challenge. Since this is impossible in the ideal protocol, \( \pi^*\text{AOS} \) does not realize \( F_{\text{AOS}} \).

The remaining part of the paper is dedicated to proving the following result.

**Theorem 2.** \( \pi_{\text{AOS}} \) realizes \( F_{\text{AOS}} \) in the \((F_{\text{ABS}}, F_{\text{ACA}}, F_{\text{SC}}, F_{\text{ASC}})\)-hybrid model under static corruption.

**Proof.** We show that for every adversary \( A \) interacting with parties running \( \pi_{\text{AOS}} \) in the \((F_{\text{ABS}}, F_{\text{ACA}}, F_{\text{SC}}, F_{\text{ASC}})\)-hybrid model, there is an ideal adversary \( S \) such that no environment \( Z \) can tell whether it is interacting with \( A \) and \( \pi_{\text{AOS}} \) or with \( S \) and \( \text{IDEAL}_{F_{\text{AOS}}} \).

As usual, \( S \) runs a simulated copy of \( A \) and forwards all messages from \( Z \) to \( A \) and back. When \( A \) corrupts a party \( P \), \( S \) corrupts \( \tilde{P} \). When \( \tilde{P} \) is corrupt, any input from \( Z \) meant for \( \tilde{P} \) goes directly to \( S \), who forwards it to \( A \) on the input tape corresponding to \( P \), and the other way around. Moreover, \( S \) can send messages to \( F \) in the name of \( \tilde{P} \), and messages from \( F \) meant for \( \tilde{P} \) go to \( S \). In
the following, when an entity \( E \) controls another entity \( E' \), the notation \( \langle E | E' \rangle \) should be read as “\( E \) in the name of \( E' \)”.

**When both \( P_1 \), \( Q \) and \( B \) are honest:** \( S \) simulates honest \( P_1 \), \( Q \) and \( B \) and functionalities \( F_{\text{ABS}}, F_{\text{ACA}}, F_{\text{SC}} \) and \( F_{\text{ASC}} \). We observe that \( S \) obtains all the information needed to perfectly simulate the above functionalities. Note that, it does not matter that \( S \) does not obtain \( P_1 \)’s identity or the content of messages sent between \( Q \) and \( P_1 \), since this information is also perfectly hidden from \( A \) in the real protocol. In the following, when we say that \( S|F \) sends a message to \( S|F \), where \( F \) is one of the above functionalities and \( P \) is either \( P_1 \), \( Q \) or \( B \), this implies that \( S \) proceeds according to the description of this functionality, and allows \( A \) to delay messages, determine keys etc. whenever appropriate.

**Simulating the set-up:** Upon receiving \((\text{AOS.Register}, \text{sid}_1)\) from \( F_{\text{AOS}} \), \( S|B \) sends \((\text{ABS.KeyGen}, \text{sid}_2)\) to \( S|F_{\text{ABS}} \). Upon receiving \((\text{ABS.Key}, \text{sid}_2, \text{pk})\) from \( S|F_{\text{ABS}} \), \( S|B \) sends \((\text{ACA.Register}, \text{sid}_3, \text{pk})\) to \( S|F_{\text{ACA}} \). Upon receiving \((\text{ACA.Registered}, \text{sid}_3)\) from \( S|F_{\text{ACA}} \), \( S|B \) sends \((\text{SC.Register}, \text{sid}_4)\) to \( S|F_{\text{SC}} \). Upon receiving \((\text{SC.Registered}, \text{sid}_4)\) from \( S|F_{\text{SC}} \), \( S \) sends \((\text{AOS.Register}, \text{sid}_1, \text{Q}, \text{ack})\) to \( F_{\text{AOS}} \).

Upon receiving \((\text{AOS.Register}, \text{sid}_1, \text{Q})\) from \( F_{\text{AOS}} \), \( S|Q \) sends \((\text{ACA.Retrieve}, \text{sid}_3, \text{B})\) to \( S|F_{\text{ACA}} \). Upon receiving \((\text{ACA.Retrieved}, \text{sid}_3, \text{B}, \text{pk})\) from \( S|F_{\text{ACA}} \), \( S|Q \) sends \((\text{ASC.Register}, \text{sid}_5)\) to \( S|F_{\text{ASC}} \). Upon receiving \((\text{ASC.Registered}, \text{sid}_5)\) from \( S|F_{\text{ASC}} \), \( S \) sends \((\text{AOS.Register}, \text{sid}_1, \text{Q}, \text{ack})\) to \( F_{\text{AOS}} \).

**Simulating the session set-up:** Upon receiving \((\text{AOS.EstablishSession}, \text{sid}_1, \text{ssid}, \text{Q})\) from \( F_{\text{AOS}} \), \( S \) chooses a random identity \( ID \) and stores \((\text{ID}, \text{ssid})\). \( S|ID \) sends \((\text{ASC.EstablishSession}, \text{sid}_5, \text{ssid}, \text{Q})\) to \( S|F_{\text{ASC}} \). Upon receiving \((\text{ASC.Established}, \text{sid}_5, \text{ssid}, \text{Q})\) from \( S|F_{\text{ASC}} \), \( S|ID \) sends \((\text{ACA.Retrieve}, \text{sid}_3, \text{B})\) to \( S|F_{\text{ACA}} \). Upon receiving \((\text{ACA.Retrieved}, \text{sid}_3, \text{B}, \text{pk})\) from \( S|F_{\text{ACA}} \), \( S|ID \) sends \((\text{ASC.Send}, \text{sid}_5, \text{ssid}, \text{AOS.Pay})\) to \( S|F_{\text{ASC}} \). Upon receiving \((\text{ASC.Receive}, \text{sid}_5, \text{ssid}, \text{AOS.Pay})\) from \( S|F_{\text{ASC}} \), \( S \) picks a random challenge \( c \), and \( S|Q \) sends \((\text{ASC.Send}, \text{sid}_5, \text{ssid}, (\text{AOS.Challenge}, c))\) to \( S|F_{\text{ASC}} \). Upon receiving \((\text{ASC.Receive}, \text{sid}_5, \text{ssid}, (\text{AOS.Challenge}, c))\) from \( S|F_{\text{ASC}} \), \( S|ID \) sends \((\text{ABS.Sign}, \text{sid}_2, \text{ssid}, \text{pk}, c)\) to \( S|F_{\text{ABS}} \). When \( S|B \) receives \((\text{ABS.Sign}, \text{sid}_2, \text{ssid}, \text{ID}, \text{request})\) from \( S|F_{\text{ABS}} \), \( S \) sends \((\text{AOS.EstablishSession}, \text{sid}_1, \text{ssid}, \text{ack})\) to \( F_{\text{AOS}} \).

If \( S \) receives \((\text{AOS.Pay}, \text{sid}_1, \text{ssid}, \text{no})\) from \( F_{\text{AOS}} \), then \( S|B \) sends \((\text{ABS.Sign}, \text{sid}_2, \text{ssid}, \text{no})\) to \( S|F_{\text{ABS}} \). Then, if \( S|ID \) receives \((\text{ABS.Sign}, \text{sid}_2, \text{ssid}, \text{no})\) from \( S|F_{\text{ABS}} \), \( S \) sends \((\text{AOS.Pay}, \text{sid}_1, \text{ssid}, \text{no}, \text{ack})\) to \( F_{\text{AOS}} \). Otherwise, if \( S \) receives \((\text{AOS.Pay}, \text{sid}_1, \text{ssid}, \text{yes})\) from \( F_{\text{AOS}} \), then \( S|B \) sends \((\text{ABS.Sign}, \text{sid}_2, \text{ssid}, \text{yes})\) to \( S|F_{\text{ABS}} \). Then, if \( S|ID \) receives \((\text{ABS.Signature}, \text{sid}_2, \text{ssid}, \sigma)\) from \( S|F_{\text{ABS}} \), then \( S|ID \) sends \((\text{ASC.Send}, \text{sid}_5, \text{ssid}, (\text{AOS.Response}, \sigma))\) to \( S|F_{\text{ASC}} \). \( S \) stores \((\text{ID}, \text{ssid}, \text{active})\) and sends \((\text{AOS.Pay}, \text{sid}_2, \text{ssid}, \text{partner}, \text{ack})\) to \( F_{\text{AOS}} \).
When $S|Q$ receives $(\text{ASC.Receive}, sid_5, ssid, (\text{AOS.Response}, \sigma))$ from $S|F_{\text{ASC}}$, $S|Q$ sends $(\text{ABS.Verify}, sid_2, pk, c, \sigma)$ to $S|F_{\text{ABS}}$. If $S|Q$ receives $(\text{ABS.Verify}, sid_2, accept)$ from $S|F_{\text{ABS}}$, then $S$ stores $(c, \sigma)$ and $(Q, ssid, active)$, and sends $(\text{AOS.Pay}, sid_1, ssid, Q, ack)$ to $F_{\text{AOS}}$.

Simulating the data exchange: Upon receiving $(\text{AOS.Send}, sid_1, ssid, partner, Q, |m|)$ from $F_{\text{AOS}}$, if $(ID, ssid, active)$ is stored, then $S$ picks a random message $m'$ of length $|m|$, and $S|ID$ sends $(\text{ASC.Send}, sid_5, ssid, m')$ to $S|F_{\text{ASC}}$. When $S|Q$ receives $(\text{ASC.Receive}, sid_5, ssid, m')$ from $S|F_{\text{ASC}}$, if $(Q, ssid, active)$ is stored, then $S$ sends $(\text{AOS.Send}, sid_1, ssid, partner, Q, ack)$ to $F_{\text{AOS}}$.

Upon receiving $(\text{AOS.Send}, sid_1, ssid, Q, partner, |m|)$ from $F_{\text{AOS}}$, if $(Q, ssid, active)$ is stored, then $S$ picks a random message $m'$ of length $|m|$, and $S|ID$ sends $(\text{ASC.Send}, sid_5, ssid, m')$ to $S|F_{\text{ASC}}$. When $S|ID$ receives $(\text{ASC.Receive}, sid_5, ssid, m')$ from $S|F_{\text{ASC}}$, if $(ID, ssid, active)$ is stored, then $S$ sends $(\text{AOS.Send}, sid_1, ssid, Q, partner, ack)$ to $F_{\text{AOS}}$.

Simulating the session ending: Upon receiving $(\text{AOS.ExpireSession}, sid_1, ssid, partner)$ from $F_{\text{AOS}}$, $S$ removes $(ID, ssid, active)$, and $S|ID$ sends $(\text{ASC.ExpireSession}, sid_5, ssid)$ to $S|F_{\text{ASC}}$.

Upon receiving $(\text{AOS.ExpireSession}, sid_1, ssid, Q)$ from $F_{\text{AOS}}$, $S$ removes $(Q, ssid, active)$, and $S|Q$ sends $(\text{ASC.ExpireSession}, sid_5, ssid)$ to $S|F_{\text{ASC}}$.

Simulating the collecting: Upon receiving $(\text{AOS.Collect}, sid_1, ssid', Q, n)$ from $F_{\text{AOS}}$, if $S$ has at least $n$ stored pairs $(c, \sigma)$, then $S|Q$ sends $(\text{SC.EstablishSession}, sid_4, ssid', B)$ to $S|F_{\text{SC}}$. When $S|Q$ receives $(\text{SC.SessionEstablished}, sid_4, ssid', B)$ from $S|F_{\text{SC}}$, $S$ lets $s$ be a set of $n$ stored pairs $(c, \sigma)$, and sends $(\text{SC.Send}, sid_4, ssid', (\text{AOS.Collect}, s))$ to $S|F_{\text{SC}}$. When $S|B$ receives $(\text{SC.Receive}, sid_4, ssid', (\text{AOS.Collect}, s))$ from $S|F_{\text{SC}}$, for every pair $(c, \sigma)$ in $s$, $S$ checks that there is no stored pair $(c, \sigma')$ for any $\sigma'$, and $S|B$ sends $(\text{ABS.Verify}, sid_2, pk, c, \sigma)$ to $S|F_{\text{ABS}}$. If, for every pair, $S|B$ receives $(\text{ABS.Verify}, sid_2, accept)$ from $S|F_{\text{ABS}}$, then $S$ stores $s$ and $S|B$ sends $(\text{SC.Send}, sid_4, ssid', (\text{AOS.Collected}))$ to $S|F_{\text{SC}}$. $S|B$ then sends $(\text{SC.ExpireSession}, sid_4, ssid')$ to $S|F_{\text{SC}}$, and $S$ sends $(\text{AOS.Collect}, sid_1, ssid', Q, ack)$ to $F_{\text{AOS}}$.

When $S|Q$ receives $(\text{SC.Receive}, sid_4, ssid', (\text{AOS.Collected}))$ from $S|F_{\text{SC}}$, $S$ removes the pairs $(c, \sigma)$ contained in $s$. $S|Q$ then sends $(\text{SC.ExpireSession}, sid_4, ssid')$ to $S|F_{\text{SC}}$, and $S$ sends $(\text{AOS.Collect}, sid_1, ssid', Q, ack)$ to $F_{\text{AOS}}$.

It can be verified that the simulation is perfect. In particular, every time $Q$ stores a pair $(c, \sigma)$ in $\pi_{\text{AOS}}$, so does $S$ in IDEAL$_{\text{F}_{\text{AOS}}}$. Also, every time such a pair is stored, $F_{\text{AOS}}$ increases balance. Moreover, every time $n$ pairs are removed by $Q$, $n$ pairs are removed by $S$, and balance is decreased by $n$. This means that the number of pairs $(c, \sigma)$ stored by $Q$ in the hybrid protocol equals the
number of such pairs stored by $S$ in the ideal protocol, which again is equal to balance. Accordingly, if $S$ has $n'$ pairs $(c, \sigma)$ to collect, the condition $n' \leq \text{balance}$ is guaranteed to hold.

**When $P$ is corrupt, while $Q$ and $B$ are honest:** $S$ simulates honest $Q$ and $B$ and functionalities $\mathcal{F}_{\text{ABS}}, \mathcal{F}_{\text{ACA}}, \mathcal{F}_{\text{SC}}$ and $\mathcal{F}_{\text{ASC}}$. We observe that $S$ obtains all the information needed to perfectly simulate the above functionalities. In the following, when we say that an entity sends a message to $S|\mathcal{F}$, where $\mathcal{F}$ is one of the above functionalities, this implies that $S$ proceeds according to the description of this functionality, and allows $A$ to delay messages, determine keys etc. whenever appropriate.

**Simulating the set-up:** Since both $Q$ and $B$ are honest, this is done exactly as when all parties are honest, as described above.

**Simulating the session set-up:** Upon receiving $(\text{ASC.Receive}, sid_5, ssid, \text{AOS.Pay})$ from $S|\mathcal{F}_{\text{ASC}}$, $S$ picks a random challenge $c$, and $S|Q$ sends $(\text{ASC.Send}, sid_5, ssid, (\text{AOS.Challenge}, c))$ to $S|\mathcal{F}_{\text{ASC}}$. When $S|B$ receives $(\text{ABS.Sign}, sid_2, ssid, P_1, \text{request})$ from $S|\mathcal{F}_{\text{ABS}}$, $S|\bar{P}_1$ sends $(\text{AOS.EstablishSession}, sid, ssid, Q)$ to $\mathcal{F}_{\text{AOS}}$. Then, upon receiving $(\text{AOS.EstablishSession}, sid, ssid, Q)$ from $\mathcal{F}_{\text{AOS}}$, $S$ sends $(\text{AOS.EstablishSession}, sid_1, ssid, \text{ack})$ to $\mathcal{F}_{\text{AOS}}$. Then, if $S$ receives $(\text{AOS.Pay}, sid_1, ssid, \text{no})$ from $\mathcal{F}_{\text{AOS}}$, $S$ sends $(\text{AOS.Pay}, sid_1, ssid, no, \text{ack})$ to $\mathcal{F}_{\text{AOS}}$. $S|B$ then sends $(\text{ABS.Sign}, sid_2, ssid, no)$ to $S|\mathcal{F}_{\text{ABS}}$. Otherwise, if $S$ receives $(\text{AOS.Pay}, sid_1, ssid, yes)$ from $\mathcal{F}_{\text{AOS}}$, $S$ sends $(\text{AOS.Pay}, sid_2, ssid, partner, \text{ack})$ to $\mathcal{F}_{\text{AOS}}$. $S|B$ then sends $(\text{ABS.Sign}, sid_2, ssid, yes)$ to $S|\mathcal{F}_{\text{ABS}}$. If $S|Q$ receives $(\text{ASC.Receive}, sid_5, ssid, (\text{AOS.Response}, \sigma))$ from $S|\mathcal{F}_{\text{ASC}}$, and $S|Q$ earlier sent $(\text{ASC.Send}, sid_5, ssid, (\text{AOS.Challenge}, c))$ to $S|\mathcal{F}_{\text{ASC}}$, then $S|Q$ sends $(\text{ABS.Verify}, sid_2, pk, c, \sigma)$ to $S|\mathcal{F}_{\text{ABS}}$. Then, if $S|Q$ receives $(\text{ABS.Verify}, sid_2, \text{accept})$ from $S|\mathcal{F}_{\text{ABS}}$, and $S$ earlier received $(\text{AOS.Pay}, sid_1, ssid, yes)$ from $\mathcal{F}_{\text{AOS}}$, then $S$ stores $(c, \sigma)$ and $(Q, ssid, \text{active})$, and sends $(\text{AOS.Pay}, sid_1, ssid, Q, \text{ack})$ to $\mathcal{F}_{\text{AOS}}$.

**Simulating the data exchange:** When $S|Q$ receives $(\text{ASC.Receive}, sid_5, ssid, m)$ from $S|\mathcal{F}_{\text{ASC}}$, if $(Q, ssid, \text{active})$ is stored, then $S|\bar{P}_1$ sends $(\text{AOS.Send}, sid_1, ssid, m)$ to $\mathcal{F}_{\text{AOS}}$, and upon receiving $(\text{AOS.Send}, sid_1, ssid, partner, Q, m)$ from $\mathcal{F}_{\text{AOS}}$, $S$ sends $(\text{AOS.Send}, sid_1, ssid, partner, Q, \text{ack})$ to $\mathcal{F}_{\text{AOS}}$. When $S$ receives $(\text{AOS.Send}, sid_1, ssid, Q, partner, m)$ from $\mathcal{F}_{\text{AOS}}$, $S$ sends $(\text{AOS.Send}, sid_1, ssid, Q, partner, \text{ack})$ to $\mathcal{F}_{\text{AOS}}$. If $(Q, ssid, \text{active})$ is stored, $S|Q$ sends $(\text{ASC.Send}, sid_5, ssid, m)$ to $S|\mathcal{F}_{\text{ASC}}$.

**Simulating the session ending:** If $S$ receives $(\text{AOS.ExpireSession}, Q, sid_1, ssid)$ from $\mathcal{F}_{\text{AOS}}$, then $S|Q$ sends $(\text{ASC.ExpireSession}, sid_5, ssid)$ to $S|\mathcal{F}_{\text{ASC}}$. 
Simulating the collecting: Since both $Q$ and $B$ are honest, this is done exactly as when all parties are honest, as described above.

It can be verified that the simulation is perfect also in this case. We note that, since $P_i$ is corrupt, $\mathcal{A}$ obtains the challenge $c$. In $\pi_{\text{AOS}}$, $c$ is randomly chosen by an honest $Q$, while in $\text{IDEAL}_{\mathcal{F}_{\text{AOS}}}$, it is randomly chosen by $S$, hence the distribution of $c$ is the same in both cases.

We also observe the following difference between $\pi_{\text{AOS}}$ and $\text{IDEAL}_{\mathcal{F}_{\text{AOS}}}$: In the hybrid protocol, in principle, a session can be established between $P_i$ and $Q$ even if $B$ has not agreed to issue a blind signature to $P_i$. By the construction of $\mathcal{F}_{\text{AOS}}$, this is impossible in the ideal protocol. However, we can show that, by the construction of $\mathcal{F}_{\text{ABS}}$, this behavior is prevented also by the hybrid protocol. Recall that $B$ and $Q$ are honest, so we can assume that the verification request $\langle \text{ABS.Verify}, \text{id}_2, pk', c, \sigma \rangle$ sent from $Q$ to $\mathcal{F}_{\text{ABS}}$ has the correct key, i.e. $pk' = pk$, and that the challenge $c$ is honestly chosen by $Q$. Accordingly, we can assume that $\mathcal{F}_{\text{ABS}}$ has never seen $c$ before, so that $\mathcal{F}_{\text{ABS}}$ has no recorded entry $\langle c, \sigma, pk, \text{accept} \rangle$. Hence, by the non-forgability condition, the signature will be rejected by $\mathcal{F}_{\text{ABS}}$.

As for the collecting, the simulation guarantees that the number of pairs $(c, \sigma)$ stored by $Q$ in the hybrid protocol equals the number of such pairs stored by $S$ in the ideal protocol. Moreover, balance is increased by $\mathcal{F}_{\text{AOS}}$ every time $S$ stores a pair $(c, \sigma)$, and decreased by $n$ every time $S$ removes $n$ pairs. So, if $S$ has $n$ pairs $(c, \sigma)$ to collect, the condition $n \leq \text{balance}$ holds.

When $Q$ is corrupt, while $P_i$ and $B$ are honest: $S$ simulates honest $P_i$ and $B$ and functionalities $\mathcal{F}_{\text{ABS}}, \mathcal{F}_{\text{ACA}}, \mathcal{F}_{\text{SC}}$ and $\mathcal{F}_{\text{ASC}}$. We observe that $S$ obtains all the information needed to perfectly simulate the above functionalities.

Simulating the set-up: Upon receiving $\langle \text{AOS.Register}, \text{id}_1 \rangle$ from $\mathcal{F}_{\text{AOS}}$, $S|B$ sends $\langle \text{ABS.KeyGen}, \text{id}_2 \rangle$ to $S|\mathcal{F}_{\text{ABS}}$. Upon receiving $\langle \text{ABS.Key}, \text{id}_2, pk \rangle$ from $S|\mathcal{F}_{\text{ABS}}$, $S|B$ sends $\langle \text{ACA.Register}, \text{id}_3, pk \rangle$ to $S|\mathcal{F}_{\text{ACA}}$. Upon receiving $\langle \text{ACA.Registered}, \text{id}_3 \rangle$ from $S|\mathcal{F}_{\text{ACA}}$, $S|B$ sends $\langle \text{SC.Register}, \text{id}_4 \rangle$ to $S|\mathcal{F}_{\text{SC}}$. Upon receiving $\langle \text{SC.Registered}, \text{id}_4 \rangle$ from $S|\mathcal{F}_{\text{SC}}$, $S$ sends $\langle \text{AOS.Register}, \text{id}_1, \text{ack} \rangle$ to $\mathcal{F}_{\text{AOS}}$.

The first time $S|Q$ receives some input, $S|Q$ sends $\langle \text{AOS.Register}, \text{id}_1 \rangle$ to $\mathcal{F}_{\text{ASC}}$, and upon receiving $\langle \text{AOS.Register}, \text{id}_1, Q \rangle$ from $\mathcal{F}_{\text{AOS}}$, $S$ sends $\langle \text{AOS.Register}, \text{id}_1, Q, \text{ack} \rangle$ to $\mathcal{F}_{\text{AOS}}$.

Simulating the session set-up: Upon receiving $\langle \text{AOS.EstablishSession}, \text{id}_1, \text{ssid}, Q \rangle$ from $\mathcal{F}_{\text{AOS}}$, $S$ chooses a random identity $ID$ and stores $(ID, \text{ssid})$. $S|ID$ sends $\langle \text{ASC.EstablishSession}, \text{id}_5, \text{ssid}, Q \rangle$ to $S|\mathcal{F}_{\text{ASC}}$. Upon receiving $\langle \text{ASC.SessionEstablished}, \text{id}_5, \text{ssid}, Q \rangle$ from $S|\mathcal{F}_{\text{ASC}}$, $S|ID$ sends $\langle \text{ACA.Retrieve}, \text{id}_3, B \rangle$ to $S|\mathcal{F}_{\text{ACA}}$. Upon receiving $\langle \text{ACA.Retrieved}, \text{id}_3, B, pk \rangle$ from $S|\mathcal{F}_{\text{ACA}}$, $S|ID$ sends $\langle \text{ASC.Send}, \text{id}_5, \text{ssid}, \text{AOS.Pay} \rangle$ to $S|\mathcal{F}_{\text{ASC}}$. 
If $S|ID$ receives $(ASC.Receive, sid_5, ssid, (AOS.Challenge, c))$ from $S|FASC$, $S|ID$ sends $(ABS.Sign, sid_2, ssid, pk, c)$ to $S|FABS$. When $S|B$ receives $(ABS.Sign, sid_2, ssid, ID, request)$ from $S|FABS$, $S$ sends $(AOS.EstablishSession, sid_1, ssid, ack)$ to $FABS$.

If $S$ receives $(AOS.Pay, sid_1, ssid, no)$ from $FABS$, then $S|B$ sends $(ABS.Sign, sid_2, ssid, no)$ to $S|FABS$. Then, if $S|ID$ receives $(ABS.Sign, sid_2, ssid, no)$ from $S|FABS$, $S$ sends $(AOS.Pay, sid_1, ssid, no, ack)$ to $FABS$.

Simulating the collecting: If $S|B$ receives $(SC.Receive, sid_4, ssid', (AOS.Collect, s))$ from $S|FSC$, for every pair $(c, \sigma)$ in $s$, $S$ checks that there is no stored pair $(c, \sigma')$ for any $\sigma'$, and $S|B$ sends $(ABS.Verify, sid_2, pk, c, \sigma)$ to $S|FABS$. If, for every pair, $S|B$ receives $(ABS.Verify, sid_2, accept)$ from $S|FABS$, then $S$ stores $s$, and $S|B$ sends $(SC.Send, sid_4, ssid', AOS.Collected)$ to $S|FSC$. $S|B$ then sends $(SC.ExpireSession, sid_4, ssid')$ to $S|FSC$. $S|Q$ sends $(AOS.Collect, sid_1, ssid', n)$ to $FABS$, where $n$ is the number of pairs in $s$, and upon receiving $(AOS.Collect, sid_1, ssid', Q, n)$ from $FABS$, $S$ sends $(AOS.Collect, sid_1, ssid', ack)$ to $FABS$. $S$ then sends $(AOS.Collect, sid_1, ssid', Q, ack)$ to $FABS$.

It can be verified that the simulation is perfect also in this case. The simulation guarantees that the number of pairs $(c, \sigma)$ output by $S|FABS$ in the ideal protocol equals the number of such pairs output by $FABS$ in the hybrid protocol. Assume that, at some point in the execution, this number is $n'$, and no pairs were yet collected by $Q$. We observe that, in the ideal protocol, we then have $balance \geq n'$. In the hybrid protocol, in principle, $Q$ can collect a larger number of challenge/signature pairs than in the ideal protocol, where this number is
limited by balance. However, we can show that, in both protocols, the number of pairs that can be collected is limited by $n'$, due to the construction of $\mathcal{F}_{\text{ABS}}$. Assume that $Q$ tries to collect $n$ pairs $(c, \sigma)$, where $n \geq n'$. Recall that $B$ is honest, so we can assume that every verification request $(\text{ABS.Verify}, \text{sid}_2, pk')$, $(c, \sigma)$ sent from $B$ to $\mathcal{F}_{\text{ABS}}$ has the correct key, i.e. $pk' = pk$. In this case, the signature will be accepted only if $(c, \sigma, pk')$, accept is stored, which is the case only if $(c, \sigma)$ was earlier output by $S|\mathcal{F}_{\text{ABS}}/\mathcal{F}_{\text{ABS}}$. Since by assumption only $n'$ pairs were output, at least one of the pairs will be rejected by $S|\mathcal{F}_{\text{ABS}}/\mathcal{F}_{\text{ABS}}$ upon verification.

Next, assume that $Q$ collects $n''$ challenge/signature pairs. The simulation then ensures that balance is decreased by $n''$. Recall that an honest $B$ ignores a collect request if $S$ contains a pair $(c, \sigma)$, such that there is a recorded entry $(c, \sigma')$ for some $\sigma'$. By a similar argument as above, we can show that the number of pairs that can be collected is now limited by $n' - n''$ in both protocols.

When $B$ is corrupt, while $P_i$ and $Q$ are honest: $S$ simulates honest $P_i$ and $Q$ and functionalities $\mathcal{F}_{\text{ABS}}$, $\mathcal{F}_{\text{ACA}}$, $\mathcal{F}_{\text{SC}}$ and $\mathcal{F}_{\text{ASC}}$. We observe that $S$ obtains all the information needed to perfectly simulate the above functionalities.

Simulating the set-up: Upon receiving $(\text{AOS.Register}, \text{sid}_1, Q)$ from $\mathcal{F}_{\text{AOS}}$, $S|Q$ sends $(\text{ACA.Retrieve}, \text{sid}_3, B)$ to $S|\mathcal{F}_{\text{ACA}}$. Upon receiving $(\text{ACA.Retrieved}, \text{sid}_3, B, pk)$ from $S|\mathcal{F}_{\text{ACA}}$, $S|Q$ sends $(\text{ASC.Register}, \text{sid}_5)$ to $S|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\text{ASC.Registered}, \text{sid}_5)$ from $S|\mathcal{F}_{\text{ASC}}$, $S$ sends $(\text{AOS.Register}, \text{sid}_1, Q, \text{ack})$ to $\mathcal{F}_{\text{AOS}}$.

Simulating the session set-up: Upon receiving $(\text{AOS.EstablishSession}, \text{sid}_1, \text{ssid}, P_i, Q)$ from $\mathcal{F}_{\text{AOS}}$, $S$ sends $(\text{AOS.EstablishSession}, \text{sid}_1, \text{ssid}, \text{ack})$ to $\mathcal{F}_{\text{AOS}}$. $S|P_i$ sends $(\text{ASC.EstablishSession}, \text{sid}_5, \text{ssid}, Q)$ to $S|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\text{ASC.Established}, \text{sid}_5, \text{ssid}, Q)$ from $S|\mathcal{F}_{\text{ASC}}$, $S|P_i$ sends $(\text{ACA.Retrieve}, \text{sid}_3, B)$ to $S|\mathcal{F}_{\text{ACA}}$. Upon receiving $(\text{ACA.Retrieved}, \text{sid}_3, B, pk)$ from $S|\mathcal{F}_{\text{ACA}}$, $S|P_i$ sends $(\text{ASC.Send}, \text{sid}_5, \text{ssid}, \text{AOS.Pay})$ to $S|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\text{ASC.Receive}, \text{sid}_5, \text{ssid}, \text{AOS.Pay})$ from $S|\mathcal{F}_{\text{ASC}}$, $S$ picks a random challenge $c$, and $S|Q$ sends $(\text{ASC.Send}, \text{sid}_5, \text{ssid}, (\text{AOS.Challenge}, c))$ to $S|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\text{ASC.Receive}, \text{sid}_5, \text{ssid}, (\text{AOS.Challenge}, c))$ from $S|\mathcal{F}_{\text{ASC}}$, $S|P_i$ sends $(\text{ABS.Sign}, \text{sid}_2, \text{ssid}, pk, c)$ to $S|\mathcal{F}_{\text{ABS}}$.

If $S|\mathcal{F}_{\text{ABS}}$ then receives $(\text{AOS.Sign}, \text{sid}_2, \text{ssid}, \text{no})$ from $A|B$, then $S|\tilde{B}$ sends $(\text{AOS.Pay}, \text{sid}_1, \text{ssid}, \text{no})$ to $\mathcal{F}_{\text{AOS}}$. When $S|P_i$ receives $(\text{AOS.Sign}, \text{sid}_2, \text{ssid}, \text{no})$ from $S|\mathcal{F}_{\text{ABS}}$, $S$ sends $(\text{AOS.Pay}, \text{sid}_1, \text{ssid}, \text{no, ack})$ to $\mathcal{F}_{\text{AOS}}$. Otherwise, if $S|\mathcal{F}_{\text{ABS}}$ receives $(\text{ABS.Sign}, \text{sid}_2, \text{ssid}, \text{yes})$ from $A|B$, then $S|\tilde{B}$ sends $(\text{AOS.Pay}, \text{sid}_1, \text{ssid}, \text{yes})$ to $\mathcal{F}_{\text{AOS}}$.

If $S|P_i$ then receives $(\text{ABS.Signature}, \text{sid}_2, \text{ssid}, \sigma)$ from $S|\mathcal{F}_{\text{ABS}}$, then $S|P_i$ sends $(\text{ASC.Send}, \text{sid}_5, \text{ssid}, (\text{AOS.Response}, \sigma))$ to $S|\mathcal{F}_{\text{ASC}}$. $S$ stores $(P_i, \text{ssid}, \text{active})$ and sends $(\text{AOS.Pay}, \text{sid}_2, \text{ssid}, \text{partner, ack})$ to $\mathcal{F}_{\text{AOS}}$. 
When \( S|Q \) receives \((\text{ASC.Receive}, \text{sid}_5, \text{ssid}, (\text{AOS.Response}, \sigma))\) from \( S|\mathcal{F}_{\text{ASC}} \), \( S|Q \) sends \((\text{ABS.Verify}, \text{sid}_2, pk, c, \sigma)\) to \( S|\mathcal{F}_{\text{ABS}} \). If \( S|Q \) receives \((\text{ABS.Verify}, \text{sid}_2, \text{accept})\) from \( S|\mathcal{F}_{\text{ABS}} \), then \( S \) stores \((c, \sigma)\) and \((Q, \text{ssid}, \text{active})\), and sends \((\text{AOS.Pay}, \text{sid}_1, \text{ssid}, Q, \text{ack})\) to \( \mathcal{F}_{\text{AOS}} \).

**Simulating the data exchange:** Since both \( P_i \) and \( Q \) are honest, this is done exactly as when all parties are honest, as described earlier. We note that we now have \( ID = P_i \).

**Simulating the session ending:** Since both \( P_i \) and \( Q \) are honest, this is done exactly as when all parties are honest, as described earlier. We note that we now have \( ID = P_i \).

**Simulating the collecting:** Upon receiving \((\text{AOS.Collect}, \text{sid}_1, \text{ssid}', Q, n)\) from \( \mathcal{F}_{\text{AOS}} \), if \( S \) has at least \( n \) stored pairs \((c, \sigma)\), then \( S|Q \) sends \((\text{SC.EstablishSession}, \text{sid}_4, \text{ssid}', B)\) to \( S|\mathcal{F}_{\text{SC}} \). When \( S|Q \) receives \((\text{SC.SessionEstablished}, \text{sid}_4, \text{ssid}', B)\) from \( S|\mathcal{F}_{\text{SC}} \), \( S \) lets \( s \) be a set of \( n \) stored pairs \((c, \sigma)\), and sends \((\text{SC.Send}, \text{sid}_4, \text{ssid}', (\text{AOS.Collect}, s))\) to \( S|\mathcal{F}_{\text{SC}} \).

If \( S|Q \) receives \((\text{SC.Receive}, \text{sid}_4, \text{ssid}', \text{AOS.Collected})\) from \( S|\mathcal{F}_{\text{SC}} \), \( S \) removes the pairs \((c, \sigma)\) contained in \( s \), and \( S|Q \) sends \((\text{SC.ExpireSession}, \text{sid}_4, \text{ssid}')\) to \( S|\mathcal{F}_{\text{SC}} \). \( S \) sends \((\text{AOS.Collect}, \text{sid}_1, \text{ssid}', \text{ack})\) to \( \mathcal{F}_{\text{AOS}} \), and then sends \((\text{AOS.Collect}, \text{sid}_1, \text{ssid}', Q, \text{ack})\) to \( \mathcal{F}_{\text{AOS}} \).

It can be verified that the simulation is perfect. We note that, since \( B \) is corrupt, the key \( pk \) retrieved from \( \mathcal{F}_{\text{ACA}} \) by \( P_i \) and \( Q \) is not necessarily honestly generated. In that case, \( A \) gets the challenge \( c \) from \( \mathcal{F}_{\text{ABS}} \). In \( \pi_{\text{AOS}} \), \( c \) is randomly chosen by an honest \( Q \), while in \( \text{IDEAL}_{\mathcal{F}_{\text{AOS}}} \), it is randomly chosen by \( S \), hence the distribution of \( c \) is the same in both situations. Also, in the hybrid protocol, the corrupt \( B \) learns \( P_i \)'s identity upon signature requests. In order to simulate this behavior in the ideal protocol, \( \mathcal{F}_{\text{AOS}} \) is constructed such that, when \( B \) is corrupt, \( P_i \)'s identity is given to \( S \) upon signature requests.

As for the collecting, the simulation guarantees that the number of pairs \((c, \sigma)\) stored by \( Q \) in the hybrid protocol equals the number of such pairs stored by \( S \) in the ideal protocol. Also, \( \text{balance} \) is increased by \( \mathcal{F}_{\text{AOS}} \) every time \( S \) stores a pair \((c, \sigma)\), and decreased by \( n \) every time \( S \) removes \( n \) pairs. Accordingly, if \( S \) has \( n \) pairs \((c, \sigma)\) to collect, the condition \( n \leq \text{balance} \) holds.

**When \( P_i \) and \( Q \) are corrupt, while \( B \) is honest:** \( S \) simulates honest \( B \) and functionalities \( \mathcal{F}_{\text{ABS}}, \mathcal{F}_{\text{ACA}}, \mathcal{F}_{\text{SC}} \) and \( \mathcal{F}_{\text{ASC}} \). We observe that \( S \) obtains all the information needed to perfectly simulate the above functionalities.

**Simulating the set-up:** This is done exactly as when \( Q \) is corrupt, while \( P_i \) and \( B \) are honest, as described earlier.
Simulating the session set-up: If $S|F_{\text{ABS}}$ sends $(\text{ABS}.\text{Sign}, \text{sid}_2, \text{ssid}, P_i, \text{request})$ to $S|B$, then $S|P_i$ sends $(\text{AOS}.\text{EstablishSession}, \text{sid}_1, \text{ssid}, Q)$ to $F_{\text{AOS}}$, and upon receiving $(\text{AOS}.\text{EstablishSession}, \text{sid}_1, \text{ssid}, Q)$ from $F_{\text{AOS}}$, $S$ sends $(\text{AOS}.\text{EstablishSession}, \text{sid}_1, \text{ssid}, Q, \text{ack})$ to $F_{\text{AOS}}$.

If $S$ receives $(\text{AOS}.\text{Pay}, \text{sid}_1, \text{ssid}, \text{no})$ from $F_{\text{AOS}}$, then $S|B$ sends $(\text{ABS}.\text{Sign}, \text{sid}_2, \text{ssid}, \text{no})$ to $S|F_{\text{ABS}}$. $S$ then sends $(\text{AOS}.\text{Pay}, \text{sid}_1, \text{ssid}, \text{no}, \text{ack})$ to $F_{\text{AOS}}$. Otherwise, if $S$ receives $(\text{AOS}.\text{Pay}, \text{sid}_1, \text{ssid}, \text{yes})$ from $F_{\text{AOS}}$, then $S|B$ sends $(\text{ABS}.\text{Sign}, \text{sid}_2, \text{ssid}, \text{yes})$ to $S|F_{\text{ABS}}$. $S$ then sends $(\text{AOS}.\text{Pay}, \text{sid}_1, \text{ssid}, \text{partner}, \text{ack})$ to $F_{\text{AOS}}$, and finally, $S$ sends $(\text{AOS}.\text{Pay}, \text{sid}_1, \text{ssid}, Q, \text{ack})$ to $F_{\text{AOS}}$.

Simulating the data exchange: Since both $P_i$ and $Q$ are corrupt, $S$ obtains the inputs of both $P_i$ and $Q$, and also controls their outputs. $S$ simulates honest $B$ and the functionalities $F_{\text{ABS}}, F_{\text{ACA}}, F_{\text{SC}}$ and $F_{\text{ASC}}$, and $F_{\text{AOS}}$ is never activated.

Simulating the collecting: This is done exactly as when $Q$ is corrupt, while $P_i$ and $B$ are honest, as described earlier.

It can be verified that the simulation is perfect. As for the collecting, the same argument holds as for the case where $Q$ is corrupt, while $P_i$ and $B$ are honest.

When $P_i$ and $B$ are corrupt, while $Q$ is honest: $S$ simulates honest $Q$ and functionalities $F_{\text{ABS}}, F_{\text{ACA}}, F_{\text{SC}}$ and $F_{\text{ASC}}$. We observe that $S$ obtains all the information needed to perfectly simulate the above functionalities.

Simulating the set-up: This is done exactly as when $B$ is corrupt, while $P_i$ and $Q$ are honest, as described earlier.

Simulating the session set-up: Upon receiving $(\text{ASC}.\text{Receive}, \text{sid}_5, \text{ssid}, \text{AOS}.\text{Pay})$ from $S|F_{\text{ASC}}$, $S$ picks a random challenge $c$, and $S|Q$ sends $(\text{ASC}.\text{Send}, \text{sid}_5, \text{ssid}, (\text{AOS}.\text{Challenge}, c))$ to $S|F_{\text{ASC}}$.

If $S|Q$ receives $(\text{ASC}.\text{Receive}, \text{sid}_5, \text{ssid}, (\text{AOS}.\text{Response}, \sigma))$ from $S|F_{\text{ASC}}$, and $S|Q$ earlier sent $(\text{ASC}.\text{Send}, \text{sid}_5, \text{ssid}, (\text{AOS}.\text{Challenge}, c))$ to $S|F_{\text{ASC}}$, then $S|Q$ sends $(\text{AOS}.\text{Verify}, \text{sid}_2, \text{pk}, c, \sigma)$ to $S|F_{\text{ABS}}$. Then, if $S|Q$ receives $(\text{AOS}.\text{Verify}, \text{sid}_2, \text{accept})$ from $S|F_{\text{ABS}}$, $S$ proceeds as follows: $S|P_i$ sends $(\text{AOS}.\text{EstablishSession}, \text{sid}_1, \text{ssid}, P_i, Q)$ to $F_{\text{AOS}}$, and then $S$ sends $(\text{AOS}.\text{EstablishSession}, \text{sid}_1, \text{ssid}, Q, \text{ack})$ to $F_{\text{AOS}}$. $S$ then sends $(\text{AOS}.\text{Pay}, \text{sid}_1, \text{ssid}, \text{yes})$ to $F_{\text{AOS}}$. $S$ sends $(\text{AOS}.\text{Pay}, \text{sid}_2, \text{ssid}, \text{partner}, \text{ack})$ to $F_{\text{AOS}}$, and then $S$ stores $(c, \sigma)$ and $(Q, \text{ssid}, \text{active})$, and sends $(\text{AOS}.\text{Pay}, \text{sid}_1, \text{ssid}, Q, \text{ack})$ to $F_{\text{AOS}}$.

Simulating the data exchange: This is done exactly as when $P_i$ is corrupt, while $B$ and $Q$ are honest, as described earlier.

Simulating the session ending: This is done exactly as when $P_i$ is corrupt, while $B$ and $Q$ are honest, as described earlier.
Simulating the collecting: This is done exactly as when $B$ is corrupt, while $P_i$ and $Q$ are honest, as described earlier.

It can be verified that the simulation is perfect. As for the collecting, the same argument holds as for the case where $B$ is corrupt, while $P_i$ and $Q$ are honest.

When $Q$ and $B$ are corrupt, while $P_i$ is honest: $S$ simulates honest $P_i$ and functionalities $F_{ABS}$, $F_{ACA}$, $F_{SC}$ and $F_{ASC}$. We observe that $S$ obtains all the information needed to perfectly simulate the above functionalities.

Simulating the set-up: The first time $S|\tilde{Q}$ receives some input, $S|\tilde{Q}$ sends $(AOS.Register, sid_1)$ to $F_{AOS}$, and upon receiving $(AOS.Register, sid_1, Q)$ from $F_{AOS}$, $S$ sends $(AOS.Register, sid_1, Q, ack)$ to $F_{AOS}$.

Simulating the session set-up: Upon receiving $(AOS.EstablishSession, sid_1, ssid, P_i, Q)$ from $F_{AOS}$, $S$ sends $(AOS.EstablishSession, sid_1, ssid, Q, ack)$ to $F_{AOS}$. $S|P_i$ then sends $(ASC.EstablishSession, sid_5, ssid, Q)$ to $S|F_{ASC}$. Upon receiving $(ASC.SessionEstablished, sid_5, ssid, Q)$ from $S|F_{ASC}$, $S|P_i$ sends $(ACA.Retrieve, sid_3, B)$ to $S|F_{ACA}$. Upon receiving $(ACA.Retrieved, sid_3, B, pk)$ from $S|F_{ACA}$, $S|P_i$ sends $(ASC.Send, sid_5, ssid, AOS.Pay)$ to $S|F_{ASC}$.

If $S|P_i$ receives $(ASC.Receive, sid_5, ssid, (AOS.Challenge, c))$ from $S|F_{ASC}$, $S|P_i$ sends $(ABS.Sign, sid_2, ssid, pk, c)$ to $S|F_{ABS}$.

If $S|F_{ABS}$ then receives $(ABS.Sign, sid_2, ssid, no)$ from $A|B$, then $S|\tilde{B}$ sends $(AOS.Pay, sid_1, ssid, no)$ to $F_{AOS}$. When $S|P_i$ receives $(ABS.Sign, sid_2, ssid, no)$ from $S|F_{ABS}$, $S$ sends $(AOS.Pay, sid_1, ssid, no, ack)$ to $F_{AOS}$. Otherwise, if $S|F_{ABS}$ receives $(ABS.Sign, sid_2, ssid, yes)$ from $A|B$, then $S$ sends $(AOS.Pay, sid_1, ssid, Q, ack)$ to $F_{AOS}$. $S|\tilde{B}$ then sends $(AOS.Pay, sid_1, ssid, yes)$ to $F_{AOS}$.

If $S|P_i$ then receives $(ABS.Signature, sid_2, ssid, \sigma)$ from $S|F_{ABS}$, then $S|P_i$ sends $(ASC.Send, sid_5, ssid, (AOS.Response, \sigma))$ to $S|F_{ASC}$. $S$ stores $(P_i, ssid, active)$ and sends $(AOS.Pay, sid_2, ssid, partner, ack)$ to $F_{AOS}$.

Simulating the data exchange: This is done exactly as when $Q$ is corrupt, while $P_i$ and $B$ are honest, as described earlier.

Simulating the session ending: This is done exactly as when $Q$ is corrupt, while $P_i$ and $B$ are honest, as described earlier.

Simulating the collecting: Since both $Q$ and $B$ are corrupt, $S$ obtains the inputs of both $Q$ and $\tilde{B}$, and also controls their outputs. $S$ simulates honest $P_i$ and the functionalities $F_{ABS}$, $F_{ACA}$, $F_{SC}$ and $F_{ASC}$, and $F_{AOS}$ is never activated.

It can be verified that the simulation is perfect.
When both $P_i$, $Q$ and $B$ are corrupt: In this case, $S$ obtains the inputs of both $\tilde{P}_i$, $\tilde{Q}$ and $\tilde{B}$, and also controls their outputs. $S$ simulates the functionalities $\mathcal{F}_{\text{ABS}}$, $\mathcal{F}_{\text{ACA}}$, $\mathcal{F}_{\text{SC}}$ and $\mathcal{F}_{\text{ASC}}$, and $\mathcal{F}_{\text{AOS}}$ is never activated. In this case, it is obvious that the simulation is perfect.

We conclude that $\pi_{\text{AOS}}$ realizes $\mathcal{F}_{\text{AOS}}$ in the $(\mathcal{F}_{\text{ABS}}, \mathcal{F}_{\text{ACA}}, \mathcal{F}_{\text{SC}}, \mathcal{F}_{\text{ASC}})$-hybrid model under static corruption. □

REFERENCES


\( F_{AOS} \) proceeds as follows, with a bank \( B \), a server \( Q \), users \( P_1, \ldots, P_n \) and an ideal adversary \( S \).

- **Set-up**
  - On the first message \((AOS.\text{Register}, sid)\) from \( B \), where \( sid = (B, sid') \), send \((AOS.\text{Register}, sid)\) to \( S \). Wait to receive \((AOS.\text{Register}, sid, ack)\) from \( S \), send \((AOS.\text{Registered}, sid)\) to \( B \).
  - On the first message \((AOS.\text{Register}, sid)\) from \( Q \), where \( sid = (B, sid') \), send \((AOS.\text{Register}, sid, Q)\) to \( S \). Wait to receive \((AOS.\text{Register}, sid, Q, ack)\) from \( S \), and store \( Q \).

- **Session Set-up**
  On message \((AOS.\text{EstablishSession}, sid, ssid, Q)\) from \( P_i \), where \( sid = (B, sid') \) and \( ssid \) has not been used before:
  1. Send \((AOS.\text{EstablishSession}, sid, ssid, Q)\) to \( S \) if \( B \) is honest, and \((AOS.\text{EstablishSession}, sid, ssid, P_i, Q)\) otherwise.
  2. Upon receipt of \((AOS.\text{EstablishSession}, sid, ssid, Q, ack)\) from \( S \), if \( Q \) is stored, then send \((AOS.\text{Pay}, sid, ssid, P_i, request)\) to \( B \).
  3. Upon receipt of \((AOS.\text{Pay}, sid, ssid, P_i, no)\) from \( B \), send \((AOS.\text{Pay}, sid, ssid, no)\) to \( S \). Wait for \((AOS.\text{Pay}, sid, ssid, no, ack)\) from \( S \), send \((AOS.\text{Pay}, sid, ssid, no)\) to \( P_i \) and stop. Otherwise, upon receipt of \((AOS.\text{Pay}, sid, ssid, P_i, yes)\) from \( B \), send \((AOS.\text{Pay}, sid, ssid, yes)\) to \( S \).
  4. Upon receipt of \((AOS.\text{Pay}, sid, ssid, partner, ack)\) from \( S \), store \((P_i, ssid, active)\) and send \((AOS.\text{SessionEstablished}, sid, ssid, Q)\) to \( P_i \).
  5. Upon receipt of \((AOS.\text{Pay}, sid, ssid, Q, ack)\) from \( S \), increase balance and store \((Q, ssid, active)\). Send \((AOS.\text{SessionEstablished}, sid, ssid, Q)\) to \( Q \).

- **Data Exchange**
  - On message \((AOS.\text{Send}, sid, ssid, m)\) from \( P_i \): If \((P_i, ssid, active)\) is stored, then send \((AOS.\text{Send}, sid, ssid, partner, Q, [m])\) to \( S \) if both \( P_i \) and \( Q \) are honest, and \((AOS.\text{Send}, sid, ssid, partner, Q, m)\) otherwise. Wait for \((AOS.\text{Send}, sid, ssid, partner, Q, ack)\) from \( S \). If \((Q, ssid, active)\) is stored, then send \((AOS.\text{Receive}, sid, ssid, m)\) to \( Q \).
  - On message \((AOS.\text{Send}, sid, ssid, m)\) from \( Q \): If \((Q, ssid, active)\) is stored, then send \((AOS.\text{Send}, sid, ssid, Q, partner, [m])\) to \( S \) if both \( P_i \) and \( Q \) are honest, and \((AOS.\text{Send}, sid, ssid, Q, partner, m)\) otherwise. Wait for \((AOS.\text{Send}, sid, ssid, Q, partner, ack)\) from \( S \). If \((P_i, ssid, active)\) is stored, then send \((AOS.\text{Receive}, sid, ssid, m)\) to \( P_i \).

- **Session Ending**
  On message \((AOS.\text{ExpireSession}, sid, ssid)\) from \( P_i / Q \): Remove \((P_i, ssid, active)/(Q, ssid, active)\), and send \((AOS.\text{ExpireSession}, sid, ssid, partner/Q)\) to \( S \).

- **Collect**
  On message \((AOS.\text{Collect}, sid, ssid', n)\) from \( Q \): Send \((AOS.\text{Collect}, sid, ssid', Q, n)\) to \( S \). Upon receiving \((AOS.\text{Collect}, sid, ssid', ack)\) from \( S \), check that \( n \leq \text{balance} \), let \( \text{balance} \leftarrow \text{balance} - n \), and send \((AOS.\text{Deposit}, sid, ssid', Q, n)\) to \( B \). Upon receiving \((AOS.\text{Collect}, sid, ssid', Q, ack)\) from \( S \), send \((AOS.\text{Deposit}, sid, ssid', n)\) to \( Q \).

**Figure 1.** The anonymous online service functionality \( F_{AOS} \).
\( F_{ASC} \) proceeds as follows, with users \( \tilde{P}_1, \ldots, \tilde{P}_n \) and an ideal adversary \( S \).

- **Set-up**
  In the first activation, expect to receive \((\text{ASC.Register}, sid)\) from some party \( \tilde{P}_i \). Send \((\text{ASC.Register}, sid, P_i)\) to \( S \). Wait to receive \((\text{ASC.Register}, sid, ack)\) from \( S \), store \( P_i \), and send \((\text{ASC.Registered}, sid)\) to \( \tilde{P}_i \).

- **Session Set-up**
  Upon receiving \((\text{ASC.EstablishSession}, sid, ssid, P_i)\) from some party \( \tilde{P}_j \), where \( ssid \) has not been used before, send \((\text{ASC.EstablishSession}, sid, ssid, P_i)\) to \( S \), and check that \( P_i \) is stored. Upon receiving \((\text{ASC.EstablishSession}, sid, ssid, partner, ack)\) from \( S \), store \( (P_j, ssid, active) \), and send \((\text{ASC.SessionEstablished}, sid, ssid, P_i, ack)\) from \( S \), store \((P_i, ssid, active)\), and send \((\text{ASC.SessionEstablished}, sid, ssid)\) to \( \tilde{P}_i \).

- **Data Exchange**
  - Upon receiving \((\text{ASC.Send}, sid, ssid, m)\) from \( \tilde{P}_i \), if \((P_i, ssid, active)\) is stored, then send \((\text{ASC.Send}, sid, ssid, P_i, partner, |m|)\) to \( S \) if both \( P_i \) and \( P_j \) are honest, and \((\text{ASC.Send}, sid, ssid, P_i, partner, m)\) otherwise. Wait to receive \((\text{ASC.Send}, sid, ssid, P_i, partner, ack)\) from \( S \). If \((P_j, ssid, active)\) is stored, then send \((\text{ASC.Receive}, sid, ssid, m)\) to \( \tilde{P}_j \).
  - Upon receiving \((\text{ASC.Send}, sid, ssid, m')\) from \( \tilde{P}_j \), if \((P_j, ssid, active)\) is stored, then send \((\text{ASC.Send}, sid, ssid, partner, P_i, |m'|)\) to \( S \) if both \( P_i \) and \( P_j \) are honest, and \((\text{ASC.Send}, sid, ssid, partner, P_i, m')\) otherwise. Wait to receive \((\text{ASC.Send}, sid, ssid, partner, P_i, ack)\) from \( S \). If \((P_i, ssid, active)\) is stored, then send \((\text{ASC.Receive}, sid, ssid, m')\) to \( \tilde{P}_i \).

- **Session Ending**
  Upon receiving \((\text{ASC.ExpireSession}, sid, ssid)\) from \( \tilde{P}_i/\tilde{P}_j \), remove \((P_i, ssid, active)/(P_j, ssid, active)\), and send \((\text{ASC.ExpireSession}, sid, ssid, P_i/partner)\) to \( S \).

**Figure 2.** The anonymous secure channel functionality \( F_{ASC} \).

\( F_{AN} \) proceeds as follows, with users \( \tilde{P}_1, \ldots, \tilde{P}_n \) and an ideal adversary \( S \).

- In the first activation, expect to receive \((\text{AN.Register}, sid)\) from some party \( \tilde{P}_j \). Choose a random identity \( ID \). Send \((\text{AN.Registered}, sid, ID)\) to \( \tilde{P}_j \) and store \((P_j, ID)\).

- Upon receiving \((\text{AN.Send}, sid, m, ID')\) from some party \( \tilde{P}_i \), if \( i = j \), then send \((\text{AN.Send}, sid, m, ID, ID')\) to \( S \), otherwise send \((\text{AN.Send}, sid, m, P_i, ID')\) to \( S \). Upon receiving \((\text{AN.Send}, sid, m', ID'')\) from \( S \), if \( ID'' = ID \), then send \((\text{AN.Receive}, sid, m')\) to \( \tilde{P}_j \), otherwise send \((\text{AN.Receive}, sid, m')\) to \( ID'' \).

**Figure 3.** The anonymous network functionality \( F_{AN} \).
\( \mathcal{F}_{\text{ACA}} \) proceeds as follows, with parties \( \tilde{P}_1, \ldots, \tilde{P}_n \) and an ideal adversary \( S \).

- In the first activation, expect to receive \((\text{ACA.} \text{Register}, \text{sid}, \text{pk})\) from some party \( \tilde{P}_i \). Send \((\text{ACA.} \text{Register}, \text{sid}, P_i, \text{pk})\) to \( S \). Wait to receive \((\text{ACA.} \text{Register}, \text{sid}, \text{ack})\) from \( S \), record the pair \((P_i, \text{pk})\), and send \((\text{ACA.} \text{Registered}, \text{sid})\) to \( \tilde{P}_i \).
- Upon receiving a message \((\text{ACA.} \text{Retrieve}, \text{sid}, P_i)\) from some party \( \tilde{P}_j \), send \((\text{ACA.} \text{Retrieve}, \text{sid}, P_i)\) to \( S \), and wait to receive \((\text{ACA.} \text{Retrieve}, \text{sid}, \text{ack})\) from \( S \). If there is a recorded pair \((P_i, \text{pk})\), then send \((\text{ACA.} \text{Retrieved}, \text{sid}, P_i, \text{pk})\) to \( \tilde{P}_j \). Otherwise, send \((\text{ACA.} \text{Retrieved}, \text{sid}, P_i, \perp)\) to \( \tilde{P}_j \).

**Figure 4.** The ideal anonymous certificate authority functionality \( \mathcal{F}_{\text{ACA}} \).
\( \mathcal{F}_{\text{KEM-DEM}} \) proceeds as follows, with parties \( \tilde{P}_1, \ldots, \tilde{P}_n \) and an ideal adversary \( S \).

- **Set-up**
  - In the first activation, expect to receive \( (\text{KEM.KeyGen}, \text{sid}) \) from some party \( \tilde{P}_i \). Send \( (\text{KEM.KeyGen}, \text{sid}) \) to \( S \). Upon receiving \( (\text{KEM.Key}, \text{sid}, \text{pk}) \) from \( S \), send \( (\text{KEM.Key}, \text{sid}, \text{pk}) \) to \( \tilde{P}_i \).

- **Session Set-up**
  - Upon receiving \( (\text{KEM.Encrypt}, \text{sid}, \text{ssid}, \text{pk}') \) from some party \( \tilde{P}_j \):
    - If there is no recorded entry \( (\tilde{P}_j, \text{ssid}, C, \text{active}) \) for any \( C \), then send \( (\text{KEM.Encrypt}, \text{sid}, \text{ssid}, \text{pk}') \) to \( S \). Upon receiving \( (\text{EncryptedSharedKey}, \text{sid}, \text{ssid}, \text{pk}', C_0) \) from \( S \), send \( (\text{EncryptedSharedKey}, \text{sid}, \text{ssid}, \text{pk}', C_0) \) to \( \tilde{P}_j \), and store \( (\text{pk}', \text{ssid}, C_0) \) and \( (\tilde{P}_j, \text{ssid}, C_0, \text{active}) \).
    - Otherwise, do nothing.
  - Upon receiving \( (\text{KEM.Decrypt}, \text{sid}, \text{ssid}, C'_0) \) from \( \tilde{P}_i \) (and \( \tilde{P}_i \) only), send \( (\text{KEM.Decrypt}, \text{sid}, \text{ssid}, C'_0) \) to \( S \). Upon receiving \( (\text{KEM.Decrypt}, \text{sid}, \text{ssid}, \text{ack}) \) from \( S \):
    - If there is no stored entry \( (\tilde{P}_i, \text{ssid}, C, \text{active}) \) for any \( C \), send \( (\text{KEM.Decrypt}, \text{sid}, \text{ssid}, \text{ok}) \) to \( \tilde{P}_i \) and store \( (\tilde{P}_i, \text{ssid}, C'_0, \text{active}) \).

- **Data Exchange**
  - Upon receiving \( (\text{DEM.Encrypt}, \text{sid}, \text{ssid}, m) \) from \( \tilde{P}_i/\tilde{P}_j \):
    - If \( (\tilde{P}_i/\tilde{P}_j, \text{ssid}, C_0, \text{active}) \) is stored:
      * If both \( \tilde{P}_i \) and \( \tilde{P}_j \) are honest, then send \( (\text{DEM.Encrypt}, \text{sid}, \text{ssid}, \|m\|) \) to \( S \). Upon receiving \( (\text{DEM.Ciphertext}, \text{sid}, \text{ssid}, c) \) from \( S \), send \( (\text{DEM.Ciphertext}, \text{sid}, \text{ssid}, c) \) to \( \tilde{P}_i/\tilde{P}_j \), and store \( (\text{ssid}, m, c, C_0) \).
      * Otherwise, send \( (\text{DEM.Encrypt}, \text{sid}, \text{ssid}, m) \) to \( S \). Upon receiving \( (\text{DEM.Ciphertext}, \text{sid}, \text{ssid}, c) \) from \( S \), send \( (\text{DEM.Ciphertext}, \text{sid}, \text{ssid}, c) \) to \( \tilde{P}_i/\tilde{P}_j \), and store \( (\text{ssid}, m, c, C_0) \).
  - Upon receiving \( (\text{DEM.Decrypt}, \text{sid}, \text{ssid}, c') \) from \( \tilde{P}_i/\tilde{P}_j \), send \( (\text{DEM.Decrypt}, \text{sid}, \text{ssid}, c') \) to \( S \). Upon receiving \( (\text{DEM.Plaintext}, \text{sid}, \text{ssid}, m') \) from \( S \):
    - If there is a recorded entry \( (\tilde{P}_i/\tilde{P}_j, \text{ssid}, C, \text{active}) \) for some \( C \):
      1. If there is a recorded entry \( (\text{ssid}, m, c', C) \), then send \( (\text{DEM.Plaintext}, \text{sid}, \text{ssid}, m) \) to \( \tilde{P}_i/\tilde{P}_j \).
      2. Otherwise, if both \( \tilde{P}_i \) and \( \tilde{P}_j \) are honest, then store \( (\text{ssid}, \bot, c', C) \) and send \( (\text{DEM.Plaintext}, \text{sid}, \text{ssid}, \bot) \) to \( \tilde{P}_i/\tilde{P}_j \).
      3. Otherwise, if there is a recorded entry \( (\text{ssid}, \bot, c', C) \), then send \( (\text{DEM.Plaintext}, \text{sid}, \text{ssid}, \bot) \) to \( \tilde{P}_i/\tilde{P}_j \).
      4. Otherwise, send \( (\text{DEM.Plaintext}, \text{sid}, \text{ssid}, m') \) to \( \tilde{P}_i/\tilde{P}_j \), and store \( (\text{ssid}, m', c', C) \).

- **Session Ending**
  - Upon receiving \( (\text{KEM.ExpireSession}, \text{sid}, \text{ssid}) \) from \( \tilde{P}_i/\tilde{P}_j \), remove \( (\tilde{P}_i, \text{ssid}, C_0, \text{active})/ (\tilde{P}_j, \text{ssid}, C'_0, \text{active}) \).

**Figure 5.** The KEM-DEM functionality \( \mathcal{F}_{\text{KEM-DEM}} \).
\(\pi_{\text{ASC}}\) runs as follows, with users \(P_1, \ldots, P_n\) and an adversary \(A\).

**Set-up**

Upon the first input \((\text{ASC.\,Register}, sid_1)\), \(P_i\) sends \((\text{KEM.\,KeyGen}, sid_2)\) to \(F_{\text{KEM-DEM}}\). Upon receiving \((\text{KEM.\,Key}, sid_2, PK_i)\) from \(F_{\text{KEM-DEM}}\), \(P_i\) sends \((\text{ACA.\,Register}, sid_3, PK_i)\) to \(F_{\text{ACA}}\). Upon receiving \((\text{ACA.\,Registered}, sid_3)\) from \(F_{\text{ACA}}\), \(P_i\) outputs \((\text{ASC.\,Registered}, sid_1)\).

**Session Set-up**

1. Upon the first input \((\text{ASC.\,EstablishSession}, sid_1, ssid, P_i)\), \(P_j\) sends \((\text{AN.\,Register}, (sid_4, ssid))\) to \(F_{\text{AN}}\), and waits to receive \((\text{AN.\,Registered}, (sid_4, ssid), ID)\) from \(F_{\text{AN}}\). Then \(P_j\) sends \((\text{ACA.\,Retrieve}, sid_3, P_i)\) to \(F_{\text{ACA}}\).
2. Upon receiving \((\text{ACA.\,Retrieved}, sid_3, P_i, PK_i)\) from \(F_{\text{ACA}}\), \(P_i\) sends \((\text{KEM.\,Encrypt}, sid_2, ssid, PK_i)\) to \(F_{\text{KEM-DEM}}\) and receives \((\text{EncryptedSharedKey}, sid_2, ssid, PK_i, C_0)\) from \(F_{\text{KEM-DEM}}\).
3. \(P_j\) stores \((ssid, \text{active})\), sets variables \(ssid - sn\) and \(ssid - psn\) equal to 1, sends \((\text{AN.\,Send}, (sid_4, ssid), (\text{ASC.\,EstablishSession}, sid_1, ssid, C_0, ID), P_i)\) to \(F_{\text{AN}}\), and outputs \((\text{ASC.\,SessionEstablished}, sid_1, ssid, P_i)\).
4. Upon receiving \((\text{AN.\,Receive}, (sid_4, ssid), (\text{ASC.\,EstablishSession}, sid_1, ssid, C'_0, ID'))\) from \(F_{\text{AN}}\), \(P_i\) sends \((\text{KEM.\,Decrypt}, sid_2, ssid, C'_0)\) to \(F_{\text{KEM-DEM}}\). If \((\text{KEM.\,Decrypt}, sid_2, ssid, ok)\) is returned from \(F_{\text{KEM-DEM}}\), \(P_i\) stores \((ssid, \text{active})\), sets variables \(ssid - sn\) and \(ssid - psn\) equal to 1 and outputs \((\text{ASC.\,SessionEstablished}, sid_1, ssid)\).

**Data Exchange**

1. Upon input \((\text{ASC.\,Send}, sid_1, ssid, M)\), \(P_i/P_j\) checks that \((ssid, \text{active})\) is stored, let \(m = M||ID||ssid - sn\), increases \(ssid - sn\) and sends \((\text{DEM.\,Encrypt}, sid_2, ssid, m)\) to \(F_{\text{KEM-DEM}}\).
2. Upon receiving \((\text{DEM.\,Ciphertext}, sid_2, ssid, c)\) from \(F_{\text{KEM-DEM}}\), \(P_i/P_j\) sends \((\text{AN.\,Send}, (sid_4, ssid), c, ID/P_i)\) to \(F_{\text{AN}}\).
3. Upon receiving \((\text{AN.\,Receive}, (sid_4, ssid), c)\) from \(F_{\text{AN}}\), \(P_j/P_i\) checks that \((ssid, \text{active})\) is stored, and sends \((\text{DEM.\,Decrypt}, sid_2, ssid, c)\) to \(F_{\text{KEM-DEM}}\).
4. Upon receiving \((\text{DEM.\,Plaintext}, sid_2, ssid, m)\) from \(F_{\text{KEM-DEM}}\), \(P_j/P_i\) checks that \(m = M||ID||ssid - psn\), increases \(ssid - psn\) and outputs \((\text{ASC.\,Receive}, sid_1, ssid, M)\).

**Session Ending**

Upon input \((\text{ASC.\,ExpireSession}, sid_1, ssid)\), \(P_i/P_j\) removes \((ssid, \text{active})\) and sends \((\text{KEM.\,ExpireSession}, sid_2, ssid)\) to \(F_{\text{KEM-DEM}}\).

**Figure 6.** The anonymous secure channel protocol \(\pi_{\text{ASC}}\).
\( F_{\text{ABS}} \) proceeds as follows, with signer \( \hat{B} \), users \( \hat{P}_1, \ldots, \hat{P}_n \) and an ideal adversary \( S \).

- Upon receiving the first \( (\text{ABS.KeyGen}, \text{sid}) \) from the signer \( \hat{B} \), where \( \text{sid} = (B, \text{sid}') \), send \( (\text{ABS.KeyGen}, \text{sid}) \) to \( S \). Upon receiving \( (\text{ABS.KeyGen}, \text{sid}, \text{pk}, \Pi) \) from \( S \), store \( (\text{pk}, \Pi) \), send \( (\text{ABS.KeyGen}, \text{sid}, \text{pk}) \) to \( \hat{B} \) and stop.

- Upon receiving \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{pk}', \text{m}) \) from a user \( \hat{P}_i \), where \( \text{sid} = (B, \text{sid}') \), proceed as follows:
  1. If \( \text{pk}' = \text{pk} \) and \( \hat{P}_i \) is honest, then send \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{pk}') \) to \( S \), otherwise send \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{pk}', \text{m}) \) to \( S \).
  2. Upon receiving \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{pk}', \text{ack}) \) from \( S \), send \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{P}_i, \text{request}) \) to \( \hat{B} \).
  3. Upon receiving \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{P}_i, \text{no}) \) from \( \hat{B} \), send \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{no}) \) to \( S \). Wait for \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{no}, \text{ack}) \) from \( S \), and send \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{no}) \) to \( \hat{P}_i \).
  4. Upon receiving \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{P}_i, \text{yes}) \) from \( \hat{B} \), send \( (\text{ABS.Sign}, \text{sid}, \text{ssid}, \text{yes}) \) to \( S \).
    (a) Upon receiving \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \text{signer completed}) \) from \( S \), send \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \text{P}_i) \) to \( \hat{B} \).
    (b) Upon receiving \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \text{signer not completed}) \) from \( S \), send \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \text{P}_i, \text{signer not completed}) \) to \( \hat{B} \).
    (c) Upon receiving \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \text{user completed}, \sigma') \) from \( S \), proceed as follows: If \( \hat{P}_i \) is honest and \( \text{pk}' = \text{pk} \), then generate \( \sigma \leftarrow \Pi(m) \). If \( (m, \sigma, \text{pk}', \text{reject}) \) is stored, then stop, otherwise store \( (m, \sigma, \text{pk}', \text{accept}) \) and send \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \sigma) \) to \( \hat{P}_i \). Otherwise, if \( \hat{P}_i \) is corrupt or \( \text{pk}' \neq \text{pk} \), and if there is no stored entry \( (m, \sigma', \text{pk}', \text{reject}) \), then store \( (m, \sigma', \text{pk}', \text{accept}) \) and send \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \sigma') \) to \( \hat{P}_i \).
    (d) Upon receiving \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \text{user failed}) \) from \( S \), send \( (\text{ABS.Signature}, \text{sid}, \text{ssid}, \text{user failed}) \) to \( \hat{P}_i \).

- Upon receiving \( (\text{ABS.Verify}, \text{sid}, \text{pk}', m, \sigma) \) from an honest user \( \hat{P}_j \), send \( (\text{ABS.Verify}, \text{sid}, \text{pk}', m, \sigma) \) to \( S \). Upon receiving \( (\text{ABS.Verify}, \text{sid}, \text{pk}', m, \sigma, \phi) \) from \( S \), proceed as follows:
  1. If \( \text{pk}' = \text{pk} \) and \( (m, \sigma, \text{pk}, \text{accept}) \) is stored, then let \( f = \text{accept} \) (completeness condition).
  2. Otherwise, if \( \text{pk}' = \text{pk} \), \( B \) is honest and there is no recorded entry \( (m, \sigma, \text{pk}, \text{accept}) \), then let \( f = \text{reject} \) and store \( (m, \sigma, \text{pk}, \text{reject}) \) (non-forgability condition).
  3. Otherwise, if there is a recorded entry \( (m, \sigma, \text{pk}', f') \), then let \( f = f' \) (consistency condition).
  4. Otherwise, let \( f = \phi \) and store \( (m, \sigma, \text{pk}', f) \).

Send \( (\text{ABS.Verify}, \text{sid}, f) \) to \( \hat{P}_j \).

Figure 7. The anonymous blind signature functionality \( F_{\text{ABS}} \).
\( \mathcal{F}_\text{SC} \) proceeds as follows, with users \( \tilde{P}_1, \ldots, \tilde{P}_n \) and an ideal adversary \( S \).

- **Set-up**
  
  In the first activation, expect to receive \((\text{SC}.\text{Register}, \text{sid})\) from some party \( \tilde{P}_i \). Send \((\text{SC}.\text{Register}, \text{sid}, P_i)\) to \( S \). Wait to receive \((\text{SC}.\text{Register}, \text{sid}, \text{ack})\) from \( S \), store \( P_i \), and send \((\text{SC}.\text{Registered}, \text{sid})\) to \( \tilde{P}_i \).

- **Session Set-up**
  
  Upon receiving \((\text{SC}.\text{EstablishSession}, \text{sid}, \text{ssid}, P_i)\) from some party \( \tilde{P}_j \), send \((\text{SC}.\text{EstablishSession}, \text{sid}, \text{ssid}, P_i, P_j)\) to \( S \), and check that \( P_i \) is stored. Upon receiving \((\text{SC}.\text{EstablishSession}, \text{sid}, \text{ssid}, P_j, \text{ack})\) from \( S \), store \((P_j, \text{ssid}, \text{active})\), and send \((\text{SC}.\text{SessionEstablished}, \text{sid}, \text{ssid}, P_i)\) to \( \tilde{P}_j \). Upon receiving \((\text{SC}.\text{EstablishSession}, \text{sid}, \text{ssid}, P_i, \text{ack})\) from \( S \), store \((P_i, \text{ ssid, active})\), and send \((\text{SC}.\text{SessionEstablished}, \text{sid}, \text{ssid}, P_j)\) to \( \tilde{P}_i \).

- **Data Exchange**
  
  - Upon receiving \((\text{SC}.\text{Send}, \text{sid}, \text{ssid}, m)\) from \( \tilde{P}_i \), if \((P_i, \text{ ssid, active})\) is stored, then send \((\text{SC}.\text{Send}, \text{sid}, \text{ ssid, }P_i, P_j, |m|)\) to \( S \) if both \( P_i \) and \( P_j \) are honest, and \((\text{SC}.\text{Send}, \text{sid}, \text{ ssid, }P_i, P_j, m)\) otherwise. Wait to receive \((\text{SC}.\text{Send}, \text{sid}, \text{ ssid, }P_i, P_j, \text{ack})\) from \( S \). If \((P_j, \text{ ssid, active})\) is stored, then send \((\text{SC}.\text{Receive}, \text{sid}, \text{ ssid, }m)\) to \( \tilde{P}_j \).
  
  - Upon receiving \((\text{SC}.\text{Send}, \text{sid}, \text{ssid, m})\) from \( \tilde{P}_j \), if \((P_j, \text{ ssid, active})\) is stored, then send \((\text{SC}.\text{Send}, \text{sid}, \text{ ssid, }P_j, P_i, |m|)\) to \( S \) if both \( P_i \) and \( P_j \) are honest, and \((\text{SC}.\text{Send}, \text{sid}, \text{ ssid, }P_i, P_j, m)\) otherwise. Wait to receive \((\text{SC}.\text{Send}, \text{sid}, \text{ ssid, }P_i, P_j, \text{ack})\) from \( S \). If \((P_i, \text{ ssid, active})\) is stored, then send \((\text{SC}.\text{Receive}, \text{sid}, \text{ ssid, }m)\) to \( \tilde{P}_i \).

- **Session Ending**
  
  Upon receiving \((\text{SC}.\text{ExpireSession}, \text{sid}, \text{ssid})\) from \( \tilde{P}_i/\tilde{P}_j \), remove \((P_i, \text{ ssid, active})/(P_j, \text{ ssid, active})\), and send \((\text{SC}.\text{ExpireSession}, \text{sid}, \text{ssid}, P_i/P_j)\) to \( S \).

**Figure 8.** The secure channel functionality \( \mathcal{F}_\text{SC} \).
\(\pi_{\text{AOS}}\) runs as follows, with a bank \(B\), a server \(Q\), users \(P_1, \ldots, P_n\) and an adversary \(A\).

**Set-up**
- Upon the first input \((\text{AOS.Register}, sid_1)\), where \(sid_1 = (B, sid')\), \(B\) sends \((\text{ABS.KeyGen}, sid_2)\) to \(\mathcal{F}_{\text{ABS}}\). Upon receiving \((\text{ABS.Key}, sid_2, pk)\) from \(\mathcal{F}_{\text{ABS}}\), \(B\) sends \((\text{ACA.Register}, sid_3, pk)\) to \(\mathcal{F}_{\text{ACA}}\). Upon receiving \((\text{ACA.Registered}, sid_1)\) from \(\mathcal{F}_{\text{ACA}}\), \(B\) sends \((\text{SC.Register}, sid_4)\) to \(\mathcal{F}_{\text{SC}}\). Upon receiving \((\text{SC.Registered}, sid_4)\) from \(\mathcal{F}_{\text{SC}}\), \(B\) outputs \((\text{AOS.Registered}, sid_1)\).
- Upon the first input \((\text{AOS.Register}, sid_1)\), where \(sid_1 = (B, sid')\), \(Q\) sends \((\text{ACA.Retrieve}, sid_3, B)\) to \(\mathcal{F}_{\text{ACA}}\). Upon receiving \((\text{ACA.Retrieved}, sid_3, B, pk)\) from \(\mathcal{F}_{\text{ACA}}\), \(Q\) sends \((\text{ASC.Register}, sid_5)\) to \(\mathcal{F}_{\text{ASC}}\), and waits to receive \((\text{ASC.Registered}, sid_5)\) from \(\mathcal{F}_{\text{ASC}}\).

**Session Set-up**
1. Upon input \((\text{AOS.EstablishSession}, sid_1, ssid, Q)\), where \(sid_1 = (B, sid')\), \(P_i\) sends \((\text{ASC.EstablishSession}, sid_5, ssid, Q)\) to \(\mathcal{F}_{\text{ASC}}\). Upon receiving \((\text{ASC.Established}, sid_5, ssid, Q)\) from \(\mathcal{F}_{\text{ASC}}\), \(P_i\) sends \((\text{ACA.Retrieve}, sid_3, B)\) to \(\mathcal{F}_{\text{ACA}}\). Upon receiving \((\text{ACA.Retrieved}, sid_3, B, pk)\) from \(\mathcal{F}_{\text{ACA}}\), \(P_i\) sends \((\text{ASC.Send}, sid_5, ssid, \text{AOS.Pay})\) to \(\mathcal{F}_{\text{ASC}}\).
2. Upon receiving \((\text{ASC.Receive}, sid_5, ssid, \text{AOS.Pay})\) from \(\mathcal{F}_{\text{ASC}}\), \(Q\) picks a random challenge \(c\), and sends \((\text{ASC.Send}, sid_5, ssid, (\text{AOS.Challenge}, c))\) to \(\mathcal{F}_{\text{ASC}}\).
3. Upon receiving \((\text{ASC.Receive}, sid_5, ssid, (\text{AOS.Challenge}, c))\) from \(\mathcal{F}_{\text{ASC}}\), \(P_i\) sends \((\text{ABS.Sign}, sid_2, ssid, pk, c)\) to \(\mathcal{F}_{\text{ABS}}\).
4. Upon receiving \((\text{ABS.Sign}, sid_2, ssid, P_i, request)\) from \(\mathcal{F}_{\text{ABS}}\), \(B\) outputs \((\text{AOS.Pay}, sid_1, ssid, P_i, request)\). \(B\) proceeds as follows:
   - Upon input \((\text{AOS.Pay}, sid_1, ssid, no)\), \(B\) sends \((\text{ABS.Sign}, sid_2, ssid, no)\) to \(\mathcal{F}_{\text{ABS}}\).
   - Upon input \((\text{AOS.Pay}, sid_1, ssid, yes)\), \(B\) sends \((\text{ABS.Sign}, sid_2, ssid, yes)\) to \(\mathcal{F}_{\text{ABS}}\).
5. \(P_i\) proceeds as follows:
   - Upon receiving \((\text{ABS.Sign}, sid_2, ssid, no)\) from \(\mathcal{F}_{\text{ABS}}\), \(P_i\) outputs \((\text{AOS.Pay}, sid_1, ssid, no)\).
   - Upon receiving \((\text{ABS.Signature}, sid_2, ssid, \sigma)\) from \(\mathcal{F}_{\text{ABS}}\), \(P_i\) sends \((\text{ASC.Send}, sid_5, ssid, (\text{AOS.Response}, \sigma))\) to \(\mathcal{F}_{\text{ASC}}\), sets a boolean variable \(active\) and outputs \((\text{AOS.Established}, sid_1, ssid, Q)\).
6. Upon receiving \((\text{ASC.Receive}, sid_5, ssid, (\text{AOS.Response}, \sigma))\) from \(\mathcal{F}_{\text{ASC}}\), \(Q\) sends \((\text{ABS.Verify}, sid_2, pk, c, \sigma)\) to \(\mathcal{F}_{\text{ABS}}\). Upon receiving \((\text{ABS.Verify}, sid_2, \text{accept})\) from \(\mathcal{F}_{\text{ABS}}\), \(Q\) stores \((c, \sigma)\), sets a boolean variable \(active\) and outputs \((\text{AOS.Established}, sid_1, ssid)\).

Figure 9. The anonymous online service protocol \(\pi_{\text{AOS}}\), part 1.
\( \pi_{\text{AOS}} \) runs as follows, with a bank \( B \), a server \( Q \), users \( P_1, \ldots, P_n \) and an adversary \( A \).

- **Data Exchange**
  1. On input \((\text{AOS.Send}, sid_1, ssid, m)\), if \( P_i/Q \) is active, then he sends \((\text{ASC.Send}, sid_5, ssid, m)\) to \( \mathcal{F}_\text{ASC} \).
  2. Upon receiving \((\text{ASC.Receive}, sid_5, ssid, m)\) from \( \mathcal{F}_\text{ASC} \), if \( Q/P_i \) is active, then he outputs \((\text{AOS.Receive}, sid_1, ssid, m)\).

- **Session Ending**
  Upon input \((\text{AOS.ExpireSession}, sid_1, ssid)\), \( P_i/Q \) unsets the active variable and sends \((\text{ASC.ExpireSession}, sid_5, ssid)\) to \( \mathcal{F}_\text{ASC} \).

- **Collect**
  1. On input \((\text{AOS.Collect}, sid_1, ssid', n)\), if \( Q \) has at least \( n \) stored pairs \((c, \sigma)\), \( Q \) sends \((\text{SC.EstablishSession}, sid_4, ssid', B)\) to \( \mathcal{F}_\text{SC} \).
  2. Upon receiving \((\text{SC.SessionEstablished}, sid_4, ssid', B)\) from \( \mathcal{F}_\text{SC} \), \( Q \) lets \( s \) be a set of \( n \) pairs \((c, \sigma)\), and sends \((\text{SC.Send}, sid_4, ssid', (\text{AOS.Collect}, s))\) to \( \mathcal{F}_\text{SC} \).
  3. Upon receiving \((\text{SC.Receive}, sid_4, ssid', (\text{AOS.Collect}, s))\) from \( \mathcal{F}_\text{SC} \), for every pair \((c, \sigma)\) in \( s \), \( B \) checks that there is no stored pair \((c, \sigma')\) for any \( \sigma' \), and sends \((\text{ABS.Verify}, sid_2, pk, c, \sigma)\) to \( \mathcal{F}_\text{ABS} \). If, for every pair, \( B \) receives \((\text{ABS.Verify}, sid_2, \text{accept})\) from \( \mathcal{F}_\text{ABS} \), \( B \) stores \( s \) and sends \((\text{SC.Send}, sid_4, ssid', \text{AOS.Collect})\) to \( \mathcal{F}_\text{SC} \). \( B \) then sends \((\text{SC.ExpireSession}, sid_4, ssid')\) to \( \mathcal{F}_\text{SC} \) and outputs \((\text{AOS.Deposit}, sid_1, ssid', Q, n)\).
  4. Upon receiving \((\text{SC.Receive}, sid_4, ssid', \text{AOS.Collect})\) from \( \mathcal{F}_\text{SC} \), \( Q \) removes the pairs \((c, \sigma)\) corresponding to \( s \), \( Q \) then sends \((\text{SC.ExpireSession}, sid_4, ssid')\) to \( \mathcal{F}_\text{SC} \) and outputs \((\text{AOS.Collect}, sid_1, ssid', n)\).

**Figure 10.** The anonymous online service protocol \( \pi_{\text{AOS}} \), part 2.
Paper V

Secure Messaging from Signcryption:
An Application of Formal Methods to Universal Composability

Lillian Kråkmo

Preprint
SECURE MESSAGING FROM SIGNCRYPTION: AN APPLICATION OF FORMAL METHODS TO UNIVERSAL COMPOSABILITY

LILLIAN KRÁKMO

Abstract. The framework of universally composable security has received a lot of attention due to its strong security guarantees for cryptographic protocols running in arbitrary environments. A concern is that security proofs within this framework are typically long and prone to errors. This paper presents an application of state machine theory, which may be considered as a first step towards allowing for automatic verification of such proofs. As an example, we consider a hybrid protocol using functionalities for a public key infrastructure and signcryption, and prove that the protocol realizes a functionality for secure messaging.

1. Introduction

Traditionally, we have seen two main approaches for security analysis of cryptographic protocols. In computational models, cryptographic primitives are treated as algorithms, and adversaries are computationally bounded entities with access to the inputs and outputs of these algorithms. Security is typically defined in a probabilistic sense, and relies on computational intractability assumptions. A variety of such models have been proposed [9, 10, 3]. A common feature of these models is that security notions tend to get relatively complex, even for simple protocols. Furthermore, the security proofs require some level of human creativity, since breaking the security of the protocol in question must be reduced to breaking some underlying hard problem. Accordingly, such proofs are not likely to be amenable to automation. Symbolic models, on the other hand, treat cryptographic primitives as symbolic operations, which immediately guarantee a set of idealized security properties. As a result, protocol analysis becomes considerably simpler, and easier to mechanize. Examples of such models are the Dolev-Yao model [8], the BAN logic [4] and the Spi-calculus [1]. However, when it comes to guaranteeing security for protocols running in realistic settings, the computational approach is the only one that is obviously sound.

In recent years, many attempts have been made at combining the above approaches, in order to obtain both soundness and automation of analysis. One such approach is represented by models that guarantee security-preserving composition [11, 2, 5]. In such models, primitives are represented by idealized abstractions. These abstractions are realizable by actual protocols, and may also
be deployed as subroutines by higher-level protocols, thus allowing for a more mechanical security analysis. At the same time, soundness is guaranteed by a strong composition theorem, which ensures that a protocol using an abstraction retains its security when the abstraction is replaced by a realizing protocol. Our work focuses on one specific such model, namely the framework of universally composable (UC) security, developed by Canetti [5]. In the UC framework, the idealized abstractions appear as ideal functionalities, and protocols that use ideal functionalities as subroutines are referred to as hybrid protocols. We refer to [5] for a full overview of this framework.

Although the UC framework simplifies analysis of complex protocols, it still requires that security proofs be obtained within a full-fledged cryptographic model, hence proofs tend to get tedious and error-prone. This problem is addressed by Canetti and Herzog in [7, 6], where they demonstrate how symbolic analysis within a simple model can be used to argue about the UC security of a concrete protocol. More specifically, they show how a concrete protocol in the UC framework can be translated into a simpler, symbolic protocol in the Dolev-Yao model, and prove that, if the translated protocol satisfies a certain symbolic criterion, then the original protocol is UC secure.

This paper, which is an extension of an unpublished manuscript by Gjøsteen [12], presents an alternative approach to solve this problem, by using state machine theory. In the UC framework, the parties running a protocol, as well as the adversarial entities, are represented as interactive Turing machines (ITMs). Security of a protocol is defined by a game, where the environment machine \( Z \) tries to decide whether it is interacting with the real protocol or with the ideal protocol for the task at hand. The basic idea behind our approach is simple: The above scenarios give rise to two systems of ITMs. The set of possible messages in such a system can be viewed as an alphabet, and the string of messages sent in a run of the system can be considered a word. Then, the language consisting of all such words completely describes the possible communication patterns of the system. When trying to distinguish between the two systems, the only information \( Z \) can benefit from is the messages it exchanges with the rest of the system. We argue that, if the languages of the systems are identical when restricted to these messages, and the machines representing the protocols are all deterministic, then the two systems look identical from \( Z \)'s point of view. This means that the protocol securely realizes the functionality for the task in question.

As an example, we consider a hybrid protocol using functionalities for a public key infrastructure and signcryption, and prove that the protocol realizes a functionality for secure messaging. Most of the steps involved in the proof boil down to mechanical manipulations of state machines, which should be easy to implement on a computer. Our approach can thus be considered as a first step towards automatic verification of such proofs. We emphasize that this paper presents ongoing work. For instance, our approach currently only applies to protocols where the Turing machines representing the parties are all deterministic. Eliminating this restriction is an object of further study.
In Section 2, we introduce some basic notions concerning state machines and languages. Several lemmas are derived, which will be useful when proving our main result. Moreover, we obtain a theorem, which allows us to use state machine theory to prove security of certain protocols within the UC framework. In Section 3, we present our hybrid protocol along with the relevant functionalities, and construct state machines that recognize the languages arising from $\mathcal{Z}$ interacting with either the hybrid protocol or the ideal protocol for secure messaging. Finally, in Section 4, we prove that the hybrid protocol realizes the secure messaging functionality.

2. Languages and State Machines

2.1. State Machines. A deterministic state machine $M$ is a tuple $(S, I, f, W, s_0)$ where $S$ and $I$ are sets, $f : S \times I \to S$ is a partial function, $W$ is a non-empty subset of $S$, and $s_0 \in S$. $S$ is the set of states, $I$ is the machine’s input alphabet, $f$ is the state transition function, and $W$ is the set of wait states or accepting states, and $s_0$ is the initial state.

Note that we will sometimes consider the partial function $f$ to be a subset of $S \times I \times S$. Also, $\text{dom}(f)_2$ denotes the subset of elements $w$ in $I$ for which at least one state $s$ exists such that $f(s, w)$ is defined.

Let $M_1 = (S_1, f_1, I_1, W_1, s_1)$ and $M_2 = (S_2, f_2, I_2, W_2, s_2)$ be state machines. A map $\mu : M_1 \to M_2$ is a pair $(\sigma : S_1 \to S_2, \tau : I_1 \to I_2)$, where $\sigma$ is a function and $\tau$ is a partial function, such that $\sigma(s_1) = s_2$, $\sigma(W_1) \subseteq W_2$ and for all $s \in S_1$, $w \in I_1$, if $f_1(s, w)$ is defined, then

$$\sigma(f_1(s, w)) = f_2(\sigma(s), \tau(w))$$

if $\tau(w)$ is defined, and $\sigma(f_1(s, w)) = \sigma(s)$ if $\tau(w)$ is undefined. Furthermore, if $\sigma$ is a bijection, $\tau$ restricted to $\text{dom}(f_1)_2$ is a bijection on $\text{dom}(f_2)_2$, $\sigma(W_1) = W_2$ and $f_1(s, w)$ is defined if and only if $f_2(\sigma(s), \tau(w))$ is defined, then the map $\mu$ is an isomorphism.

2.2. Languages. An alphabet $I$ is a set of letters. Given an alphabet $I$, $I^*$ denotes the set of strings of elements from $I$. Furthermore, a language $L$ over $I$ is a subset of $I^*$.

A state machine $(S, I, f, W, s_0)$ can process a string $w_1w_2\ldots w_n \in I^*$ using the following procedure: The machine is in state $s_{i-1}$ when processing input letter $w_i$. If $f(s_{i-1}, w_i)$ is not defined, the machine stops and the input string is not recognized. Otherwise, the machine moves to state $s_i = f(s_{i-1}, w_i)$ and processes the next input letter. If the machine reaches a state $s_n$ without stopping, the machine recognizes the input string if $s_n \in W$, otherwise it does not recognize the input string.

The language recognized by a machine $M$ is the subset of strings in $I^*$ that are recognized by the machine, denoted $L(M)$.

Let $I_1$ and $I_2$ be two alphabets. From a partial function $\tau : I_1 \to I_2$ we get a function $\tau : I_1^* \to I_2^*$ by applying $\tau$ to each letter, removing letters for which $\tau$ is undefined.
Lemma 1. Let \( M_1 \) and \( M_2 \) be state machines. If \((\sigma, \tau)\) is a map from \( M_1 \) to \( M_2 \), then
\[
\tau(L(M_1)) \subseteq L(M_2).
\]

If \((\sigma, \tau)\) is an isomorphism, then the map \( \tau : L(M_1) \to L(M_2) \) is a bijection.

Proof. Let \( M_1 = (S_1, f_1, I_1, W_1, s_1) \) and \( M_2 = (S_2, f_2, I_2, W_2, s_2) \). Consider a word \( w_1 w_2 \ldots w_n \in I_1^* \). Suppose \( M_1 \) recognizes this word. Then we have states \( s_{1,0}, s_{1,1}, \ldots, s_{1,n} \in S_1 \) where \( s_{1,0} = s_1 \), \( s_{1,i} = f_1(s_{1,i-1}, w_i) \) for \( 1 \leq i \leq n \) and \( s_{1,n} \in W_1 \).

By the map properties, we have that \( \sigma(s_{1,0}) = s_{2,0}, f_2(\sigma(s_{1,i-1}), \tau(w_i)) = \sigma(s_{1,i}) \) if \( \tau(w_i) \) is defined, otherwise \( \sigma(s_{1,i-1}) = \sigma(s_{1,i}) \), for \( 1 \leq i \leq n \), and \( \sigma(s_{1,n}) \in W_2 \). Therefore \( M_2 \) recognizes \( \tau(w_1 w_2 \ldots w_n) \), and \( \tau(L(M_1)) \subseteq L(M_2) \).

Assume that \((\sigma, \tau)\) is an isomorphism, and consider a word \( v_1 v_2 \ldots v_m \in L(M_2) \). We will show that there is a unique word \( w_1 w_2 \ldots w_{m'} \in L(M_1) \) such that \( \tau(w_1 w_2 \ldots w_{m'}) = v_1 v_2 \ldots v_m \). Since \( v_1 v_2 \ldots v_m \in L(M_2) \), we have states \( s_{2,0}, s_{2,1}, \ldots, s_{2,m} \in S_2 \), where \( s_{2,0} = s_2 \), \( s_{2,i} = f_2(s_{2,i-1}, v_i) \) for \( 1 \leq i \leq m \) and \( s_{2,m} \in W_2 \). Recall that \( \sigma(s_{1,0}) = s_{2,0} \). Since \((\sigma, \tau)\) is an isomorphism, \( \sigma \) is a bijection, so for \( 1 \leq i \leq m \), we have a unique state \( s_{1,i} \in S_1 \) such that \( \sigma(s_{1,i}) = s_{2,i} \).

Moreover, \( \tau \) restricted to \( dom(f_1)_2 \) is a bijection on \( dom(f_2)_2 \), so for \( 1 \leq i \leq m \), we have a unique letter \( w_i \in dom(f_1)_2 \) such that \( \tau(w_i) = v_i \). This means that, for \( 1 \leq i \leq m \), \( f_2(\sigma(s_{1,i-1}), \tau(w_i)) \) is defined, which implies that \( f_1(s_{1,i-1}, w_i) \) is defined. Then, by the map properties, \( \sigma(f_1(s_{1,i-1}, w_i)) = f_2(s_{2,i-1}, v_i) = s_{2,i} \), so \( f_1(s_{1,i-1}, w_i) = s_{1,i} \). Consequently, after processing \( w_1 w_2 \ldots w_m, M_1 \) will be in state \( s_{1,m} \). Since \( \sigma(s_{1,m}) = s_{2,m} \) and \( s_{2,m} \in W_2 \), we have \( s_{1,m} \in W_1 \), hence \( w_1 w_2 \ldots w_3 \in L(M_1) \), and \( \tau(w_1 w_2 \ldots w_m) = v_1 v_2 \ldots v_m \).

Now assume that there is another word \( w'_1 w'_2 \ldots w'_{m'} \in L(M_1) \) such that \( \tau(w'_1 w'_2 \ldots w'_{m'}) = v_1 v_2 \ldots v_m \). Then we have states \( s'_{1,0}, s'_{1,1}, \ldots, s'_{1,m'} \in S_1 \) where \( s'_{1,0} = s_1 \), \( s'_{1,i} = f_1(s'_{1,i-1}, w'_i) \) for \( 1 \leq i \leq m' \) and \( s'_{1,m'} \in W_1 \). When \( f_1(s'_{1,i-1}, w'_i) \) is defined, so is \( f_2(\sigma(s'_{1,i-1}), \tau(w'_i)) \), which in particular means that \( \tau(w'_i) \) is defined. Hence \( m' = m \), and for \( 1 \leq i \leq m' \), we must have \( \tau(w'_i) = v_i \). Since restricted to \( dom(f_1)_2 \) is a bijection on \( dom(f_2)_2 \), this implies that \( w'_i = v_i \), so \( w'_1 w'_2 \ldots w'_{m'} = w_1 w_2 \ldots w_m \).

We want to make changes to a machine without essentially changing the language recognized by the machine. Some transitions are not essential, in the sense that the letter recognized by the transition does not contain any new information; the transition and the next state are completely determined by the previous state. The following lemma allows us to modify a machine by adding or removing certain states and transitions, while obtaining a bijection between the languages recognized by the original and the resulting machine.

Lemma 2. Let \( M = (S, f, I, W, s_0) \) be a machine, and let \( S_0 \subseteq S \) be a subset such that \( S_0 \cap (W \cup \{s_0\}) = \emptyset \). Suppose there exist a partial function \( g : S_0 \to I \) and a function \( h : S_0 \to S \setminus S_0 \) such that:

1. for all \( s \in S, w \in I \) and \( s' \in S_0 \), if \((s, w, s') \in f \), then the only transition out of \( s' \) in \( f \) is \((s', g(s'), h(s')) \),
(2) for all \( w \in g(S_0) \), if \( (s, w, s') \in f \) then \( s \in S_0 \).

Let \( M' = (S \setminus S_0, f', I \setminus I_0, W, s_0) \), where \( I_0 \subseteq g(S_0) \) and

\[
f' = \{(s, w, s') \in f \mid s, s' \notin S_0\} \cup \{(s, w, s'') \mid \exists s' \in S_0 : (s, w, s') \in f \land h(s') = s''\}.
\]

Then the map \( \tau : L(M) \to L(M') \) induced by

\[
\tau(w) = \begin{cases} 
  w & \text{if } w \notin g(S_0), \\
  \text{undefined} & \text{otherwise},
\end{cases}
\]

is a bijection.

**Proof.** Let \( w = w_1 w_2 \ldots w_n \in L(M') \). We shall prove that there is a unique word \( v = v_1 v_2 \ldots v_m \in L(M) \) such that \( \tau(v) = w \), hence \( \tau \) will be a bijection.

We construct the string \( v(i) \) from \( v(i-1) \). Suppose \( \tau(v(i-1)) = w_1 w_2 \ldots w_{i-1} \), and that both \( M \) and \( M' \) are in state \( s \) after processing \( v(i-1) \) and \( w_1 \ldots w_{i-1} \), respectively. Now consider \( w_i \), and let \( (s, w_i, s') \in f' \). If \( (s, w_i, s') \in f \), we set \( v(i) = v(i-1) w_i \). Then

\[
\tau(v(i)) = \tau(v(i-1)) \tau(w_i) = w_1 \ldots w_{i-1} w_i,
\]

and after processing \( w_i \), both \( M \) and \( M' \) will be in state \( s' \).

If \( (s, w_i, s') \notin f \), then there is an \( s'' \in S_0 \) such that \( h(s'') = s' \) and \( (s, w_i, s'') \in f \). Let \( w'_i = g(s'') \), and set \( v(i) = v(i-1) w_i w'_i \). Then

\[
\tau(v(i)) = \tau(v(i-1)) \tau(w_i) \tau(w'_i) = w_1 \ldots w_{i-1} w_i,
\]

and after processing \( w_i \), both \( M \) and \( M' \) will be in state \( s' \).

Let \( v(0) \) be the empty word. With the above rule, we construct the sequence \( v(0), v(1), \ldots, v(n) \) and set \( v = v(n) \). Since \( M' \) is in a wait state after processing \( w \), so must \( M \) be after processing \( v \), so \( v \in L(M) \) and by construction, \( \tau(v) = w \). This shows that \( \tau \) is onto. Also, \( v \) is because of the functions \( g \) and \( h \) the only possible preimage of \( w \), hence \( \tau \) is also injective.

It remains to prove that \( \tau(L(M)) \subseteq L(M') \). Assume that \( v_1 v_2 \ldots v_m \in L(M) \). Suppose \( \tau(v_1 v_2 \ldots v_{i-1}) = w(i-1) \), and that both \( M \) and \( M' \) are in state \( s \in S_0 \) after processing \( v_1 v_2 \ldots v_{i-1} \) and \( w(i-1) \), respectively. Consider \( v_i \), and assume that \( (s, v_i, s') \in f \). Then, since \( s \notin S_0 \), we have \( \tau(v_i) = v_i \). If \( s' \notin S_0 \), then \( (s, v_i, s') \in f' \), so after processing \( v_1 v_2 \ldots v_i \) and \( \tau(v_1 v_2 \ldots v_i) = w(i-1)v_i \), respectively, both \( M \) and \( M' \) are in state \( s' \notin S_0 \). If \( s' \in S_0 \), then the only valid transition out of \( s' \) in \( f \) is given by \( (s', g(s'), h(s')) \). This means that \( (s, v_i, h(s')) \in f' \). Furthermore, \( g(s') = v_{i+1} \) and \( \tau(v_{i+1}) \) is undefined. Accordingly, after processing \( v_1 v_2 \ldots v_{i+1} \) and \( \tau(v_1 v_2 \ldots v_{i+1}) = w(i-1)v_i \), respectively, both \( M \) and \( M' \) are in state \( h(s') \notin S_0 \).

Initially, both \( M \) and \( M' \) are in state \( s_0 \notin S_0 \). Assume that \( M \) is in state \( s_m \) after processing \( v_1 v_2 \ldots v_m \). Since \( v_1 v_2 \ldots v_m \in L(M) \), \( s_m \) is a wait state, so \( s_m \notin S_0 \). This means that, according to the above rule, both \( M \) and \( M' \) will be in state \( s_m \) after processing \( v_1 v_2 \ldots v_m \) and \( \tau(v_1 v_2 \ldots v_m) \), respectively. Since \( M \)
is in a wait state at this point, so is \( M' \), and we conclude that \( \tau(v_1v_2\ldots v_m) \in L(M') \).

\[ \square \]

Suppose we have a language \( L_1 \) over an alphabet \( I_1 \), and let \( I \) be a subset of \( I_1 \). The language \( L_1 \) restricted to the alphabet \( I \), denoted \( L_1|_I \), is formed by removing from the strings in \( L_1 \) any letter that is not in \( I \). Thus \( L_1|_I \subseteq I^* \). Any string consisting entirely of letters not in \( I \) would result in the empty string. If \( L_2 \) is a second language over an alphabet \( I_2 \), \( I \subseteq I_2 \), then we say that \( L_1 \) and \( L_2 \) are equivalent with respect to \( I \) if \( L_1|_I = L_2|_I \).

We say that two machines with input alphabets \( I_1 \) and \( I_2 \) are equivalent with respect to an alphabet \( I \subseteq I_1 \cap I_2 \) if the languages recognized by the machines are equivalent with respect to \( I \). Note that this relation is transitive, symmetric and reflexive. If \( \tau : I_1 \to I_2 \) is a partial function such that the induced language map is bijective and \( \tau \) keeps \( I \) fixed, then the two machines are equivalent with respect to \( I \). We state this result in a lemma for further reference.

**Lemma 3.** Let \( M_1 \) and \( M_2 \) be state machines with input alphabets \( I_1 \) and \( I_2 \), respectively, and let \( I \subseteq I_1 \cap I_2 \). If \( \tau : I_1 \to I_2 \) is a partial function such that the induced language map is bijective and \( \tau \) restricted to \( I \) is the identity map, then the two machines are equivalent with respect to \( I \).

### 2.3. Composition of State Machines

We shall define a composition of two state machines, to model the intuitive notion of two processors cooperating in recognizing a language. For every letter in the input string, one or both machines may recognize the input letter. If only one machine recognizes the input letter, the input letter must not be part of the other machine’s input alphabet, and the other machine must be in a wait state for the composite machine to recognize the input letter.

Let \( M_1 = (S_1, I_1, f_1, W_1, s_{1,0}) \) and \( M_2 = (S_2, I_2, f_2, W_2, s_{2,0}) \) be two state machines. Consider the set \( f_{12} \) given by

\[
f_{12} = \{ ((s_{1i}, s_{2k}), w, (s_{1j}, s_{2l})) \mid (s_{1i}, w, s_{1j}) \in f_1 \lor (w \notin I_1 \land s_{1i} = s_{1j} \land s_{1i} \in W_1) \land (s_{2k}, w, s_{2l}) \in f_2 \lor (w \notin I_2 \land s_{2k} = s_{2l} \land s_{2k} \in W_2) \}. \]

It is easy to verify that \( f_{12} \) is a partial function from \( S_1 \times S_2 \times (I_1 \cup I_2) \) to \( S_1 \times S_2 \).

**Definition 4.** The composite machine \( M_1 + M_2 \) is the tuple \((S_1 \times S_2, I_1 \cup I_2, f_{12}, W_1 \times W_2, (s_{1,0}, s_{2,0}))\).

It is straightforward to show the following lemma, which says that the composition operation is essentially commutative and associative.

**Lemma 5.** For state machines \( M_1, M_2 \) and \( M_3 \), \( M_1 + M_2 \) is isomorphic to \( M_2 + M_1 \), and \( (M_1 + M_2) + M_3 \) is isomorphic to \( M_1 + (M_2 + M_3) \).

Recall that, in our definition of an isomorphism, we require that \( \sigma \) be a bijection between the sets of states of the respective machines. In the following, we will call a state \( s \) of a machine reachable if there exists an input string that can
take the machine from the initial state to state $s$. In the composition $M_1 + M_2$, the machines $M_1$ and $M_2$ do not run independently, but are subject to the restrictions implied by the above definition. This means that the set of reachable states for $M_1 + M_2$ is typically small compared to the product $S_1 \times S_2$. A similar argument applies when considering the input alphabet of a composition of machines. We will call a letter $w$ in a machine’s input alphabet significant if there exists a reachable state $s$ such that the transition function is defined on input $(s, w)$. It is intuitive that, when arguing about the language recognized by a machine, it should be sufficient to consider the reachable states and the significant letters. On the other hand, identifying these states and letters is generally a non-trivial task. In many cases, it will be more convenient to identify certain subsets of states/letters, containing all reachable states/significant letters, and base arguments on these subsets.

Regarding the sufficiency of considering reachable states, it is straightforward to verify the following result.

**Corollary 6.** *Lemma 2 still holds when restricting the first condition to the reachable $s \in S$.*

In order to formalize the above ideas, we introduce the notion of a minimal machine as follows: If $M = (S, I, f, W, s_0)$, then $\text{minimal}(M) = (\hat{S}, \hat{I}, \hat{f}, \hat{W}, s_0)$, where $\hat{S}$ consists of the reachable states in $S$, and $\hat{I}$ consists of the significant letters in $I$. Furthermore, $\hat{f}$ contains exactly those triples $(s, w, s')$ in $f$ where $s$ is a reachable state, and $\hat{W}$ consists of the wait states contained in $\hat{S}$. Moreover, a machine $\hat{M} = (\hat{S}, \hat{I}, \hat{f}, \hat{W}, s_0)$ is said to include $\text{minimal}(M)$ if $\hat{S} \subseteq \hat{S} \subseteq S$, $\hat{I} \subseteq \hat{I} \subseteq I$, $\hat{f}$ contains exactly those triples $(s, w, s')$ in $f$ such that $s \in \hat{S}$ and $w \in \hat{I}$, and $\hat{W}$ consists of the wait states contained in $\hat{S}$. For ease of presentation, we will sometimes speak of the machine $M$ restricted to some subset $\hat{S}$ of $S$, implying that the necessary changes are made to $f$ and $W$ in order to satisfy the above conditions.

Clearly, any machine including $\text{minimal}(M)$ recognizes the same language. This means in particular that, if $I$ is a subset of $\hat{I}$, then any pair of machines including $\text{minimal}(M)$ are equivalent with respect to $I$. The following lemma is included for further reference.

**Lemma 7.** *Let $M$ be a state machine, let $M_1$ and $M_2$ be machines including $\text{minimal}(M)$, and let $I$ be a subset of the input alphabet of $\text{minimal}(M)$. Then $M_1$ and $M_2$ are equivalent with respect to $I$.***

**Corollary 8.** *Let $M_1$ and $M_2$ be state machines, let $M'_1$ and $M'_2$ be machines including $\text{minimal}(M_1)$ and $\text{minimal}(M_2)$, respectively, and let $I$ be a common subset of the input alphabets of $\text{minimal}(M_1)$ and $\text{minimal}(M_2)$. If $M'_1$ and $M'_2$ are equivalent with respect to $I$, then $M_1$ and $M_2$ are equivalent with respect to $I$.***

2.4. **Realizing Functionalities.** In the UC framework, an environment interacting with a protocol is modeled as a system of communicating Turing machines.
A message sent from TM A to TM B in such a system can be represented as a tuple \((A, B, \text{Name}, \text{content})\). We can consider the set of possible messages in the system to be an alphabet. The string of all the messages actually sent in a run of the system can be considered a word, and the language \(L\) containing all such words completely describes all possible communication patterns for the system.

The environment \(Z\)'s job is to determine something about the rest of the system. The only information it can benefit from is the messages it exchanges with the rest of the system. In this paper, we assume security within the UC framework to be defined with respect to the dummy adversary, thus messages to and from the adversary may be considered as messages to and from \(Z\). Under this assumption, if we let \(I\) be the set of messages where \(Z\) is either the sender or recipient, then the restricted language \(L|_I\) completely describes \(Z\)'s information.

Suppose we have two protocols \(\pi_1\) and \(\pi_2\), and \(Z\) is trying to determine which protocol it is talking to. This gives rise to two systems, one with \(Z\) and the Turing machines of \(\pi_1\), and one with \(Z\) and the Turing machines of \(\pi_2\), defining two languages of possible communication patterns \(L_1\) and \(L_2\). Let \(I_1\) and \(I_2\) be the sets of possible messages to and from \(Z\) in these systems.

Note that if a protocol's Turing machines are all deterministic, the protocol's responses to \(Z\)'s input will be completely determined by the input. If in this case \(I_1 = I = I_2\) and \(L_1|_I = L_2|_I\), then for a given sequence of previous messages from \(Z\), the next response by the protocols must be the same. This means that no matter what input \(Z\) gives to the protocols, the responses will always be identical. In other words, it will be impossible for \(Z\) to decide which protocol it is talking to. We have therefore proved the following theorem:

**Theorem 9.** Let \(\pi_1\) and \(\pi_2\) be protocols consisting entirely of deterministic Turing machines. Let \(L_1\) and \(L_2\) be the languages determined by the system where an environment \(Z\) interacts with \(\pi_1\) and \(\pi_2\), respectively. Let \(I_1\) and \(I_2\) be the sets of possible messages to and from \(Z\) in these systems. If \(I_1 = I_2 = I\) and \(L_1|_I = L_2|_I\), then \(\pi_1\) securely realizes \(\pi_2\).

This theorem allows us to apply the state machine theory. If we can find two machines \(M_1\) and \(M_2\) recognizing the languages \(L_1\) and \(L_2\), respectively, such that \(M_1\) and \(M_2\) are equivalent with respect to the alphabet \(I\), then we have proved that \(\pi_1\) realizes \(\pi_2\).

**3. Functionalities and State Machines**

In general, while a functionality is processing a message from a party, no further messages from that party will be accepted by the functionality until the current processing is done. Messages from other parties will be accepted and processed, however.

We want to prove that a hybrid protocol \(\pi_{SM}\) using PKI and signcryption functionalities \(\mathcal{F}_{PKI}\) and \(\mathcal{F}_{SC}\) realizes a secure messaging functionality \(\mathcal{F}_{SM}\). We now proceed to describe the functionalities informally, together with state machines.
that recognize the possible communication patterns arising from an environment \( Z \) interacting either with the hybrid protocol \( \pi_{SM} \) or the ideal functionality \( F_{SM} \).

The descriptions of the functionalities include some behavior that is not relevant in this context. To simplify the descriptions of the state machines, this behavior is not modeled. Also, in order to save space, we do not consider corruption of parties.

The state machine descriptions specify the possible states, the wait states and the initial states. The state transition function is described through triples of the form

\[
(s, w, s') \quad \text{and} \quad (s, \text{conditions}/w, s'),
\]

meaning that \( f(s, w) = s' \) if the optionally specified conditions hold for \( s, w \) and \( s' \). The state \( s \) will be given in an abbreviated form whenever the complete description can be inferred from the rest of the table. The input alphabet (the set of possible messages recognized by the machine) is implicitly determined by the description of the state transition function.

Let \( \Sigma \) be the set of bit strings, and let \( T \) be the set of tuples of bit strings. We shall also consider sets of tuples where each entry is either a bit string or blank (denoted by the special symbol \(-\)). Note that a blank entry is distinct from an entry with an empty string. These tuples are called patterns, and a pattern \( x = (x_1, x_2, \ldots, x_n) \) matches a tuple of bit strings \( y = (y_1, y_2, \ldots, y_m) \) if and only if \( m = n \) and for \( 1 \leq i \leq n \), either \( x_i \) is a blank entry or \( x_i = y_i \). We denote this by \( y \sim x \).

**Buffer Machine.** Modeling the communication patterns is simplified by introducing a “buffer machine”. A buffer machine essentially manages a list of tuples of bit strings. It accepts messages that either add tuples to the list, or ask if tuples matching a certain pattern are in the list. The per-player machine \( M_{PKI}(i) \) is

\[
(\{s_0, s_1, \ldots, s_{10}\} \times (\Sigma \cup \{-\}), I, f, \{s_0, s_2, s_7\} \times (\Sigma \cup \{-\}), (s_0, -)),
\]

**PKI Functionality.** The PKI functionality is given in Figure 2. The machine modeling the PKI functionality is composed of one machine for each player together with a buffer machine, \( B_{PKI} \). The per-player machine \( M_{PKI}(i) \) is

\[
(\{s_0, s_1, \ldots, s_{10}\} \times (\Sigma \cup \{-\}), I, f, \{s_0, s_2, s_7\} \times (\Sigma \cup \{-\}), (s_0, -)),
\]
where the state transition function $f$ is described in Figure 3. The input alphabet $I$ is implicit in the description of $f$.

The PKI machine is

$$M_{PKI} = B_{PKI} + \sum_i M_{PKI}(i).$$

When $(PKI.\text{Register}, v)$ is received from $\tilde{P}_i$:

1. If $F_{PKI}$ has already processed a $PKI.\text{Register}$ message from $\tilde{P}_i$, send $PKI.\text{Error}$ to $\tilde{P}_i$ and stop.
2. Send $(PKI.\text{Register}, P_i, v)$ to $A$ and wait for $(PKI.\text{Register}.\text{OK}, P_i)$ from $A$.
3. Store $(P_i, v)$ in the registration buffer and send $(PKI.\text{Registered})$ to $\tilde{P}_i$.

When $(PKI.\text{Retrieve}, P_i)$ is received from $\tilde{P}_j$:

1. Send $(PKI.\text{Retrieve}, P_j, P_i)$ to $A$ and wait for $(PKI.\text{Retrieve}.\text{OK}, P_j, P_i)$ from $A$.
2. If there is an entry $(P_i, v)$ in the registration buffer, send $(PKI.\text{Retrieved}, v)$ to $\tilde{P}_j$, otherwise send $(PKI.\text{Retrieved}, \bot)$ to $\tilde{P}_j$.

**Figure 2.** The PKI functionality $F_{PKI}$.

**Signcryption Functionality.** The signcryption functionality is given in Figure 4. The machine modeling the signcryption functionality is composed of one machine for each player together with a buffer machine $B_{SC}$. The per-player

<table>
<thead>
<tr>
<th>$s$</th>
<th>$w$</th>
<th>$s'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$(P_i, F_{PKI}, PKI.\text{Register}, v)$</td>
<td>$(s_1, v)$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$(F_{PKI}, A_{PKI}, PKI.\text{Register}, P_i, v)$</td>
<td>$(s_2, v)$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$(A_{PKI}, F_{PKI}, PKI.\text{Register}.\text{Ok}, P_i)$</td>
<td>$(s_3, v)$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$(F_{PKI}, B_{PKI}, B.\text{Add}, (P_i, v))$</td>
<td>$(s_4, -)$</td>
</tr>
<tr>
<td>$s_4$</td>
<td>$(F_{PKI}, P_i, PKI.\text{Register}.\text{Ok})$</td>
<td>$(s_5, -)$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>$(P_i, F_{PKI}, PKI.\text{Retrieve}, P_j)$</td>
<td>$(s_6, P_j)$</td>
</tr>
<tr>
<td>$s_6$</td>
<td>$(F_{PKI}, A_{PKI}, PKI.\text{Retrieve}, P_i, P_j)$</td>
<td>$(s_7, P_j)$</td>
</tr>
<tr>
<td>$s_7$</td>
<td>$(A_{PKI}, F_{PKI}, PKI.\text{Retrieve}.\text{Ok}, P_i, P_j)$</td>
<td>$(s_8, P_j)$</td>
</tr>
<tr>
<td>$s_8$</td>
<td>$(F_{PKI}, B_{PKI}, B.\text{Lookup}, (P_j, -), (P_j, v))$</td>
<td>$(s_9, v)$</td>
</tr>
<tr>
<td>$s_9$</td>
<td>$(F_{PKI}, P_i, PKI.\text{Retrieve}.\text{Ok}, v)$</td>
<td>$(s_5, -)$</td>
</tr>
<tr>
<td>$s_{10}$</td>
<td>$(F_{PKI}, B_{PKI}, B.\text{Lookup}, (P_j, -), \bot)$</td>
<td>$(s_{10}, -)$</td>
</tr>
<tr>
<td>$s_{10}$</td>
<td>$(F_{PKI}, P_i, PKI.\text{Retrieve}.\text{Fail})$</td>
<td>$(s_{5}, -)$</td>
</tr>
</tbody>
</table>

**Figure 3.** PKI per-player machine for player $i$. 
machine $M_{SC}(i)$ is
\[
(\{s_0, s_1, \ldots, s_{19}\} \times (T \cup \{-\}), I, f, \{s_0, s_2, s_5, s_8, s_{10}, s_{14}\} \times (T \cup \{-\})), (s_0, -)),
\]
where the state transition function $f$ is described in Figure 5. The input alphabet $I$ is implicit in the description of $f$.

The signcryption machine is
\[
M_{SC} = B_{SC} + \sum_i M_{SC}(i).
\]

**Protocol Machines.** The protocol machines for the protocol $\pi_{SM}$ are given in Figure 6. The machine $M_{\pi}(i)$ modeling the protocol machine used by the $i$'th player is
\[
(\{s_0, s_1, \ldots, s_{20}\} \times (T \cup \{-\}), I, f, \{s_0, s_2, s_4, s_6, s_8, s_{10}, s_{15}, s_{18}\} \times (T \cup \{-\})), (s_0, -)),
\]
where the state transition function $f$ is described in Figure 5. The input alphabet $I$ is implicit in the description of $f$.

**Dummy Parties.** When the environment interacts with the ideal protocol, it sends messages to the secure messaging functionality via dummy parties that simply forward messages. The machine $M_{\tilde{P}}(i)$ is
\[
(\{s_0, s_1, s_2\} \times (2T \cup \{-\}), I, f, \{(s_0, -)\}, (s_0, -)),
\]
where the state transition function $f$ is described in Figure 8. The input alphabet $I$ is implicit in the description of $f$.

**Secure Messaging Functionality.** The secure messaging functionality is given in Figure 9. The machine modeling the secure messaging functionality is composed of one machine for each player together with a buffer machine $B_{SM}$. The per-player machine $M_{SM}(i)$ is
\[
(\{s_0, s_1, \ldots, s_{19}\} \times (T \cup \{-\}), I, f, \{s_0, s_2, s_5, s_7, s_{13}\} \times (T \cup \{-\})), (s_0, -)),
\]
where the state transition function $f$ is described in Figure 10. The input alphabet $I$ is implicit in the description of $f$.

The secure messaging machine is
\[
M_{SM} = B_{SM} + \sum_i M_{SM}(i).
\]

Note that we ignore the dummy per-player protocol machines.

**Ideal Adversary.** The ideal adversary used in the proof is given in Figure 11. The machine modeling the ideal adversary is composed of one machine for each player together with a buffer machine $B_S$. The per-player machine $M_S(i)$ is
\[
(\{s_0, s_1, \ldots, s_{19}\} \times (T \cup \{-\}), I, f, \{s_0, s_2\} \times (T \cup \{-\})), (s_0, -)),
\]
where the state transition function $f$ is described in Figure 10. The input alphabet $I$ is implicit in the description of $f$.

The ideal adversary machine is
\[
M_S = B_S + \sum_i M_S(i).
\]
When \textbf{SC.KeyGen} is received from $\tilde{P}_i$:

(1) If $F_{SC}$ has already processed an \textbf{SC.KeyGen} message from $\tilde{P}_i$, send \textbf{SC.Error} to $\tilde{P}_i$ and stop.

(2) Send (\textbf{SC.KeyGen}, $P_i$) to the adversary $A$ and wait for (\textbf{SC.Key}, $P_i$, $pk_i^s$, $pk_i^r$) from $A$. (We note that $A$ is free to choose ($pk_i^s$, $pk_i^r$), subject to the restriction that $pk_i^s$ or $pk_i^r$ should not have appeared before in messages between $A$ and $F_{SC}$.)

(3) Store ($P_i$, $pk_i^s$, $pk_i^r$) in the public key buffer and send (\textbf{SC.Key}, $pk_i^s$, $pk_i^r$) to $\tilde{P}_i$.

When (\textbf{SC.Encrypt}, $pk^r$, $m$) is received from $\tilde{P}_i$:

(1) If an entry ($P_i$, $pk_i^s$, $\cdot$) is not stored in the public key buffer, send \textbf{SC.Error} to $\tilde{P}_i$ and stop.

(2) If an entry ($P_j$, $\cdot$, $pk^r$) is stored in the public key buffer:
   (a) Send (\textbf{SC.Encrypt}, $pk_i^s$, $pk_i^r$, $|m|$) to $A$ and wait for (\textbf{SC.Ciphertext}, $pk_i^s$, $pk_i^r$, $c$) from $A$. (We note that $A$ is free to choose $c$, subject to the restriction that $c$ should not have appeared before in an \textbf{SC.Ciphertext} message from $A$ to $F_{SC}$, nor in an \textbf{SC.Decrypt} message to $A$ from $F_{SC}$.)
   (b) Store ($pk_i^s$, $pk_i^r$, $c$, $m$) in the ciphertext buffer, send (\textbf{SC.Ciphertext}, $c$) to $\tilde{P}_i$ and stop.

(3) Send (\textbf{SC.Encrypt}, $pk_i^s$, $pk_i^r$, $m$) to $A$ and wait for (\textbf{SC.Ciphertext}, $pk_i^s$, $pk_i^r$, $c$) from $A$. (We note that the above restrictions on $c$ also apply here.)

(4) Send (\textbf{SC.Ciphertext}, $c$) to $\tilde{P}_i$.

When (\textbf{SC.Decrypt}, $pk^s$, $c$) is received from $\tilde{P}_j$:

(1) If an entry ($P_j$, $\cdot$, $pk_j^r$) is not stored in the public key buffer, send \textbf{SC.Error} to $\tilde{P}_j$ and stop.

(2) Send (\textbf{SC.Decrypt}, $pk^s$, $pk_j^r$, $c$) to $A$ and wait for (\textbf{SC.Plaintext}, $pk^s$, $pk_j^r$, $m'$) from $A$.

(3) If there is an entry ($pk^s$, $pk_j^r$, $c$, $m$) in the ciphertext buffer, send (\textbf{SC.Plaintext}, $m$) to $\tilde{P}_j$ and stop.

(4) If there is an entry ($P_i$, $pk^s$, $\cdot$) in the public key buffer, send (\textbf{SC.Plaintext}, $\perp$) to $\tilde{P}_j$ and stop.

(5) Send (\textbf{SC.Plaintext}, $m'$) to $\tilde{P}_j$.

\begin{center}
\begin{figure}[!h]
\centering
\includegraphics[width=\textwidth]{image}
\caption{The signcryption functionality $F_{SC}$.}
\end{figure}
\end{center}

\textbf{Composite Machines.} We can now compose these building blocks into two machines,

\[ M'_1 = M_{SC} + M_{PKI} + \sum_i M_i(i) \] \[ M'_2 = M_{SM} + M_{S}. \]
The protocol

The machines

1

Signcryption per-player machine for player

4. Equivalence Proof

Theorem 10. The protocol \( \pi_{SM} \) securely realizes \( F_{SM} \) in the \((F_{PKI}, F_{SC})\)-hybrid model.

Proof. The machines \( M_1' \) and \( M_2' \) with input alphabets \( I_1 \) and \( I_2 \) described in the previous sections recognize the languages \( L_1 \) and \( L_2 \), respectively. Note that, when assuming that the adversaries are dummy machines, the set of possible messages going to or from the environment \( Z \) is the same for both machines, so we call this set \( I \). We are going to prove that the two machines are equivalent with respect to \( I \), allowing us to apply Theorem 9.

In order to simplify further analysis, we replace \( M_1' \) and \( M_2' \) by the machines \( M_1 \) and \( M_2 \) given by

\[
M_1 = B_{SC} + B_{PKI} + \sum_i (M_{SC}(i) + M_{\pi}(i) + M_{PKI}(i)),
\]

and

\[
M_2 = B_{SM} + B_{S} + \sum_i (M_{SM}(i) + M_{S}(i)).
\]
When $P_i$ receives $\text{SM\.Register}$ from the environment:

1. If $P_i$ has already processed $\text{SM\.Register}$, send $\text{SM\.Error}$ to the environment and stop.
2. Send $\text{SC\.KeyGen}$ to $F_{SC}$ and wait for $(\text{SC\.Key}, pk_i^s, pk_i^r)$ from $F_{SC}$.
3. Send $(\text{PKI\.Register}, (pk_i^s, pk_i^r))$ to $F_{PKI}$ and wait for $\text{PKI\.Registered}$ from $F_{PKI}$.
4. Send $\text{SM\.Register\.Ok}$ to the environment.

When $P_i$ receives $(\text{SM\.Encrypt}, m)$ from the environment:

1. If $P_i$ has not processed $\text{SM\.Register}$, send $\text{SM\.Error}$ to the environment and stop.
2. Send $(\text{PKI\.Retrieve}, P_j)$ to $F_{PKI}$ and wait.
3. If $F_{PKI}$ replies with $(\text{PKI\.Retrieve\.Ok}, \bot)$, send $\text{SM\.Not\.Registered}$ to the environment and stop.
4. Otherwise $F_{PKI}$ replies with $(\text{PKI\.Retrieve\.Ok}, (\cdot, pk_j^r))$.
5. Send $(\text{SC\.Encrypt}, pk_j^r, m)$ to $F_{SC}$ and wait for $(\text{SC\.Ciphertext}, c)$ from $F_{SC}$.
6. Send $(\text{SM\.Ciphertext}, c)$ to the environment.

When $P_j$ receives $(\text{SM\.Decrypt}, c)$ from the environment:

1. If $P_j$ has not processed $\text{SM\.Register}$, send $\text{SM\.Error}$ to the environment and stop.
2. Send $(\text{PKI\.Retrieve}, P_i)$ to $F_{PKI}$.
3. If $F_{PKI}$ replies with $(\text{PKI\.Retrieve\.Ok}, \bot)$, send $\text{SM\.Not\.Registered}$ to the environment and stop.
4. Otherwise $F_{PKI}$ replies with $(\text{PKI\.Retrieve\.Ok}, (pk_i^s, \cdot))$.
5. Send $(\text{SC\.Decrypt}, pk_i^s, c)$ to $F_{SC}$ and wait for $(\text{SC\.Plaintext}, m)$ from $F_{SC}$.
6. Send $(\text{SM\.Plaintext}, m)$ to the environment.

Figure 6. The secure messaging protocol $\pi_{SM}$.

By Lemma 5, $M'_1$ and $M_1$ are isomorphic. In particular, it is easy to define an isomorphism $(\sigma, \tau) : M'_1 \rightarrow M_1$, where $\tau$ is the identity map. Lemmas 1 and 3 then imply that $M'_1$ and $M_1$ are equivalent with respect to $I$. The same argument applies to $M'_2$ and $M_2$.

Let $M_1 = (S_1, I_1, f_1, W_1, s_{1,0})$ and $M_2 = (S_2, I_2, f_2, W_2, s_{2,0})$. The overall strategy of the proof is to let $M_1$ and $M_2$ undergo a series of modifications, making them more and more similar, until we can easily define an isomorphism between them. In the following, a modified version of a machine $M$ will be referred to as $M^{(i)}$ for some integer $i$, and if $M = (S, I, f, W, s_1)$, then $M^{(i)} = (S^{(i)}, I^{(i)}, f^{(i)}, W^{(i)}, s_1^{(i)})$.

By inspection of the machines constituting $M_1$, we claim that the reachable states and the significant messages of $M_{SC}(i) + M_s(i) + M_{PKI}(i)$ when running
<table>
<thead>
<tr>
<th>$s$</th>
<th>$w$</th>
<th>$s'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$(Z, P_i, \text{SM.Register})$</td>
<td>$(s_1, -)$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$(P_i, \mathcal{F}_{SC}, \text{SC.KeyGen})$</td>
<td>$(s_2, -)$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$(\mathcal{F}_{SC}, P_i, \text{SC.Key}, pk^s, pk^r)$</td>
<td>$(s_3, pk^s, pk^r)$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$(P_i, \mathcal{F}_{PKI}, \text{PKI.Register},(pk^s, pk^r))$</td>
<td>$(s_4, -)$</td>
</tr>
<tr>
<td>$s_4$</td>
<td>$(\mathcal{F}_{PKI}, P_i, \text{PKI.Register.Ok})$</td>
<td>$(s_5, -)$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>$(P_i, Z, \text{SM.Register.Ok})$</td>
<td>$(s_6, -)$</td>
</tr>
<tr>
<td>$s_6$</td>
<td>$(Z, P_i, \text{SM.Encrypt}, P_j, m)$</td>
<td>$(s_7, P_j, m)$</td>
</tr>
<tr>
<td>$s_7$</td>
<td>$(P_i, \mathcal{F}_{PKI}, \text{PKI.Retrieve}, P_j)$</td>
<td>$(s_8, m)$</td>
</tr>
<tr>
<td>$s_8$</td>
<td>$(\mathcal{F}_{PKI}, P_i, \text{PKI.Retrieve.Fail})$</td>
<td>$(s_9, -)$</td>
</tr>
<tr>
<td>$s_9$</td>
<td>$(P_i, \mathcal{F}_{SC}, \text{SC.Encrypt}, pk^r_j, m)$</td>
<td>$(s_{10}, -)$</td>
</tr>
<tr>
<td>$s_{10}$</td>
<td>$(\mathcal{F}_{SC}, P_i, \text{SC.Ciphertext}, c)$</td>
<td>$(s_{11}, c)$</td>
</tr>
<tr>
<td>$s_{11}$</td>
<td>$(P_i, Z, \text{SM.Ciphertext}, c)$</td>
<td>$(s_6, -)$</td>
</tr>
<tr>
<td>$s_6$</td>
<td>$(Z, P_i, \text{SM.Decrypt}, P_j, c)$</td>
<td>$(s_{12}, P_j, c)$</td>
</tr>
<tr>
<td>$s_{12}$</td>
<td>$(P_i, \mathcal{F}_{PKI}, \text{PKI.Retrieve}, P_j)$</td>
<td>$(s_{13}, c)$</td>
</tr>
<tr>
<td>$s_{13}$</td>
<td>$(\mathcal{F}_{PKI}, P_i, \text{PKI.Retrieve.Fail})$</td>
<td>$(s_{14}, -)$</td>
</tr>
<tr>
<td>$s_{14}$</td>
<td>$(\mathcal{F}_{PKI}, P_i, \text{PKI.Retrieve.Ok}, (pk^s_j, pk^r_j))$</td>
<td>$(s_{15}, pk^r_j, c)$</td>
</tr>
<tr>
<td>$s_{15}$</td>
<td>$(P_i, Z, \text{SM.Not.Registered})$</td>
<td>$(s_6, -)$</td>
</tr>
<tr>
<td>$s_{16}$</td>
<td>$(P_i, \mathcal{F}_{SC}, \text{SC.Decrypt}, pk^s_j, c)$</td>
<td>$(s_{17}, -)$</td>
</tr>
<tr>
<td>$s_{17}$</td>
<td>$(\mathcal{F}_{SC}, P_i, \text{SC.Plaintext}, m)$</td>
<td>$(s_{18}, m)$</td>
</tr>
<tr>
<td>$s_{18}$</td>
<td>$(P_i, Z, \text{SM.Plaintext}, m)$</td>
<td>$(s_6, -)$</td>
</tr>
<tr>
<td>$s_6$</td>
<td>$(Z, P_i, \text{SM.Encrypt}, P_j, m)$</td>
<td>$(s_{19}, -)$</td>
</tr>
<tr>
<td>$s_{19}$</td>
<td>$(Z, P_i, \text{SM.Decrypt}, P_j, c)$</td>
<td>$(s_{19}, -)$</td>
</tr>
<tr>
<td>$s_6$</td>
<td>$(Z, P_i, \text{SM.Register})$</td>
<td>$(s_{20}, -)$</td>
</tr>
<tr>
<td>$s_{19}$</td>
<td>$(P_i, Z, \text{SM.Error})$</td>
<td>$(s_6, -)$</td>
</tr>
</tbody>
</table>

**Figure 7.** Protocol machine for player $i$. 

<table>
<thead>
<tr>
<th>$s$</th>
<th>$w$</th>
<th>$s'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$(Z, P_i, x)$</td>
<td>$(s_1, x)$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$(P_i, \mathcal{F}_{SM}, x)$</td>
<td>$(s_0, -)$</td>
</tr>
<tr>
<td>$s_0$</td>
<td>$(\mathcal{F}_{SM}, P_i, x)$</td>
<td>$(s_2, x)$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$(P_i, Z, x)$</td>
<td>$(s_0, -)$</td>
</tr>
</tbody>
</table>

**Figure 8.** Dummy machine for player $i$. 


When \textit{SM.Register} is received from $\tilde{P}_i$:

1. If $\mathcal{F}_{SM}$ has already processed an \textit{SM.Register} message from $\tilde{P}_i$, send \textit{SM.Error} to $\tilde{P}_i$ and stop.
2. Send $(\textit{SM.Register}, P_i)$ to the adversary $\mathcal{A}$ and wait for $(\textit{SM.Register.OK}, P_i)$ from $\mathcal{A}$.
3. Store $P_i$ in the registration buffer and send \textit{SM.Register.Ok} to $\tilde{P}_i$.

When $(\textit{SM.Encrypt}, P_j, m)$ is received from $\tilde{P}_i$:

1. If $P_i$ is not stored in the registration buffer, send \textit{SM.Error} to $\tilde{P}_i$ and stop.
2. Send $(\textit{SM.Encrypt}, P_i, P_j, |m|)$ to the adversary $\mathcal{A}$ and wait for $(\textit{SM.Ciphertext}, P_i, P_j, c)$ from $\mathcal{A}$. (We note that $\mathcal{A}$ is free to choose $c$, subject to the restriction that $c$ should not have appeared before in an \textit{SM.Ciphertext} message from $\mathcal{A}$ to $\mathcal{F}_{SM}$, nor in an \textit{SM.Decrypt} message to $\mathcal{A}$ from $\mathcal{F}_{SM}$ that resulted in an \textit{SM.Plaintext} being sent from $\mathcal{F}_{SM}$.)
3. If $P_j$ is not stored in the registration buffer, send \textit{SM.Not.Registered} to $\tilde{P}_i$ and stop.
4. Store $(P_i, P_j, c, m)$ in the ciphertext buffer and send $(\textit{SM.Ciphertext}, c)$ to $\tilde{P}_i$.

When $(\textit{SM.Decrypt}, P_i, c)$ is received from $\tilde{P}_j$:

1. If $P_j$ is not stored in the registration buffer, send \textit{SM.Error} to $\tilde{P}_j$ and stop.
2. Send $(\textit{SM.Decrypt}, P_j, P_i, c)$ to the adversary $\mathcal{A}$ and wait for $(\textit{SM.Plaintext}, P_j, P_i, m')$ from $\mathcal{A}$.
3. If $P_i$ is not stored in the registration buffer, send \textit{SM.Not.Registered} to $\tilde{P}_j$ and stop.
4. If an entry $(P_i, P_j, c, m)$ is stored in the ciphertext buffer, send $(\textit{SM.Plaintext}, m)$ to $\tilde{P}_j$, otherwise send $(\textit{SM.Plaintext}, \bot)$ to $\tilde{P}_j$.

**Figure 9.** The secure messaging functionality $\mathcal{F}_{SM}$.

In $M_1$ is given by Figures 13, 14, 15 and 16. These figures also describe the reachable states and the significant messages of $M_{SM}(i) + M_S(i)$ when running in $M_2$. Concerning the interpretation of the figures, note that each transition takes the machines from the states in the above row to the states in the same row. If no state is given for a particular machine, the state of this machine is unchanged by the transition in question. For the present, the parameters shown in red should be ignored.

For the most part, the reachable states and the significant messages of $M_{SC}(i) + M_s(i) + M_{PKI}(i)$ and $M_{SM}(i) + M_S(i)$ given in the figures are evident from the descriptions of the individual machines. However, we briefly comment on the
For ease of presentation, Figures 14 and 15 only include the case where \( P_j \) is no pair containing \( P_j \) to reach this state, either \( M_i \) by \( B(j) \) (pend on the state of some buffer machine. We start by considering the message \((F_{PKI}, B_{PKI}, B_{Lookup}, (P_j, -), \bot/(P_j, (pk_i^s, pk_i^r)))\), which can be recognized by \( M_{PKI}(i) \) in state \((s_8, P_j)\). By tracing backwards, we see that for \( M_{PKI}(i) \) to reach this state, either \( P_j \) has registered, in which case there is a unique pair \((P_j, (pk_i^s, pk_i^r))\) in the state of \( B_{PKI} \), or \( P_j \) has not registered, in which case there is no pair containing \( P_j \) in the state of \( B_{PKI} \). This means that \((F_{PKI}, B_{PKI}, B_{Lookup}, (P_j, -), (P_j, v))\) is recognized only if \( v = (pk_i^s, pk_i^r) \), hence we need only include \((F_{PKI}, B_{PKI}, B_{Lookup}, (P_j, -), \bot/(P_j, (pk_i^s, pk_i^r)))\) in the figures. For ease of presentation, Figures 14 and 15 only include the case where \( P_j \)
When $S$ receives $(\text{SM.Register}, P_i)$ from $F_{SM}$:
(1) Send $(\text{SC.KeyGen}, P_i)$ to $A_{SC}$ and wait for $(\text{SC.Key}, P_i, pk_i^S, pk_i^P)$.
(We note that $A_{SC}$ is free to choose $(pk_i^S, pk_i^P)$, subject to the restriction that $pk_i^S$ or $pk_i^P$ should not have appeared before in messages between $A_{SC}$ and $S$.)
(2) Send $(\text{PKI.Register}, (P_i, pk_i^S, pk_i^P))$ to $A_{PKI}$ and wait for $(\text{PKI.Register.Ok}, P_i)$ from $A_{PKI}$.
(3) Store $(P_i, pk_i^S, pk_i^P)$ in the public key buffer.
(4) Send $(\text{SM.Register.Ok}, P_i)$ to $F_{SM}$.

When $S$ receives $(\text{SM.Encrypt}, P_i, P_j, |m|)$ from $F_{SM}$:
(1) Send $(\text{PKI.Retrieve}, P_i, P_j)$ to $A_{PKI}$ and wait for $(\text{PKI.Retrieve.Ok}, P_i, P_j)$ from $A_{PKI}$.
(2) If there is no entry $(P_j, \cdots)$ in public key buffer, send $(\text{SM.Ciphertext}, P_i, P_j, \bot)$ to $F_{SM}$ and stop.
(3) Find entries $(P_i, pk_i^S, \cdots)$ and $(P_j, \cdot, pk_j^P)$ in public key buffer, send $(\text{SC.Encrypt}, pk_i^S, pk_j^P, |m|)$ to $A_{SC}$, and wait for $(\text{SC.Ciphertext}, pk_i^S, pk_j^P, c)$ from $A_{SC}$.
(4) Send $(\text{SM.Ciphertext}, P_i, P_j, c)$ to $F_{SM}$.

When $S$ receives $(\text{SM.Decript}, P_j, P_i, c)$ from $F_{SM}$:
(1) Send $(\text{PKI.Retrieve}, P_j, P_i)$ to $A_{PKI}$ and wait for $(\text{PKI.Retrieve.Ok}, P_j, P_i)$ from $A_{PKI}$.
(2) If there is no entry $(P_i, \cdots)$ in public key buffer, send $(\text{SM.Plaintext}, P_j, P_i, \bot)$ to $F_{SM}$ and stop.
(3) Find entries $(P_i, pk_i^S, \cdots)$ and $(P_j, \cdot, pk_j^P)$ in public key buffer, send $(\text{SC.Decrypt}, pk_i^S, pk_j^P, c)$ to $A_{SC}$, and wait for $(\text{SC.Plaintext}, pk_i^S, pk_j^P, m')$ from $A_{SC}$.
(4) Send $(\text{SM.Plaintext}, P_j, P_i, m')$ to $F_{SM}$.

Figure 11. The ideal adversary $S$.

has registered, while the case where $P_j$ has not registered is handled in Figure 16. The same reasoning applies to the message $(S, B_S, \text{B.Lookup}, (P_j, \bot), \bot/(P_j, (pk_j^S, pk_j^P)))$, which is treated in the same way.

We proceed by considering the message $(F_{SC}, B_{SC}, \text{B.Lookup}, (-, - , pk_j^P), (P_j, pk_j^S, pk_j^P))$, which can be recognized by $M_{SC}(i)$ in state $(s_6, pk_i^S, pk_i^P, pk_j^S, m)$. By tracing backwards, we see that for $M_{SC}(i)$ to reach this state, there must be a pair $(P_j, (pk_j^S, pk_j^P))$ contained in the state of $B_{PKI}$, which is the case only if the state of $B_{SC}$ contains the triple $(P_j, pk_j^S, pk_j^P)$. Since public keys are assumed to be unique, this is the only triple containing $pk_j^P$ in the state of $B_{SC}$. This means that $(F_{SC}, B_{SC}, \text{B.Lookup}, (-, - , pk_j^P), \bot)$ is not recognized, and that $(F_{SC}, B_{SC}, \text{B.Lookup}, (-, - , pk_j^S), (P_k, pk_k^S, pk_k^P))$ is recognized if and only if $P_k = P_j$ and $pk_k^S = pk_j^S$, hence we need only include $(F_{SC}, B_{SC}, \text{B.Lookup}, (-, - , pk_j^P), \bot)$.
s | w | s'  
---|---|---
\(s_0\) | \((\mathcal{F}_{SM}, S, \text{SM.Register}, P_i)\) | \((s_1, -)\)  
\(s_1\) | \((S, A_{SC}, \text{SC.KeyGen}, P_i)\) | \((s_2, -)\)  
\(s_2\) | \((A_{SC}, S, \text{SC.Key}, P_i, pk^*_i, pk^r_i)\) | \((s_3, pk^*_i, pk^r_i)\)  
\(s_3\) | \((S, A_{PKI}, \text{PKI.Register}, P_i, pk^*_i, pk^r_i)\) | \((s_4, pk^*_i, pk^r_i)\)  
\(s_4\) | \((A_{PKI}, S, \text{PKI.Register.Ok}, P_i)\) | \((s_5, pk^*_i, pk^r_i)\)  
\(s_5\) | \((S, B_{S}, \text{B.Add}, (P_i, (pk^*_i, pk^r_i)))\) | \((s_6, pk^*_i, pk^r_i)\)  
\(s_6\) | \((S, \mathcal{F}_{SM}, \text{SM.Register.Ok}, P_i)\) | \((s_7, pk^*_i, pk^r_i)\) 
\(s_7\) | \((\mathcal{F}_{SM}, S, \text{SM.Encrypt}, P_i, P_j, |m|)\) | \((s_8, pk^*_i, pk^r_i, P_j, |m|)\)  
\(s_8\) | \((S, A_{PKI}, \text{PKI.Retrieve}, P_i, P_j)\) | \((s_9, pk^*_i, pk^r_i, P_j, |m|)\)  
\(s_9\) | \((A_{PKI}, S, \text{PKI.Retrieve.Ok}, P_i, P_j)\) | \((s_10, pk^*_i, pk^r_i, P_j, |m|)\)  
\(s_{10}\) | \((S, B_{S}, \text{B.Lookup}, (P_j, -), \bot)\) | \((s_{11}, pk^*_i, pk^r_i, P_j)\)  
\(s_{11}\) | \((S, B_{S}, \text{B.Lookup}, (P_j, -), \bot, (pk^*_i, pk^r_i, |m|), P_j, (pk^*_i, P_j))\) | \((s_{12}, pk^*_i, pk^r_i, P_j, |m|)\)  
\(s_{12}\) | \((S, A_{SC}, \text{SC.Encrypt}, pk^*_i, pk^r_i, |m|)\) | \((s_{13}, pk^*_i, pk^r_i, P_j, pk^r_i)\)  
\(s_{13}\) | \((S, B_{S}, \text{B.Encrypt}, pk^*_i, pk^r_i, c)\) | \((s_{14}, pk^*_i, pk^r_i, P_j, c)\)  
\(s_{14}\) | \((S, \mathcal{F}_{SM}, \text{SM.Encrypt}, P_i, P_j, c)\) | \((s_{15}, pk^*_i, pk^r_i, P_j, c)\)  
\(s_{15}\) | \((S, A_{PKI}, \text{PKI.Encrypt}, P_i, P_j)\) | \((s_{16}, pk^*_i, pk^r_i, P_j, c)\)  
\(s_{16}\) | \((A_{PKI}, S, \text{PKI.Encrypt.Ok}, P_i, P_j)\) | \((s_{17}, pk^*_i, pk^r_i, P_j, c)\)  
\(s_{17}\) | \((S, B_{S}, \text{B.Encrypt}, (P_j, -), \bot)\) | \((s_{18}, pk^*_i, pk^r_i, P_j)\)  
\(s_{18}\) | \((S, \mathcal{F}_{SM}, \text{SM.Decrypt}, P_i, P_j, c)\) | \((s_{19}, pk^*_i, pk^r_i, P_j, pk^*_i, c)\)  
\(s_{19}\) | \((S, A_{PKI}, \text{PKI.Decrypt}, P_i, P_j)\) | \((s_{20}, pk^*_i, pk^r_i, P_j, pk^r_i)\)  
\(s_{20}\) | \((A_{PKI}, S, \text{PKI.Decrypt.Ok}, P_i, P_j)\) | \((s_{21}, pk^*_i, pk^r_i, P_j, m')\)  
\(s_{21}\) | \((S, \mathcal{F}_{SM}, \text{SM.Plaintext}, P_i, P_j, m')\) | \((s_{22}, pk^*_i, pk^r_i, P_j)\) 

**Figure 12.** Ideal adversary per-player machine for player \(i\).

\((-,-, pk^*_j), (P_j, pk^*_i, pk^r_i)\) in Figure 14. Similar arguments apply to the messages \((\mathcal{F}_{SC}, B_{SC}, B_{lookup}, (-,-), \bot, (P_i, pk^*_j, pk^r_j))\) and \((\mathcal{F}_{SM}, B_{SM}, B_{lookup}, P_j, \bot, P_j)\).

Alternatively, we may view the description of \(M_{SC}(i) + M_{PKI}(i) + M_{\pi}(i)\) when running in \(M_i\) as the description of a single machine. Let this machine be denoted by \(C_1(i)\). The set of states of \(C_1(i)\) consists of triples \((s_{M_{SC}(i)}, s_{M_{\pi}(i)}, s_{M_{PKI}(i)})\), where \(s_{M}\) is the state of machine \(M\), and is equal to the set of reachable states of \(M_{SC}(i) + M_{\pi}(i) + M_{PKI}(i)\) in \(M_i\). Moreover, the input alphabet of \(C_1(i)\) is exactly the set of significant messages of \(M_{SC}(i) + M_{\pi}(i) + M_{PKI}(i)\) in \(M_i\).

We now modify \(M_i\) into a new machine \(M_i^{(1)}\) by replacing \(M_{SC}(i) + M_{\pi}(i) + M_{PKI}(i)\) in the above expression by \(C_1(i)\), that is,

\[M_i^{(1)} = B_{SC} + B_{PKI} + \sum_i C_1(i).\]
Lemma 11. The machines $M_1$ and $M_1^{(1)}$ are equivalent with respect to $I$.

Proof. If we let $S_M$ denote the set of states of machine $M$, the sets $S_1$ and $S_1^{(1)}$ of states of $M_1$ and $M_1^{(1)}$, respectively, are given by

$$S_1 = S_{B_{sc}} \times S_{B_{PKI}} \times \prod_{i=1}^{n} (S_{M_{sc}(i)} \times S_{M_{\pi}(i)} \times S_{M_{PKI}(i)}),$$

$$S_1^{(1)} = S_{B_{sc}} \times S_{B_{PKI}} \times \prod_{i=1}^{n} S_{C_1(i)}.$$

Since $S_{C_1(i)}$ contains all reachable states of $M_{sc}(i) + M_{\pi}(i) + M_{PKI}(i)$ in $M_1$, we observe that $S_1^{(1)}$ is a subset of $S_1$ containing all reachable states in $S_1$. Moreover, since the input alphabet of $C_1(i)$ contains all significant messages of $M_{sc}(i) + M_{\pi}(i) + M_{PKI}(i)$ in $M_1$, we observe that $I_1^{(1)}$ is a subset of $I_1$ containing all significant messages in $I_1$. This means that $M_1^{(1)}$ includes $\text{minimal}(M_1)$. Lemma 7 then implies that $M_1$ and $M_1^{(1)}$ are equivalent with respect to $I$. \[\square\]

By essentially repeating the above procedure, $M_2$ is modified into a new machine $M_2^{(1)}$ by replacing $M_{SM}(i) + M_S(i)$ by a machine $C_2(i)$, whose set of states and input alphabet contain the reachable states and the significant messages of $M_{SM}(i) + M_S(i)$ when running in $M_2$. That is,

$$M_2^{(2)} = B_{SM} + B_S + \sum_i C_2(i).$$

The proof of the following lemma corresponds to the previous proof.

Lemma 12. The machines $M_2$ and $M_2^{(1)}$ are equivalent with respect to $I$.

The next step in the modification procedure concerns the red colored parameters in Figures 14 and 15. In detail, $M_1^{(1)}$ is modified into a new machine $M_1^{(2)}$ by adding the identity $P_k$ from the pair $(P_k, (pk^s_k, pk^r_k))$ in the state of $B_{PKI}$ to the following states: $s_9$ of $M_{PKI}(i)$, $s_{10}$ of $M_{\pi}(i)$ and $s_6$, $s_9$, $s_{10}$ and $s_{11}$ of $M_{sc}(i)$. Furthermore, we add the identity $P_l$ from the pair $(P_l, (pk^s_l, pk^r_l))$ in the state of $B_{PKI}$ to the following states: $s_{16}$ of $M_{\pi}(i)$ and $s_{13}$, $s_{14}$, $s_{15}$ and $s_{17}$ of $M_{sc}(i)$. If the identity $P_k$ or $P_l$ is not well-defined, then the state is left unchanged.

Lemma 13. The machines $M_1^{(1)}$ and $M_1^{(2)}$ are equivalent with respect to $I$.

Furthermore, when considering only reachable states of $M_1^{(1)}$, $P_k$ and $P_l$ in the above description are well-defined, and we have $P_k = P_l = P_j$.

Proof. Let $S$ and $S'$ be the sets of reachable states of $S_1^{(2)}$ and $S_1^{(1)}$, respectively. Note that, since the above modification does not affect the transition function, $S$ and $S'$ are equal, except from the added identity contained in certain states of $S$. Assume that $s$ is a reachable state in $S_1$, such that some machine is in one of the above listed states. By tracing backwards, we see that for $s$ to be reached,
there must be a pair \((P_j, (pk_j^s, pk_j^r))\) in the state of \(B_{PKI}\). Since public keys are assumed to be unique, we must have \(P_k = P_i = P_j\).

Let \(M\) and \(M'\) be the machines \(M^{(2)}\) and \(M^{(1)}\) restricted to \(S\) and \(S'\), respectively, and let \(\sigma : S \rightarrow S'\) be defined as follows: Let \(s\) be a state in \(S\). If \(s\) corresponds to some machine being in one of the above listed states, \(\sigma\) removes the added identity \(P_j\) from this state, and applies the identity map to the states of all other machines. Since the states from which \(P_j\) is removed also contains the key \(pk_j^s\) or \(pk_j^r\), \(P_j\) can be reconstructed from the pair \((P_j, (pk_j^s, pk_j^r))\) in the state of \(B_{PKI}\), hence no information is lost when applying \(\sigma\). It can be verified that \((\sigma, \tau) : M \rightarrow M'\), where \(\tau : I^{(2)} \rightarrow I^{(1)}\) is the identity map, is an isomorphism. It follows from Lemma 1 that the induced language map \(\tau : L(M) \rightarrow L(M')\) is a bijection. \(\tau\) leaves \(I\) fixed, so by Lemma 3, \(M\) and \(M'\) are equivalent with respect to \(I\). Since \(M\) includes \(\text{minimal}(M^{(2)})\) and \(M'\) includes \(\text{minimal}(M^{(1)})\), Corollary 8 implies that \(M^{(1)}\) and \(M^{(2)}\) are equivalent with respect to \(I\). \(\square\)

As for \(M^{(1)}\), this machine is modified into a new machine \(M^{(2)}\) by adding the key \(pk_j^s\) from the pair \((P_j, (pk_j^s, pk_j^r))\) in the state of \(B_S\) to the following states: \(s_{14}\) of \(M_S(i)\) and \(s_{10}\) of \(M_{SM}(i)\). Furthermore, we add the key \(pk_j^s\) from the pair \((P_j, (pk_j^s, pk_j^r))\) in the state of \(B_S\) to the following states: \(s_{21}\) of \(M_S(i)\) and \(s_{14}\) and \(s_{16}\) of \(M_{SM}(i)\). If the key \(pk_j^r\) or \(pk_j^s\) is not well-defined, then the state is left unchanged.

**Lemma 14.** The machines \(M^{(1)}\) and \(M^{(2)}\) are equivalent with respect to \(I\). In particular, when considering only reachable states of \(M^{(1)}\), \(pk_j^s\) and \(pk_j^r\) in the above description are well-defined.

**Proof.** Let \(S\) and \(S'\) be the sets of reachable states of \(S^{(2)}\) and \(S^{(1)}\), respectively. Note that, since the above modification does not affect the transition function, \(S\) and \(S'\) are equal, except from the added key contained in certain states of \(S\). Assume that \(s\) is a reachable state in \(S_1\), such that some machine is in one of the above listed states. By tracing backwards, we see that for \(s\) to be reached, there must be a pair \((P_j, (pk_j^s, pk_j^r))\) in the state of \(B_S\). Since key generation only happens once per player, there can be at most one such pair, so \(pk_j^r\) and \(pk_j^s\) are well-defined.

Let \(M\) and \(M'\) be the machines \(M^{(2)}\) and \(M^{(1)}\) restricted to \(S\) and \(S'\), respectively, and let \(\sigma : S \rightarrow S'\) be defined as follows: Let \(s\) be a state in \(S\). If \(s\) corresponds to some machine being in one of the above listed states, \(\sigma\) removes the added key \(pk_j^s\) or \(pk_j^r\) from this state, and applies the identity map to the states of all other machines. Since the states from which \(pk_j^s\) or \(pk_j^r\) is removed also contains the identity \(P_j\), \(pk_j^s\) or \(pk_j^r\) can be reconstructed from the pair \((P_j, (pk_j^s, pk_j^r))\) in the state of \(B_S\), hence no information is lost when applying \(\sigma\). It can be verified that \((\sigma, \tau) : M \rightarrow M'\), where \(\tau : I^{(2)} \rightarrow I^{(1)}\) is the identity map, is an isomorphism. It follows from Lemma 1 that the induced language map \(\tau : L(M) \rightarrow L(M')\) is a bijection. \(\tau\) leaves \(I\) fixed, so by Lemma 3, \(M\) and
$M'$ are equivalent with respect to $I$. Since $M$ includes $\text{minimal}(M_2^{(2)})$ and $M'$ includes $\text{minimal}(M_2^{(1)})$, Corollary 8 implies that $M_2^{(1)}$ and $M_2^{(2)}$ are equivalent with respect to $I$. □

Considering the implications of Lemma 13 and Lemma 14 concerning reachable states, we modify $M_1^{(2)}$ and $M_2^{(2)}$ into new machines $M_1^{(3)}$ and $M_2^{(3)}$, where the states of $C_1(i)$ and $C_2(i)$ also include the parameters shown in red in Figures 14 and 15. By essentially repeating the strategy used for proving Lemma 11, it can be shown that $M_2^{(2)}$ and $M_2^{(3)}$ are equivalent with respect to $I$, and correspondingly for $M_2^{(2)}$ and $M_2^{(3)}$.

We would like to add a buffer machine $B_{SM}^*$ to $M_1^{(2)}$, as an equivalent to $B_{SM}$ in $M_2^{(2)}$. Before doing this, we must modify the input alphabet $I_1^{(3)}$: We create $M_1^{(4)}$ from $M_1^{(3)}$ by adding messages recognized by $B_{SM}^*$ to $I_1^{(3)}$, that is, $I_1^{(4)} = I_1^{(3)} \cup \{(P, B_{SM}^*, \text{B.Add}, m)\} \cup \{(P, B_{SM}^*, \text{B.Lookup}, p, m)\} \cup \{(P, B_{SM}^*, \text{B.Lookup}, p, \bot)\}$.

**Lemma 15.** The machines $M_1^{(3)}$ and $M_1^{(4)}$ are equivalent with respect to $I$.

**Proof.** Note that $\text{dom}(f_1^{(3)})_2 = \text{dom}(f_1^{(4)})_2$, since the additional messages in $I^{(4)}$ have no transitions related to them. Let $(\sigma, \tau) : M_1^{(4)} \rightarrow M_1^{(3)}$ be defined as follows: $\sigma : S_1^{(4)} \rightarrow S_1^{(3)}$ is the identity map, while $\tau : I_2^{(4)} \rightarrow I_2^{(3)}$ applies the identity map to all messages except the messages meant for $B_{SM}^*$, for which it is undefined. It is clear that $(\sigma, \tau)$ is an isomorphism. Then, by Lemma 1, the induced language map $\tau : L(M_1^{(4)}) \rightarrow L(M_1^{(3)})$ is a bijection. $\tau$ leaves $I$ fixed, and it follows from Lemma 3 that $M_1^{(3)}$ and $M_1^{(4)}$ are equivalent with respect to $I$. □

Next we create $M_1^{(5)}$ from $M_1^{(4)}$ by adding a buffer machine $B_{SM}^*$ to $M_1^{(4)}$, i.e. $M_1^{(5)} = M_1^{(4)} + B_{SM}^*$.

**Lemma 16.** The machines $M_1^{(4)}$ and $M_1^{(5)}$ are equivalent with respect to $I$.

**Proof.** We start by showing that, in the composed machine $M_1^{(5)} = M_1^{(4)} + B_{SM}^*$, the only reachable state of $B_{SM}^*$ is $(s_0, \emptyset)$. Note that the messages potentially capable of changing the state of $B_{SM}^*$ are all on the form $(P, B_{SM}^*, \text{B.Add}, m)$. Throughout this paragraph, let $M_1 = M_1^{(4)}$, $M_2 = B_{SM}^*$, and consider the definition of the transition function $f_{12}$ of the composed machine $M_1 + M_2$. For $B_{SM}^*$ to change its state, there must be an entry $((s_1, i, s_2, k), w, (s_1, j, s_2, l))$ in $f_{12}$, where $w$ is a message on the above form, and $s_2, k \neq s_2, l$. We have $w \in I_1$ and $w \in I_2$, so then we must have $(s_1, i, w, s_1, j) \in f_1$ and $(s_2, k, w, s_2, l) \in f_2$. But $f_1$ has no transitions involving $w$, i.e. $w \notin (\text{dom} f_1)_2$, so there can be no such entries in $f_{12}$. This means that $(s_0, \emptyset)$ is the only reachable state of $B_{SM}^*$. 


Recall that the set $S^{(4)}_1$ of states of $M^{(4)}_1$ is given by

$$S^{(4)}_1 = S_{B_{SC}} \times S_{B_{PKI}} \times \prod_{i=1}^{n} C_1(i),$$

so the set $S^{(5)}_1$ of states of $M^{(5)}_1 = M^{(4)}_1 + B^{*}_{SM}$ is given by

$$S^{(5)}_1 = S_{B^{*}_{SM}} \times S_{B_{SC}} \times S_{B_{PKI}} \times \prod_{i=1}^{n} C_1(i).$$

From the above argument, we deduce that a subset $S'$ containing all reachable states of $M^{(5)}_1$ is obtained as

$$S' = (s_0, \emptyset) \times S_{B_{SC}} \times S_{B_{PKI}} \times \prod_{i=1}^{n} C_1(i).$$

Let $M'$ be the machine $M^{(5)}_1$ restricted to $S'$, and let $(\sigma, \tau) : M' \to M^{(4)}_1$ be defined as follows: $\sigma : S' \to S^{(4)}_1$ applies the identity map to each state except that of $B^{*}_{SM}$, and $\tau : I^{(5)}_1 \to I^{(4)}_1$ is the identity map. Note that, since $f^{(5)}_1$ has no transitions involving $B^{*}_{SM}$, it has essentially the same transitions as $f^{(4)}_1$. It can be verified that $(\sigma, \tau)$ is an isomorphism. By Lemma 1, the induced language map $\tau : L(M') \to L(M^{(4)}_1)$ is a bijection. $\tau$ leaves $I$ fixed, and it follows from Lemma 3 that $M^{(4)}_1$ and $M'$ are equivalent with respect to $I$. Since $M'$ includes $\text{minimal}(M^{(5)}_1)$, Corollary 8 implies that $M^{(4)}_1$ and $M^{(5)}_1$ are equivalent with respect to $I$. \qed

We would also like to add a buffer machine $B^{*}_{SC}$ to $M^{(3)}_2$, as an equivalent to $B_{SC}$ in $M^{(3)}_1$. Before doing this, we must modify the input alphabet $I^{(3)}_2$: We create $M^{(4)}_2$ from $M^{(3)}_2$ by adding messages recognized by $B^{*}_{SC}$ to $I^{(3)}_2$, that is, $I^{(4)}_2 = I^{(3)}_2 \cup \{(P, B^{*}_{SC}, B_{Add}, m)\} \cup \{(P, B^{*}_{SC}, B_{Lookup}, p, m)\} \cup \{(P, B^{*}_{SC}, B_{Lookup}, p, \perp)\}$. The proof of the following lemma is similar to the proof of Lemma 15.

**Lemma 17.** The machines $M^{(3)}_2$ and $M^{(4)}_2$ are equivalent with respect to $I$.

Next we create $M^{(5)}_2$ from $M^{(4)}_2$ by adding a buffer machine $B^{*}_{SC}$ to $M^{(4)}_2$, i.e. $M^{(5)}_2 = M^{(4)}_2 + B^{*}_{SC}$. The proof of the following lemma corresponds to the proof of Lemma 16.

**Lemma 18.** The machines $M^{(4)}_2$ and $M^{(5)}_2$ are equivalent with respect to $I$.

As we would like the behavior of $C_1(i)$ to resemble that of $C_2(i)$, we now add certain transitions to $C_1(i)$, such that for every transition in $C_2(i)$, there is an equivalent transition in $C_1(i)$. Each time a transition is added, we use Lemma 2 to obtain a bijection between the languages recognized by the modified and the original machine. Since the bijection keeps $I$ fixed, Lemma 3 implies that the
machines are equivalent with respect to I. The final result of these modifications can be seen from Figures 17, 18, 19 and 20.

We start by modifying $M_1^{(5)}$ into a new machine $M_1^{(6)}$ by replacing the line

$$(s_5, -) \quad (s_5, -) \quad (F_{PKI}, P_i, PKI.Register.Ok)$$

in the description of $C_1(i)$ by the following two lines:

$$(s_4^{(1)}, -) \quad (s_5, -) \quad (F_{PKI}, P_i, PKI.Register.Ok)$$
$$(s_5, -) \quad (P_i, B^*_{SM}, B.Add, P_i).$$

We note that $((s_5, pk^s_i, pk^s_j), (s_4^{(1)}, -), (s_5, -))$ is not a wait state of $C_1(i)$.

**Lemma 19.** The machines $M_1^{(5)}$ and $M_1^{(6)}$ are equivalent with respect to I.

**Proof.** Using the construction in Lemma 2, we modify $M_1^{(6)}$ by removing the state $((s_5, pk^s_i, pk^s_j), (s_4^{(1)}, -), (s_5, -))$ and the transition out of it from $C_1(i)$, such that the resulting machine is $M_1^{(5)}$. Let $S_0$ be the set of states of $M_1^{(6)}$ where at least one $C_1(i)$ is in this state. In particular, assume $s$ to be a state in $S_0$ such that, for each $i$ in some index set $I$, $C_1(i)$ is in state $((s_5, pk^s_i, pk^s_j), (s_4^{(1)}, -), (s_5, -))$.

Let the function $h : S_0 \rightarrow S_0 \setminus S_0$ be defined as follows: $h$ maps $s$ to the same state, except that for each $i \in I$, $C_1(i)$ is in state $((s_5, pk^s_i, pk^s_j), (s_5, -), (s_5, -))$. It is clear that $h$ is well-defined.

Furthermore, assume that $s'$ is a state in $S_0$ such that $(s'', w, s') \in I_1^{(6)}$ for some $w$ in $I_1^{(6)}$, some $s''$ in $S_1^{(6)}$. Note that $s'$ corresponds to exactly one $C_1(i)$ being in state $((s_5, pk^s_i, pk^s_j), (s_4^{(1)}, -), (s_5, -))$, since this is not a wait state. Let the partial function $g : S_0 \rightarrow I_1^{(6)}$ be defined as follows: $g$ maps each such $s'$ to the message $(P_i, B^*_{SM}, B.Add, P_i)$, and is undefined for other states in $S_0$. $g$ is clearly well-defined.

We observe that $(s', g(s'), h(s'))$ is the only valid transition out of $s'$, and that $g(s')$ cannot be recognized by a state not in $S_0$. This means that $M_1^{(6)}$ can be modified using Lemma 2. Since messages containing $B.Add$ meant for $B^*_{SM}$ are included in $I_1^{(5)}$, $M_1^{(5)}$ and $M_1^{(6)}$ have the same input alphabet, so we let $I_0 = \emptyset$ to ensure that the resulting machine is $M_1^{(5)}$. We thus obtain a bijection $\tau : L(M_1^{(6)}) \rightarrow L(M_1^{(5)})$. Since $\tau$ leaves $I$ fixed, Lemma 3 implies that $M_1^{(5)}$ and $M_1^{(6)}$ are equivalent with respect to $I$. 

The next step is to modify $M_1^{(6)}$ into a new machine $M_1^{(7)}$ by replacing the line

$$(s_7, P_j, m) \quad (Z, P_i, SM.Encrypt, P_j, m)$$

in the description of $C_1(i)$ by the following two lines:

$$(s_6^{(1)}, P_j, m) \quad (Z, P_i, SM.Encrypt, P_j, m)$$
$$(s_7, P_j, m) \quad (P_i, P_i, SM.Encrypt, P_j, |m|).$$
We note that \(((s_5, pk_i^s, pk_i^r), (s_6^{(1)}, P_j, m), (s_5, -))\) is not a wait state of \(C_1(i)\).

Using the same strategy as in the previous proof, we can show that the machines \(M_1^{(6)}\) and \(M_1^{(7)}\) are equivalent with respect to \(I\).

We proceed by modifying \(M_1^{(7)}\) into a new machine \(M_1^{(8)}\) by replacing the line

\[ (s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) \quad (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathcal{SC}.Ciphertext, pk_i^s, pk_i^r, c) \]

in the description of \(C_1(i)\) by the following two lines:

\[ (s_{10}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c) \quad (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathcal{SC}.Ciphertext, pk_i^s, pk_i^r, c) \]

\[ (s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) \quad (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \mathcal{SM}.Ciphertext, P_i, P_j, c). \]

We note that \(((s_{10}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c), (s_{11}, -), (s_5, -))\) is not a wait state of \(C_1(i)\).

Again, we can show that the machines \(M_1^{(7)}\) and \(M_1^{(8)}\) are equivalent with respect to \(I\), using the strategy in the proof of Lemma 19.

Next we modify \(M_1^{(8)}\) into a new machine \(M_1^{(9)}\) by replacing the line

\[ (s_{15}, pk_i^s, pk_i^r, pk_j^r, c, m', P_j) \quad (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathcal{SC}.Plaintext, pk_j^s, pk_i^s, m') \]

in the description of \(C_1(i)\) by the following two lines:

\[ (s_{14}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m') \quad (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathcal{SC}.Plaintext, pk_j^s, pk_i^s, m') \]

\[ (s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j) \quad (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \mathcal{SM}.Plaintext, P_i, P_j, m'). \]

We note that \(((s_{14}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m'), (s_{17}, -), (s_5, -))\) is not a wait state of \(C_1(i)\).

By essentially repeating the argument in the proof of Lemma 19, we can show that the machines \(M_1^{(8)}\) and \(M_1^{(9)}\) are equivalent with respect to \(I\).

The next step is to modify \(M_1^{(9)}\) into a new machine \(M_1^{(10)}\) by replacing the line

\[ (s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) \quad (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \mathcal{SM}.Ciphertext, P_i, P_j, c) \]

in the description of \(C_1(i)\) by the following two lines:

\[ (s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c) \quad (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \mathcal{SM}.Ciphertext, P_i, P_j, c) \]

\[ (s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) \quad (\mathcal{F}_{SC}, B_{SM}^*, \mathcal{B}.Lookup, P_j, P_j/\bot). \]

Furthermore, we replace the line

\[ (s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j) \quad (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \mathcal{SM}.Plaintext, P_i, P_j, m') \]

by the following two lines:

\[ (s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m') \quad (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \mathcal{SM}.Plaintext, P_i, P_j, m') \]

\[ (s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j) \quad (\mathcal{F}_{SC}, B_{SM}^*, \mathcal{B}.Lookup, P_j, P_j/\bot). \]

We note that \(((s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c), (s_{11}, -), (s_5, -))\) and \(((s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m'), (s_{17}, -), (s_5, -))\) are not wait states of \(C_1(i)\).
Lemma 20. The machines $M_1^{(9)}$ and $M_1^{(10)}$ are equivalent with respect to $I$. Moreover, in $M_1^{(10)}$, the message $(F_{SC}, B_{SM}^{*}, B\text{.Lookup}, P_j, \bot)$ is not significant.

Proof. Using the construction in Lemma 2, we modify $M_1^{(10)}$ by removing the states $((s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c), (s_{11}, -), (s_5, -))$ and $((s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, c, m'), (s_{17}, -), (s_5, -))$ and the transitions out of them from $C_1(i)$, such that the resulting machine is $M_1^{(9)}$. Let $S_0$ be the set of states of $M_1^{(10)}$ where at least one $C_1(i)$ is in such a state. In particular, assume $s$ to be a state in $S_0$ such that, for each $i$ in some index set $I$, $C_1(i)$ is in state $((s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, m_i, c_i), (s_{11}, -), (s_5, -))$, and for each $i'$ in some index set $I'$, $C_1(i')$ is in state $((s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, c_i, m_i), (s_{11}, -), (s_5, -))$. Let the function $h : S_0 \rightarrow S_1^{(10)} \setminus S_0$ be defined as follows: $h$ maps $s$ to the same state, except that for each $i \in I$, $C_1(i)$ is in state $((s_{11}, pk_i^s, pk_i^r, pk_j^r, m_i, c_i, P_j), (s_{11}, -), (s_5, -))$, and for each $i' \in I', C_1(i')$ is in state $((s_{15}, pk_i^s, pk_i^r, pk_j^r, c_i, m_i, P_j'), (s_{17}, -), (s_5, -))$. It is clear that $h$ is well-defined.

Furthermore, assume that $s'$ is a state in $S_0$ such that $(s'', w, s') \in f_1^{(10)}$ for some $w$ in $I_1^{(10)}$, some $s''$ in $S_1^{(10)}$. Note that $s'$ corresponds to exactly one $C_1(i)$ being in state $((s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c), (s_{11}, -), (s_5, -))$ or $((s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, c, m'), (s_{17}, -), (s_5, -))$, since these are not wait states. Let the partial function $g : S_0 \rightarrow I_1^{(10)}$ be defined as follows: $g$ maps each such $s'$ to the message $(F_{SC}, B_{SM}^{*}, B\text{.Lookup}, P_j, P_j)$ if $P_j$ is contained in the state of $B_{SM}^{*}$, and $(F_{SC}, B_{SM}^{*}, B\text{.Lookup}, P_j, \bot)$ otherwise. $g$ is undefined for other states in $S_0$. $g$ is clearly well-defined.

We observe that $(s', g(s'), h(s'))$ is the only valid transition out of $s'$, and that $g(s')$ cannot be recognized by a state not in $S_0$. This means that $M_1^{(10)}$ can be modified using Lemma 2. Since messages containing $B\text{.Lookup}$ meant for $B_{SM}^{*}$ are included in $I^{(9)}$, $M_1^{(9)}$ and $M_1^{(10)}$ have the same input alphabet, so we let $I_0 = \emptyset$ to ensure that the resulting machine is $M_1^{(9)}$. We thus obtain a bijection $\tau : L(M_1^{(10)}) \rightarrow L(M_1^{(9)})$. Since $\tau$ leaves $I$ fixed, Lemma 3 implies that $M_1^{(9)}$ and $M_1^{(10)}$ are equivalent with respect to $I$.

It remains to prove that, in $M_1^{(10)}$, the message $(F_{SC}, B_{SM}^{*}, B\text{.Lookup}, P_j, \bot)$ is not significant. By tracing backwards, we see that for $M_1^{(10)}$ to reach a state $s$ such that $C_1(i)$ is in state $((s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c), (s_{11}, -), (s_5, -))$ or $((s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, c, m'), (s_{17}, -), (s_5, -))$, there must be a tuple $(P_j, (pk_j^r, pk_j^r))$ in the state of $B_{PKI}$, which implies that $P_j$ must have been added to $B_{SM}^{*}$. This means that, when $M_1^{(10)}$ is in state $s$, $(F_{SC}, B_{SM}^{*}, B\text{.Lookup}, P_j, \bot)$ is not recognized by $B_{SM}^{*}$, hence it is not recognized by $M_1^{(10)}$. We conclude that no reachable state $s$ exists such that $f_1^{(10)}$ is defined on input $(s, (F_{SC}, B_{SM}^{*}, B\text{.Lookup}, P_j, \bot))$. \qed
We proceed by modifying $M_1^{(10)}$ into a new machine $M_1^{(11)}$ by replacing the line

$$(s_{11}, pk^s_i, pk^r_j, m, c, P_j) \quad (F_{SC}, B^*_{SM}, B, \text{Lookup}, P_j, P_j/\perp)$$

in the description of $C_1(i)$ by the following two lines:

$$(s_{10}^{(3)}, pk^s_i, pk^r_j, P_j, pk^s_j, m, c) \quad (F_{SC}, B^*_{SM}, B, \text{Lookup}, P_j, P_j/\perp)$$

$$(s_{11}, pk^s_i, pk^r_j, m, c, P_j) \quad (F_{SC}, B^*_{SM}, B, \text{Add}, (P, P_j, c, m)).$$

We note that $((s_{10}^{(3)}, pk^s_i, pk^r_j, P_j, pk^s_j, m, c), (s_{11}, -), (s_5, -))$ is not a wait state of $C_1(i)$. The machines $M_1^{(10)}$ and $M_1^{(11)}$ can be shown to be equivalent with respect to $I$, using a similar argument as in the proof of Lemma 19.

Since the functionality $F_{SC}$ ensures that public keys and ciphertexts are unique, we can restrict the set of states of the buffers $B_{SC}$ and $B^*_{SM}$ by only considering states where no ciphertext or public key occurs twice. Let $M_1^{(12)}$ be the machine $M_1^{(11)}$ subject to this restriction. It is straightforward to show that $M_1^{(11)}$ and $M_1^{(12)}$ are equivalent with respect to $I$.

Next we modify $M_1^{(12)}$ into a new machine $M_1^{(13)}$ by replacing the line

$$(s_{13}, P_j, c) \quad (Z, P_i, \text{SM.Decrypt}, P_j, c)$$

in the description of $C_1(i)$ by the following two lines:

$$(s_6^{(2)}, P_j, c) \quad (Z, P_i, \text{SM.Decrypt}, P_j, c)$$

$$(s_{13}, P_j, c) \quad (P_i, P_i, \text{SM.Decrypt}, P_j, c).$$

We note that $(s_5, pk^s_i, pk^r_j), (s_6^{(2)}, P_j, c), (s_5, -)$ is not a wait state of $C_1(i)$. Again, by repeating the strategy used for proving Lemma 19, we can show that $M_1^{(12)}$ and $M_1^{(13)}$ are equivalent with respect to $I$.

Next we modify $M_1^{(13)}$ into a new machine $M_1^{(14)}$ by replacing the line

$$(s_{17}, pk^s_i, pk^r_j, pk^s_j, c, P_j) \quad (F_{SC}, B_{SC}, B, \text{Lookup}, (-, pk^s_j, -), (P_j, pk^s_j, pk^r_j))$$

in the description of $C_1(i)$ by the following two lines:

$$(s_{15}^{(1)}, pk^s_i, pk^r_j, P_j, pk^s_j, c) \quad (F_{SC}, B_{SC}, B, \text{Lookup}, (-, pk^s_j, -), (P_j, pk^s_j, pk^r_j))$$

$$(s_{17}, pk^s_i, pk^r_j, pk^s_j, c, P_j) \quad (F_{SC}, B^*_{SM}, B, \text{Lookup}, (P_j, P_i, c, -), (P_j, P_i, c, m)/\perp).$$

We note that $((s_{15}^{(1)}, pk^s_i, pk^r_j, P_j, pk^s_j, c), (s_{17}, -), (s_5, -))$ is not a wait state of $C_1(i)$.

**Lemma 21.** The machines $M_1^{(13)}$ and $M_1^{(14)}$ are equivalent with respect to $I$.

**Proof.** Using the construction in Lemma 2, we modify $M_1^{(14)}$ by removing the state $((s_{15}^{(1)}, pk^s_i, pk^r_j, P_j, pk^s_j, c), (s_{17}, -), (s_5, -))$ and the transition out of it from $C_1(i)$, such that the resulting machine is $M_1^{(13)}$. Let $S_0$ be the set of states of $M_1^{(14)}$ where at least one $C_1(i)$ is in this state. In particular, assume $s$ to be a state in $S_0$ such that, for each $i$ in some index set $I$, $C_1(i)$ is in state $((s_{15}^{(1)}, pk^s_i, pk^r_j,$
Let the function $h : S_0 \rightarrow S_1^{(14)} \setminus S_0$ be defined as follows: $h$ maps $s$ to the same state, except that for each $i \in I$, $C_1(i)$ is in state $((s_{17}, pk_i^s, pk_i^r, P_j, pk_{j_1}^s, c_i), (s_{17}, -), (s_{5}, -))$. It is clear that $h$ is well-defined.

Furthermore, assume that $s'$ is a state in $S_0$ such that $(s'', w, s') \in f_1^{(14)}$ for some $w$ in $I_1^{(14)}$, some $s''$ in $S_1^{(14)}$. Note that $s'$ corresponds to exactly one $C_1(i)$ being in state $((s_{15}^{(1)}, pk_i^s, pk_i^r, P_j, pk_{j_1}^s, c_i), (s_{17}, -), (s_{5}, -))$, since this is not a wait state. Let the partial function $g : S_0 \rightarrow I_1^{(14)}$ be defined as follows: $g$ maps each such $s'$ to the message $(F_{SC}, B_{SM}^{*}, B_{Lookup}, (P_j, P_i, c, -), (P_j, P_i, c, m))$ if an entry $(P_j, P_i, c, m)$ is contained in the state of $B_{SM}^{*}$, and $(F_{SC}, B_{SM}^{*}, B_{Lookup}, (P_j, P_i, c, -), \bot)$ otherwise. $g$ is undefined for other states in $S_0$. Since ciphertexts are unique, there can be at most one entry $(P_j, P_i, c, m)$ in the state of $B_{SM}^{*}$, hence $g$ is well-defined.

We observe that $(s', g(s'), h(s'))$ is the only valid transition out of $s'$, and that $g(s')$ cannot be recognized by a state not in $S_0$. This means that $M_1^{(14)}$ can be modified using Lemma 2. Since messages containing $B_{Lookup}$ meant for $B_{SM}^{*}$ are included in $I^{(13)}$, $M_1^{(13)}$ and $M_1^{(14)}$ have the same input alphabet, so we let $I_0 = \emptyset$ to ensure that the resulting machine is $M_1^{(13)}$. We thus obtain a bijection $\tau : L(M_1^{(14)}) \rightarrow L(M_1^{(13)})$. Since $\tau$ leaves $I$ fixed, Lemma 3 implies that $M_1^{(13)}$ and $M_1^{(14)}$ are equivalent with respect to $I$.

Next $M_1^{(14)}$ is modified into a new machine $M_1^{(15)}$ by replacing the two lines

$$(5, pk_i^s, pk_i^r) \quad (8, m) \quad (s_8, P_j) \quad (s_{10}, -) \quad (F_{PKI}, B_{PKI}, B_{Lookup}, (P_j, -), (P_j, \bot))$$

in the description of $C_1(i)$ by the following three lines:

$$(5, pk_i^s, pk_i^r) \quad (8, m) \quad (s_8, P_j) \quad (s_8^{(1)}, P_j) \quad (F_{PKI}, B_{PKI}, B_{Lookup}, (P_j, -), (P_j, \bot)) \quad (F_{PKI}, F_{PKI}, SM, Ciphertext, P_i, P_j, \bot).$$

We note that $((s_5, pk_i^s, pk_i^r), (s_8, m), (s_8^{(1)}, P_j))$ is not a wait state of $C_1(i)$. By using the same strategy as for proving Lemma 19, we can prove that the machines $M_1^{(14)}$ and $M_1^{(15)}$ are equivalent with respect to $I$.

Next $M_1^{(15)}$ is modified into a new machine $M_1^{(16)}$ by replacing the two lines

$$(5, pk_i^s, pk_i^r) \quad (14, c) \quad (s_8, P_j) \quad (s_{10}, -) \quad (F_{PKI}, B_{PKI}, B_{Lookup}, (P_j, -), (P_j, \bot))$$

in the description of $C_1(i)$ by the following three lines:

$$(5, pk_i^s, pk_i^r) \quad (14, c) \quad (s_8, P_j) \quad (s_8^{(1)}, P_j) \quad (F_{PKI}, B_{PKI}, B_{Lookup}, (P_j, -), (P_j, \bot)) \quad (F_{PKI}, F_{PKI}, SM, Plaintext, P_i, P_j, \bot).$$
We note that \(((s_5, pk_i^s, pk_i^r), (s_{14}, c), (s_8^{(1)}, P_j))\) is not a wait state of \(C_1(i)\). By essentially repeating the argument in the proof of Lemma 19, we can show that the machines \(M_1^{(15)}\) and \(M_1^{(16)}\) are equivalent with respect to \(I\).

Next \(M_1^{(16)}\) is modified into a new machine \(M_1^{(17)}\) by replacing the line
\[
(s_{10}, -) \quad (\mathcal{F}_\text{PKI}, \mathcal{F}_\text{PKI}, \text{SM.Ciphertext}, P_i, P_j, \bot)
\]
in the description of \(C_1(i)\) by the following two lines:
\[
(s_8^{(2)}, P_j) \quad (\mathcal{F}_\text{PKI}, \mathcal{F}_\text{PKI}, \text{SM.Ciphertext}, P_i, P_j, \bot)
\]
\[
(s_{10}, -) \quad (\mathcal{F}_\text{SM}, B_{SM}^*, \text{B.Lookup}, P_j, P_j/\bot).
\]
Furthermore, we replace the line
\[
(s_{10}, -) \quad (\mathcal{F}_\text{PKI}, \mathcal{F}_\text{PKI}, \text{SM.Plaintext}, P_i, P_j, \bot)
\]
by the following two lines:
\[
(s_8^{(2)}, P_j) \quad (\mathcal{F}_\text{PKI}, \mathcal{F}_\text{PKI}, \text{SM.Plaintext}, P_i, P_j, \bot)
\]
\[
(s_{10}, -) \quad (\mathcal{F}_\text{SM}, B_{SM}^*, \text{B.Lookup}, P_j, P_j/\bot).
\]
We note that \(((s_5, pk_i^s, pk_i^r), (s_8, m), (s_8^{(2)}, P_j))\) and \(((s_5, pk_i^s, pk_i^r), (s_{14}, c), (s_8^{(2)}, P_j))\) are not wait states of \(C_1(i)\). The proof of the following lemma is similar to the proof of Lemma 20.

**Lemma 22.** The machines \(M_1^{(16)}\) and \(M_1^{(17)}\) are equivalent with respect to \(I\). Moreover, in \(M_1^{(17)}\), the message \((\mathcal{F}_\text{SM}, B_{SM}^*, \text{B.Lookup}, P_j, P_j)\) is not significant.

We proceed by adding certain transitions to \(C_2(i)\), such that for every transition in \(C_1(i)\), there is an equivalent transition in \(C_2(i)\). The final result of these modifications can be seen from Figures 17, 18, 19 and 20.

We start by modifying \(M_2^{(5)}\) into a new machine \(M_2^{(6)}\) by replacing the line
\[
(A_{\text{SC}}, \mathcal{F}_{\text{SC}}, \text{SC.Key}, P_i, pk_i^s, pk_i^r) \quad (s_3, pk_i^s, pk_i^r)
\]
in the description of \(C_2(i)\) by the following two lines:
\[
(A_{\text{SC}}, \mathcal{F}_{\text{SC}}, \text{SC.Key}, P_i, pk_i^s, pk_i^s) \quad (s_2^{(1)}, pk_i^s, pk_i^r)
\]
\[
(S, B_{SC}^*, \text{B.Add}, (P_i, pk_i^s, pk_i^r)) \quad (s_3, pk_i^s, pk_i^r).
\]
We note that \(((s_2, -), (s_2^{(1)}, pk_i^s, pk_i^r))\) is not a wait state of \(C_2(i)\). By copying the strategy used for proving Lemma 19, we can show that the machines \(M_2^{(5)}\) and \(M_2^{(6)}\) are equivalent with respect to \(I\).

Since the ideal adversary \(S\) ensures that public keys are unique, we can restrict the set of states of the buffers \(B_{SC}^*\) and \(B_S\) by only considering states where no public key occurs twice. Let \(M_2^{(7)}\) be the machine \(M_2^{(6)}\) subject to this restriction. It is straightforward to show that \(M_2^{(6)}\) and \(M_2^{(7)}\) are equivalent with respect to \(I\).
The next step is to modify $M_2^{(7)}$ into a new machine $M_2^{(8)}$ by replacing the line

$$(S, B^s_{SC}, \mathbf{B.Add}, (P_i, pk^s_i, pk^r_i)) \quad (s_3, pk^s_i, pk^r_i)$$

in the description of $C_2(i)$ by the following two lines:

$$(S, B^s_{SC}, \mathbf{B.Add}, (P_i, pk^s_i, pk^r_i)) \quad (s_2^{(2)}, pk^s_i, pk^r_i)$$
$$(S, S, \mathbf{SC.Key}, P_i, pk^s_i, pk^r_i) \quad (s_3, pk^s_i, pk^r_i).$$

We note that $((s_2, -), (s_2^{(2)}, pk^s_i, pk^r_i))$ is not a wait state of $C_2(i)$. It can be shown that the machines $M_2^{(7)}$ and $M_2^{(8)}$ are equivalent with respect to $I$, by essentially repeating the argument in the proof of Lemma 19.

We now modify $M_2^{(8)}$ into a new machine $M_2^{(9)}$ by replacing the line

$$(S, S, \mathbf{SC.Key}, pk^s_i, pk^r_i) \quad (s_3, pk^s_i, pk^r_i)$$

in the description of $C_2(i)$ by the following two lines:

$$(S, S, \mathbf{SC.Key}, pk^s_i, pk^r_i) \quad (s_2^{(3)}, pk^s_i, pk^r_i)$$
$$(S, S, \mathbf{PKI.Register}, P_i, (pk^s_i, pk^r_i)) \quad (s_3, pk^s_i, pk^r_i).$$

We note that $((s_2, -), (s_2^{(3)}, pk^s_i, pk^r_i))$ is not a wait state of $C_2(i)$. Again, the strategy used for proving Lemma 19 can be applied to show that the machines $M_2^{(8)}$ and $M_2^{(9)}$ are equivalent with respect to $I$.

Now we modify $M_2^{(9)}$ into a new machine $M_2^{(10)}$ by replacing the line

$$(\mathcal{F}_{SM}, S, \mathbf{SM.Encrypt}, P_i, P_j, \vert m\vert) \quad (s_7, P_j, m) \quad (s_8, pk^s_i, pk^r_i, P_j, \vert m\vert)$$

in the description of $C_2(i)$ by the following two lines:

$$(\mathcal{F}_{SM}, S, \mathbf{SM.Encrypt}, P_i, P_j, \vert m\vert) \quad (s_7, P_j, m) \quad (s_8^{(1)}, pk^s_i, pk^r_i, P_j, \vert m\vert)$$
$$(S, S, \mathbf{PKI.Retrieve}, P_j) \quad (s_8, pk^s_i, pk^r_i, P_j, \vert m\vert).$$

Furthermore, we replace the line

$$(\mathcal{F}_{SM}, S, \mathbf{SM.Decrypt}, P_i, P_j, c) \quad (s_{13}, P_j, c) \quad (s_{15}, pk^s_i, pk^r_i, P_j, c)$$

by the following two lines:

$$(\mathcal{F}_{SM}, S, \mathbf{SM.Decrypt}, P_i, P_j, c) \quad (s_{13}, P_j, c) \quad (s_7^{(2)}, pk^s_i, pk^r_i, P_j, c)$$
$$(S, S, \mathbf{PKI.Retrieve}, P_j) \quad (s_{15}, pk^s_i, pk^r_i, P_j, c).$$

We note that $((s_7, P_j, m), (s_7^{(1)}, pk^s_i, pk^r_i, P_j, \vert m\vert))$ and $((s_{13}, P_j, c), (s_7^{(2)}, pk^s_i, pk^r_i, P_j, c))$ are not wait states of $C_2(i)$. The machines $M_2^{(9)}$ and $M_2^{(10)}$ can be shown to be equivalent with respect to $I$, using an argument similar to that used for proving Lemma 20.

Next $M_2^{(10)}$ is modified into a new machine $M_2^{(11)}$ by replacing the line

$$(S, B_S, \mathbf{B.Lookup}, (P_j, -), (P_j, (pk^s_j, pk^r_j))) \quad (s_{12}, pk^s_i, pk^r_i, P_j, pk^r_j, \vert m\vert)$$

by the following two lines:

$$(S, B_S, \mathbf{B.Lookup}, (P_j, -), (P_j, (pk^s_j, pk^r_j))) \quad (s_{12}, pk^s_i, pk^r_i, P_j, pk^r_j, \vert m\vert).$$
in the description of \( C_2(i) \) by the following two lines:

\[
(S, B_S, \text{B.Lookup}, (P_j, -), (P_j, (pk_{j_i}^s, pk_{j_i}^r))) \quad (s_{10}^{(1)}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^s, pk_{j_i}^r, m_i)
\]

\[
(S, S, \text{PKI.Retrieve.Ok}, (pk_{j_i}^s, pk_{j_i}^r)) \quad (s_{12}^{(1)}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^s, m_i).
\]

Furthermore, we replace the line

\[
(S, B_S, \text{B.Lookup}, (P_j, -,-), (P_j, pk_{j_i}^s, pk_{j_i}^r)) \quad (s_{19}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^s, c)
\]

by the following two lines:

\[
(S, B_S, \text{B_LOOKUP}, (P_j, -,-), (P_j, pk_{j_i}^s, pk_{j_i}^r)) \quad (s_{17}^{(1)}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^s, pk_{j_i}^r, c)
\]

\[
(S, S, \text{PKI.Retrieve.Ok}, (pk_{j_i}^s, pk_{j_i}^r)) \quad (s_{19}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^s, c).
\]

We note that \((s_7, P_j, m), (s_{10}^{(1)}, pk_{i}^s, pk_{i}^r, P_j, pk_{j_i}^s, pk_{j_i}^r, |m|)\) and \((s_{13}, P_j, c), (s_{17}^{(1)}, pk_{i}^s, pk_{i}^r, P_j, pk_{j_i}^s, pk_{j_i}^r, c)\) are not wait states of \( C_2(i) \). By arguing similarly as in the proof of Lemma 20, the machines \( M_2^{(10)} \) and \( M_2^{(11)} \) can be shown to be equivalent with respect to \( I \).

We now modify \( M_2^{(11)} \) into a new machine \( M_2^{(12)} \) by replacing the line

\[
(S, S, \text{PKI.Retrieve.Ok}, P_i, (pk_{j_i}^s, pk_{j_i}^r)) \quad (s_{12}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^r, |m|)
\]

in the description of \( C_2(i) \) by the following two lines:

\[
(S, S, \text{PKI.Retrieve.Ok}, P_i, (pk_{j_i}^s, pk_{j_i}^r)) \quad (s_{12}^{(2)}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^r, |m|)
\]

\[
(S, S, \text{SC.Encrypt}, P_i, pk_{j_i}^r, m) \quad (s_{12}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^r, |m|).
\]

We note that \((s_7, P_j, m), (s_{12}^{(2)}, pk_{i}^s, pk_{i}^r, P_j, pk_{j_i}^r, |m|)\) is not a wait state of \( C_2(i) \). Again, by arguing as in the proof of Lemma 19, we can show that the machines \( M_2^{(11)} \) and \( M_2^{(12)} \) are equivalent with respect to \( I \).

Next \( M_2^{(12)} \) is modified into a new machine \( M_2^{(13)} \) by replacing the line

\[
(S, S, \text{SC.Encrypt}, P_i, pk_{j_i}^r, m) \quad (s_{12}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^r, |m|)
\]

in the description of \( C_2(i) \) by the following two lines:

\[
(S, S, \text{SC.Encrypt}, P_i, pk_{j_i}^r, m) \quad (s_{12}^{(3)}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^r, |m|)
\]

\[
(S, B_{SC}^*, \text{B.Lookup}, (-,-, pk_{j_i}^r), (P_k, pk_{k}^s, pk_{j_i}^r)/\bot) \quad (s_{12}, pk_{i}^s, pk_{j_i}^r, P_j, pk_{j_i}^r, |m|).
\]

We note that \((s_7, P_j, m), (s_{12}^{(3)}, pk_{i}^s, pk_{i}^r, P_j, pk_{j_i}^r, |m|)\) is not a wait state of \( C_2(i) \).

**Lemma 23.** The machines \( M_2^{(12)} \) and \( M_2^{(13)} \) are equivalent with respect to \( I \). Moreover, in \( M_2^{(13)} \) the message \((S, B_{SC}^*, \text{B.Lookup}, (-,-, pk_{j_i}^r), \bot)\) is not significant, and \((S, B_{SC}^*, \text{B.Lookup}, (-,-, pk_{j_i}^r), (P_k, pk_{k}^s, pk_{j_i}^r))\) is significant only if \( P_k = P_j \) and \( pk_{k}^s = pk_{j_i}^s \).

**Proof.** Using the construction in Lemma 2, we modify \( M_2^{(13)} \) by removing the state \((s_7, P_j, m), (s_{12}^{(3)}, pk_{i}^s, pk_{i}^r, P_j, pk_{j_i}^r, |m|)\) and the transition out of it from \( C_2(i) \), such that the resulting machine is \( M_2^{(12)} \). Let \( S_0 \) be the set of states of \( M_2^{(13)} \) where at least one \( C_2(i) \) is in this state. In particular, assume \( s \) to be a state in \( S_0 \) such that, for each \( i \) in some index set \( I \), \( C_2(i) \) is in state \((s_7, P_j, m_i)\),
Let the function \( h : S_0 \to S_2^{(13)} \setminus S_0 \) be defined as follows: \( h \) maps \( s \) to the same state, except that for each \( i \in I, C_2(i) \) is in state \((s_7, P_j, m_i), (s_{12}, p_{k_i}^s, pk_r^j, P_j, pk_r^j, m_i)\). It is clear that \( h \) is well-defined.

Furthermore, assume that \( s' \) is a state in \( S_0 \) such that \((s'', w, s') \in f_2^{(13)} \) for some \( w \) in \( I_2^{(13)} \), some \( s'' \) in \( S_2^{(13)} \). Note that \( s' \) corresponds to exactly one \( C_2(i) \) being in state \((s_7, P_j, m), (s_{10}, pk_i^s, pk_r^j, P_j, pk_r^j, m_i)\), since this is not a wait state. Let the partial function \( g : S_0 \to I_2^{(13)} \) be defined as follows: \( g \) maps each such \( s' \) to the message \((S, B_{SC}^*, B_{Lookup}, (-, -, pk_r^j), (P_k, pk_k^s, pk_r^j))\) if there is a triple \((P_k, pk_k^s, pk_r^j)\) in the state of \( B_{SC}^* \), and \((S, B_{SC}^*, B_{Lookup}, (-, -, pk_r^j), \bot)\) otherwise. \( g \) is undefined for other states in \( S_0 \). Since public keys are assumed to be unique, there is at most one triple containing \( pk_r^j \) in the state of \( B_{SC}^* \), hence \( g \) is well-defined.

We observe that \((s', g(s'), h(s'))\) is the only valid transition out of \( s' \), and that \( g(s') \) cannot be recognized by a state not in \( S_0 \). This means that \( M_2^{(13)} \) can be modified using Lemma 2. Since messages containing \( B_{Lookup} \) meant for \( B_{SC}^* \) are included in \( I^{(12)}, M_2^{(12)} \) and \( M_2^{(13)} \) have the same input alphabet, so we let \( I_0 = \emptyset \) to ensure that the resulting machine is \( M_2^{(12)} \). We thus obtain a bijection \( \tau : L(M_2^{(13)}) \to L(M_2^{(12)}) \). Since \( \tau \) leaves \( I \) fixed, Lemma 3 implies that \( M_2^{(12)} \) and \( M_2^{(13)} \) are equivalent with respect to \( I \).

It remains to prove that, in \( M_2^{(13)} \), the message \((S, B_{SC}^*, B_{Lookup}, (-, -, pk_r^j), \bot)\) is not significant, and that \((S, B_{SC}^*, B_{Lookup}, (-, -, pk_r^j), (P_k, pk_k^s, pk_r^j))\) is significant only if \( P_k = P_j \) and \( pk_k^s = pk_r^j \). By tracing backwards, we see that for \( M_2^{(13)} \) to reach a state \( s \) such that \( C_2(i) \) is in state \((s_7, P_j, m), (s_{10}, pk_i^s, pk_r^j, P_j, pk_r^j, m_i)\), there must be a pair \((P_j, (pk_k^s, pk_r^j))\) in the state of \( B_2 \), which is true only if \( B_{SC}^* \) contains the triple \((P_j, pk_k^s, pk_r^j)\). Since keys are assumed to be unique, this is the only triple containing \( pk_r^j \) in the state of \( B_{SC}^* \). This means that, when \( M_2^{(13)} \) is in state \( s \), the message \((S, B_{SC}^*, B_{Lookup}, (-, -, pk_r^j), \bot)\) is not recognized by \( B_{SM}^* \), and \((S, B_{SC}^*, B_{Lookup}, (-, -, pk_r^j), (P_k, pk_k^s, pk_r^j))\) can be recognized only if \( P_k = P_j \) and \( pk_k^s = pk_r^j \).

Next we modify \( M_2^{(13)} \) into a new machine \( M_2^{(14)} \) by replacing the line \((F_{SM}, B_{SM}, B_{Add}, (P_i, P_j, c, m)) \ (s_{11}, c)\) in the description of \( C_2(i) \) by the following two lines:

\[
(F_{SM}, B_{SM}, B_{Add}, (P_i, P_j, c, m)) \quad (s_{10}, P_j, pk_r^j, m, c) \\
(F_{SM}, B_{SC}^*, B_{Add}, (pk_k^s, pk_r^j, c, m)) \quad (s_{11}, c).
\]

We note that \((s_{10}, P_j, pk_r^j, m, c), (s_7, pk_k^s, pk_r^j)\) is not a wait state of \( C_2(i) \). By arguing as in the proof of Lemma 19, we can show that the machines \( M_2^{(13)} \) and \( M_2^{(14)} \) are equivalent with respect to \( I \). Since the functionality \( F_{SM} \) ensures that ciphertexts are unique, we can restrict the set of states of the buffers \( B_{SC}^* \) and
$B_{SM}$ by only considering states where no ciphertext occurs twice. Let $M_2^{(15)}$ be the machine $M_2^{(14)}$ subject to this restriction. It is straightforward to show that $M_2^{(14)}$ and $M_2^{(15)}$ are equivalent with respect to $I$.

We proceed by modifying $M_2^{(15)}$ into a new machine $M_2^{(16)}$ by replacing the line

$$(F_{SM}, B_{SC}^*, B.Add, (pk_i^s, pk_j^r, c, m)) \quad (s_{11}, c)$$

in the description of $C_2(i)$ by the following two lines:

$$(F_{SM}, B_{SC}^*, B.Add, (pk_i^s, pk_j^r, c, m)) \quad (s_{10}^{(2)}, c)$$
$$(F_{SM}, F_{SM}, SC.Ciphertext, P_i, c) \quad (s_{11}, c).$$

We note that $((s_{10}^{(2)}, c), (s_7, pk_i^s, pk_j^r))$ is not a wait state of $C_2(i)$. Again, we can prove that the machines $M_2^{(15)}$ and $M_2^{(16)}$ are equivalent with respect to $I$, by arguing as in the proof of Lemma 19.

We now modify $M_2^{(16)}$ into a new machine $M_2^{(17)}$ by replacing the line

$$(S, S, PKI.Retrieve.Ok, P_i, (pk_j^s, pk_j^r)) \quad (s_{19}, pk_i^s, pk_i^r, P_j, pk_j^s, c)$$

in the description of $C_2(i)$ by the following two lines:

$$(S, S, PKI.Retrieve.Ok, P_i, (pk_j^s, pk_j^r)) \quad (s_{17}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c)$$
$$(S, S, SC.Decrypt, P_i, pk_j^s, c) \quad (s_{19}, pk_i^s, pk_i^r, P_j, pk_j^s, c).$$

We note that $((s_{13}, P_j, c), (s_{17}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c))$ is not a wait state of $C_2(i)$. The machines $M_2^{(16)}$ and $M_2^{(17)}$ can be shown to be equivalent with respect to $I$, by applying the strategy in the proof of Lemma 19.

We continue by modifying $M_2^{(17)}$ into a new machine $M_2^{(18)}$ by replacing the line

$$(F_{SM}, B_{SM}, B.Lookup, P_j, P_j) \quad (s_{16}, P_j, c, pk_j^s)$$

in the description of $C_2(i)$ by the following two lines:

$$(F_{SM}, B_{SM}, B.Lookup, P_j, P_j) \quad (s_{14}^{(1)}, P_j, pk_j^s, c, m')$$
$$(F_{SM}, B_{SC}^*, B.Lookup, (-, pk_j^s, -), (Pk, pk_j^s, pk_k^r) / \perp) \quad (s_{16}, P_j, c, pk_j^s).$$

We note that $((s_{14}^{(1)}, P_j, c, m'), (s_7, pk_i^s, pk_j^r))$ is not a wait state of $C_2(i)$. The below lemma is obtained by essentially repeating the argument in the proof of Lemma 23.

**Lemma 24.** The machines $M_2^{(17)}$ and $M_2^{(18)}$ are equivalent with respect to $I$. Moreover, in $M_2^{(18)}$ the message $(F_{SM}, B_{SC}^*, B.Lookup, (-, pk_j^s, -), \perp)$ is not significant, and $(F_{SM}, B_{SC}^*, B.Lookup, (-, pk_j^s, -), (P_k, pk_j^s, pk_k^r))$ is significant only if $P_k = P_j$ and $pk_k^r = pk_j^r$. 

We proceed by modifying $M^{(18)}_2$ into a new machine $M^{(19)}_2$ by replacing the line
\[(\mathcal{F}_{SM}, B_{SM}, \mathbf{B}. \text{Lookup}, (P_j, P_i, c, -), \perp/((P_j, P_i, c, m)) \quad (s_{17}, \perp)/(s_{18}, m)\]
in the description of $C_2(i)$ by the following two lines:
\[(\mathcal{F}_{SM}, B_{SM}, \mathbf{B}. \text{Lookup}, (P_j, P_i, c, -), \perp/((P_j, P_i, c, m)) \quad (s^{(1)}_i, P_j, pk^s_j, c)\]
\[(\mathcal{F}_{SM}, B^*_SC, \mathbf{B}. \text{Lookup}, (pk^s_j, pk^r_i, c, -), \perp/((pk^s_j, pk^r_i, c, m)) \quad (s_{17}, \perp)/(s_{18}, m).\]

We note that $((s^{(1)}_i, P_j, pk^s_j, c), (s_7, pk^s_i, pk^r_i))$ is not a wait state of $C_2(i)$.

**Lemma 25.** The machines $M^{(18)}_2$ and $M^{(19)}_2$ are equivalent with respect to $I$.

*Proof.* In this proof, we will apply Corollary 6, which allows us to replace the first condition in Lemma 2 by the following: for all reachable $s \in S$, and for all $w \in I$ and $s' \in S_0$, if $(s, w, s') \in f$, then there is unique $w' \in I$, $s'' \in S$ such that $(s', w', s'') \in f$, and $g(s') = w'$, $h(s') = s''$. Using the construction in this lemma, we modify $M^{(19)}_2$ by removing the state $((s^{(1)}_i, P_j, pk^s_j, c), (s_7, pk^s_i, pk^r_i))$ and the transition out of it from $C_2(i)$, such that the resulting machine is $M^{(18)}_2$. Let $S_0$ be the set of states of $M^{(19)}_2$ where at least one $C_2(i)$ is in this state. In particular, assume $s$ to be a state in $S_0$ such that, for each $i$ in some index set $I$, $C_2(i)$ is in state $((s^{(1)}_i, P_j, pk^s_j, c_i), (s_7, pk^s_i, pk^r_i))$. Let the function $h : S_0 \rightarrow S_2^{(19)} \setminus S_0$ be defined as follows: $h$ maps $s$ to the same state, except that for each $i \in I$, $C_2(i)$ is in state $((s_{18}, m_i), (s_7, pk^s_i, pk^r_i))$ if there is a tuple $(P_j, P_i, c_i, m_i)$ in the state of $B_{SM}$, and $((s_{17}, \bot), (s_7, pk^s_i, pk^r_i))$ otherwise. Since ciphertexts are unique, there is at most one tuple containing $c_i$ in the state of $B_{SM}$, so $h$ is well-defined. Furthermore, assume that $s'$ is a state in $S_0$ such that $(s'', w, s') \in f^{(19)}_2$ for some $w$ in $I^{(19)}_2$, some reachable $s''$ in $S^{(19)}_2$. Note that $s'$ corresponds to exactly one $C_2(i)$ being in state $((s^{(1)}_i, P_j, pk^s_j, c_i), (s_7, pk^s_i, pk^r_i))$, since this is not a wait state. Let the partial function $g : S_0 \rightarrow I^{(19)}_2$ be defined as follows: $g$ maps each such $s'$ to the message $\mathcal{F}_{SM}, B^*_SC, \mathbf{B}. \text{Lookup}, (pk^s_j, pk^r_i, c, -), (pk^s_j, pk^r_i, c, m)$ if there is a tuple $(pk^s_j, pk^r_i, c, m)$ in the state of $B^*_SC$, and $(\mathcal{F}_{SM}, B^*_SC, \mathbf{B}. \text{Lookup}, (pk^s_j, pk^r_i, c, -), \bot)$ otherwise. $g$ is undefined for other states in $S_0$. Again, since ciphertexts are unique, we can conclude that $g$ is well-defined.

In order to verify that $(s', g(s'))$, $h(s')$ is the only valid transition out of $s'$, we note the following: When $M^{(19)}_2$ is in state $s'$, there is a tuple $(P_j, P_i, c, m)$ in the state of $B_{SM}$ if and only if there is a tuple $(pk^s_j, pk^r_i, c, m)$ in the state of $B^*_SC$. This follows from the fact that $s'$ is a reachable state, that key generation only happens once per player, and the uniqueness of public keys. Consequently, $g(s') = (\mathcal{F}_{SM}, B^*_SC, \mathbf{B}. \text{Lookup}, (pk^s_j, pk^r_i, c, -), (pk^s_j, pk^r_i, c, m))$ implies that $h(s') = ((s_{18}, m), (s_7, pk^s_i, pk^r_i))$, while $g(s') = (\mathcal{F}_{SM}, B^*_SC, \mathbf{B}. \text{Lookup}, (pk^s_j, pk^r_i, c, -), \bot)$ implies that $h(s') = ((s_{17}, \bot), (s_7, pk^s_i, pk^r_i))$. We infer that $(s', g(s'), h(s'))$ is the only valid transition out of $s'$. Furthermore, we observe that $g(s')$ cannot be recognized by a state not in $S_0$. This means that $M^{(19)}_2$
can be modified using Lemma 2. Since messages containing $\text{B.Lookup}$ meant for $B_{SC}^*$ are included in $I_2^{(18)}$, $M_2^{(18)}$ and $M_2^{(19)}$ have the same input alphabet, so we let $I_0 = \emptyset$ to ensure that the resulting machine is $M_2^{(18)}$. We thus obtain a bijection $\tau : L(M_2^{(19)}) \to L(M_2^{(18)})$. Since $\tau$ leaves $I$ fixed, Lemma 3 implies that $M_2^{(18)}$ and $M_2^{(19)}$ are equivalent with respect to $I$. □

Next we modify $M_2^{(19)}$ into a new machine $M_2^{(20)}$ by replacing the line
\[
(F_{SM}, B_{SC}^*, \text{B.Lookup}, (pk_j^s, pk_i^r, c, -), \perp/(pk_j^s, pk_i^r, c, m)) \quad (s_{17}, \perp)/(s_{18}, m).
\]
in the description of $C_2(i)$ by the following two lines:
\[
(F_{SM}, B_{SC}^*, \text{B.Lookup}, (pk_j^s, pk_i^r, c, -), \perp/(pk_j^s, pk_i^r, c, m)) \quad (s_{16}^{(2)}, \perp)/(s_{16}^{(3)}, m)
(F_{SM}, F_{SM}, \text{SC.Plaintext}, P_i, \perp/m) \quad (s_{17}, \perp)/(s_{18}, m).
\]
We note that $((s_{16}^{(2)}, \perp), (s_7, pk_i^s, pk_i^r))$ and $((s_{16}^{(3)}, m), (s_7, pk_i^s, pk_i^r))$ are not wait states of $C_2(i)$.

**Lemma 26.** The machines $M_2^{(19)}$ and $M_2^{(20)}$ are equivalent with respect to $I$.

**Proof.** Using the construction in Lemma 2, we modify $M_2^{(20)}$ by removing the states $((s_{16}^{(2)}, \perp), (s_7, pk_i^s, pk_i^r))$ and $((s_{16}^{(3)}, m), (s_7, pk_i^s, pk_i^r))$ and the transitions out of them from $C_2(i)$, such that the resulting machine is $M_2^{(19)}$. Let $S_0$ be the set of states of $M_2^{(20)}$ where at least one $C_2(i)$ is in such a state. In particular, assume $s$ to be a state in $S_0$ such that, for each $i$ in some index set $I$, $C_2(i)$ is in state $((s_{16}^{(2)}, \perp), (s_7, pk_i^s, pk_i^r))$, and for each $i'$ in some index set $I'$, $C_2(i')$ is in state $((s_{16}^{(3)}, m), (s_7, pk_i^s, pk_i^r))$. Let the function $h : S_0 \to S_2^{(20)} \setminus S_0$ be defined as follows: $h$ maps $s$ to the same state, except that for each $i \in I$, $C_2(i)$ is in state $((s_{17}, \perp), (s_7, pk_i^s, pk_i^r))$, and for each $i' \in I'$, $C_2(i')$ is in state $((s_{18}, m), (s_7, pk_i^s, pk_i^r))$. It is clear that $h$ is well-defined.

Furthermore, assume that $s'$ is a state in $S_0$ such that $(s', w, s') \in f_2^{(20)}$ for some $w$ in $I_2^{(20)}$, some $s''$ in $S_2^{(20)}$. Note that $s'$ corresponds to exactly one $C_2(i)$ being in state $((s_{16}^{(2)}, \perp), (s_7, pk_i^s, pk_i^r))$ or $((s_{16}^{(3)}, m), (s_7, pk_i^s, pk_i^r))$, since these are not wait states. Let the partial function $g : S_0 \to I_2^{(20)}$ be defined as follows: $g$ maps each such $s'$ to the message $(F_{SM}, F_{SM}, \text{SC.Plaintext}, \perp)$ if $C_2(i)$ is in state $((s_{16}^{(2)}, \perp), (s_7, pk_i^s, pk_i^r))$ and $(F_{SM}, F_{SM}, \text{SC.Plaintext}, m)$ if $C_2(i)$ is in state $((s_{16}^{(3)}, m), (s_7, pk_i^s, pk_i^r))$. $g$ is undefined for other states in $S_0$. $g$ is clearly well-defined.

We observe that $(s', g(s'), h(s'))$ is the only valid transition out of $s'$, and that $g(s')$ cannot be recognized by a state not in $S_0$. This means that $M_2^{(20)}$ can be modified using Lemma 2. Since $I^{(19)}$ contains no message on the form $(F_{SM}, F_{SM}, \text{SC.Plaintext}, \perp/m)$, we let $I_0 = g(S_0)$ to ensure that the resulting machine is $M_2^{(19)}$. We thus obtain a bijection $\tau : L(M_2^{(20)}) \to L(M_2^{(19)})$. Since $\tau$ leaves $I$ fixed, Lemma 3 implies that $M_2^{(19)}$ and $M_2^{(20)}$ are equivalent with respect to $I$. □
Next $M_2^{(20)}$ is modified into a new machine $M_2^{(21)}$ by replacing the line

$$(F_{SM}, B_{SM}, \textbf{B.Lookup}, P_j, \perp) \quad (s_9, \perp)$$

in the description of $C_2(i)$ by the following two lines:

$$(F_{SM}, B_{SM}, \textbf{B.Lookup}, P_j, \perp) \quad (s_8^{(1)}, m, \perp)$$
$$(F_{SM}, F_{SM}, \textbf{PKI.Retrieve.Fail}, P_i) \quad (s_9, \perp).$$

Furthermore, the line

$$(F_{SM}, B_{SM}, \textbf{B.Lookup}, P_j, \perp) \quad (s_{15}, \perp)$$

is replaced by the following two lines:

$$(F_{SM}, B_{SM}, \textbf{B.Lookup}, P_j, \perp) \quad (s_{14}^{(2)}, c, \perp)$$
$$(F_{SM}, F_{SM}, \textbf{PKI.Retrieve.Fail}, P_i) \quad (s_{15}, \perp).$$

We note that $((s_8^{(1)}, m, \perp), (s_7, pk_i^s, pk_i^r))$ and $((s_{14}^{(2)}, c, \perp), (s_7, pk_i^s, pk_i^r))$ are not wait states of $C_2(i)$. The machines $M_2^{(20)}$ and $M_2^{(21)}$ can be shown to be equivalent with respect to $I$, by arguing as in the proof of Lemma 20.

Finally, we modify the machines $M_1^{(17)}$ and $M_2^{(21)}$ into new machines $M_1^{(18)}$ and $M_2^{(22)}$, where the messages proven not significant by the previous lemmas are disregarded. It is straightforward to show that $M_1^{(17)}$ and $M_1^{(18)}$ are equivalent with respect to $I$, and correspondingly for $M_2^{(21)}$ and $M_2^{(22)}$. The machines $C_1(i)$ and $C_2(i)$ are now completely described by Figures 17, 18, 19 and 20.

It remains to prove the following lemma:

**Lemma 27.** The machines $M_1^{(18)}$ and $M_2^{(22)}$ are equivalent with respect to $I$.

**Proof.** The strategy of the proof is to construct an isomorphism $(\sigma, \tau)$ between the two machines, such that $\tau$ leaves $I$ fixed. For ease of notation, we refer to $M_1^{(18)}$ as $M_1$ and to $M_2^{(22)}$ as $M_2$ throughout the proof, and assume that $M_1 = (S_1, f_1, I_1, W_1, s_1)$ and $M_2 = (S_2, f_2, I_2, W_2, s_2)$.

We start by defining the function $\tau$, which is fully described by Figures 17, 18, 19 and 20: $\tau : I_1 \to I_2$ simply maps each message in the left column to the adjacent message in the right column. Simply by inspection of these messages, it can be verified that $\tau$ is a bijection.

We proceed by describing the map $\sigma$. Recall that $S_1$ and $S_2$ are given by

$$S_1 = S_{B_{SC}} \times S_{B_{PKI}} \times S_{B_{SM}} \times \prod_{i=1}^{n} S_{C_1(i)},$$

$$S_2 = S_{B_{SC}} \times S_{B_{S}} \times S_{B_{SM}} \times \prod_{i=1}^{n} S_{C_2(i)}.$$
It will be convenient to describe \( \sigma \) using the maps

\[
\begin{align*}
\sigma_{B_{sc}} & : S_{B_{sc}} \rightarrow S_{B_{sc}}, \\
\sigma_{B_{S}} & : S_{B_{PKI}} \rightarrow S_{B_{S}}, \\
\sigma_{B_{SM}} & : S_{B_{SM}}^{*} \rightarrow S_{B_{SM}}, \\
\sigma_{i} & : S_{C_{1}(i)} \rightarrow S_{C_{2}(i)}.
\end{align*}
\]

That is, for a state \( s \) in \( S_{1} \), \( \sigma(s) \) is given by

\[
\sigma(s) = (\sigma_{B_{sc}}(s_{B_{sc}}), \sigma_{B_{S}}(s_{B_{PKI}}), \sigma_{B_{SM}}(s_{B_{SM}}^{*}), \sigma_{1}(s_{C_{1}(1)}), \ldots, \sigma_{n}(s_{C_{1}(n)})).
\]

We now describe the map \( \sigma_{i} \). Since this map only depends on the state of \( C_{1}(i) \), it is fully described by Figures 17, 18, 19 and 20: For each element of \( S_{C_{1}(i)} \), the parameters in the leftmost column are simply mapped to the equivalent parameters in the rightmost column, thus giving an element of \( S_{C_{2}(i)} \). By pairwise inspection of the elements, we observe that each element of \( S_{C_{1}(i)} \) contains exactly the same information as the corresponding element of \( S_{C_{2}(i)} \). This implies that \( \sigma_{i} \) is a bijection. Another important property of \( \sigma_{i} \) is preservation of wait states. More precisely, \( \sigma_{i} \) gives a one-to-one correspondence between wait states of \( S_{C_{1}(i)} \) and wait states of \( S_{C_{2}(i)} \).

We proceed by describing the maps \( \sigma_{B_{sc}}, \sigma_{B_{S}} \) and \( \sigma_{B_{SM}} \), which simply copy the states of the respective buffers, that is, for a state \( s \) of \( S_{1}' \), \( \sigma_{B_{sc}}(s_{B_{sc}}) = s_{B_{sc}} \), \( \sigma_{B_{S}}(s_{B_{PKI}}) = s_{B_{PKI}} \) and \( \sigma_{B_{SM}}(s_{B_{SM}}^{*}) = s_{B_{SM}}^{*} \). It is clear that \( \sigma_{B_{sc}}, \sigma_{B_{S}} \) and \( \sigma_{B_{SM}} \) are all bijective. In particular, since every state of \( B_{sc}^{*}, B_{S} \) and \( B_{SM} \) is a wait state, each map gives a one-to-one correspondence between wait states.

Given that \( \sigma_{i}, \sigma_{B_{sc}}, \sigma_{B_{S}} \) and \( \sigma_{B_{SM}} \) are all bijective, we conclude that \( \sigma : S_{1} \rightarrow S_{2} \) is a bijection. We clearly have \( \sigma(s_{1}) = s_{2} \). Furthermore, the fact that \( \sigma_{i}, \sigma_{B_{sc}}, \sigma_{B_{S}} \) and \( \sigma_{B_{SM}} \) all give one-to-one correspondences between wait states implies that \( \sigma(W_{1}) = W_{2} \).

The further strategy of the proof is to consider each message \( w \) in \( I_{1} \), and verify that, for all \( s \in S_{1} \), \( f_{1}(s, w) \) is defined if and only if \( f_{2}(\sigma(s), \tau(w)) \) is defined, and if they are defined, then \( \sigma(f_{1}(s, w)) = f_{2}(\sigma(s), \tau(w)) \).

We start by considering the message \( w = (Z, P_{i}, SM/Register) \). Note that \( \tau(w) = w \). Let \( S \) be the subset of states in \( S_{1} \) where \( C_{1}(i) \) is in state \( ((s_{0}, -), (s_{0}, -), (s_{0}, -)) \) or \( ((s_{5}, p_{k_{1}^{i}}, p_{k_{1}^{i}}), (s_{6}, -), (s_{5}, -)) \), while every other machine is in a wait state. \( w \) is recognized by \( M_{1} \) exactly when its state belongs to \( S \).

On the other hand, let \( S' \) be the subset of states in \( S_{2} \) where \( C_{2}(i) \) is in state \( ((s_{0}, -), (s_{0}, -)) \) or \( ((s_{5}, -), (s_{7}, p_{k_{1}^{i}}, p_{k_{1}^{i}})) \), while every other machine is in a wait state. \( \tau(w) \) is recognized by \( M_{2} \) exactly when its state belongs to \( S' \). Note that \( \sigma_{i}((s_{0}, -), (s_{0}, -), (s_{0}, -)) = ((s_{0}, -), (s_{0}, -)), \sigma_{i}((s_{5}, p_{k_{1}^{i}}, p_{k_{1}^{i}}), (s_{6}, -), (s_{5}, -)) = ((s_{5}, -), (s_{7}, p_{k_{1}^{i}}, p_{k_{1}^{i}})) \). The one-to-one correspondences between wait states provided by \( \sigma \) then implies that \( \sigma \) gives a one-to-one correspondence between elements of \( S \) and elements of \( S' \), hence \( f_{1}(s, w) \) is defined if and only if \( f_{2}(\sigma(s), \tau(w)) \) is defined. Furthermore, note that transitions involving \( w \) only
affect the state of $C_1(i)$ when recognized by $M_1$, and the state of $C_2(i)$ when recognized by $M_2$. It can thus be verified that $\sigma(f_1(s, w)) = f_2(\sigma(s), \tau(w))$, simply by inspection of Figure 13.

Next we consider the message $w = (P_i, F_{SC}, SC.KeyGen)$. Note that $\tau(w) = (F_{SM}, S, SM.Register, P_i)$. Let $S$ be the subset of states in $S_1$ where $C_1(i)$ is in state $((s_0, -), (s_1, -), (s_0, -))$, while every other machine is in a wait state. $w$ is recognized by $M_1$ exactly when its state belongs to $S$. On the other hand, let $S'$ be the subset of states in $S_2$ where $C_2(i)$ is in state $((s_1, -), (s_0, -))$, while every other machine is in a wait state. $\tau(w)$ is recognized by $M_2$ exactly when its state belongs to $S'$. Note that $\sigma_i((s_0, -), (s_1, -), (s_0, -)) = ((s_1, -), (s_0, -))$. The one-to-one correspondences between wait states provided by $\sigma$ then implies that $\sigma$ gives a one-to-one correspondence between elements of $S$ and elements of $S'$, hence $f_1(s, w)$ is defined if and only if $f_2(\sigma(s), \tau(w))$ is defined. Furthermore, note that transitions involving $w$ only affect the state of $C_1(i)$, while transitions involving $\tau(w)$ only affect the state of $C_2(i)$. It can thus be verified that $\sigma(f_1(s, w)) = f_2(\sigma(s), \tau(w))$, simply by inspection of Figure 13.

Generally, when the transitions involving a message $w$ only concern the machine $C_1(i)$ of $M_1$, and the transitions involving $\tau(w)$ only concern $C_2(i)$ of $M_2$, the properties in question can be verified by inspection of Figures 17, 18, 19 and 20, and repeating the above argument. In the following, we only comment on messages where the transitions involve other machines as well.

We thus proceed by considering the message $w = (F_{SC}, B_{SC}, B.Add, (P_i, pk^*_i, pk^*_i))$. Note that $\tau(w) = (S, B^*_{SC}, B.Add, (P_i, pk^*_i, pk^*_i))$. As for verifying that $f_1(s, w)$ is defined if and only if $f_2(\sigma(s), \tau(w))$ is defined, note that $w$ and $\tau(w)$ will always be recognized by $B_{SC}$ and $B^*_{SC}$, respectively. The transitions involving $w$ and $\tau(w)$ are thus independent of the states of these machines, and the above strategy still applies: We identify subsets $S$ and $S'$ by inspection of Figures 17, 18, 19 and 20, and use the definition of $\sigma_i$ and the one-to-one correspondences between wait states provided by $\sigma$ to argue that $\sigma$ gives a one-to-one correspondence between elements of $S$ and elements of $S'$. As for showing that $\sigma(f_1(s, w)) = f_2(\sigma(s), \tau(w))$, we make the following observation: $\sigma(f_1(s, w))$ corresponds to first adding the triple $(P_i, pk^*_i, pk^*_i)$ to $B_{SC}$, and then copying this triple to $B^*_{SC}$. $f_2(\sigma(s), \tau(w))$ corresponds to adding the triple $(P_i, pk^*_i, pk^*_i)$ directly to $B^*_{SC}$. It is clear that, in either case, the resulting state of $B^*_{SC}$ is the same. Regarding $C_2(i)$, inspection of Figure 13 confirms that $\sigma(f_1(s, w))$ and $f_2(\sigma(s), \tau(w))$ correspond to the same states of this machine. Regarding the other messages containing $B.Add$, the properties in question can be proved by the above procedure, using that $\sigma$ copies elements from $B_{SC}$ to $B^*_{SC}$, elements from $B_{PKI}$ to $B_S$, and elements from $B^*_{SM}$ to $B_{SM}$.

We now consider the message $w = (F_{PKI}, B_{PKI}, B.Lookup, (P_j, -), (P_j, (pk^*_j, pk^*_j)))$. For $w$ to be recognized by $M_1$, it must be recognized both by $M_{PKI}(i)$ and the buffer machine $B_{PKI}$. The latter is the case if $B_{PKI}$ has a pair $(P_j, (pk^*_j, pk^*_j))$ in its state. Similarly, for $\tau(w) = (S, B_S, B.Lookup, (P_j, -), (P_j, (pk^*_j, pk^*_j)))$ to be recognized by $M_2$, it must be recognized both by $M_S(i)$ and the buffer machine...
$B_S$. The latter is the case if $B_S$ has a pair $(P_j, (pk_j^s, pk_j^r))$ in its state. Since $\sigma$ copies elements from $B_{PKI}$ to $B_S$, we deduce that $w$ is recognized by $B_{PKI}$ when $M_1$ is in state $s$ if and only if $\tau(w)$ is recognized by $B_S$ when $M_2$ is in state $\sigma(s)$. Again, we may identify subsets $S$ and $S'$ by inspection of Figures 17, 18, 19 and 20, and use the definition of $\sigma_i$ and the one-to-one correspondences between wait states provided by $\sigma$ to argue that $\sigma$ gives a one-to-one correspondence between elements of $S$ and elements of $S'$. This means that $f_1(s, w)$ is defined if and only if $f_2(\sigma(s), \tau(w))$ is defined. Finally, since the transitions only affect $C_1(i)$ and $C_2(i)$, we may conclude that $\sigma(f_1(s, w)) = f_2(\sigma(s), \tau(w))$, simply by inspection of Figures 18 and 19.

Concerning the other messages containing $B_{Lookup}$, the properties in question can be proved by the above procedure, using that $\sigma$ copies elements from $B_{SC}$ to $B_{SC}'$, elements from $B_{PKI}$ to $B_S$, and elements from $B_{SM}^*$ to $B_{SM}$.

This means that $(\sigma, \tau) : M_1^{(18)} \rightarrow M_2^{(22)}$ is an isomorphism. By Lemma 1, the induced language map is a bijection, and since $\tau$ leaves $I$ fixed, it follows from Lemma 3 that $M_1^{(18)}$ and $M_2^{(22)}$ are equivalent with respect to $I$. □

This completes our proof, and we conclude that the machines $M_1$ and $M_2$ are equivalent with respect to $I$. By Theorem 9, the protocol $\pi_{SM}$ securely realizes the functionality $\mathcal{F}_{SM}$ in the $(\mathcal{F}_{PKI}, \mathcal{F}_{SC})$-hybrid model. □

References


### Figure 13. Registration

<table>
<thead>
<tr>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1,□)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(s20,□)</td>
</tr>
<tr>
<td>(s6,□)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(s5,□)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1,□)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(s20,□)</td>
</tr>
<tr>
<td>(s6,□)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(s5,□)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1,□)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(s20,□)</td>
</tr>
<tr>
<td>(s6,□)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(s5,□)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1,□)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(s20,□)</td>
</tr>
<tr>
<td>(s6,□)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(s5,□)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
<th>(s0,□)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1,□)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(s20,□)</td>
</tr>
<tr>
<td>(s6,□)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(Pi,Z,SM.Error)</td>
<td>(s5,□)</td>
<td></td>
</tr>
</tbody>
</table>
Figure 14. Encryption
Figure 16. Special cases during encryption and decryption

<table>
<thead>
<tr>
<th>$M_{SC}(i)$</th>
<th>$M_{s}(i)$</th>
<th>$M_{PKI}(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(s_5, p^k_i, p^r_i)$</td>
<td>$(s_8, m)$</td>
<td>$(s_8, P_j)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(s_{10}, -)$</td>
</tr>
<tr>
<td>$(s_9, -)$</td>
<td>$(s_9, -)$</td>
<td>$(s_{11}, -)$</td>
</tr>
<tr>
<td>$(s_{14}, c)$</td>
<td>$(s_8, P_j)$</td>
<td>$(s_{10}, p^k_i, p^r_i, P_j, [m])$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(s_{11}, p^k_i, p^r_i, P_j)$</td>
</tr>
<tr>
<td></td>
<td>$(s_8, P_j, m, \perp)$</td>
<td>$(s_8, \perp)$</td>
</tr>
<tr>
<td>$(s_6, -)$</td>
<td>$(s_6, -)$</td>
<td>$(s_7, \perp)$</td>
</tr>
</tbody>
</table>

$\mathcal{F}_{PKI}, B_{PKI}, B$ Lookup, ($P_j, -), (P_j, \perp)$

$\mathcal{F}_{SM}, B_{SM}, B$ Lookup, ($P_j, \perp$)

$(s_7, P_j, m)$

$(s_{10}, p^k_i, p^r_i, P_j)$

$(s_9, \perp)$

$(s_8, P_j, m, \perp)$

$(s_8, \perp)$

$(s_9, \perp)$

$(s_{11}, -)$

$(s_{12}, -)$

$(s_{13}, P_j, c)$

$(s_{14}, P_j, c, \perp)$

$(s_{15}, c, \perp)$

$(s_{16}, p^k_i, p^r_i, P_j)$

$(s_{17}, p^k_i, p^r_i, P_j, [m])$
<table>
<thead>
<tr>
<th>(i, □)</th>
<th>(i, □)</th>
<th>(i, □)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s0, □)</td>
<td>(s0, □)</td>
<td>(s0, □)</td>
</tr>
<tr>
<td>(s1, □)</td>
<td>(Z, Pi, SM.Register)</td>
<td>(Z, Pi, SM.Register)</td>
</tr>
<tr>
<td>(s20, □)</td>
<td>(s6, □)</td>
<td>(Pi, Z, SM.Error)</td>
</tr>
<tr>
<td>(s5, □)</td>
<td>(Pi, Z, SM.Error)</td>
<td>(s5, □)</td>
</tr>
</tbody>
</table>

Figure 17. Modified registration
<table>
<thead>
<tr>
<th>$M_{21}(t)$</th>
<th>$M_{2}(t)$</th>
<th>$M_{EXP21}(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(s_0, -)$</td>
<td>$(s_0, -)$</td>
<td>$(s_0, -)$</td>
</tr>
<tr>
<td>$(s_0, -)$</td>
<td>$(s_0, -)$</td>
<td>$(Z, P; \text{SM Encrypt}, P; m)$</td>
</tr>
<tr>
<td>$(s_0, -)$</td>
<td>$(s_0, -)$</td>
<td>$(Z, P; \text{SM Error})$</td>
</tr>
<tr>
<td>$(s_0, -)$</td>
<td>$(s_0, -)$</td>
<td>$(s_0, -)$</td>
</tr>
</tbody>
</table>

Figure 18. Modified encryption
Figure 19. Modified decryption
### Figure 20. Special cases during modified encryption and decryption

<table>
<thead>
<tr>
<th>$M_{SC}(i)$</th>
<th>$M_{S}(i)$</th>
<th>$M_{PKI}(i)$</th>
<th>$M_{SM}(i)$</th>
<th>$M_{S}(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(s_5, pk_i)$</td>
<td>$(s_5, pk_i)$</td>
<td>$(s_8, P_j)$</td>
<td>$(s_7, P_j, m)$</td>
<td>$(s_{10}, pk_j, P_j,</td>
</tr>
<tr>
<td>$(s_5, pk_i)$</td>
<td>$(s_5, pk_i)$</td>
<td>$(F_{PKI}, F_{PKI}, B_{Lookup}, (P_j, --), (P_j, □))$</td>
<td>$(s_{12}, pk_i, pk_j, P_j)$</td>
<td>$(s_{12}, pk_i, pk_j, P_j)$</td>
</tr>
<tr>
<td>$(s_5, pk_i)$</td>
<td>$(s_5, pk_i)$</td>
<td>$(F_{PKI}, F_{PKI}, SM_{Ciphertext}, P_j, P_j, □)$</td>
<td>$(s_{13}, P_j, c)$</td>
<td>$(s_{13}, P_j, c)$</td>
</tr>
<tr>
<td>$(s_5, pk_i)$</td>
<td>$(s_5, pk_i)$</td>
<td>$(F_{PKI}, B_{SM}, B_{Lookup}, P_j, □)$</td>
<td>$(s_{14}, P_j, c, □)$</td>
<td>$(s_{14}, P_j, c, □)$</td>
</tr>
<tr>
<td>$(s_5, pk_i)$</td>
<td>$(s_5, pk_i)$</td>
<td>$(F_{PKI}, B_{SM}, B_{Lookup}, P_j, □)$</td>
<td>$(s_{16}, pk_i, pk_j, P_j)$</td>
<td>$(s_{16}, pk_i, pk_j, P_j)$</td>
</tr>
<tr>
<td>$(s_5, pk_i)$</td>
<td>$(s_5, pk_i)$</td>
<td>$(F_{PKI}, F_{PKI}, PKI_{Retrieve}, Fail)$</td>
<td>$(s_{17}, pk_i, pk_j)$</td>
<td>$(s_{17}, pk_i, pk_j)$</td>
</tr>
<tr>
<td>$(s_6, -)$</td>
<td>$(s_6, -)$</td>
<td>$(P_j, Z, SM_{Not Registered})$</td>
<td>$(s_{15}, P_j, □)$</td>
<td>$(s_{15}, P_j, □)$</td>
</tr>
<tr>
<td>$(s_6, -)$</td>
<td>$(s_6, -)$</td>
<td>$(P_j, Z, SM_{Not Registered}, c)$</td>
<td>$(s_{15}, □, □)$</td>
<td>$(s_{15}, □, □)$</td>
</tr>
</tbody>
</table>


